UNIVERSITÀ DEGLI STUDI DI PALERMO

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA INFORMATICA
SEDE DI PALERMO

PROGETTAZIONE E IMPLEMENTAZIONE DI UN LINGUAGGIO PER LA CREAZIONE ED ESECUZIONE IN SIMULAZIONE E NELLA REALTÀ DEI COMPORTAMENTI COMPLESSI PER UN ROBOT UMANOIDE ROBOVIE-M

Tesi di laurea di:

Allievo Ing. Gaspare Galuppo

Relatori:

Prof. Ing. Antonio Chella

Dott. Ing. Rosario Sorbello

ANNO ACCADEMICO 2005 - 2006

Indice

Capitolo 3

Lo studio dei movimenti

Capitolo 4

La creazione di un movimento robotico

Capitolo 5

La programmazione del movimento per "Robovie-M"

Capitolo 6

Il software "RobovieMovement"

Capitolo 7

I risultati sperimentali ottenuti su "Robovie-M"

Appendice A

I codici sorgenti

Appendice B

L'installazione

Prefazione

Sono già passati circa trent'anni da quando, al dipartimento di "Scienza ed Ingegneria" della "Waseda University" di Tokio, nasceva il primo vero robot umanoide, il robot "Wabot-1".

Da allora, anche grazie al fatto che sono nate tecnologie sempre migliori, i robot umanoidi si stanno ritagliando una posizione sempre più predominante nella vita comune, occupando svariati campi, dall'intrattenimento a campi più importanti e complessi, come quelli che riguardano l'assistenza.

La presente tesi affronta il problema della progettazione e della implementazione di un linguaggio, per la programmazione dei comportamenti di un robot umanoide, e della creazione di una classe di movimenti che costituiranno parte di una libreria.

Oggetto del mio studio è stato il robot umanoide "Robovie-M", sul quale sono stati eseguiti e raggiunti i risultati prefissati.

Il lavoro fa parte di un progetto sperimentale in corso presso il dipartimento di Ingegneria Informatica dell'Università degli Studi di Palermo, che prevede la realizzazione e l'implementazione di un software per la completa gestione del robot.

Nel presente documento vengono inizialmente analizzate le problematiche relative alla movimentazione di un robot umanoide, ripercorrendo in prima battuta la storia di questa particolare branca della robotica mobile.

Viene mostrata una descrizione delle caratteristiche concernenti lo sviluppo e la costruzione di un linguaggio e, dopo aver esposto tali principi, viene descritto quale è l'approccio classico che viene utilizzato per la programmazione di un movimento robotico, discutendo quali sono i metodi attuali che vengono impiegati per la programmazione dei movimenti del robot umanoide "Robovie-M".

Di seguito è presentata la soluzione sviluppata e proposta, mostrando le caratteristiche e le peculiarità del software creato ed in ultimo sono presentati i risultati raggiunti.

La tesi di laurea è così strutturata:

Capitolo 1: La robotica umanoide, effettua una introduzione sulla robotica umanoide e sulla robotica bipede, presentando il robot "Robovie-M V3".

Capitolo 2: **La costruzione di un linguaggio,** descrive quali sono le componenti fondamentali di un programma traduttore e le fasi del processo di traduzione.

Capitolo 3: Lo studio dei movimenti, effettua una studio analitico dei movimenti che possono essere eseguiti da un bipede ed in particolar modo da un robot.

Capitolo 4: La creazione di un movimento, illustra l'approccio classico della programmazione di un movimento robotico.

Capitolo 5: La programmazione di un movimento per "Robovie_M", riporta il modo di programmare il robot "Robovie-M".

Capitolo 6: Il software "RobovieMovement", discute gli obiettivi e le scelte progettuali prese per lo sviluppo della soluzione proposta.

Capitolo 7: I risultati sperimentali ottenuti su "Robovie-M" presenta tutti i risultati sperimentali ottenuti mediante l'utilizzo del software proposto.

Introduzione

La ricerca sui robot umanoidi è, negli ultimi anni, in forte espansione.

Questo tipo di "macchine" è generalmente considerata una particolare evoluzione del campo della robotica mobile poiché un robot, dotato di gambe ed in particolare se bipede, può operare in un ambiente umano in modo più efficiente di quelli su ruote, soprattutto se in presenza di scale o ostacoli simili.

Un robot bipede deve essere in grado di camminare con un'andatura stabile e con un'ampia tolleranza ai disturbi, ma deve anche avere comportamenti più complessi come interagire efficientemente con il proprio ambiente, talvolta cooperando con gli stessi esseri umani o con altri robot.

Sfortunatamente, questi tipi di performance allo stato attuale non sono state totalmente raggiunte e lo studio, in questo campo, si trova ancora al suo periodo iniziale.

Gli aspetti da migliorare sono molteplici, legati principalmente alla configurazione meccanica degli stessi robot ed agli algoritmi di controllo: il problema della camminata, per esempio, noto per la sua complessità, è reso particolarmente difficile dalla sua non-linearità e dal fatto di essere un processo di per se caotico e instabile, la cui dinamica è soggetta a fenomeni di discontinuità e variazioni temporali.

Non è casuale pertanto che, oggigiorno, i robot umanoidi siano considerati una piattaforma ideale per sperimentare algoritmi di intelligenza artificiale e di "*reinforcement learning*"[11].

Una delle domande più comuni rimane comunque: "Quale sarà l'impiego di questi robot e perché è interessante indagare in un campo così complesso e ampio?"

Non esiste una sola risposta: i robot bipedi possono essere utilizzati per sostituire gli umani in ambienti pericolosi, per esempio ambienti in cui sono presenti gas tossici o elementi chimici pericolosi, o per effettuare processi di bonifica di terreni minati [3].

Un'altra applicazione può essere quella che riguarda l'assistenza agli anziani, un ambito che trova numerose applicazioni in Giappone, il paese più attivo del mondo in questo settore, campo che troverà sicuramente terreno fertile dal momento che la vita media dei paesi del mondo industrializzato si sta innalzando di decennio in decennio [4].

I robot umanoidi sono infine un soggetto interdisciplinare, collegati a diversi campi quali la meccanica, l'automazione, l'informatica e la bio-meccanica; la connessione con quest'ultimo campo è probabilmente una delle più importanti e andrà aumentando in futuro.

La ricerca, infatti, presuppone di costruire una macchina che imiti una funzione degli esseri umani, senza esattamente sapere come il nostro sistema nervoso può controllare i movimento muscolo-scheletrici in maniera cosi efficiente: per questa ragione i dati accumulati dalla medicina sono di certo un buon punto di riferimento per iniziare lo studio dei movimenti umani da applicare a quelli robotici.

Questa relazione, se osservata da un altro punto di vista, porta ad immaginare che un giorno, quando si sarà in possesso di robot umanoidi dotati di una maggiore capacità di adattamento, gambe artificiali o altre protesi saranno testate prima su robot antropomorfi che sull'uomo.

Capitolo 1

La robotica umanoide

1.1 Introduzione

La "*Robotica Umanoide*" è quella "*branca della BioRobotica il cui scopo è riprodurre, il più fedelmente possibile, alcune capacità, comportamenti e caratteristiche morfologiche umane*" [1] .

Essendo la robotica umanoide una branca della *BioRobotica*, essa fa parte di quella che viene normalmente chiamata in letteratura "*robotica di servizio*".

Diverse sono le motivazioni per le quali si cerca di realizzare robot che siano simili agli esseri umani ed abbiano comportamenti affini.

Un motivo viene dato dal fatto che il robot deve poter interagire con l'uomo nella maniera più naturale e semplice possibile.

Filoni di ricerca riguardano, sotto questo settore, lo sviluppo di applicazioni basate su comunicazioni gestuali, ed esempi sono le numerose applicazioni sviluppate per utenti disabili, o comunicazioni vocali, come il robot "*CiceRobot*", che è un robot museale "parlante".

Un altro motivo è dato dal fatto che il robot deve operare in ambienti che sono stati progettati e costruiti in base alla conformazione fisica ed alle esigenze dell'uomo.

Il robot deve pertanto essere in grado di manipolare oggetti o attrezzi specificatamente progettati per l'uomo e deve sapersi muovere in ambienti "umani". Il robot, ad esempio, deve essere in grado di salire e scendere le scale o aggirare ostacoli normalmente presenti in un ambiente.

Un robot deve poter cooperare ed eventualmente effettuare assistenza ad esseri umani; il campo dell'assistenza agli anziani è un campo in cui la ricerca è in grande ascesa.

Inoltre non deve mai, in ogni caso, nuocere l'uomo.

Per poter far fronte a tutti questi obiettivi, ci si è accorti che per risolvere determinati problemi mediante applicazioni robotiche, come problemi di automazione industriale, problemi riguardanti ispezione di ambienti o tele-manipolazione, le normali architetture adottate non sono sufficienti.
E' necessario sviluppare dei robot in grado di imitare l'uomo e di riprodurne alcune funzionalità.
Il problema di progettare un umanoide che sia dotato di tutte le caratteristiche morfologiche umane, è, però, un traguardo estremamente complesso.
Molti ricercatori preferiscono affrontare problemi più a basso livello, come problemi riguardanti la movimentazione o problemi riguardanti il riconoscimento di features oppure, passando ad un più alto livello, problemi che riguardano il ragionamento automatico e la deliberazione.
Ma anche a questi livelli, difficili risulta essere il raggiungimento degli obiettivi.
Per quanto riguarda il problema della movimentazione, problema che ho affrontato da vicino durante lo sviluppo della presente tesi, se si effettua un paragone uomo-robot considerando i gradi di liberta in gioco, l'uomo comune possiede circa 92 gradi di libertà [2].
Nell'ambito della robotica umanoide, ad oggi i principali filoni di ricerca riguardano:

- Gli studi sui sistemi di locomozione su due arti (robot bipedi);
- Gli studi su sistemi di manipolazione (mano artificiali);
- Gli studi sui sistemi di coordinamento Visione-Manipolazione;
- Gli studi su sistemi di controllo ad alto livello e di ragionamento artificiale (metodi di inferenza automatica di intelligenza artificiale);
- Gli studi su sistemi di controllo in grado di esprimere semplici stati d'animo (rabbia, felicità, stupore,indifferenza,paura).

1.2 Le tappe fondamentali della robotica umanoide

Il primo sistema robotico *simil-umanoide* nasce circa tremila anni fa, all'apice della civiltà egiziana dove, statue articolate, potevano essere controllate da operatori nascosti. A Tebe, il nuovo re, era scelto da una statua articolata di Ammone che veniva controllata segretamente, tramite meccanismi sotterranei nascosti, dai Sommi Sacerdoti.

Ma è il sedicesimo secolo, con il più grande genio di tutti i tempi, *Leonardo Da Vinci*, che vede fiorire il primo progetto di robot umanoide. Il principale studente di anatomia umana del suo tempo riuscì infatti a concepire e a progettare un equivalente meccanico di un essere umano. Purtroppo questo progetto, visibile in figura 1.1, non è ancora stato realizzato.

Figura 1.1: Il cavaliere automa di Leonardo.

All'inizio del diciottesimo secolo il francese Jacques de Vaucanson costruì tre meccanismi umanoidi: un suonatore di mandolino che cantava e batteva il piede mentre suonava, un suonatore di piano che simulava la respirazione e muoveva la testa e infine un suonatore di flauto.

Tutti erano semplici meccanismi di riproduzione, costruiti per essere molto realistici, ma nessuno di essi era in grado di percepire l'ambiente.

Sempre nel diciottesimo secolo, l'orologiaio svizzero Pierre Jacquet-Droz e suo figlio Henri-Louis costruirono un certo numero di umanoidi, tra cui una suonatrice di organo che simulava la respirazione, che è possibile osservare nella figura 1.2, e la direzione dello sguardo fisso verso il pubblico.

Figura 1.2: Gli androidi Jacquet-Droz.

1.2.1 La nascita del primo umanoide: Wabot-1

L'era moderna della robotica umanoide inizia però nel 1973 grazie a Hirokazu Kato, un professore della "*Waseda University*", a Tokyo.

Fu lui a dirigere la costruzione di *Wabot-1*, che è possibile osservare nella figura 1.3a, il primo dei robot umanoidi.

Questo robot è in grado di muovere piccoli passi grazie a due gambe, afferrare semplici oggetti con due mani ed eseguire qualche interazione di discorso basilare con le persone. Il robot possiede un sistema di controllo della camminata, che gli fa compiere camminate statiche, un sistema di visione con telecamere non orientabili ed un sistema vocale.

La sua andatura è abbastanza lenta e si muove nell'ambiente grazie alla capacità di valutare direzioni e distanze dagli oggetti usando sensori esterni.

Wabot-1, in realtà, è solamente il primo di una lunga serie di prototipi costruiti nei rinomati laboratori della *Waseda University*; ne è un esempio *Wabian*, robot

umanoide di ultima generazione equipaggiato con un sistema di controllo in grado di replicare la camminata umana.

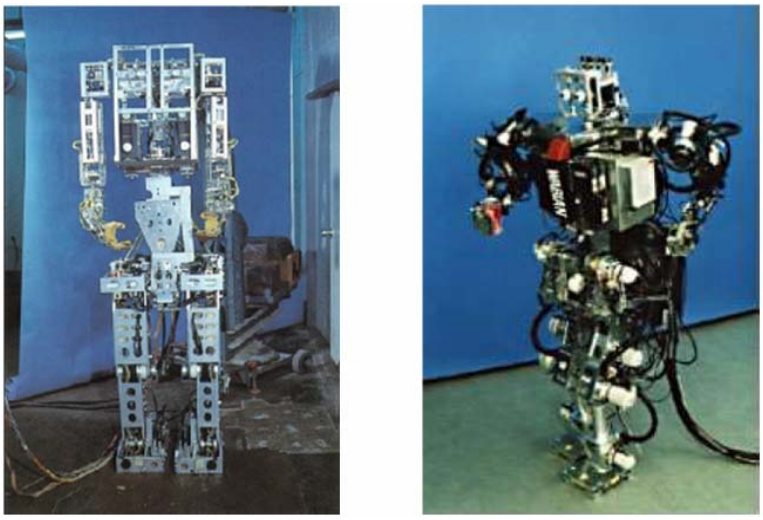

Figura 1.3: I robots Wabot-1 e Wabian.

Il robot *Wabian*, presente in figura 1.3b, è alto 1,88 m, pesa ben 130 Kg. Utilizza un sofisticato sistema di bilanciamento che, coordinando in modo opportuno il movimento delle gambe e del busto, è in grado di mantenerlo in equilibrio, in particolare la camminata è bilanciata utilizzando il movimento del tronco mentre i movimenti sono calcolati off-line con un metodo di learning automatico.

I giunti dei piedi hanno inoltre un alto coefficiente di smorzamento per assorbire gli urti.

Gli obiettivi di *Wabian* sono quelli di sviluppare il meccanismo di controllo del movimento degli esseri umani dal punto di vista della robotica e stabilire una base tecnologica per costruire il personal robot del futuro.

1.2.2 Cog e Kismet: frontiere aperte alla robotica emozionale

Nella stessa direzione ha lavorato anche il team del Prof. Roodney Brooks, dell' "*Artificial Intelligent Laboratory*" all' *MIT* di Boston, negli Stati Uniti dove, nel 1993, è partito il progetto "*Cog*".

Questo progetto aveva essenzialmente due obiettivi principali: uno di tipo ingegneristico che era quello di costruire un robot "*general purpose*", ed uno di tipo scientifico che aveva l'obiettivo di comprendere meglio i meccanismi della percezione umana.

Cog, che è possibile osservare nella figura 1.4, è un robot umanoide che possiede 21 gradi di libertà e che gli permettono di replicare, in modo apprezzabile, i movimenti della parte superiore del corpo umano.

Ogni giunto è controllato da un sistema hardware dedicato che a sua volta riceve i comandi da un sistema di controllo di più alto livello, ed è proprio quest'ultimo avere lo scopo di coordinare i movimenti di tutti i giunti per ottenere manipolazioni complesse.

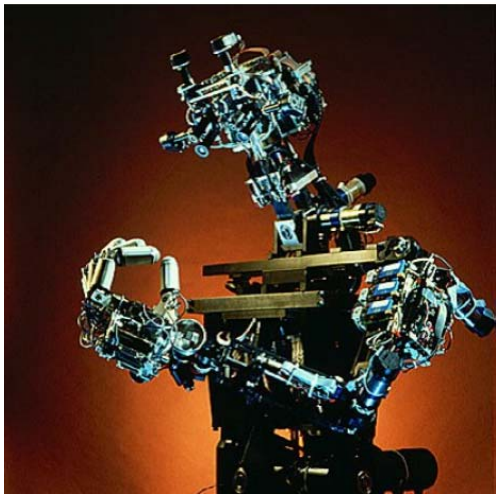

Figura 1.4: Il robot umanoide Cog.

Rilevante è la presenza di numerosi apparati sensoriali che permettono al robot di avere una percezione di se stesso e dell'ambiente che lo circonda. Il robot è in grado, grazie ad un sistema vestibolare artificiale, di orientare lo sguardo verso eventuali fonti di stimolazione esterne come suoni, oggetti in movimento o contatti fisici.

Il "cervello" del robot è contenuto in un network di calcolatori che, in tempo reale, riescono ad elaborare le informazioni provenienti dai sensori e a controllare quindi la postura e la reattività dell'umanoide.

Cog è usato anche nell'ambito della ricerca sugli aspetti dell'interazione sociale e, per questo stesso scopo e stato sviluppato *Kismet*, che e possibile osservare nella figura 1.5.

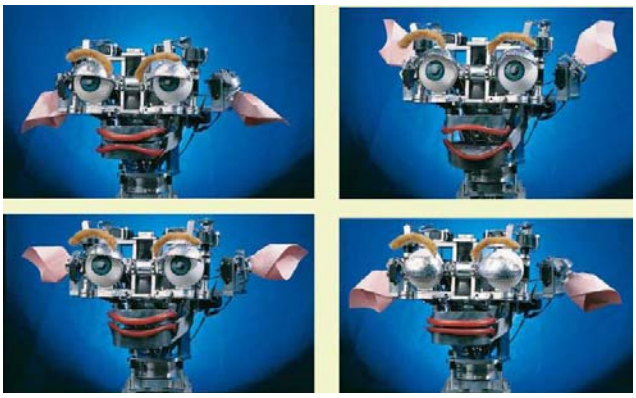

Figura 1.5: Alcune espressioni facciali del robot umanoide Kismet.

Allo stesso modo di Cog, Kismet possiede un sistema di visione attivo, cioè un sistema di visione con telecamere orientabili, che si muovono in determinate direzioni come parte di un processo di percezione.

Un sistema di "*visione umanoide*" con la stessa struttura meccanica di base degli umani e degli altri mammiferi e che segue le stesse primitive motorie usate dagli umani.

Sia in *Cog* che in *Kismet* sono state implementate varie capacità tipiche dell'occhio umano, con prestazioni paragonabili a quelle degli umani, anche se, le loro telecamere, hanno una risoluzione globale molto più bassa di quella dell'occhio.

Cog e *Kismet* possiedono capacità che consentono di percepire aspetti 3D del mondo, inoltre, sono in grado di individuare i volti umani attraverso una varietà di funzioni e stimare la direzione dello sguardo fisso di una persona, determinando la direzione che i loro occhi stanno puntando.

L'artificio di basso livello che permette questa coerenza è un *meccanismo di attenzione visuale*, che determina dove il robot sta guardando. La direzione dello sguardo del robot determina ciò che viene visto da tutti i processi di percezione di basso livello. Questo a sua volta determina quali comportamenti del robot

devono essere eseguiti. La coerenza di comportamento del robot non è determinata da qualche meccanismo di bloccaggio interno ma dalla direzione dello sguardo del robot sul mondo.
I sistemi visivi di *Cog* e *Kismet* costituiscono la base per le loro interazioni sociali. Anche un semplice osservatore umano può capire a cosa i robot stanno prestando attenzione dalla direzione del loro sguardo. Allo stesso modo i robot possono capire a cosa una persona sta prestando attenzione, dalla direzione dello sguardo della persona.

1.2.3 Il robot umanoide Asimo della Honda

La massima espressione della robotica umanoide può essere osservata nel robot Asimo, visibile in figura 1.6, importante robot umanoide sviluppato presso i laboratori giapponesi della Honda.
Asimo nasce come robot umanoide realizzato per simulare la camminata umana, sia statica che dinamica, il numero di gradi di liberta delle sue gambe, infatti, è ridotto al minimo indispensabile, con sei gradi di liberta per ogni gamba.

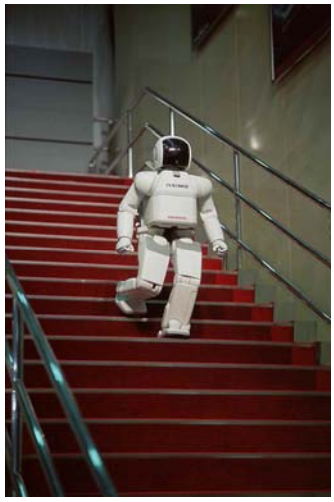

Figura 1.6: Il robot umanoide Asimo.

Ben quattordici gradi di liberta, distribuiti sul corpo del robot, permettono ad Asimo di mantenere l'equilibrio in ogni situazione.

I piedi hanno un sistema di assorbimento delle forze di impatto che consente di ammortizzare l'andatura e di avere una minore sollecitazione dei giunti.
Il robot possiede dimensioni ridotte principalmente per due motivi: innanzitutto i motori degli attuatori peserebbero troppo e occuperebbero uno spazio eccessivo, inoltre la batteria necessaria per il funzionamento di questi motori avrebbe dimensioni e peso rilevanti.
Di recente presso la Honda Motor, a Tokyo, è stato annunciato lo sviluppo di nuove tecnologie per il robot Asimo di prossima generazione.
Si cerca di raggiungere un nuovo livello di mobilità che permetterà ad Asimo di funzionare e interagire con gli esseri umani processando più velocemente le informazioni e agendo più agilmente negli ambienti del mondo reale.
Le tecnologie chiave sono le seguenti: una tecnologia di controllo della postura che renda possibile correre in modo naturale come un essere umano, una tecnologia di movimento continuo autonomo che permetta la creazione di un percorso flessibile fino alla destinazione, come la pianificazione del percorso aggiornata continuamente e tecnologie basate su sensori di forza e visivi perfezionati che permettano una interazione più facile con gli esseri umani.

1.3 Il robot Robovie-M V3

La piattaforma meccanica utilizzata in questo lavoro, è stato il robot umanoide *Robovie-M* nella sua terza versione, prodotto dalla ditta giapponese *VStone*.
Il robot è un sistema bipede completamente autonomo, nel senso che tutti i sistemi necessari alla sua movimentazione, incluse le batterie, sono on board. I movimenti vengono eseguiti dal robot, poiché trasferiti alla scheda di controllo, tramite seriale RS232.
Robovie-M V3 è alto circa 30 cm per un peso complessivo di 2 Kg ed è dotato, nella sua versione base, di un processore H8-3687 prodotto dalla ditta Renesas.
Il processore, integrato nella scheda di controllo viene alimentato ad una tensione costante di 6V, ha una potenza di calcolo di 20 Mhz.

Il robot, mostrato in figura 1.7, possiede 22 giunti di tipo *Hinge,* (giunti a cerniera) che possono ruotare fino a 255 gradi. I giunti che sono presenti negli arti inferiori del robot, sono dotati di una coppia maggiore rispetto a quelli degli arti superiori per garantire maggiore stabilità al robot.

Figura 1.7: Il robot umanoide "Robovie-M".

Il robot possiede per ciascuna gamba sei gradi di libertà gestiti da servomotori *Sanwa* modello *Hyper ERG-VB*. Tali servomotori sviluppano una coppia di circa 13 Kg per cm e possono ruotare fino ad una velocità di 60 gradi per millisecondo.

I gradi di libertà presenti su ogni braccio del robot sono quattro e vengono coordinati da servomotori di potenza minore rispetto a quelli montati sulle gambe. In particolare il modello utilizzato per gestire i gradi di libertà sulle braccia è il *Sanwa SPEC-APZ* , che sviluppa una coppia di 4 Kg per cm e può ruotare fino ad una velocità di 30 gradi per millisecondo

Il busto del robot, infine, possiede due gradi di libertà che permettono al robot di ruotare su se stesso e portare il tronco in avanti.

Sul robot sono presenti anche dei sensori, in particolare un encoder montato su ogni giunto consente di conoscere istante per istante la posizione del giunto ed un accelerometro che consente di stabilire l'inclinazione del robot.

1.3.1 RobovieMaker: il software per la movimentazione

Insieme al robot viene fornito un software per la creazione dei movimenti.
Tale software, la cui interfaccia grafica è visibile nella figura 1.8, si presenta completamente in giapponese e consente di generare, mediante la realizzazione di "*stage*", ovvero pose che il robot deve assumere durante l'esecuzione del comportamento, movimenti complessi che possono essere eseguiti dal robot.

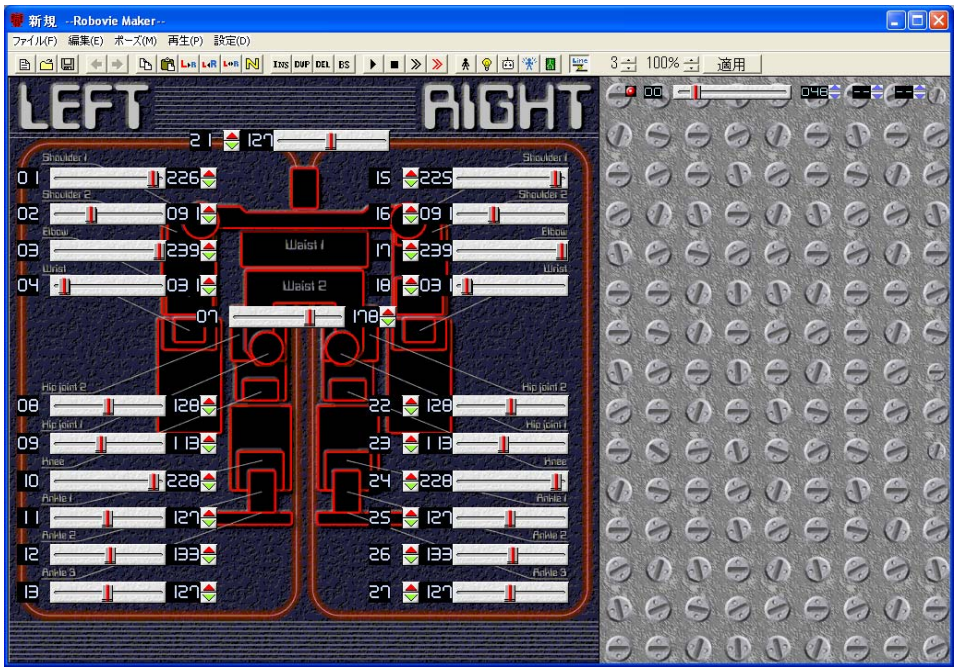

Figura 1.8: Una istantanea del software "RobovieMaker".

L'utilizzo di tale software, anche se si presenta in una lingua sconosciuta, non è molto complesso, almeno per quanto riguarda le funzioni base.
In alto, è presente una barra degli strumenti, in cui sono posizionati diversi pulsanti.

I più importanti sono il pulsante di "*Line*", utilizzato per stabilire una connessione fra il PC sul quale è istallato "*RobovieMaker*" ed il robot stesso; ed il pulsate di "*Zero*" (icona lampadina accesa), utilizzato dopo aver connesso il PC al robot, per accendere i 22 servomotori e mettere il robot nella configurazione iniziale.

Durante queste due operazioni, gli errori che possono verificarsi sono generalmente dovuti a due fattori. Il robot può infatti essere alimentato con una tensione non sufficiente e tale da impedirgli di accendere tutti i suoi servomotori o non è possibile stabilire una connessione, ed il messaggio di errore è visibile nella figura 1.9, perché la comunicazione non è settata in maniera corretta, come il numero della porta o la velocità di comunicazione.

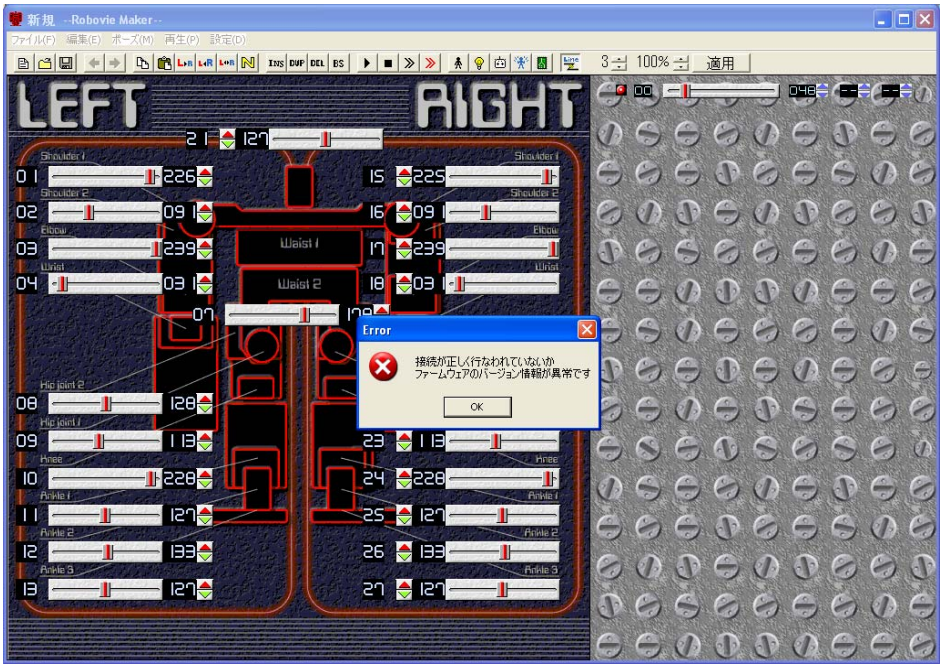

Figura 1.9: Il messaggio di errore relativo all'errato settaggio della connessione.

Nel caso nascano errori non imputabili a questi due fattori, uno dei rimedi più efficaci è quello di riavviare l'applicazione e riprovare la procedura di connessione e accensione. Dopo aver connesso fisicamente il robot al PC e aver messo il robot nella posizione di zero con i servomotori accesi, è possibile far eseguire al robot dei movimenti complessi.

Capitolo 2

La costruzione di un linguaggio

2.1 Introduzione

Prima del 1975 implementare un compilatore o un traduttore, utilizzando gli strumenti presenti in quel periodo, era un processo che richiedeva molto tempo. La svolta arrivò proprio nel 1975, ad opera di Lesk e Johnson che svilupparono due tool, Lex e Yacc, cui seguì una pubblicazione che li presentò al pubblico [5],[6].

Questi strumenti, ancora oggi largamente utilizzati nello sviluppo di programmi traduttori, semplificarono notevolmente la creazione di nuovi, riducendo, di netto, i tempi di sviluppo.

Lex genera programmi scritti in codice C, atti ad elaborare sequenze di caratteri in ingresso. Accetta specifiche ad alto livello orientate al problema del riconoscimento di stringhe di caratteri e produce un programma che riconosce linguaggi definiti da espressioni regolari.

Le espressioni regolari sono specificate dal programmatore nel sorgente delle specifiche di Lex. Il codice prodotto, allora, segmenta una sequenza di caratteri fornita in ingresso, che può essere un insieme di caratteri presente in un file di testo o uno stream da tastiera, producendo un segmento, per ogni sottosequenza che costituisce una stringa, in uno dei linguaggi definiti.

Al riconoscimento di una stringa, viene eseguita una porzione di codice, fornita dal programmatore, associata alla definizione dell'espressione regolare corrispondente. Nel caso dello sviluppo di un programma traduttore, l'azione che deve essere eseguita dopo il riconoscimento di una stringa, è quella di ritorno di un token.

I tokens sono una rappresentazione numerica delle parole chiave riconosciute, e vengono utilizzate sotto la forma numerica per semplificare il processo di

traduzione. Il valore numerico del token ed il tipo che esso rappresenta, sono contenuti in una tabella, presente in un header file che viene generato durante la compilazione di un file di specifiche yacc.

Yacc genera una funzione in codice C, chiamata *parser* (o *analizzatore sintattico*), che ha bisogno di una routine a più basso livello che esegue l'analisi lessicale. Tale routine può essere fornita dell'utente o generata mediante la compilazione di un sorgente Lex.

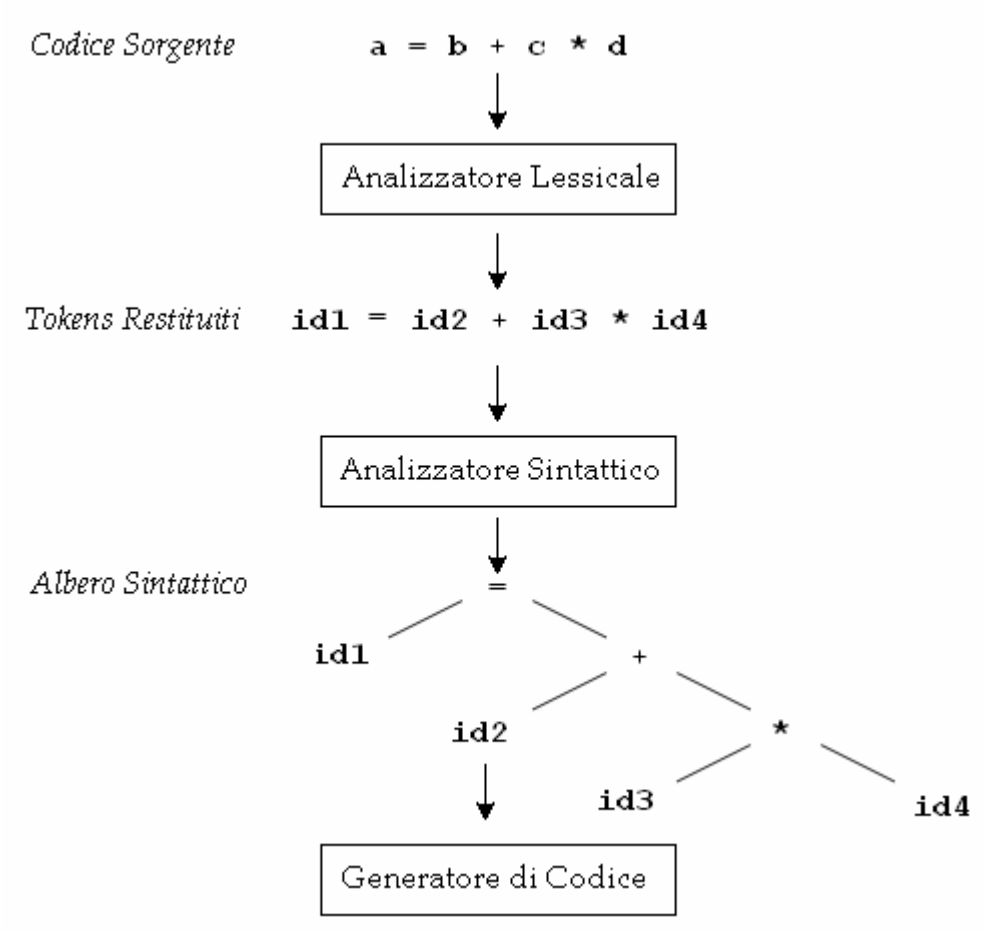

Figura 2.1: Le fasi del processo di traduzione

Yacc utilizza regole grammaticali fornite dal programmatore, allo scopo di analizzare i tokens restituiti da lex, nella fase di analisi lessicale.

Questi token devono essere organizzati secondo le regole di struttura dell'ingresso, chiamate *regole grammaticali*; quando una di queste regole è usata

per ridurre l'ingresso, allora viene invocato il codice utente relativo alla regola, cioè un'azione.
Come visibile nella figura 2.1, per una sequenza di tokens, yacc crea un albero, chiamato albero sintattico, che impone una ben determinata struttura all'ingresso. E' in questa fase che viene gestita la precedenza fra gli operatori e la loro associatività.

2.2 L'analisi lessicale mediante Lex

La prima fase del processo di traduzione è, come detto, la fase di analisi lessicale. Tale operazione consiste nel leggere lo stream in ingresso e convertirlo in una sequenza di tokens.
Ad ogni sottostringa riconosciuta è possibile associare, mediante lex, una porzione di codice che l'analizzatore eseguirà ogni volta che viene riconosciuta quella determinata espressione regolare.
Il sorgente Lex è pertanto una tabella di espressioni regolari e corrispondenti parti di codice. La tabella, dopo la compilazione, è tradotta in un programma che legge sequenze in ingresso, copiandole in una sequenza di output e separandole in stringhe che corrispondono con le espressioni date. Quando una stringa è riconosciuta il corrispondente frammento di codice è eseguito.
Il riconoscimento delle espressioni è eseguito da un automa a stati finiti deterministico generato da Lex stesso e generalmente, l'azione associata al riconoscimento di una espressione regolare, è l'azione di ritorno del valore intero cui è associato il token.
I programmi di analisi lessicale scritti con Lex possono accettare specifiche ambigue, cioè con diverse interpretazioni possibili, e scelgono per ogni caso di ambiguità, l'interpretazione che produce il frammento più lungo fra quelli possibili.
Lex non può essere considerato un linguaggio completo, ma piuttosto un generatore di programmi che può essere integrato in diversi linguaggi di

programmazione, chiamati “linguaggi ospite” (**Host Languages**). Il linguaggio ospite è usato per il codice di uscita generato da Lex, così come per le parti di programma aggiunte dal programmatore. Sono anche fornite librerie run-time compatibili per differenti linguaggi ospite che rende Lex adattabile a diversi ambienti di sviluppo (C, C++, Java) e programmatori.
Il programma generato dalla compilazione di un sorgente lex è chiamato `yylex()`. Tale programma riconosce espressioni in una sequenza in ingresso ed esegue le azioni specifiche per ogni espressione trovata.

2.3 L’analisi sintattica mediante yacc

Yacc costruisce uno strumento generale per imporre strutture all'ingresso di un programma. L'utente Yacc prepara una specifica del processo di lettura degli ingressi che include regole descriventi la struttura dell'ingresso, il codice da richiamare quando si riconoscono sottoinsieme derivato dalle singole regole, ed una routine a basso livello per analizzare l'ingresso.
Yacc quindi genera una funzione per controllare il trattamento dell'ingresso. Questa funzione, chiamata *parser* (o *analizzatore sintattico*) chiama la routine a basso livello fornita dall'utente (l'*analizzatore lessicale*) per identificare gli elementi di base (i *token*) nella sequenza dell'ingresso.
Questi token devono essere organizzati secondo le regole di struttura dell'ingresso, chiamate *regole grammaticali*; quando una di queste regole è usata per ridurre l'ingresso, allora viene invocato il codice utente relativo alla regola, cioè un'azione; le azioni hanno la possibilità di restituire valori e di far uso dei valori di altre azioni.
Il cuore delle specifiche dell'ingresso è l'insieme delle regole grammaticali. Ogni regola descrive una possibile struttura dell’ingresso e le assegna un nome.
Una parte importante del processo di analisi è svolto dall'analizzatore lessicale. Questa routine dell'utente legge la sequenza in ingresso, riconosce le strutture a livello più basso, e comunica i token identificati all'analizzatore sintattico. Per

ragioni storiche, una struttura riconosciuta dall'analizzatore lessicale è chiamata un *simbolo terminale*, mentre la struttura riconosciuta dal parser è chiamata un *simbolo nonterminale*.

L'ingresso da leggere potrebbe non essere conforme alle specifiche. Errori presenti in ingresso sono identificati in tempo, quando possibile, con una scansione da sinistra a destra; quindi non solo è possibile leggere ed elaborare dati in ingresso con errori sostanzialmente ridotti, ma i dati non validi possono essere di solito trovati rapidamente. La manipolazione degli errori, fornita come parte delle specifiche d'ingresso permette il reinserimento dei dati errati, o la continuazione del trattamento dell'ingresso saltando i dati non validi.

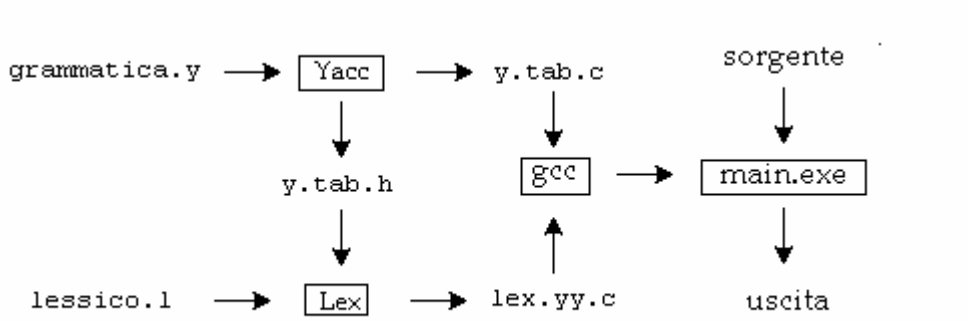

Figura 2.2: I processi di compilazione per la creazione di un traduttore.

In qualche caso, Yacc fallisce la produzione del parser a causa del fatto che le specifiche possono essere errate. Per esempio, le specifiche possono essere contraddittorie di per sé, o richiedere dei meccanismi di riconoscimento più potenti di quelli che sono presenti in Yacc. Il primo caso rappresenta un errore di progetto; il secondo spesso può essere corretto con un analizzatore lessicale più potente, o con la riscrittura di qualche regola grammaticale[7]. Nella figura 2.2 è possibile identificare i processi di compilazione che portano alla costruzione di un programma traduttore.

In ambienti linux, i comandi per generare un programma traduttore sono:

```
yacc -d grammatica.y
lex lessico.l
gcc lex.yy.c y.tab.c -o main.exe
```

Il primo comando compila, mediante lo strumento yacc, il file di specifiche yacc `grammatica.y`. L'opzione utilizzata `-d` serve per generare la tabella in cui è presente la corrispondenza token-valore numerico. Tale tabella è presente nell'header file `y.tab.h`. La compilazione genera inoltre un file `y.tab.c` dove è presente la funzione `yyparse()`, funzione che eseguirà l'analisi sintattica.
Il secondo comando compila, mediante lex, il file di specifiche `lessico.l`. L'uscita di questa compilazione è il file `lex.yy.c` che è il file c all'interno del quale è contenuta la funzione `yylex()` che avrà il compito di eseguire l'analisi lessicale e cui si appoggerà il parser durante l'opera di traduzione.
Il terzo ed ultimo comando compila mediante gcc, i due file precedentemente generati, creando in uscita un programma, che sarà il traduttore.

Capitolo 3

Lo studio dei movimenti

3.1 Introduzione alla locomozione bipede

Numerosi sono stati, nel corso degli anni, i lavori che sono stati pubblicati riguardanti la locomozione bipede. Gli approcci al sistema proposti sono stati molteplici, sia dal punto di vista teorico, sia da quello sperimentale, con la realizzazione di numerosi prototipi. Per avere un'idea della varietà dei progetti che riguardano la locomozione bipede, si può fare riferimento al *Walking Machine Catalogue* [18].

Di fronte ad una così grande vastità, è normale chiedersi quali siano i tratti fondamentali comuni alle macchine bipedi ed, in generale, al processo che le governa.

Un criterio di classificazione molto utile, è quello che classifica i robot in base alla caratteristiche della camminata.

I robot che camminano rimanendo sempre in equilibrio statico, ovvero i robot che mantengono la proiezione del proprio centro di massa sempre all'interno dell'area di appoggio, sono classificati nella categoria degli esecutori di camminata statica (*static walkers)*.

La seconda categoria, chiamata esecutori di camminata dinamica (*dynamic walkers)*, comprende i sistemi robotici che durante il loro moto si trovano ad essere in equilibrio dinamico, ed hanno la particolarità di avere attuatori montati su piedi e caviglie.

Un discorso a parte merita la categoria degli esecutori di camminata passiva (*passive walkers*), meccanismi passivi che sono in grado di camminare su due gambe sfruttando solamente la forza di gravità mediante una opportuna distribuzione sulle masse.

3.2 I "Passive walkers"

I "*passive walkers*" sono dei particolari meccanismi passivi, in grado di camminare su due gambe sfruttando solamente la forza di gravità, mediante un'opportuna disposizione delle masse. Data la loro semplicità, questi sistemi costituiscono un buon mezzo per comprendere i tratti fondamentali della locomozione bipede.

Il pioniere nel campo dei *passive walkers* è *T. McGeer* [8][9], che ha pubblicato i primi studi nel 1990.

L'idea di base è molto semplice e parte dal modello della ruota senza cerchione (*rimless wheel)*

Nella figura 3.1a è rappresentata la ruota dei carri (*wagon wheel)*. Se si rimuove il cerchione esterno si ottiene la ruota senza cerchione (*rimless wheel)*, rappresentata nella figura 3.1b, i cui raggi possono essere visti come una serie di gambe.

La *wagon wheel* può rotolare in maniera fluida su una superficie piana, la *rimless wheel* al contrario, subirà una serie di urti che ne ridurranno progressivamente l'energia cinetica.

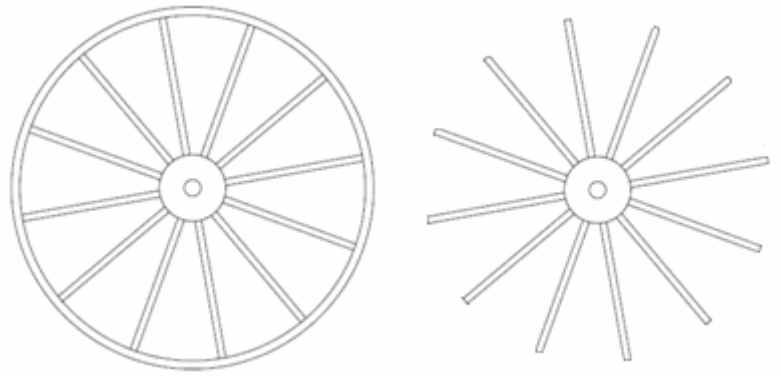

Figura 3.1: Le ruote "wagon wheel" e "rimless wheel".

Per calcolare questa perdita, si faccia riferimento alla figura 3.2: nell'istante *t*- che precede il cambio di supporto, la ruota è in contatto con il suolo nel punto *P*-. Nell'istante immediatamente successivo *t*+ il punto di contatto si è trasferito

nel punto *P*+. I punti *P* rappresentano i centri istantanei di rotazione di tutto il sistema, il cui centro di massa si trova in *O*.

Le velocità *v*+ e *v*- rappresentano le velocità lineari del centro della ruota nei rispettivi istanti *t*- e *t*+. A causa del cambio di supporto istantaneo si vede che c'è una brusca variazione nel vettore velocità: questo è un fenomeno di urto, che qui viene ipotizzato anelastico.

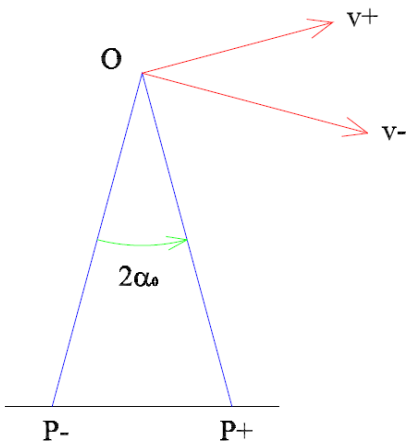

Figura 3.2: La variazione dei vettori velocità negli istanti di cambio di supporto.

Si calcola allora il momento della quantità di moto del sistema prima e dopo l'urto:

$$\begin{aligned} H^{-} &= (\cos 2\alpha_0 + \mathrm{r}_{\mathrm{gyr}}^2) m l^2 \Omega^{-}; \\ H^{+} &= (1 + \mathrm{r}_{\mathrm{gyr}}^2) m l^2 \Omega^{+}. \end{aligned} \tag{3.1}$$

dove Ω indica la velocità angolare, m la massa della ruota, l la lunghezza della gamba, $\mathrm{r}_{\mathrm{gyr}}$ il raggio di girazione normalizzato secondo la lunghezza l; si ricorda, inoltre che il raggio di girazione $\mathrm{r}_{\mathrm{gyr}}$ è definito dalla relazione

$$I = m * r_{gyr}^2 \tag{3.2}$$

dove I è il momento di inerzia ed m la massa del sistema. $2\alpha_0$ indica l'angolo tra due gambe consecutive, come indicato in figura 3.2.

I simboli recanti il (-) indicano delle grandezza precedenti l'urto, viceversa quelle con il (+).

Per il principio della conservazione del momento della quantità di moto si possono confrontare queste due equazioni ricavando la variazione della velocità angolare:

$$\frac{\Omega^{+}}{\Omega^{-}} = \frac{\cos 2\alpha_0 + r_{gyr}^2}{1 + r_{gyr}^2} = \eta \tag{3.3}$$

dove η indica il coefficiente di perdita di velocità.

Su una superficie piana, la *rimless wheel*, perderà molto rapidamente la sua energia cinetica fino a fermarsi. Ma se la superficie è inclinata, l'energia persa potrà essere reintegrata dall'energia gravitazionale.

Sorge però il problema della continuità del moto. Infatti il baricentro della *rimless wheel* percorrerà degli archi di circonferenza raccordati con delle cuspidi. Si avranno quindi brusche variazioni di velocità e accelerazione, oltre ad un'oscillazione dell'altezza del baricentro. Si deduce allora che la *rimless wheel* non produce delle caratteristiche di moto ideali.

Un risultato migliore si ottiene in questo modo: si prende la *wagon wheel*, si seziona il cerchione esterno in corrispondenza dell'angolo bisettore fra due raggi consecutivi, si eliminano tutti i raggi tranne due e si rende mobile il collegamento fra questi tramite un perno nel centro della ruota.

Quello che si ottiene è il meccanismo illustrato nella figura 3.3.

Tale meccanismo ha la stessa cinematica della *wagon wheel* quando uno dei due piedi è in contatto con il suolo; inoltre, ora le gambe possono ruotare una rispetto all'altra.

Il problema è ora quello di portare la gamba che non è in contatto con il terreno in avanti, in modo da sostituire la gamba in contatto con il suolo al momento giusto.

Se questo accade si realizza una locomozione simile a quella della *wagon wheel*.

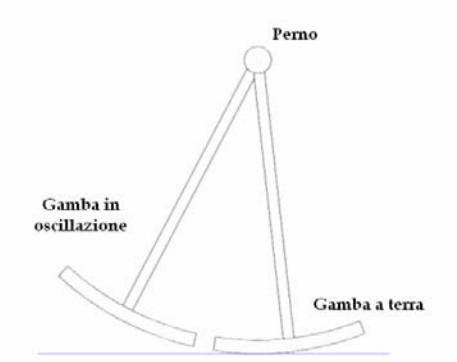

Figura 3.3: Il meccanismo ottenuto modificando la wagon wheel.

Qualitativamente parlando, un sistema è stabile se riesce sempre a raggiungere delle determinate condizioni di equilibrio, partendo da un qualsiasi stato iniziale. Il sistema in esame, la *wagon wheel modificata*, presenta un moto ciclico che vede il ripetersi di due fasi: gamba destra a terra e sinistra in aria, e viceversa.

L'evoluzione di ciascuna di queste fasi dipende unicamente dalle condizioni iniziali, ed è evidente che queste corrispondono alle condizioni finali della fase precedente. Se queste variano si avranno fasi del moto diverse, ma se si riesce in qualche modo a fare si che le condizioni iniziali siano ogni volta le stesse si ottiene allora un ciclo di fasi sempre identiche, che ripete all'infinito se stesso.

Questo è quello che nella teoria dei controlli automatici, viene chiamato ciclo limite [17].

Se intervengono degli errori che allontanano il sistema da queste condizioni, il sistema può o allontanarsi dal ciclo limite, ed in questo caso l'energia persa negli urti sarà maggiore di quella che il sistema reintegra con l'energia gravitazionale, o riportarsi autonomamente in questa condizione. In quest'ultima situazione si assiste ad un'ulteriore caratteristica, la stabilità, e si parla allora di ciclo limite stabile [17].

L'obiettivo è quindi quello di realizzare dei *passive walkers* che abbiano dei cicli limite *stabili*, in modo tale che questi possano essere fatti partire in un

determinato intervallo di configurazioni iniziali, e possano autonomamente raggiungere il proprio ciclo limite.

Il problema viene risolto concentrando la maggior parte della massa in corrispondenza del centro della ruota *(hip)*, in modo tale che la dinamica della gamba che effettua il passo (*swing leg)* non disturbi il moto complessivo del sistema, e individuando opportune condizioni iniziali tali da permettere a questa di effettuare il moto desiderato.

Nelle figure 3.4a e 3.4b sono illustrati due sistemi realizzati secondo tali principi e quindi capaci di camminare in modo stabile lungo dei piani inclinati.

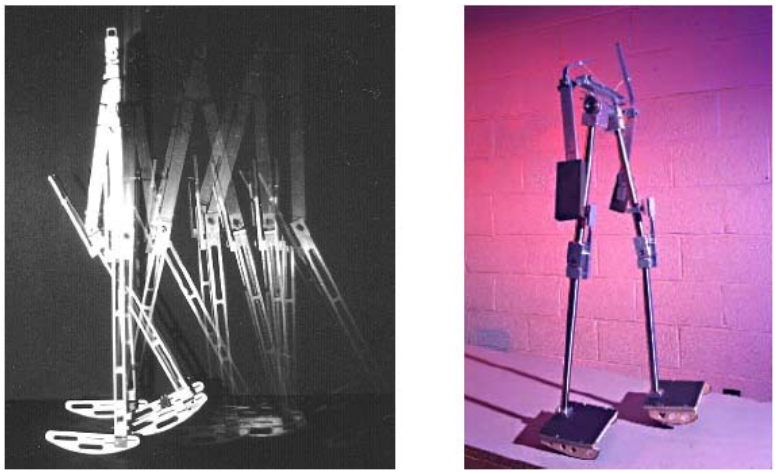

Figura 3.4: Alcuni sistemi che implementano la camminata passiva.

Esperimenti condotti su questo tipo di macchine ha permesso di stabilire che la locomozione bipede realizzata dai *passive walker* è energeticamente molto efficiente. La pendenza necessaria per garantire la stabilità del moto per questi sistemi è molto bassa. Si sono ottenute delle camminate stabili anche con lo 0.01% di pendenza.

Il processo che si realizza, inoltre, è asintoticamente stabile sotto un'ampia gamma di condizioni iniziali, quindi il sistema è in grado di convergere al ciclo limite anche partendo da configurazioni diverse da quelle ottimali.

In base a questi risultati si deduce che la locomozione bipede realizzata mediante *passive walkers*, è in generale efficiente dal punto di vista energetico, e

produce un moto del centro di massa molto fluido, al limite perfettamente rettilineo, qualora questo coincida con il centro di rotazione.
Per la fluidità dell'intero movimento è fondamentale la presenza di un piede opportunamente sagomato, capace di mantenere costante l'altezza del centro di massa rispetto al suolo inoltre è bene che la massa sia concentrata nel corpo principale e non nelle gambe, in quanto il movimento di queste nelle varie fasi deve interferire il meno possibile con la dinamica dell'anca.

3.3 Gli "Active walkers".

Il passo che porta i "*passive walkers*" a diventare "*active walkers*" è quello di aggiungere al sistema degli attuatori e dei controllori, in modo tale da poter fornire energia in modo autonomo, senza dover ricorrere alla forza di gravità.
In questo caso ogni grado di libertà del bipede è equipaggiato con un'attuatore.
Vengono fissate a priori delle traiettorie per i piedi e per il bacino in funzione del tipo di andatura che si vuole eseguire; diverse allora saranno le andature dei giunti quando si hanno diverse saranno le andature (passeggio o corsa).
Questo approccio è attualmente adottato dai robot giapponesi (*Wabian, Wabian-2, Wabot, Asimo*) ed americani (*M2, DB*). *Robovie-M V3* fa parte anch'esso di questa categoria di robot bipedi.
In questa categoria è possibile catalogare due sottocategorie di camminatori "(*walkers)*": gli esecutori di camminata statica (*static walkers)* ed gli esecutori di camminata dinamica (*dynamic walkers*).

3.4 La camminata statica

Il modo più semplice per realizzare una camminata bipede consiste nel fare in modo che il robot sia in ogni istante in equilibrio statico. Questo discorso ha senso solo se gli effetti dinamici sono tali da poter essere trascurati, in modo tale che si debba tenere conto esclusivamente delle forze gravitazionali. Se queste

condizioni sono verificate, allora la strategia di controllo risulta effettivamente molto semplice.

Nella figura 3.5 si vede il piede del robot che poggia a terra. Il dato fondamentale di cui si deve tenere conto è la proiezione sul suolo del baricentro dell'intero sistema, calcolabile con le relazioni:

$$x_{cog} = \frac{\sum_{i=1}^{n} m_i x_i}{\sum_{i=1}^{n} m_i}$$
$$y_{cog} = \frac{\sum_{i=1}^{n} m_i y_i}{\sum_{i=1}^{n} m_i} \qquad (3.4)$$

dove x_i e y_i i rappresentano le coordinate dei centri di massa dei vari link ed m_i la loro massa. x_{cog} e y_{cog} sono le coordinate del punto cercato (centro di gravità).

Come è possibile osservare nella figura 3.6, nel caso in cui la proiezione del baricentro sia nella posizione *P1*, non si ha equilibrio statico che si ottiene, invece, nel caso in cui questa cada in *P2*.

Se si riesce a pianificare i diversi movimenti necessari alla locomozione facendo rimanere la proiezione del baricentro sempre all'interno del poligono di appoggio, coincidente con l'area di contatto del piede con il suolo, si ottiene una serie di movimenti in cui il robot avanza senza mai cadere.

Particolari dei robot che effettuano questa camminata sono nei piedi molto grandi che aumentano l'area di appoggio.

I giunti dell'anca sono inoltre equipaggiati con grossi motori per bilanciare il momento generato dal corpo durante l'alzata di una gamba.

I movimenti risultano però essere movimenti lenti, per mantenere gli effetti dinamici trascurabili rispetto alle forze statiche.

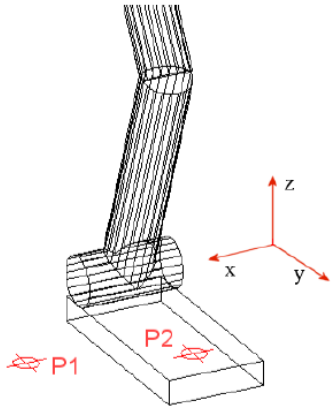

Figura 3.5: Lo studio dell'equilibrio di un robot in relazione alla proiezione del baricentro.

A fronte della semplicità nel controllo, si hanno quindi andature con velocità molto ridotte, motivo per cui si tende in genere ad orientarsi verso altri tipi di locomozione.

3.5 La camminata dinamica

Il problema principale degli *static walker*, come si è avuto modo di intuire nel paragrafo precedente, risiede nella loro bassa velocità di avanzamento. L'unica maniera per incrementare tale velocità è quella di ricorrere a sistemi più complessi che sfruttino pienamente la dinamica.

Una classe di robot che opera in questo modo è quella raggruppata sotto il nome di "*dynamic walkers*".

Durante la locomozione questi robot in genere non sono in equilibrio statico, bensì dinamico: questo è un estensione del concetto di equilibrio, in quanto include nel calcolo, oltre alle forze gravitazionali, anche le forze di inerzia.

Nella figura 3.5 si erano illustrate le condizioni per il sussistere dell'equilibrio statico, ovvero la proiezione del baricentro totale del sistema deve ricadere all'interno del poligono di appoggio.

Per ottenere un equilibrio dinamico è necessario includere nel calcolo le forze di inerzia ricavate con le equazioni della dinamica del robot.
Per il corpo i-esimo queste sono date da:

$$F_i = \begin{pmatrix} m_i \ddot{x}_i \\ m_i \ddot{y}_i \\ m_i \ddot{z}_i \end{pmatrix} \tag{3.5}$$

Queste sono le basi per il principio dello Zero Moment Point (ZMP)[10]: per un insieme di corpi rigidi (si veda la figura 3.6) si possono scrivere le equazioni della dinamica nella forma:

$$\begin{aligned} &\sum_{i=1}^{n} m_i \ddot{a}_i = F; \\ &\sum_{i=1}^{n} ((r_i - P) \times m_i (\ddot{a} - g)) + T = 0. \end{aligned} \tag{3.6}$$

equazioni che esprimono il bilancio delle forze e dei momenti agenti sul sistema. m_i è la massa del link i-esimo, $\ddot{a}_i - i$ il suo vettore accelerazione, F la risultante delle forze esterne. r_i è il vettore posizione del link, P il punto rispetto al quale vengono calcolati i momenti, g è il vettore accelerazione di gravità, e T il momento risultante delle forze esterne.
L'intero sistema si trova in equilibrio se, nel punto *P*, si ha l'applicazione di una forza F e di una coppia T opportuni. Si può scegliere il punto *P* in maniera tale che la coppia *T* sia nulla, e quindi il sistema risulti equilibrato semplicemente mediante l'applicazione di una forza F : in questo caso *P* si definisce appunto *Zero Moment Point*.
In realtà di punti con questa caratteristica, ne esistono infiniti, tutti allineati a formare una retta; ma nel campo della locomozione interessa solo il punto intersezione di tale retta con il piano del suolo: l'obiettivo finale, infatti, è quello

di far cadere lo *ZMP* all'interno dell'area del piede di appoggio, area che giace appunto sul piano del terreno.

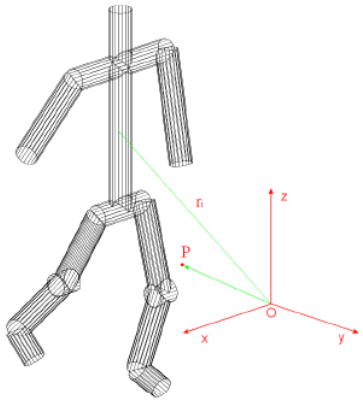

Figura 3.6: Assi di riferimento per il calcolo dello ZMP.

Per calcolare il punto in questione si impone la coordinata z di P uguale a zero:

$$O = \begin{bmatrix} O_x & O_y & 0 \end{bmatrix} \tag{3.7}$$

e si ricavano le coordinate O_x ed O_y dalla seconda delle (3.6)

$$\begin{aligned} O_x &= x_{zmp} = \frac{\sum_{i=1}^{n} m_i x_i (\ddot{z}_i - g_z) - \sum_{i=1}^{n} m_i z_i (\ddot{x}_i - g_x)}{\sum_{i=1}^{n} m_i (\ddot{z}_i - g_z)}; \\ O_y &= y_{zmp} = \frac{\sum_{i=1}^{n} m_i y_i (\ddot{z}_i - g_z) - \sum_{i=1}^{n} m_i z_i (\ddot{y}_i - g_x)}{\sum_{i=1}^{n} m_i (\ddot{z}_i - g_z)} \end{aligned} \tag{3.8}$$

dove g è il vettore accelerazione di gravità che, ponendo il piano xy come piano del suolo e l'asse z ortogonale e diretto verso l'alto assume la forma:

$$g = \begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -9.81 \end{bmatrix} \tag{3.9}$$

E' importante sottolineare che lo *ZMP* sul piano, esiste sempre a meno che il denominatore comune alle espressioni di x_{zmp} e y_{zmp} non diventi nullo.

Questo accade quando la componente $\ddot{z}$ compensa esattamente l'accelerazione di gravità.

L'applicazione del principio dello *ZMP* alla locomozione consta di diverse fasi. In primo luogo si pianifica il movimento dei vari corpi per realizzare il movimento [12],[15]. La situazione ideale è quella in cui lo *ZMP* si trova in ogni istante in corrispondenza del giunto che collega la gamba al piede di appoggio, situazione chiaramente molto difficile da ottenere nella realtà. Lo *ZMP* seguirà infatti una traiettoria sul suolo dipendente dall'evoluzione della dinamica dell'intero sistema.

Quello che si vuole ottenere è che lo *ZMP* rimanga sempre all'interno del piede andando a verificare che la traiettoria pianificata non varchi mai i confini del poligono di appoggio.

Se questo è vero, allora la camminata è stabile: le forze necessarie al mantenimento dell'equilibrio possono infatti essere fornite dal piede tramite l'azione dei motori della caviglia.

Se invece lo *ZMP* esce dall'area del piede, allora non è più possibile riequilibrare il sistema: in questo caso è necessario modificare i movimenti pianificati o sfruttare il moto di altri corpi, come le braccia e il busto, per introdurre effetti dinamici che riposizionino opportunamente lo *ZMP*.

Il grande vantaggio di questa tecnica risiede nella possibilità di introdurre un controllo in controreazione sulla movimentazione [13]. E' infatti possibile ricostruire in tempo reale la posizione dello *ZMP*, in modo tale da correggere il movimento che si sta effettuando per rimanere in equilibrio dinamico.

Capitolo 4

La creazione di un movimento robotico

4.1 La suddivisione del movimento in stage

Spesso, quando si vede concludere un movimento complesso ad un robot umanoide, non ci si rende conto di quali e quanti siano state, effettivamente, le configurazioni che esso ha assunto durante lo svolgimento dello stesso.
Un qualsiasi movimento complesso, infatti, è l'insieme di tutta una serie di configurazioni, che vengono settate all'atto della programmazione del movimento e che il robot "percorre" per eseguire l'azione programmata.
Ogni configurazione, chiamata "stage", identifica univocamente la posizione del robot; ad ogni "stage", infatti, è associato, per ogni grado di libertà del robot, un valore relativo alla posizione che tale variabile ha nello spazio.
La creazione di un movimento complesso consiste allora nel programmare l'insieme di stage che costituiscono il movimento, fornendo, per ogni posa, il valore relativo alla posizione che il giunto, grado di libertà del sistema, deve assumere.
Il passaggio da uno stage all'altro, che in termini di posizioni dei giunti individua il passaggio da una posizione ad un'altra, è ciò che fa muovere il robot e gli fa compiere l'azione complessa programmata.
Appena il robot si ritrova ad essere nella configurazione dello stage, è pronto per iniziare una nuova transizione per portarsi su un nuovo stage, ovvero in una nuova configurazione successiva.
Supponendo di sviluppare un movimento complesso che consista nel far alzare il braccio destro ad un robot umanoide di una certa quantità, iniziando il movimento dalla posizione di zero, che per definizione è una posizione di equilibrio statico, e di farlo ritornare nella posizione di zero subito dopo che il

braccio del robot ha assunto la posizione desiderata,è chiaro che il numero di stage che costituisce il movimento robotico è tre.
In particolare si ha che:

- Lo stage 0 è lo stage di partenza, con certi valori di angolazioni dei giunti;
- Lo stage 1 è lo stage nel quale i valori delle angolazioni dei giunti sono immutati rispetto allo stage 0 tranne il valore del giunto che fa muovere il braccio destro del robot verso l'alto;
- Lo stage 2 è l'ultimo stage ed i valori dei giunti, in questo stage, è uguale al valore che essi stessi avevano nello stage 0.

Il passaggio da uno stage ad un successivo può avvenire con tempi di esecuzione diversi, possono infatti entrare in gioco le velocità di passaggio da uno stage ad un altro e quindi gli effetti dinamici del movimento possono arrivare ad essere non trascurabili.
Generalmente alte velocità di transizione tra gli stage vengono utilizzate quando si costruiscono movimenti in cui, nello stage, le posizioni sono posizioni di equilibrio in senso dinamico, ed il robot deve riuscire a portarsi in poco tempo in una configurazione diversa prima che esso cada.
Durante la programmazione da zero di un movimento complesso, nella maggioranza dei casi, si cerca di fare in modo che il robot assuma posizioni di equilibrio statico in ogni configurazione che esso deve assumere.
Questo avviene, come visto nel capitolo 3, quando la proiezione del baricentro del sistema sul piano di appoggio si trova all'interno dell'area delimitata dalla parte del corpo del robot che è in contatto col piano di appoggio stesso.
Osservando la figura 4.1, ci si rende conto come, durante le fasi di una camminata, il baricentro può continuamente cambiare posizione. Nel caso della camminata statica, la proiezione del baricentro del piede di appoggio cade sempre all'interno della convex hull delimitata dalla base dei piedi di appoggio.

I movimenti statici, come le camminate statiche, hanno la caratteristica che, fermando l'azione in un qualsiasi momento, il robot si troverà sempre in una posizione di equilibrio stabile.

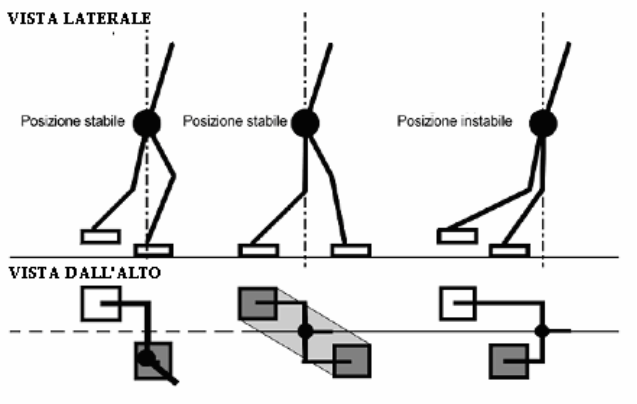

Figura 4.1: Le posizioni di equilibrio stabili ed instabili durante le fasi di una camminata.

Un movimento statico è generalmente un movimento sicuro per il robot, perché piccole perturbazioni lo fanno ritornare ad una posizione di equilibrio, ma è un movimento che viene compiuto lentamente.

La soluzione per eliminare il vincolo delle basse velocità di esecuzione dei movimenti complessi è come detto quella di aumentare le velocità di transizioni fra gli stage.

Il controllore che deve essere implementato su questi sistemi dovrà però necessariamente essere un controllore che tenga conto della dinamica del sistema, e pertanto tenere in considerazione, oltre all'effetto delle forze gravitazionali, i momenti di inerzia, le accelerazioni angolari e le accelerazioni lineari.

Se si considerano sia le accelerazioni sia le inerzie, è possibile che la proiezione del centro di massa esca dalla *convex hull*, costituita dalla base dei piedi in appoggio, senza che, per questo, il robot cada. Quello che si ottiene è un movimento complesso molto veloce, ma costituito da posizioni in cui anche piccole perturbazioni possono far uscire il sistema dalla posizione di equilibrio.

4.2 Il testing mediante simulatore

In qualsiasi applicazione informatica, dopo aver sviluppato un lavoro, è generalmente consuetudine testare la bontà dello stesso, provando gli algoritmi di controllo sviluppati, sotto tutte le condizioni possibili.

Anche per le applicazioni robotiche è possibile effettuare le stesse considerazioni. Sia per gli algoritmi di visione che per i movimenti, è buona norma, prima di eseguirli sul sistema robotico, provare la loro efficacia ed efficienza su opportuni software costruiti appositamente.

Per testare un generico algoritmo di visione, per esempio, è buona regola provare lo stesso sotto appositi strumenti per ottenere un riscontro visivo dell'algoritmo sviluppato.

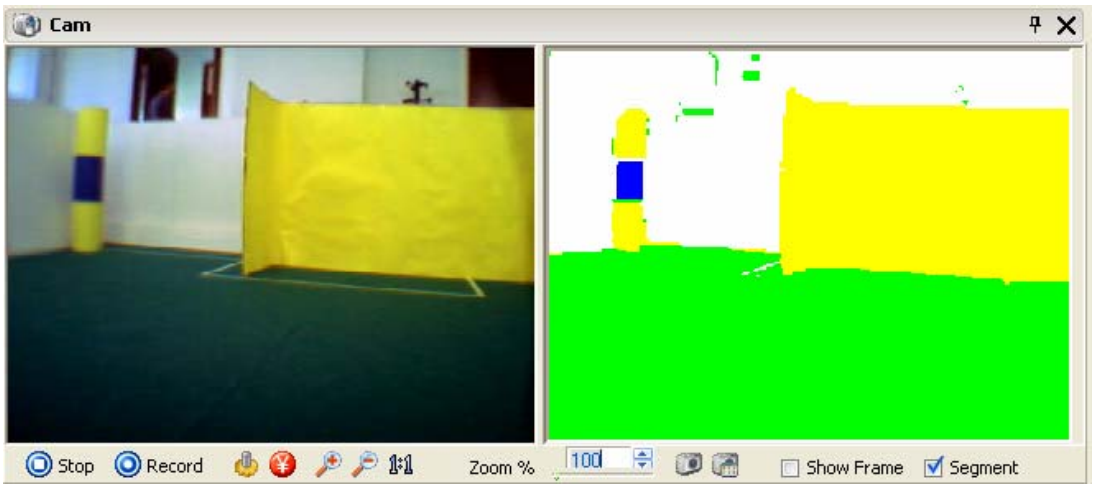

Figura 4.2: Istantanea del software "*CamTool*"

Una applicazione per testare gli algoritmi di visione, sviluppato dal collega Alessandro Cimino, è il "*CamTool*", di cui viene riportata una istantanea nella figura 4.2. Tale strumento è in grado di acquisire le immagini da una telecamera, effettuare la segmentazione delle stesse ed effettuare il salvataggio di un video o dei singoli frame costituenti.

Un altro esempio di strumento per la prova degli algoritmi di visione, è "*RoboVision*", di cui è possibile vedere una istantanea in figura 4.3.

Tale strumento consente di avere un riscontro effettivo degli algoritmi di segmentazione precedentemente sviluppati e ritorna gli oggetti riconosciuti e le relative distanze.

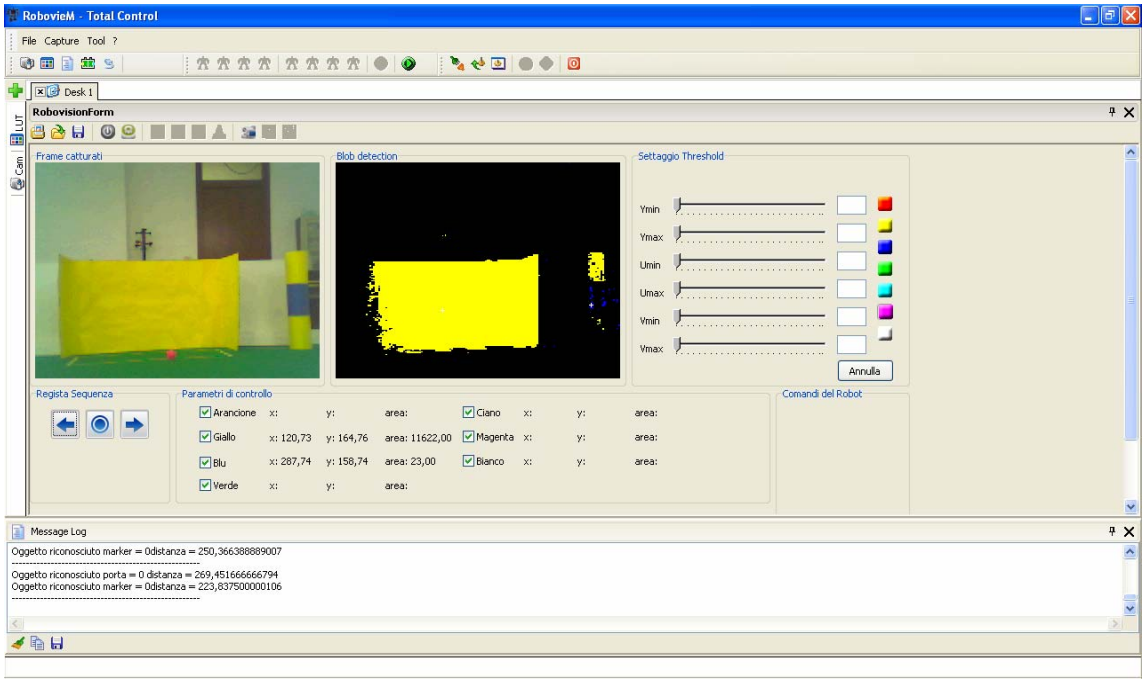

Figura 4.3: Una istantanea del software "Robovision".

Per quanto riguarda la simulazione dei movimenti, la simulazione viene effettuata sotto opportuni strumenti grafici 3D che visualizzano, istante per istante, le configurazioni che il robot assume.

Questi simulatori generalmente implementano anche la fisica del sistema, pertanto possono simulare il movimento facendo uso di tutte le grandezze di cui bisogna tener presente e ritornare, per esempio, dopo aver calcolato opportunamente momenti angolari, momenti di inerzia ed accelerazioni, il fatto che un robot sia in equilibrio o no [14].

4.3 La simulazione dei movimenti per il robot Robovie-M

Presso l'Università degli Studi di Padova, l'allievo ing. Tommaso Guseo, ha sviluppato uno strumento per la simulazione dei comportamenti complessi nel robot umanoide "Robovie-M" [16].

Purtroppo questo software, che è possibile osservare nella figura 4.5, non implementa la fisica del sistema, ma, essendo l'unico dove poter simulare un comportamento complesso sviluppato in questa tesi, è stato scelto come simulatore per la esecuzione dei movimenti.

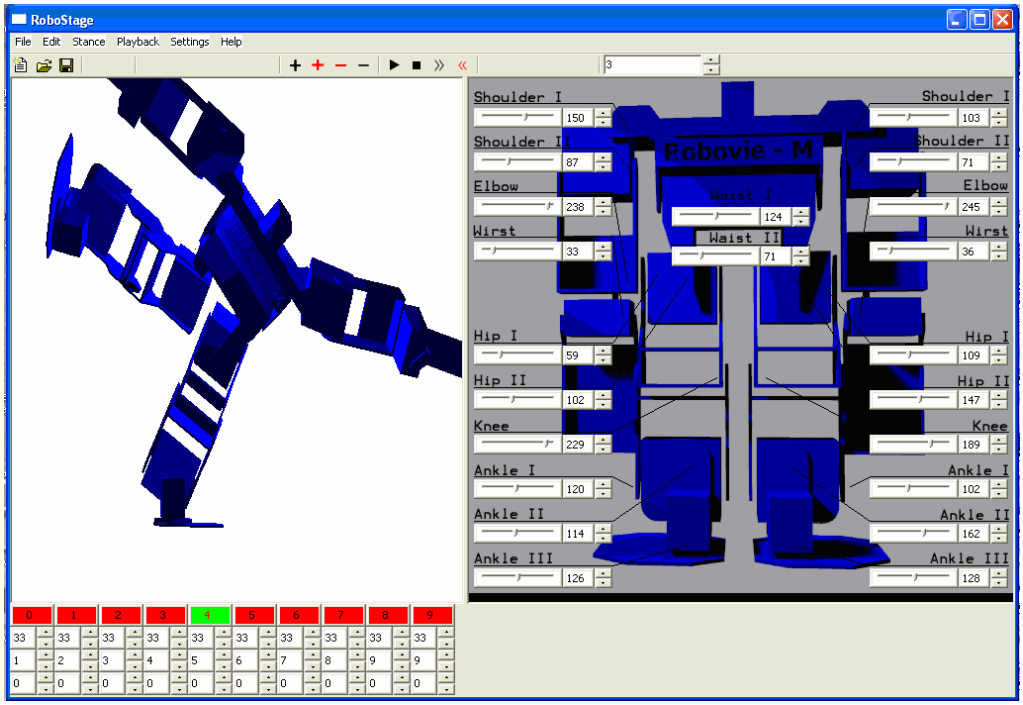

Figura 4.5: Una istantanea del software "RoboStage".

Questo simulatore accetta in ingresso un file di testo contenente stringhe opportunamente costruite.

Ogni stringa, presente nel file, rappresenta uno stage, e porta con se tutte le informazioni relative allo stage, le informazioni relative alle angolazioni dei giunti, le informazioni relative alla velocità di transizione degli stage e le informazioni riguardante i cicli.

Si analizza adesso una singola stringa e si da una spiegazione dettagliata di ogni parte che la compone.

In dettaglio, si prende in esempio la stringa relativa alla posizione iniziale, ovvero la posizione del zero del robot.

La stringa in questione è una stringa composta da 75 caratteri:

```
@21A7fB8eC1bD80E7aF80G1eHa4I10Je0K80L71Me4N7fO85P7fQe1R5bSefT1fUb2V80,+1!00
```

e porta con se tutte le informazioni che riguardano lo stage.
Se si suddivide la stringa a gruppi di tre caratteri, si ottiene una sequenza composta da 25 sottostringhe composte da tre caratteri:

```
@21, A7f, B8e, C1b, D80, E7a, F80, G1e, Ha4, I10, Je0, K80, L71, Me4, N7f, O85, P7f, Qe1, R5b, Sef, T1f, Ub2, V80,+1, !00
```

Si può subito notare come ogni gruppo di tre caratteri è composto da un carattere identificativo (@, A, B, .., V, “,”, “!”) e due caratteri successivi.
Per il primo gruppo di tre caratteri, ovvero il gruppo che inizia con il carattere @, il primo carattere identifica che l’informazione data dai due caratteri seguenti è relativa alla velocità con la quale l’azione di passaggio allo stage deve essere eseguita, i due caratteri successivi, invece, rappresentano un intero compreso nell’intervallo [0, 255] e rappresentato secondo la notazione esadecimale [00, ff].
Dal secondo gruppo in poi, il primo carattere è una lettera maiuscola che identifica univocamente il giunto sul quale si sta facendo riferimento, i due caratteri successivi, invece, rappresentano un intero compreso nell’intervallo [0,255], e rappresentato sempre secondo la notazione esadecimale [00, ff].
Per i gruppi di tre caratteri in cui il primo carattere è “,” o “!”, sono presenti informazioni riguardante i cicli.
Si fornisce ora una osservazione pratica sul significato di ogni parte che compone la stringa. Supponendo di inserire la stessa all’interno di un file di testo che rappresenta un movimento complesso, si utilizzi “RoboStage” per visualizzare il comportamento in simulazione.

Il file di testo caricato, rappresentante il movimento complesso, è, come si può vedere in basso della figura 4.6, composto da 4 stage. Lo stage 0 è quello relativo alla stringa che si sta analizzando ed è in verde poiché è selezionato.

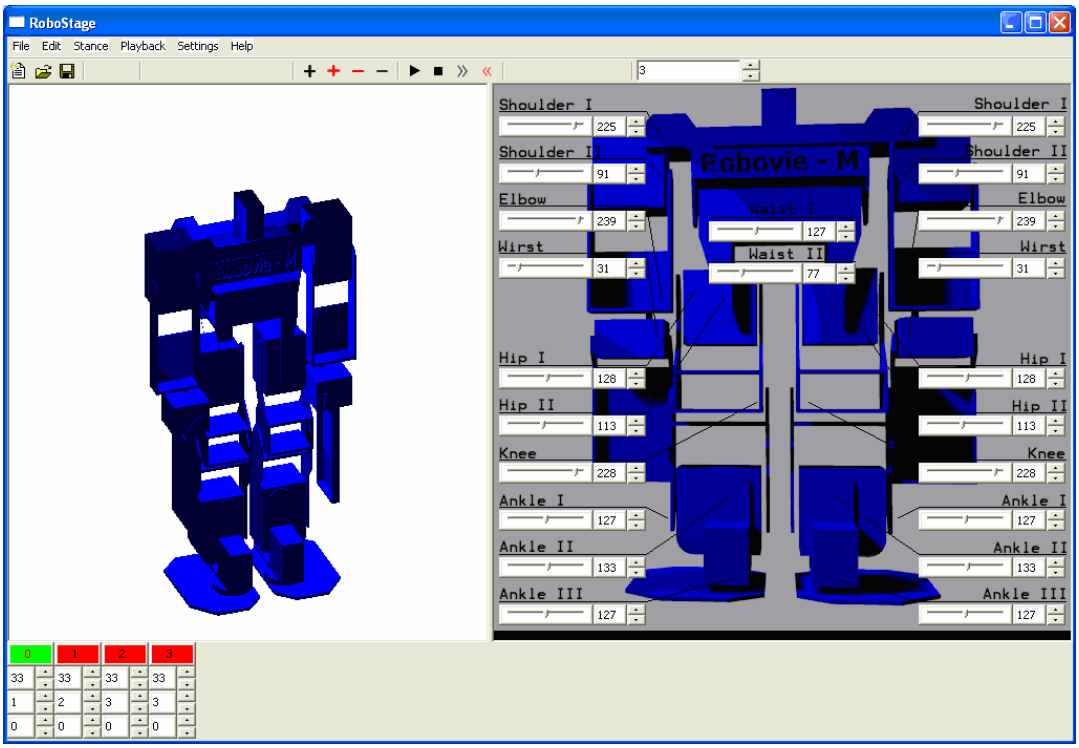

Figura 4.6: Lo studio dello stage iniziale su "RoboStage".

Sulla destra è possibile notare i valori dell'angolazione di ogni giunto del robot, mentre in basso i tre interi presenti nella selezione dello stage sono: la velocità, che in questo caso è costante e vale 33, lo stage successivo allo stage considerato, che sarà lo stage 1 e lo stage dal quale ripartire nel caso in cui il comportamento è costituito da cicli.

Nella stringa, il primo terzetto di caratteri che si incontra è il terzetto @xx. Come dapprima accennato, la @ fa riferimento al fatto che i due caratteri successivi portano informazioni riguardo la velocità di transizione allo stage.

Nella stringa relativa alla posizione di zero, la sottostringa considerata è @21.

Poiché 21 nella base decimale è 33, il significato dei tre caratteri è allora chiaro: il passaggio allo stage 0 deve avvenire a velocità 33, e questo viene riportato nel simulatore come è possibile osservare in figura 4.7.

Figura 4.7: Il particolare relativo alla velocità di transizione allo stage.

4.3.1 I giunti relativi alla parte destra del robot

Per i tutti i giunti di destra del robot, le cui informazioni relative alle angolazioni sono riportate nella sottostringa Axx…Jxx, l'identificazione del valore che il giunto ha nello stage viene calcolato effettuando una conversione particolare.

Come detto la coppia di caratteri seguenti la lettera maiuscola identificante il giunto, rappresenta un valore numerico esadecimale. Tale valore è un intero compreso nell'intervallo [0,255].

Figura 4.8: Il particolare relativo all'angolazione del giunto "Hip 1 Destro".

Il valore reale del giunto viene calcolato effettuando una complementazione a 255; tale valore viene calcolato come $(ff\text{-}xx)_{hex}$, dove xx è la coppia di caratteri presenti nella stringa rappresentante lo stage, e convertito nella base decimale.

La prima informazione sull'angolazione dei giunti è quella che porta la sottostringa Axx, che si riferisce all'angolazione del giunto hip 1 destro.

In particolare, nella posizione di zero, la sottostringa sarà A7f.

Il valore 7f vale, nella base decimale 127, ma 127 non è il valore dell'angolazione del giunto; il valore esatto è 255-127 ovvero 128. Nella figura 4.8 è possibile notare che il valore dell'angolazione del giunto hip 1 destro è proprio 128.
Poiché ogni carattere maiuscolo presente nella stringa rappresentante lo stage, fa riferimento ad un particolare giunto del robot, si rende necessario l'utilizzo di una tabella che riporta la corrispondenza lettera - giunto.

Carattere	Giunto	Valore in stringa	Valore reale
A	Hip 1	7f	128
B	Hip 2	8e	113
C	Knee	1b	228
D	Ankle 1	80	127
E	Ankle 2	7b	113
F	Ankle 3	80	127
G	Shoulder 1	1e	225
H	Shoulder 2	a4	91
I	Elbow	10	239
J	Wrist	e0	31

Tabella 4.1: La corrispondenza lettera-giunto di destra e le relative posizioni iniziali in simulazione.

4.3.2 I Giunti relativi alla parte sinistra del robot

Per i giunti di sinistra del robot, le cui informazioni relative alle angolazioni sono riportate nella sottostringa Kxx…Txx, l'identificazione del valore che il giunto ha nello stage viene calcolato direttamente dai valori letti nella stringa, senza avere bisogno di ulteriori conversioni particolari.
Come detto la coppia di caratteri seguenti la lettera maiuscola identificante il giunto, rappresenta un valore numerico esadecimale. Tale valore è un intero compreso nell'intervallo [0,255].
Il valore reale del giunto viene calcolato effettuando la conversione da esadecimale a decimale, della coppia di caratteri letti nella stringa ed identificanti il giunto.
La sottostringa K80, nella stringa che rappresenta lo stage 0, è l'informazione relativa alla angolazione del giunto Hip 1 sinistra.

Il valore 80 vale, nella base decimale, 128 che è il vero valore dell'angolazione dello Hip 1 sinistro come è possibile vedere il figura 4.9.

Figura 4.8: Il particolare relativo alla angolazione del giunto "Hip 1 sinistro".

Anche in questo caso, si riporta la tabella lettera-giunto in modo da identificare univocamente la corrispondenza fra essi.

Carattere	Giunto	Valore in stringa	Valore reale
K	Hip 1	80	128
L	Hip 2	71	113
M	Knee	e4	228
N	Ankle 1	7f	127
O	Ankle 2	85	113
P	Ankle 3	7f	127
Q	Shoulder 1	e1	225
R	Shoulder 2	5b	91
S	Elbow	ef	239
T	Wrist	1f	31

Tabella 4.2: Le corrispondenze lettere-giunto di sinistra e le relative posizioni iniziali.

4.3.3 I giunti relativi al busto del robot

La sottostringa che porta le informazioni relative ai giunti del busto, è la sottostringa UxxVxx che nella posizione di zero è Ub2V80.

La U fa riferimento al giunto Waist 2, il giunto che consente di portare il busto del robot verso il basso e verso l'alto. L'informazione che il carattere U porta con se, ovvero l'esadecimale b2, convertito in base decimale, vale 178.

Waist 2 ha la caratteristica viene considerato nel simulatore come se fosse un giunto di destra, pertanto lo si complementa a 255.

La V fa invece riferimento al giunto Waist 1 , il giunto che consente di ruotare il busto del robot in senso orario ed antiorario. L'informazione che la sottostringa V80 porta, ovvero l' informazione sull'angolazione del giunto vale, portato nella base decimale, 127.

Waist 1 viene considerato dal simulatore come se fosse un giunto di sinistra pertanto il valore letto è quello esatto.
Il valore dell'angolazione del giunto Waist 1 sarà per quanto detto 127 mentre il valore di Waist 2 sarà 77. I risultati sono visibili in figura 4.10.

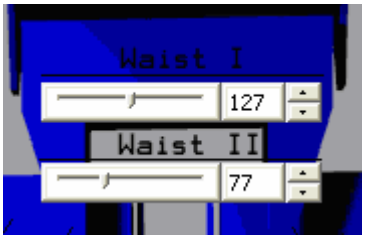

Figura 4.10: Il particolare relativo alle angolazioni di "Waist 1" e "Waist 2".

4.3.4 Le informazioni relative ai cicli

Si è in presenza di un ciclo quando il robot percorre più volte gli stessi stage.
Si pensi per esempio ad un comportamento in cui sono presenti 5 stage. Lo stage 0 è la situazione di partenza (zero del robot), lo stage 1 porta il robot in una certa configurazione e così via fino allo stage 3. Lo stage 4 lo riporta alla condizione iniziale.
Se si vuole effettuare un movimento in cui il robot percorra gli stage nel modo 0-1-2-3-1-2-3-1-2-3-4, sarebbe sconveniente sviluppare una azione complessa con 11 stage in cui 3 stage (1-2-3) portano il robot nella stessa configurazione. Si rende allora necessario l'utilizzo dei cicli.
In questo campo sia "*RobovieMaker*", il software preposto all'esecuzione del movimento, che "*RoboStage*", falliscono completamente.
Il numero di cicli, infatti, è modificabile solo tramite interfaccia grafica e non esistono informazioni riguardanti il loro numero, sia nei file di ingresso al simulatore, che quelli di ingresso al "*RobovieMaker*".
In ogni caso, fissato il numero di cicli da interfaccia (in "RoboStage" è quell'intero posto in alto appena a destra della toolbar) le informazioni riguardanti i cicli vengono portati dai tre caratteri ",xx" e dai caratteri "!xx".

L'informazione che porta la sottostringa ",xx" è relativa a quale sarà lo stage successivo dopo quello che si sta analizzando.

Il significato di tale sottostringa è che lo stage successivo a *quello attuale* è *quello attuale* più o meno *un valore intero*. Il valore intero è presente in forma esadecimale nell'ultimo carattere della sottostringa e rappresenta un numero compreso tra 0 e 15.

Supponendo di analizzare la stringa relativa allo stage 0, la presenza di una informazione del tipo ",+1" indica che lo stage successivo allo stage 0, lo stage che appunto si sta analizzando, è lo stage 1.

In assenza di cicli, allora, tutti gli stage avranno sempre il terzetto ",+1", tranne l'ultimo stage che avrà ,+0 relativamente al fatto che non c'è uno stage successivo.

Nel caso in cui il movimento complesso sia composto da cicli, sarà presente almeno uno stage in cui, la stringa che lo rappresenta, avrà un terzetto che sarà del tipo ",-n" .

La configurazione del robot nello stage successivo all'attuale sarà quella dello stage attuale-n.

Il gruppo di caratteri del tipo !xx porta informazioni solo se nel comportamento è presente un ciclo, altrimenti viene ignorato.

Il significato è relativo allo stage da quale ripartire,nel momento in cui il ciclo si esaurisce.

Generalmente, nel caso in cui il comportamento non presenta cicli, il valore dei caratteri è lasciato a zero pertanto le stringhe relative agli stage avranno la sottostringa "!00".

Il valore dei due caratteri è un valore esadecimale.

Capitolo 5

La programmazione del movimento per Robovie-M

5.1 L'istallazione ed il settaggio di "RobovieMaker"

La creazione di un movimento complesso da fare eseguire al robot umanoide "Robovie-M V3", viene effettuata mediante il software "RobovieMaker".
Il software, per l'istallazione, richiede i seguenti requisiti minimi:

- OS: Microsoft Windows XP;
- CPU: Intel Pentium 4, 1 Ghz;
- RAM: 128 MB;
- HDD: 10 MB liberi;
- PC previsto di porta seriale RS232.

Una volta che si è istallato il software, che necessita, se istallato in una versione di Microsoft Windows europea, il supporto per le lingue orientali, la prima operazione da compiere è il settaggio del def.ini presente nella directory di istallazione del programma.
Tale file contiene informazioni riguardanti lo zero dei motori (in particolare sono presenti i valori delle angolazioni dei giunti nello zero di default) ed informazioni riguardanti la porta seriale utilizzata per la comunicazione con il robot.
Dopo aver salvato sul def.ini il numero della COM relativa alla porta seriale presente nel sistema, ed eventualmente avere variato lo zero di default del robot, bisogna settare le impostazioni della porta che permettere lo scambio di stringhe fra PC e robot.

In particolare la connessione seriale deve essere settata ad una velocità di 38400 bit per secondo, 8 bit per i dati, nessuna parità ed un bit per lo stop.
Dopo aver effettuato queste operazioni è possibile connettere il robot alla seriale, tramite apposito cavo fornito in dotazione, ed iniziare a programmare i movimenti.

5.2 La programmazione di un movimento

Affinché il software funzioni correttamente, è necessario che il robot sia sempre presente, connesso al PC ed alimentato ad una tensione di 6V.
Questo perché, attraverso "RobovieMaker" la programmazione degli stage viene effettuata mediante riscontro reale sul robot.
Per evitare improvvisi sbalzi di tensione, che potrebbero rovinare la scheda di controllo presente, di norma il processo di connessione ed accensione dei motori, avviene secondo questo iter:

- Inserimento del cavo seriale su PC;
- Inserimento cavo seriale sul robot;
- Inserimento cavo alimentazione sul robot;
- Reset dei motori mediante bottoncino presente sul retro della scheda.
- Da software richiesta di connessione;
- Da software accensione servomotori.

Dopo aver connesso ed azzerato i motori, la cui procedura è descritta minuziosamente nel capitolo 1 del presente documento, per la creazione di un nuovo stage è necessario cliccare sul pulsante INS, presente sulla barra degli strumenti.
Come è possibile osservare nella figura 5.1, sulla destra del software viene visualizzato il nuovo stage e, su questo nuovo stage, è possibile cambiare l'angolazione dei servomotori agendo sugli slider presenti a sinistra.

Si fornisce adesso un esempio pratico di programmazione di un movimento complesso.

Sempre in riferimento all'esempio visto nel capitolo 4, si supponga di volere programmare un movimento complesso che consista nel fare alzare il braccio destro del robot di una certa quantità, per esempio di 50 gradi e di riportarlo alla posizione di zero.

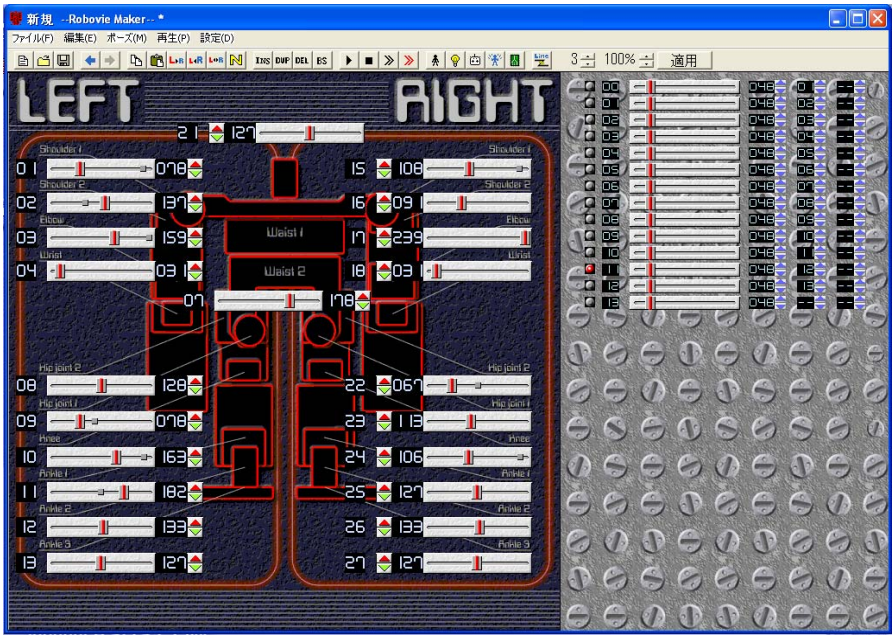

Figura 5.1: Una istantanea del software "RobovieMaker" .

Gli stage che compongono questo movimento sono tre ed in particolare:

- Lo stage 0 è lo stage di zero del robot;
- Lo stage 1 è lo stage in cui tutte le angolazioni dei giunti sono uguali allo stage allo stage 0 tranne il valore del giunto shoulder 1 dx che viene variato di una quantità pari a 50;
- Lo stage 2 in cui le angolazioni dei giunti sono uguali a quelli presenti nello stage 0.

Per creare il suddetto movimento bisogna allora creare tre stage agendo sul pulsante INS dell'interfaccia e modificare il valore dello stage 1 agendo sullo slider relativo al giunto Shoulder 1 dx.
Il movimento al robot viene fatto eseguire agendo sul pulsante play presente nella toolbar.

5.3 L'esecuzione di un movimento preprogrammato

Il software "RobovieMaker" consente all'utente la possibilità di eseguire un movimento preprogrammato, caricando la descrizione del movimento da eseguire direttamente da un file. Tale file deve presentare un insieme di stringhe, ognuna associata ad ogni stage che compongono il movimento.
Ogni stringa porterà informazioni relative ai valori delle angolazioni dei giunti per lo stage cui è associata.
Si analizza adesso una singola stringa e si effettua una spiegazione dettagliata di ogni parte che la compone. In dettaglio, si prenda in esempio la stringa relativa alla posizione iniziale, ovvero la posizione di zero del robot, come si è gia fatto per la descrizione della stringa di zero per il simulatore.
La stringa in questione è:

```
@21A7fB8eC1bD80E7aF80G1eHa4I10Je0K80L71Me4N7fO85P7fQe1R5bSefT1fUb2V80
W80X80,+1!00
```

Suddividendo la stessa a gruppi di tre caratteri; si ottiene:

```
@21, A7f, B8e, C1b, D80, E7a, F80, G1e, Ha4, I10, Je0, K80, L71, Me4, N7f, O85,
P7f, Qe1, R5b, Sef, T1f, Ub2, V80, W80, X80,+1, !00
```

Anche per questo software, si può subito notare come ogni gruppo di tre caratteri è composto da un carattere identificativo (@,A,B,..,V,",",!) e due caratteri.

Il gruppo di tre caratteri in cui il primo è la @, identifica la velocità con la quale l'azione di passaggio allo stage deve essere eseguita, in particolare i due caratteri successivi alla @ rappresentano un intero compreso nell'intervallo [0, 255], e rappresentato secondo la notazione esadecimale [00, ff].

Per i gruppi di tre caratteri in cui il primo carattere è A, B, …, V, il primo carattere identifica univocamente il giunto sul quale si sta facendo riferimento, i due caratteri successivi rappresentano un intero, compreso nell'intervallo [0,255] e rappresentato secondo la base esadecimale [00, ff].

Per i gruppi di tre caratteri in cui il primo carattere è "," o "!", i caratteri successivi presenti portano informazioni riguardante i cicli.

Si fornisce ora una osservazione pratica sul significato di ogni parte che compone la stringa. Supponendo di inserire la stessa all'interno di un file di testo che rappresenta un movimento complesso, si utilizzi "RobovieMaker" per visualizzare il comportamento.

Gli effetti di tale operazione sono visibili in figura 5.2,dove è possibile notare, sulla parte sinistra dell'interfaccia, il valore delle angolazioni di ogni giunto.

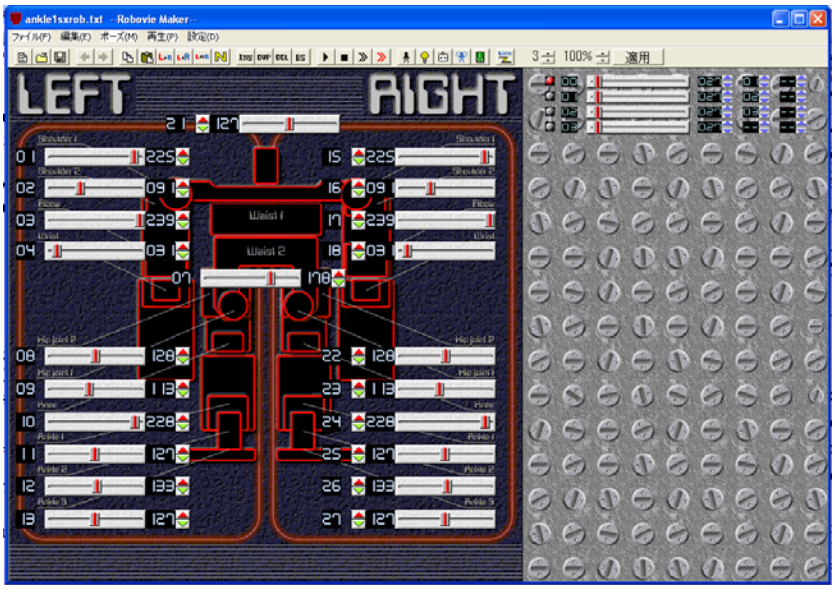

Figura 5.2: La visualizzazione su "RobovieMaker di un comportamento composto da 4 stage.

Il file di testo caricato, rappresentante il movimento complesso, è, come si può vedere sulla destra della figura 5.2, composto da 4 stage. Lo stage 0 è quello

relativo alla stringa che si sta analizzando (il pallino alla destra è visualizzato in rosso poiché lo stage 0 è lo stage selezionato).

Inoltre, non ci si faccia fuorviare dal fatto che il software presenti in alto le scritture Left e Right riferendosi al robot; da prove sperimentali è risultato che se viene effettuata qualche operazione che nel "RobovieMaker" risulta essere a destra (Right), quella stessa operazione il robot la fa a sinistra.

Il primo terzetto di caratteri che si incontra è il terzetto @xx. Come dapprima accennato, la @ fa riferimento al fatto che i due caratteri successivi portano informazioni riguardo la velocità di transizione allo stage.

Nella stringa relativa alla posizione di zero, la sottostringa considerata è @21.

"RobovieMaker" considera la velocità letta dal file, moltiplicando la stessa per un fattore di scala pari a circa 0,82745, ed arrotondando il risultato all'intero più vicino, pertanto il valore esatto della velocità di transizione allo stage è 33*FattScala ≈ 27.3, che arrotondato per difetto all'intero più vicino vale 27. Nella figura 5.3 è visualizzato il particolare di "RobovieMovement"relativo alla velocità di transizione allo stage.

Figura 5.3: Il particolare relativo alla velocità di transizione allo stage 0.

Il fattore di scala è stato scoperto facendo leggere a "RobovieMaker" uno stage in cui la sottostringa relativa alla velocità era @ff. In quel caso l'uscita che si è presentata non è stata 255 bensì 211.

5.3.1 I giunti relativi alla parte sinistra del robot.

Per i tutti i giunti di sinistra del robot, le cui informazioni relative alle angolazioni sono riportate nella sottostringa Axx…Jxx, l'identificazione del valore che il giunto ha nello stage viene calcolato effettuando la stessa conversione vista per i giunti di destra in simulazione.

Come detto la coppia di caratteri seguenti la lettera maiuscola identificante il giunto, rappresenta un valore numerico esadecimale. Tale valore è un intero compreso nell'intervallo [0,255].

Il valore reale del giunto viene calcolato effettuando una complementazione a 255; tale valore viene calcolato come $(ff\text{-}xx)_{hex}$, dove xx è la coppia di caratteri presenti nella stringa rappresentante lo stage, e convertito nella base decimale.

La prima informazione sull'angolazione dei giunti è quella che porta la sottostringa Axx, che si riferisce all'angolazione del giunto hip 2 sinistro.

In particolare, nella posizione di zero, la sottostringa sarà A7f.

Il valore 7f vale, nella base decimale 127, ma 127 non è il valore dell'angolazione del giunto; il valore esatto è 255-127 ovvero 128.

In figura 5.4 è possibile notare che il valore dell'angolazione del giunto hip 2 sinistro è proprio 128.

Figura 5.4: Il particolare relativo al valore dell'angolazione del giunto Hip 2 sinistro.

Poiché anche in questo caso ogni carattere maiuscolo presente nella stringa rappresentante lo stage, fa riferimento ad un particolare giunto del robot, si rende necessario l'utilizzo di una tabella che riporta la corrispondenza fra la lettera presente nella stringa rappresentante ed il giunto cui essa è associata.

Carattere	Giunto	Valore in stringa	Valore reale
A	Hip 2	7f	128
B	Hip 1	8e	113
C	Knee	1b	228
D	Ankle 1	80	127
E	Ankle 2	7b	113
F	Ankle 3	80	127
G	Shoulder 1	1e	225
H	Shoulder 2	a4	91
I	Elbow	10	239
J	Wrist	e0	31

Tabella 5.1: Le corrispondenze lettere-giunti di sinistra e le relative posizioni iniziali.

5.3.2 I giunti relativi alla parte destra del robot

Per i giunti di destra del robot, le cui informazioni relative alle angolazioni sono riportate nella sottostringa Kxx…Txx, l'identificazione del valore che il giunto ha nello stage viene calcolato direttamente dai valori letti nella stringa, senza avere bisogno di ulteriori conversioni particolari.

Come detto la coppia di caratteri seguenti la lettera maiuscola identificante il giunto, rappresenta un valore numerico esadecimale e tale valore è un numero intero compreso nell'intervallo [0,255].

Il valore reale del giunto viene calcolato effettuando la conversione da esadecimale a decimale, della coppia di caratteri letti nella stringa ed identificanti il giunto.

La sottostringa K80, nella stringa che rappresenta lo stage zero, è l'informazione relativa alla angolazione del giunto Hip 2 destro.

Il valore 80 vale, nella base decimale, 128 che è il vero valore dell'angolazione dello Hip 2 destro come è possibile vedere il figura 5.5.

Figura 5.5: Il particolare relativo all'angolazione del giunto Hip 2 destro.

Anche in questo caso, si riporta la la tabella lettera-giunto in modo da identificare univocamente la corrispondenza fra essi.

Carattere	Giunto	Valore in stringa	Valore reale
K	Hip 2	80	128
L	Hip 1	71	113
M	Knee	e4	228
N	Ankle 1	7f	127
O	Ankle 2	85	113
P	Ankle 3	7f	127
Q	Shoulder 1	e1	225
R	Shoulder 2	5b	91
S	Elbow	ef	239
T	Wrist	1f	31

Tabella 5.2: Le corrispondenze lettere-giunti di destra e le relative posizioni iniziali.

5.3.3 I giunti relativi al busto del robot

La sottostringa UxxVxx è la sottostringa che porta le informazioni relative alle angolazioni dei due gradi di libertà del busto.

La U fa riferimento al giunto Waist 2, il giunto che consente di ruotare il busto del robot verso il basso e verso l'alto mentre la lettera maiuscola V fa riferimento al giunto Waist 2, il giunto che consente di ruotarlo in senso orario ed antiorario.

Nello stage di zero, la sottostringa UxxVxx vale Ub2V80. Sia il giunto Waist 1 che il giunto Waist 2 vengono considerati in "RobovieMovement" giunti di destra del robot, ciò comporta che i valori riportati sul software saranno i valori letti dal file e convertiti in decimale.

Come è possibile vedere in figura 5.6, il valore relativo al giunto Waist 1 è proprio 127 ovvero 80 convertito in decimale mentre il valore relativo al giunto Waist 2 è 178, ovvero b2 in base dieci.

Figura 5.6: Un particolare relativo alle angolazioni dei giunti Waist 1 e Waist 2.

5.3.4 Le informazioni non adoperate e le informazioni sui cicli

I due terzetti di caratteri W80 e X80 sono caratteri che, nell'attuale configurazione meccanica del robot, non portano alcuna informazione.

Prove sperimentali ci hanno permesso di affermare che in questa configurazione meccanica possono anche venire omesse.

Per quanto riguarda le informazioni relative ad i cicli, le considerazioni che devono essere fatte sono le stesse viste per la simulazione.

Capitolo 6

Il software "Robovie-Movement"

6.1 Introduzione

La programmazione di un movimento complesso mediante il software "*RobovieMaker*", non risulta essere tanto efficiente.

Ai vantaggi del fatto che si programmi mediante software proprietario e sia possibile visualizzare immediatamente e direttamente sul robot la configurazione programmata, si contrappongono numerosi svantaggi.

Innanzitutto vi è, come detto, la necessità che il robot sia materialmente presente durante tutta la durata della programmazione.

Se si considera che per la programmazione di un movimento complesso composto da una decina stage, utilizzando un approccio sperimentale si impiegano mediamente dalle 10 alle 12 ore, per tutta la durata della programmazione il robot può essere utilizzato esclusivamente da coloro che programmano il movimento, precludendo agli altri la possibilità di lavorare anch'essi sul robot.

Inoltre, il robot deve essere continuamente alimentato a 6V.

Se l'alimentazione viene fornita da uno stabilizzatore di tensione collegato alla rete elettrica, il problema dell'alimentazione non sussiste, ma se essa viene fornita da batterie, allora nasce il problema che in poco tempo la carica delle pile si esaurisce e bisogna aspettare il tempo opportuno per la ricarica.

La programmazione mediante l'utilizzo degli slider può, inoltre, essere non intuitiva. Spesso, infatti, durante la programmazione di uno stage, dopo aver settato i valori delle angolazioni dei giunti uno per uno, ci si rende conto che l'angolazione di qualche giunto non è quella corretta. Poiché non esiste in "*RobovieMovement*" qualcosa che tiene traccia di quali giunti siano stati variati nella programmazione dello stage, come per esempio un comando di undo,

quando si commette un errore, si ritorna indietro di uno stage e si riprogramma il nuovo da capo, con conseguente perdita di tempo e conseguente consumo di batteria.

Per non parlare poi del fatto che l'interfaccia si presenta completamente in giapponese, lingua sconosciuta ai più, che limita pesantemente l'utilizzo del software alle sue funzioni base.

Il lavoro che è stato sviluppato per questa tesi, il software "*RobovieMovement*", non si vuole completamente sostituire al software "*RobovieMaker*", piuttosto cerca, mediante un utilizzo più intelligente delle risorse, una alternativa alla programmazione dei movimenti, accessibile a tutti e soprattutto, semplice da utilizzare.

Per gli utenti che avessero già particolare dimestichezza con "*RobovieMaker*", "*RobovieMovement*" è una buona alternativa per la programmazione dei movimenti in quanto l'uscita, oltre che passata direttamente al robot ed eseguita dallo stesso, grazie al lavoro di tesi dell'allievo Ing. Marisa Saporito, può essere anche caricata come file di movimento su "*RobovieMaker*" e pertanto possono essere visualizzate sul software le proprietà di ogni stage e, se necessario, modificate.

Mediante il software sviluppato, inoltre, è possibile creare in memoria una libreria di comportamenti complessi. Movimenti che, in qualsiasi istante, possono essere richiamati per l'esecuzione da parte del robot e modificati, se si ritiene opportuno.

6.2 Gli obiettivi

Gli obiettivi che sono stati proposti all'inizio del lavoro, erano quelli di sviluppare un software, il più semplice ed intuitivo possibile, che riuscisse a costruire, mediante una descrizione verbale delle variazioni ai giunti, movimenti complessi che potessero essere simulati da un opportuno software di simulazione 3D ed eseguiti dal robot.

Per lo sviluppo di questo software era richiesta la progettazione e l'implementazione di un traduttore che riuscisse a convertire delle espressioni scritte in un particolare linguaggio, in un particolare insieme di stringhe che avessero senso per il simulatore "*RoboStage*" e per il software preposto all'esecuzione del movimento, ovvero il software "*RobovieMaker*".
Mediante questo software doveva essere inoltre possibile costruire una libreria di comportamenti che, in un qualsiasi istante, potesse essere accessibile da parte di altre applicazioni.
Alla fine del lavoro, è possibile affermare che gli obiettivi preposti sono stati ampiamente raggiunti, inoltre sono stati sviluppati numerosi comportamenti complessi funzionanti.

6.3 Le scelte progettuali

Le scelte progettuali per lo sviluppo e l'implementazione di "*RobovieMovement*" sono state molteplici.
Il cuore del software è un traduttore, eseguibile che risulta essere esterno all'interfaccia ed istallato nella stessa directory dove si trova applicazione. Questo presuppone il fatto che, anche in assenza dell'interfaccia, si ha la possibilità di lavorare con il software, utilizzando solamente l'eseguibile dapprima accennato per effettuare le traduzioni.
Il traduttore è stato sviluppato utilizzando i seguenti strumenti:

- DJGPP V2.0, utilizzato come ambiente per lo sviluppo di software in C/C++ in ambiente Microsoft DOS;
- Flex, versione per Microsoft DOS di Lex, utilizzato per lo sviluppo dell'analizzatore lessicale;
- GNU Bison, versione per Microsoft DOS di Yacc, utilizzato per lo sviluppo dell'analizzatore sintattico;
- Gcc, compilatore C utilizzato per la compilazione del traduttore.

Sia per rendere più semplice la programmazione, sia per imparare più rapidamente il linguaggio, la scelta su quale linguaggio implementare è ricaduta in un linguaggio simil-parlato.
La descrizione di un movimento viene effettuata pertanto nella maniera più naturale possibile, per esempio la movimentazione di un giunto di una certa quantità viene descritta come:

muovi shoulder 1 destro di 35 gradi verso l'esterno

mentre per la movimentazione contemporanea di più giunti, è necessario separare le primitive, dalla congiunzione " *e* ":

muovi shoulder 1 destro di 35 gradi verso l'esterno e muovi shoulder 1 sinistro di 35 gradi verso l'esterno

E' inoltre possibile settare la velocità di esecuzione del movimento, dichiarandola semplicemente come:

muovi shoulder 1 destro di 35 gradi alla velocità di 30

L'interfaccia grafica di "*RobovieMovement*", è stata creata grazie all'ambiente di sviluppo *Borland C++ Developer Studio 2006*, un ottimo compilatore RAD che ci ha permesso di creare l'applicazione in C++ in maniera visuale, riducendo notevolmente i tempi di sviluppo del software. Inoltre, la possibilità di espandere lo stesso mediante nuovi strumenti, ci ha permesso di inserire numerose funzionalità, utili per l'utilizzo corretto del linguaggio.
Come già accennato, l'interfaccia è semplicemente un ponte tra l'utente ed il linguaggio; essa ha pertanto il compito di gestione dei file di input e di output, di gestione dell'eseguibile traduttore.exe e dell'interfacciamento con il simulatore "*RoboStage*" e con il software per la movimentazione "*RobovieMaker*".

Il programma, sviluppato in maniera modulare, ed è costituito da due form, ognuno dei quali svolge un ben preciso compito.

In dettaglio il form principale, chiamata *RobovieMovementUnit*, è il form più importante.

E' lui a gestire l'input e l'output, l'eseguibile per la traduzione, il salvataggio dei movimenti programmati in libreria e l'interfacciamento con i software esterni.

Il form *SettingPathUnit* è invece il form adibito al settaggio dei percorsi. Saranno allora presenti in questa classe i percorsi dove sono presenti i file di libreria, gli eseguibili del simulatore e del traduttore e dei sorgenti.

La descrizione dettagliata di tutte le funzionalità del software è fornita nei paragrafi successivi.

6.4. La descrizione dell'interfaccia

L'interfaccia utente di "*RobovieMovement*", molto semplice ed intuitiva, che è possibile osservare in figura 6.1, è formata, dall'alto verso il basso, da un menù del titolo dove sono posizionati i pulsanti per la riduzione ad icona, e per la chiusura, da una serie di pulsanti nella barra degli strumenti, per le azioni più comuni e tre caselle di testo, ognuna con compiti ben specifici.

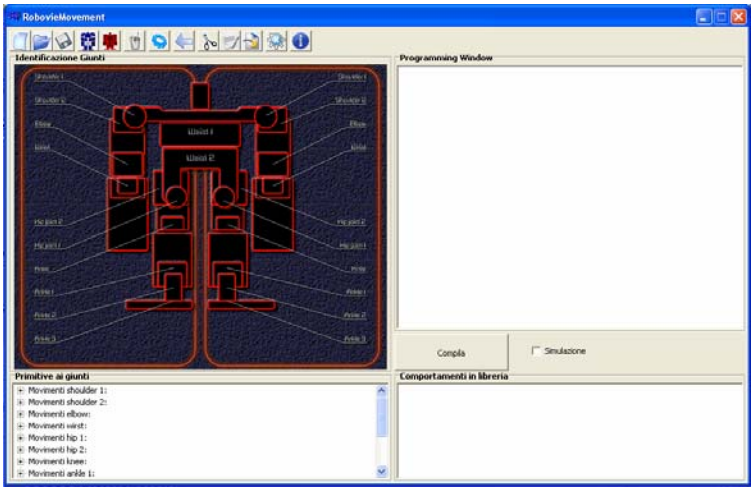

Figura 6.1: Una istantanea del software "RobovieMovement"

La casella di testo a sinistra in basso, posizionata all'interno del pannello "Primitive ai giunti" è una lista ad albero dove sono presenti esempi di espressioni corrette per la movimentazione di un singolo giunto.
Ogni giunto che è possibile muovere, può essere identificato dall'immagine presente in alto a sinistra, all'interno del pannello "Identificazione Giunti".
A destra si può trovare, invece, dall'alto, la casella di testo dove vengono inserite le espressioni da tradurre, anch'esso inserito all'interno di un pannello chiamato "Programming Window" un pulsante "Traduci" che, se premuto effettua la traduzione delle espressioni in ingresso, un checkbox che, se spuntato effettua la traduzione delle espressioni in ingresso per la simulazione ed una casella di testo dove sono riportati tutti i comportamenti che sono presenti in libreria.

6.4.1 La barra degli strumenti

Nella barra degli strumenti, visibile in figura 6.2, sono presenti una serie di pulsanti utili durante l'utilizzo di "RobovieMovement".

Figura 6.2: La barra degli strumenti del software "RobovieMovement".

I piu importanti pulsanti presenti nella toolbar sono i pulsanti di:

- Salva: consente di salvare il movimento programmato e di inserire lo stesso, dopo opportune conferme, nella libreria;
- RobovieMaker: consente di aprire l'applicativo "RobovieMaker" utilizzato per l'esecuzione del movimento programmato.
- Traduci: pulsante che effettua la traduzione delle espressioni inserite.
- Invia al Robot: consente di inviare direttamente al robot tramite seriale la stringa che consente di eseguire il movimento.

6.4.2 La lista ad albero delle primitive ai giunti

La lista ad albero delle primitive è una lista in cui soni presenti, raggruppati per categorie, esempi di corretta dettatura.
L'utilizzo dell'albero è, oltre quello di evitare ogni volta di consultare il manuale per ricordare il modo di dettare le espressioni, quello di riuscire a costruire espressioni più complesse, che in un secondo momento dovranno essere tradotte.
Il suo uso è molto semplice: con un click sugli elementi che non sono foglie, si espande la radice relativa; con un doppio click sulle foglie si inserisce l'espressione selezionata nel componente di testo relativo alle espressioni da tradurre.

6.5 La programmazione di un movimento

La programmazione di un movimento complesso mediante "RobovieMovement", viene effettuato inserendo all'interno della programming window le stringhe che compongono il movimento.
Ad ogni riga di codice scritta nella programming window, corrisponde uno stage nel movimento che viene creato. Più righe implementano pertanto un movimento complesso composto da più stage ed il movimento che verrà creato inizierà con la posizione di zero e terminerà con la stessa posizione.

6.5.1 La programmazione di un movimento da simulare

Si supponga di voler sviluppare un movimento complesso da simulare con "*RoboStage*" composto da due movimenti semplici.
Il primo movimento deve consiste nel muovere il giunto "shoulder 1 sinistro" di 35 gradi ed il secondo nel muovere il giunto "shoulder 1 destro" di 35 gradi.
Entrambi i giunti devono essere mossi verso l'esterno ed il movimento dello shoulder 1 destro deve avvenire dopo il movimento dello shoulder 1 sinistro.

Il modo di costruire questa espressione utilizzando la lista ad albero è:

- Si ricerca all'interno delle espressioni di movimentazione del giunto "shoulder 1", l'espressione corretta;
- Si doppioclicca l'espressione voluta in modo da inserirla nella "Programming Window";
- Si sostituiscono gli elementi dell'espressione con elementi opportuni;
- Si ripete la stessa operazione la movimentazione di shoulder 1 destro.

Come è possibile vedere nella figura 6.3, poiché l'ultima azione va eseguita solo dopo aver conclusa l'azione precedente, l'espressione della nuova movimentazione va scritta nella programming window in una riga diversa.
Questo comporta la creazione di un nuovo stage.

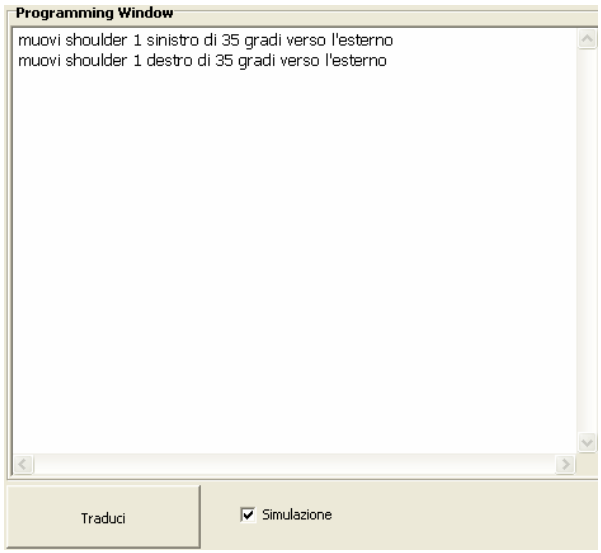

Figura 6.2: Il particolare relativo alla programming window di "RobovieMovement"

Dopo aver tradotto le espressioni per la simulazione, se si invia il file al software "RoboStage", è possibile visualizzare in simulazione il movimento programmato.

Il risultato di tale operazione è visibile nella figura 6.3 a,b,c,d,da notare il fatto che l'azione è composta da quattro stage. Il primo stage e l'ultimo sono posizioni dei zero.

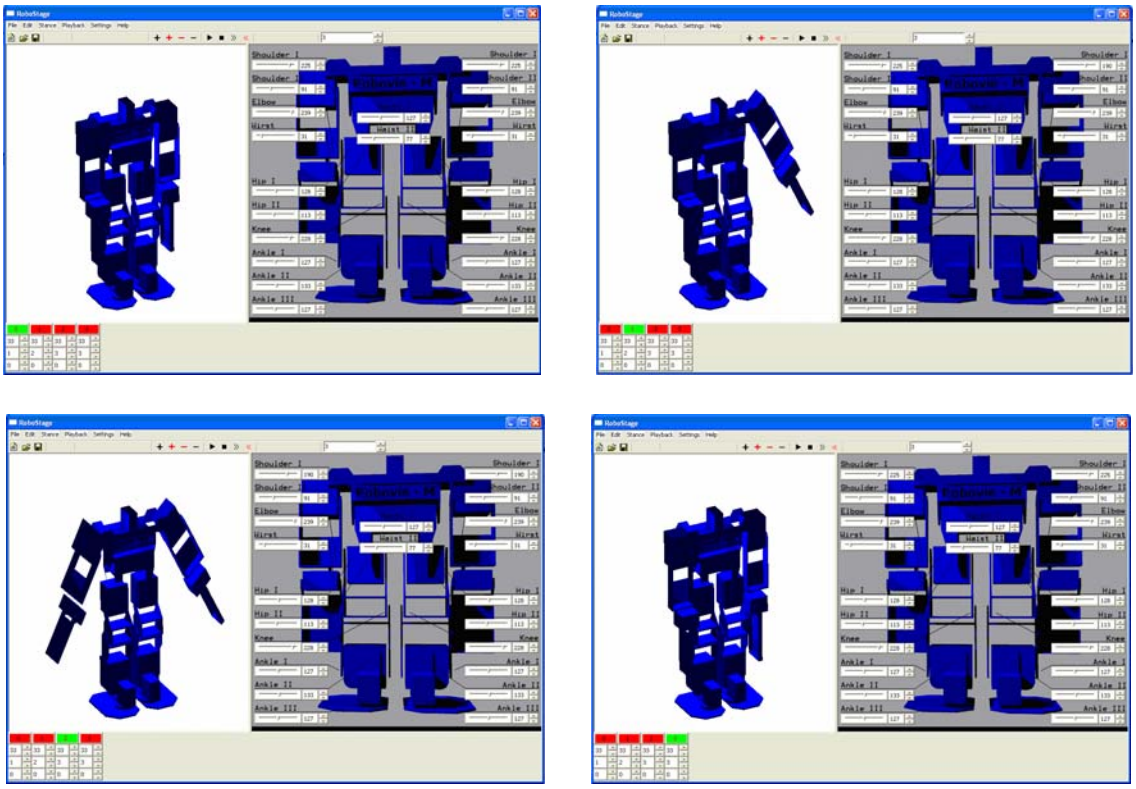

Figura 6.3: Il risultato della simulazione del movimento programmato.

6.5.2 Lo studio del file di uscita per il simulatore

Il file di testo generato con "RobovieMovement", utilizzato per simulare il movimento visto nel paragrafo precedente, è composto da quattro stringhe, ognuna associata agli stage di cui il movimento è composto.

Nella fattispecie le stringhe generate da "RobovieMovement" sono:

@21A7fB8eC1bD80E7aF80G1eHa4I10Je0K80L71Me4N7fO85P7f**Qe1**R5bSefT1fUb2V80,+1!00

@21A7fB8eC1bD80E7aF80**G1e**Ha4I10Je0K80L71Me4N7fO85P7f**Qbe**R5bSefT1fUb2V80,+1!00

@21A7fB8eC1bD80E7aF80**G41**Ha4I10Je0K80L71Me4N7fO85P7fQbeR5bSefT1fUb2V80,+1!00

@21A7fB8eC1bD80E7aF80G1eHa4I10Je0K80L71Me4N7fO85P7fQe1R5bSefT1fUb2V80,+0!00

La movimentazione di shoulder 1 sinistro, che è la prima azione che viene fatta eseguire al robot, apporta l'unica modifica che avviene alla prima stringa per creare la seconda.

In particolare la sottostringa Qxx passa da Qe1 nella stringa relativa alla posizione di zero, a Qbe nella stringa relativa allo stage 1.

Questo comporta lo spostamento del giunto shoulder 1 sinistro, da una posizione di 225 gradi (identificata dall'esadecimale e1) ad una posizione di 190 (identificata dall'esadecimale be).

Da ciò si evince che per muovere verso l'esterno il giunto shoulder 1 sinistro di una certa quantità, il valore che deve essere scritto nella stringa relativa al nuovo stage deve essere uguale al valore che il giunto aveva nello stage precedente meno l' entità dello spostamento $(e1\text{-}23=be)_{hex}$.

Si analizza adesso quello che accade nel terzo stage, lo stage relativo alla movimentazione di shoulder 1 destro. Analizzando la terza stringa in relazione alla seconda, si nota che l'unica variazione che si ha è quella relativa alle informazioni che portano i caratteri Gxx che passano da G1e a G41.

In particolare, il valore passa da 225 (è un giunto di destra, il valore è ottenuto come 255-30, dove 30 è il corrispondente decimale di 1e in esadecimale), a 190 (valore ottenuto come 255-65, dove 65 è il corrispondente decimale di 41 in esadecimale).

In questo caso per muovere verso l'esterno il giunto shoulder 1 destro di una certa quantità, il valore che deve essere presente nella stringa relativa allo stage deve essere uguale al valore che lo stesso aveva nello stage precedente più il valore desiderato $(1e+23=41)_{hex}$.

Si rende necessario allora riportare una tabella in cui sono presenti le operazioni che devono essere eseguite, per tutte le movimentazioni possibili in ogni singolo giunto.

Nella tabella 6.1 sono riportati gli identificativi dei giunti, i movimenti che essi possono compiere e l'operazione (somma o sottrazione) che va applicata al valore corrispondente all'angolazione del giunto dallo stage precedente.

ID Giunto	Movimentazione	Operazione
Shoulder 1 destro	verso l'esterno	+
Shoulder 2 destro	verso l'alto	-
Elbow destro	in senso orario	-
Wrist destro	verso l'esterno	+
Hip 1 destro	verso l'esterno	+
Hip 2 destro	in avanti	-
Knee destro	in avanti	-
Ankle 1 destro	in avanti	+
Ankle 2 destro	in senso orario	-
Ankle 3 destro	in senso orario	+
Shoulder 1 sinistro	verso l'esterno	-
Shoulder 2 sinistro	verso l'alto	+
Elbow sinistro	in senso orario	-
Wrist sinistro	verso l'esterno	-
Hip 1 sinistro	verso l'esterno	-
Hip 2 sinistro	in avanti	+
Knee sinistro	in avanti	+
Ankle 1 sinistro	in avanti	-
Ankle 2 sinistro	in senso orario	-
Ankle 3 sinistro	in senso orario	+

Tabella 6.1: Le operazioni che vengono effettuate per la movimentazione dei giunti.

6.5.3 La programmazione di un movimento da eseguire

Si supponga adesso di sviluppare un movimento complesso da far eseguire al robot mediante l'utilizzo di "RobovieMovement", composto da due movimenti contemporanei.

Il movimento deve consistere nel muovere il giunto "shoulder 1 sinistro" di 35 gradi ed il giunto "shoulder 1 destro" di 35 gradi.

Entrambi i giunti devono essere mossi verso l'esterno.

Per programmare il movimento con "RobovieMovement" :

- Si ricerca all'interno delle espressioni di movimentazione del giunto "shoulder 1", l'espressione corretta;
- Si effettua un doppio click sull'espressione voluta in modo da inserirla nella "Programming Window";
- Si aggiunge la stringa " e " in modo da effettuare l'operazione successiva contemporaneamente e si inserisce il template dell'altro movimento;
- Si inseriscono gli elementi necessari e si traduce l'espressione così costruita per l'esecuzione del movimento.

Come è possibile osservare nella figura 6.4, l'espressione costruita è solo su di una riga proprio perchè il movimento degli shoulder 1 destro e sinistro deve essere eseguito contemporaneamente. Il programma, inoltre, deve fornire un file per il programma "RobovieMaker", pertanto non c'è la spunta sul checkbox.

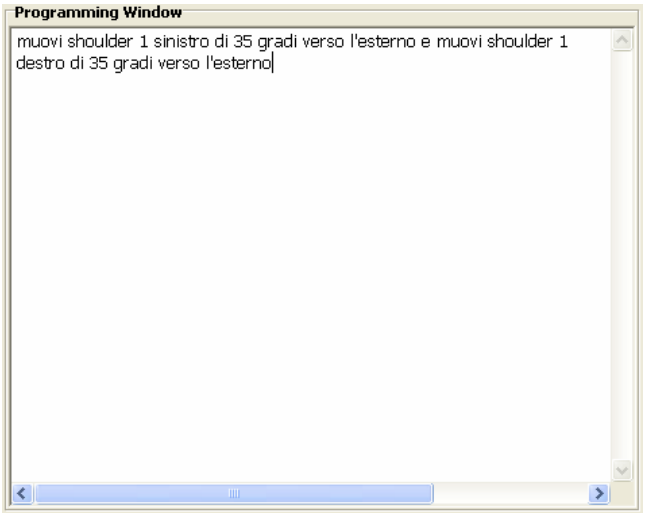

Figura 6.4: Il particolare della programming window relativa al software "RobovieMovement".

Caricato il file di uscita su "*RobovieMaker*", l'esecuzione del comportamento viene effettuato inviando opportune stringhe tramite seriale mediante il tasto *play* presente sulla barra degli strumenti.

6.5.4 Lo studio del file di uscita per "RobovieMaker"

Il file di testo generato con "RobovieMovement", utilizzato per eseguire mediante il software "RobovieMaker" il movimento programmato nel paragrafo precedente, è composto da tre stringhe, ognuna associata agli stage di cui il movimento è composto.

Nella fattispecie le stringhe generate da "RobovieMovement" sono:

@21A7fB8eC1bD80E7aF80**G1e**Ha4I10Je0K80L71Me4N7fO85P7f**Qe1**R5bSefT1fUb2V80,+1!00
@21A7fB8eC1bD80E7aF80**G41**Ha4I10Je0K80L71Me4N7fO85P7f**Qbe**R5bSefT1fUb2V80,+1!00
@21A7fB8eC1bD80E7aF80G1eHa4I10Je0K80L71Me4N7fO85P7fQe1R5bSefT1fUb2V80,+0!00

Come si può osservare, la movimentazione contemporanea di shoulder 1 sinistro e shoulder 1 destro, apporta due modifiche sulla prima stringa per creare la seconda.

La modifica relativa alla informazione che portano i caratteri Qxx, che passano da Qe1 a Qbe, è relativa alla movimentazione di shoulder 1 destro, la modifica alle informazioni di Gxx, che passano da G1e a G41, è relativa alla movimentazione di shoulder 1 sinistro.

Lo spostamento del giunto shoulder 1 sinistro, che va da una posizione di 225 gradi (identificata dall'esadecimale 1e, calcolata come 255-30 poiché si sta considerando un giunto di sinistra) ad una posizione di 190 (identificata dall'esadecimale 41 e calcolata sempre come 255-65), causa lo spostamento di shoulder 1 sinistro verso l'esterno.

Da ciò si evince che per muovere verso l'esterno il giunto shoulder 1 sinistro di una certa quantità, il valore che deve essere scritto nella stringa deve essere uguale al valore che il giunto aveva nello stage precedente, meno la quantità desiderata $(1e\text{-}23=41)_{hex}$, dove $23_{hex} = 35_{dec.}$

In particolare, il valore passa da 225 (è un giunto di destra, il valore è ottenuto come 255-30, dove 30 è il corrispondente decimale di 1e in esadecimale), a 190 (valore ottenuto come 255-65, dove 65 è il corrispondente decimale di 41 in esadecimale).

ID Giunto	Movimentazione	Operazione
Shoulder 1 destro	verso l'esterno	-
Shoulder 2 destro	verso l'alto	+
Elbow destro	in senso orario	+
Wrist destro	verso l'esterno	-
Hip 1 destro	verso l'esterno	-
Hip 2 destro	in avanti	+
Knee destro	in avanti	+
Ankle 1 destro	in avanti	-
Ankle 2 destro	in senso orario	+
Ankle 3 destro	in senso orario	-
Shoulder 1 sinistro	verso l'esterno	+
Shoulder 2 sinistro	verso l'alto	-
Elbow sinistro	in senso orario	+
Wrist sinistro	verso l'esterno	+
Hip 1 sinistro	verso l'esterno	+
Hip 2 sinistro	in avanti	-
Knee sinistro	in avanti	-
Ankle 1 sinistro	in avanti	+
Ankle 2 sinistro	in senso orario	+
Ankle 3 sinistro	in senso orario	-

Tabella 6.2: Le operazioni che vengono effettuate per la movimentazione di un giunto per l'esecuzione

Nella tabella 6.2 sono riportati gli identificativi dei giunti, i movimenti che essi possono compiere e l'operazione di somma o sottrazione che va applicata al valore corrispondente all'angolazione del giunto dallo stage precedente.

6.6 La libreria di comportamenti

Oltre ad essere un nuovo strumento per la creazione dei movimenti complessi, "RobovieMovement" offre ai programmatori la possibilità di sviluppare nuovi comportamenti e di inserire gli stessi all'interno di una libreria.
Lo scopo della libreria è quello di potere accedere a questi comportamenti in un qualsiasi momento.

Tale libreria, sviluppata come lavoro per questa tesi, è composta dai comportamenti più comuni che possono essere fatti eseguire dal robot ed è costituita, per ogni comportamento, da quattro file.
All'atto dell'inserimento in libreria di un comportamento complesso, che viene normalmente effettuato dopo aver provato sul robot l'efficienza dello stesso, vengono salvati in directory diverse:

- Un file di testo che è il file sorgente del comportamento nel linguaggio "RobovieMovement";
- Un file con estensione .rbs, che è l'insieme di stringhe che rappresentano il movimento per il simulatore "RoboStage";
- Un file di testo che è l'insieme di stringhe che rappresentano il movimento da poter eseguire sul robot mediante il software "RobovieMaker";
- Un file contenente stringhe che, inviate al robot mediante un cavo seriale RS232, fanno eseguire al robot il comportamento complesso sviluppato.

Capitolo 7

I risultati sperimentali ottenuti su "Robovie-M"

7.1 Introduzione

Come ambiente di test per il robot umanoide è stato scelto la RoboCup, in particolare la Humanoid Kid Size League. Si è reso quindi necessario andare a creare una serie di comportamenti adatti all'ambiente in cui il robot si potesse trovare ad operare.

Sono stati realizzati i principali movimenti necessari per il funzionamento del robot in maniera autonoma, fra i quali: il calcio della palla, la parata centrale, la parata a destra, la parata a sinistra, la distesa per terra ed i sollevamenti da terra in entrambi i casi ovvero quando il robot si trova a faccia in su e quando si trova a faccia in giù.

Insieme alle camminate frontali, laterali ed alle rotazioni questo insieme di movimenti si è dimostrato sufficiente per il funzionamento autonomo del robot per l'ambiente di test scelto.

7.2 Il calcio

Per un robot che gioca a calcio indispensabile è saper calciare la palla.

Si è implementato, per raggiungere lo scopo prefissato, un movimento che consentisse al robot di poter calciare la palla quando questa si presenta davanti a lui. Il movimento che ne è risultato è un movimento stabile e nello stesso tempo veloce.

Alla palla viene impressa una forza tale da poterla spostare di circa 50 cm. Nelle prove sperimentali effettuate, su un campo conforme alla Humanoid Kid Size League, il robot è riuscito a far goal da fuori dall'area del portiere.

Nel programmare questo particolare comportamento oltre a determinare la sequenza di stage corretta si è dovuto prestare molta attenzione alle velocità di transizioni tra di essi, al fine di ottenere una comportamento stabile e sicuro, ma nello stesso tempo veloce.

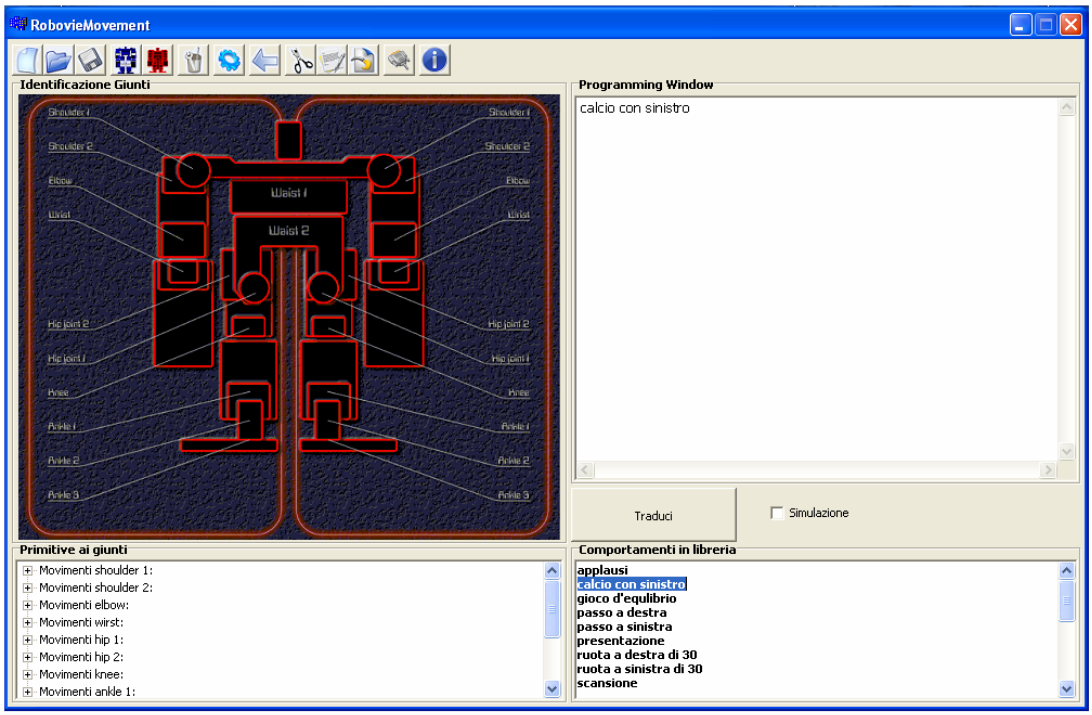

Figura 7.1: La programmazione del calcio col piede sinistro direttamente da libreria.

La figura 7.2 riporta la sequenza presa dal video del comportamento sviluppato mentre la figura 7.1 riporta la possibilità di programmare il movimento sviluppato come comportamento di libreria.

Come è possibile vedere nella foto 7.2b, il robot compie come primo movimento l'apertura delle braccia ovvero la movimentazione di entrambi gli shoulder verso l'esterno.

Questo consente di equilibrare il peso del robot opportunamente, in maniera tale che, durante l'azione del calcio quando esso si trova in equilibrio su di una gamba, non ribalta.

Il robot ritorna, dopo aver calciato la palla con una certa forza, nella posizione di zero ed è pronto subito per effettuare qualsiasi altro comportamento.

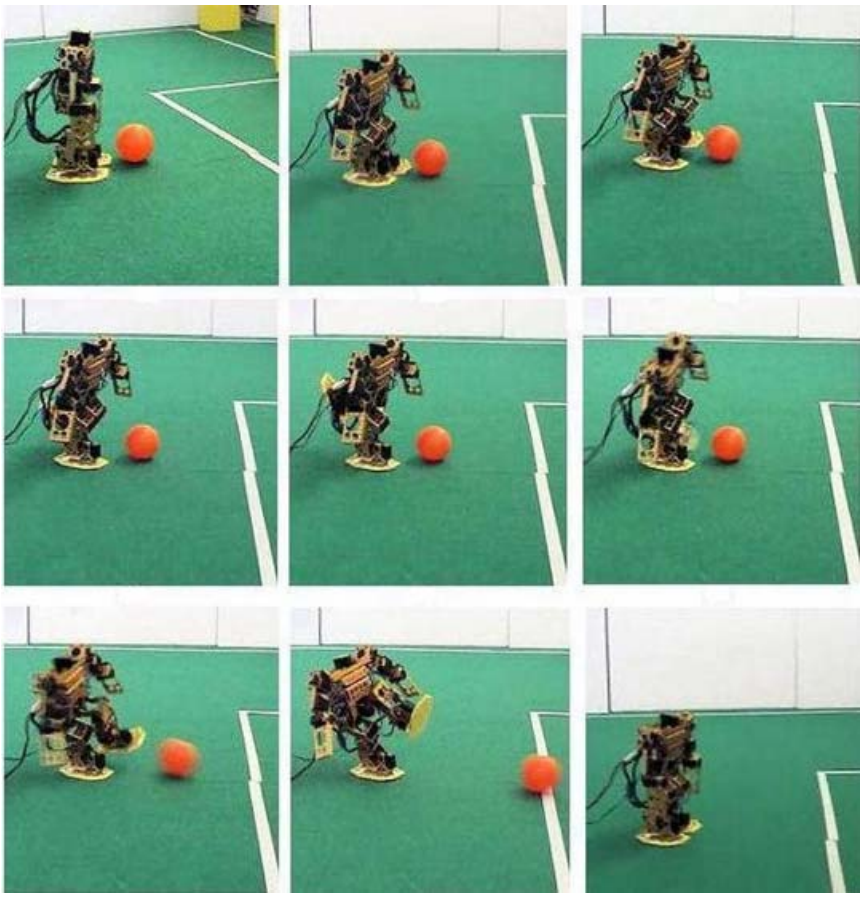

Figura 7.2: La sequenza dei movimenti che il robot effettua per calciare la palla.

Nella figura 7.3, che riprende la stessa sequenza vista di fronte, è maggiormente possibile apprezzare l'altezza della gamba con il quale il robot effettua il calcio e la sequenza dei movimenti di ritorno.
Da notare sono inoltre i minimi bilanciamenti che forniscono le movimentazioni dei giunti del braccio.

Poiché il movimento del calcio della palla, è un movimento che viene effettuato ponendo il robot in equilibrio su di una gamba, è stato possibile sviluppare questo comportamento grazie alla potenza che offrono i servomotori montati sulle caviglie, che durante questo movimento sono i giunti che subiscono maggiore sollecitazione.

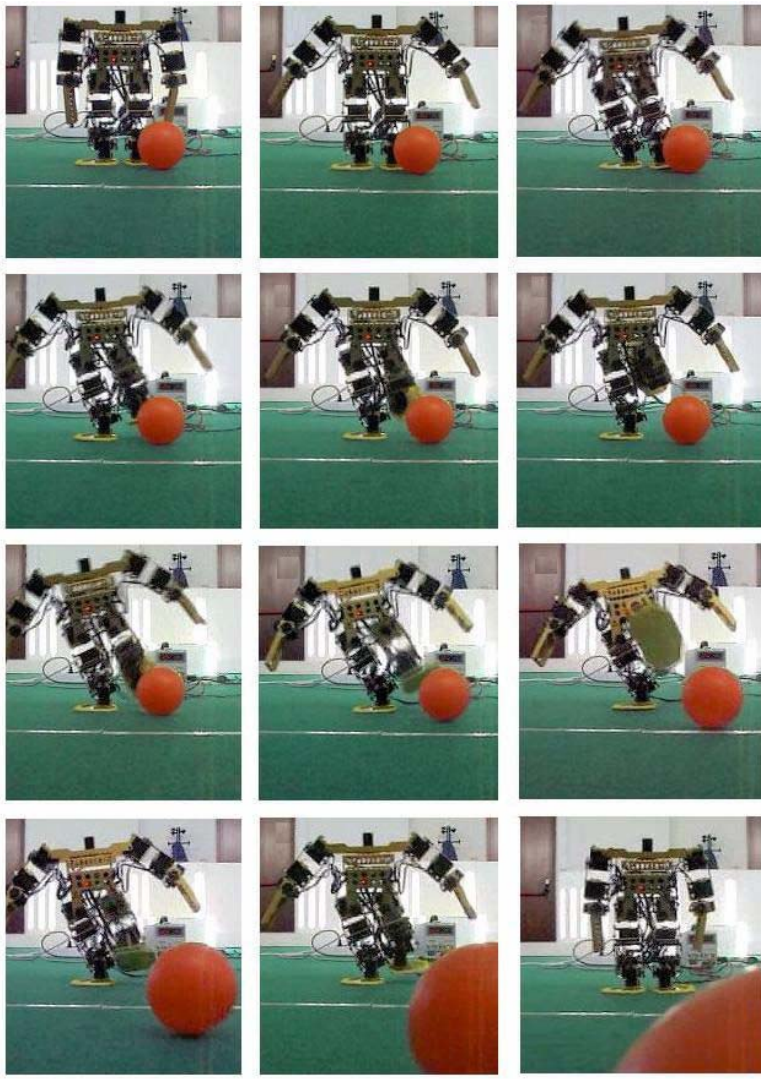

Figura 7.3: La sequenza del calcio della palla vista da una posizione frontale

Come è possibile osservare nella figura 7.3, subito dopo aver calciato la palla, il robot rimane in equilibrio e ritorna tranquillamente nella sua posizione iniziale.

Poiché la palla non è molto pesante, la forza impressa per spostarla non causa al robot perdita di stabilità, inoltre l'impatto dal piede sulla palla non procura agli attuatori montati sui piedi, grosse sollecitazioni.

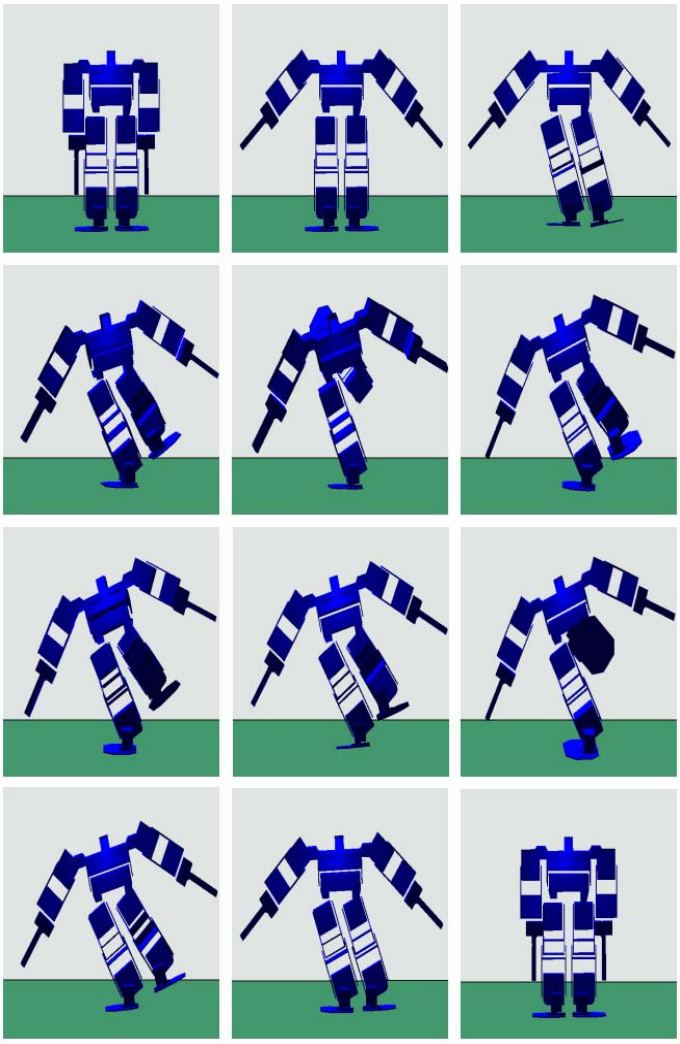

Figura 7.4: La sequenza del movimento relativo al calcio della palla in simulazione.

Nella immagine 7.4 è possibile osservare la sequenza di pose che il robot percorre per eseguire il suo comportamento in simulazione. Particolare risulta essere il movimento di posizionamento prima che la gamba oscillante colpisca la palla.

Per effettuare questo movimento, di importanza rilevante sono state le movimentazioni dei giunti posizionati nelle anche e nelle caviglie.

7.3 La parata centrale

Altri comportamenti messi a punto e strettamente connessi con il gioco del calcio sono le parate.

I tipi di parata sviluppati sono state tre. In tutti i casi il robot rimane nella posizione di double support ovvero entrambi i piedi sono poggiati a terra, ma riesce a ricoprire una porzione di porta maggiore che quando si trova in posizione di zero.

Volutamente, per ragioni di sicurezza, non sono state sviluppate parate che prevedessero la distensione del corpo del robot per terra.

In ogni caso un comportamento di questo tipo non sarebbe difficile da sviluppare poiché basta perdere l'equilibrio su di un lato e lasciarlo cadere.

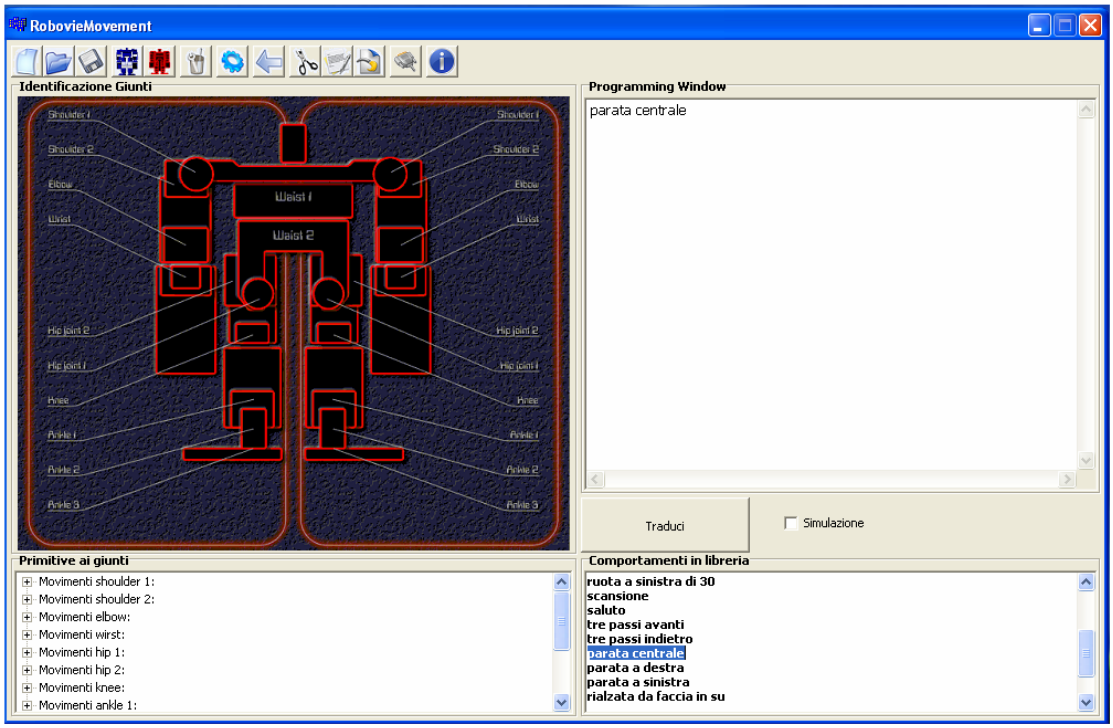

Figura 7.5: La programmazione della parata centrale come comportamento di libreria.

Nella figura 7.5 è riportato uno screenshot del software "*RobovieMovement*" in cui viene visualizzata la programmazione della parata centrale come comportamento di libreria.

Come è possibile osservare nella figura 7.6, il movimento di parata centrale cerca di far ricoprire al robot una porzione di porta molto ampia, grazie alla rotazione di polsi, che consente al robot di porsi in una posizione di attesa della palla.
Inoltre, vengono effettuati delle rotazioni sulle anche e sulle caviglie in modo tale da posizionare il robot a gambe divaricate.

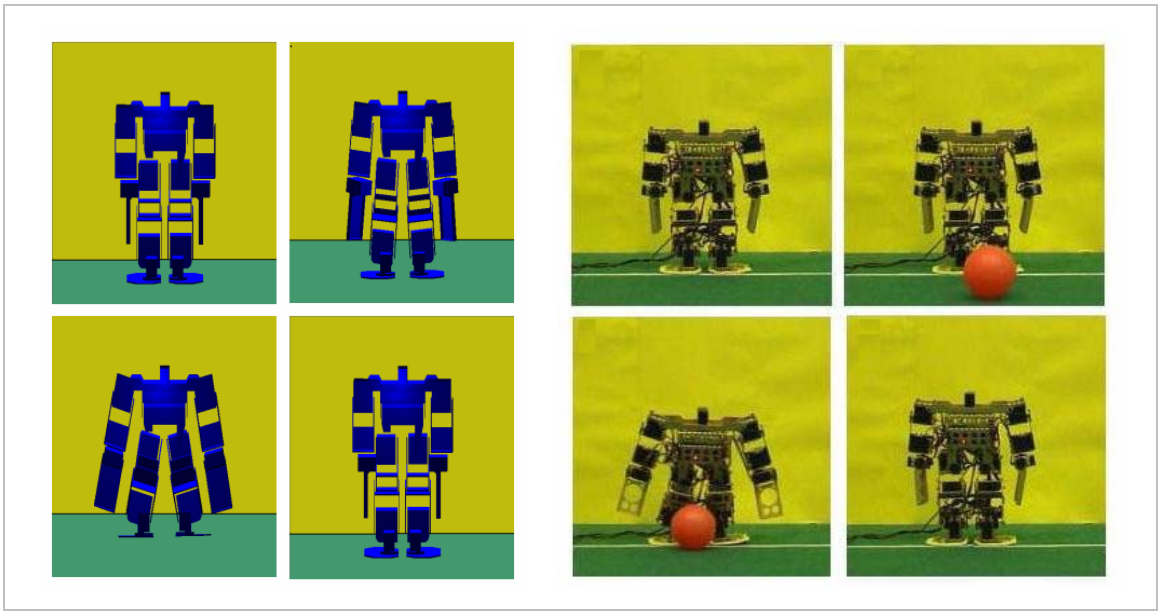

Figura 7.6: La parata centrale vista in simulazione e nella realtà.

7.4 La parata a destra ed a sinistra

La parata a destra e quella a sinistra, vengono effettuate facendo rimanere il robot in double support, pertanto anche in questo caso la base di appoggio del robot è abbastanza ampia e tale da potergli consentire un movimento veloce e sicuro.
Il piegamento sulle ginocchia è dovuto a variazioni di angolazioni sugli hip mentre la rotazione dei polsi fa assumere al robot una posizione di copertura della porta maggiore.
Come è possibile osservare nella figura 7.7 anche il comportamento di parata a destra e quello di parata a sinistra sono comportamenti di libreria pertanto è

possibile programmare il movimento utilizzando la libreria messa a disposizione da "*RobovieMovement*"

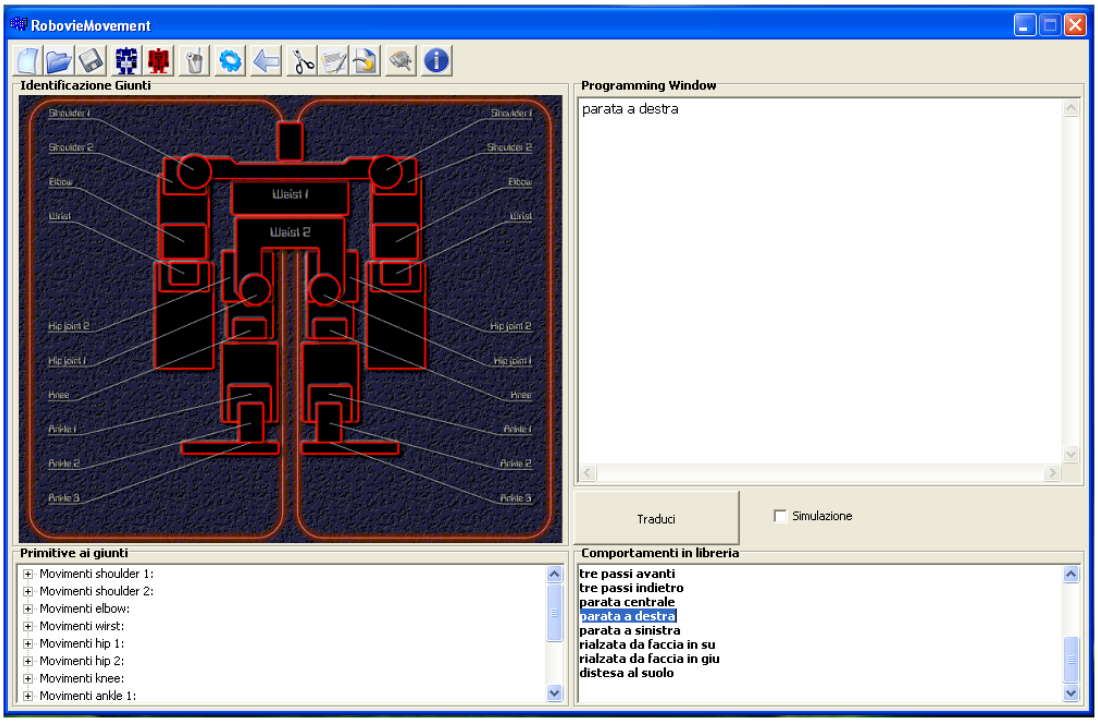

Figura 7.7: La programmazione del movimento parata a destra come comportamento di libreria.

Nelle figure 7.8 e 7.9 sono riportate le due parate discusse sia in simulazione che nella realtà.

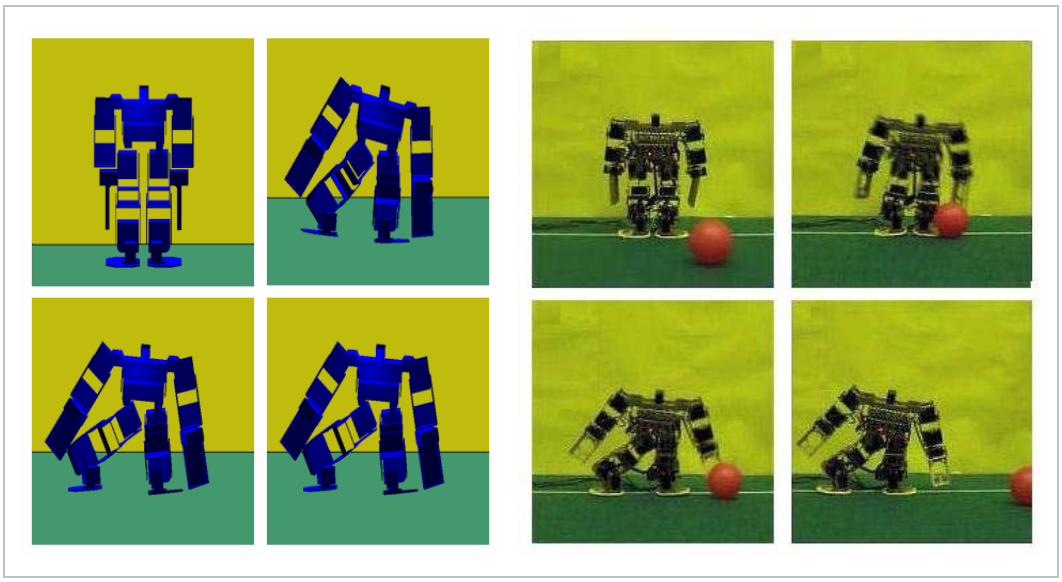

Figura 7.8: La parata a sinistra vista in simulazione e nella realtà.

Le due parate sviluppate sono perfettamente simmetriche e possono essere eseguite anche in seguito a passi laterali.

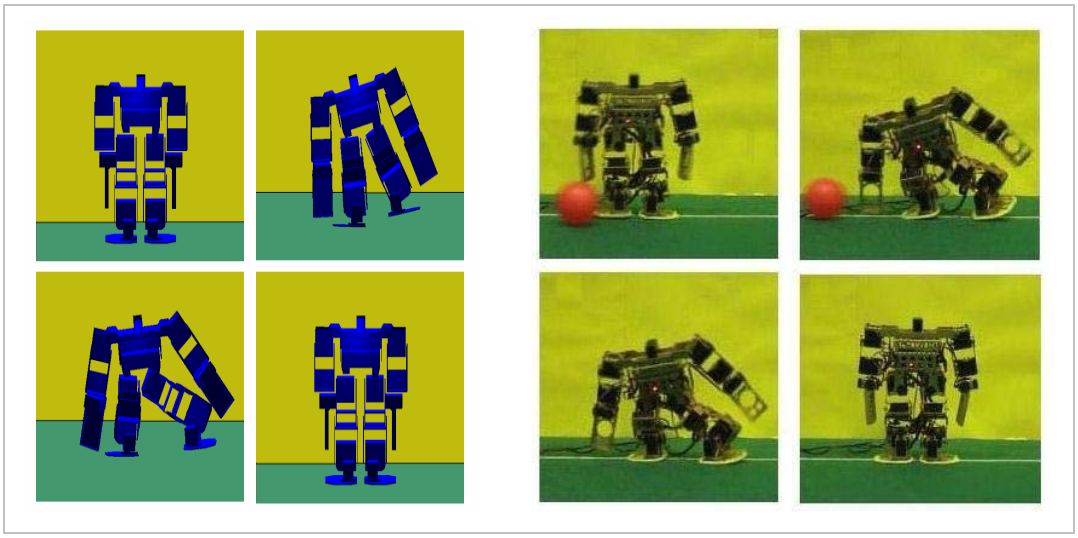

Figura 7.9: La parata a destra vista in simulazione e nella realtà.

Particolari sono le rotazioni che vengono eseguite dai giunti delle anche e delle caviglie, giunti che, per l'esecuzione del movimento sono quelli più sollecitati.

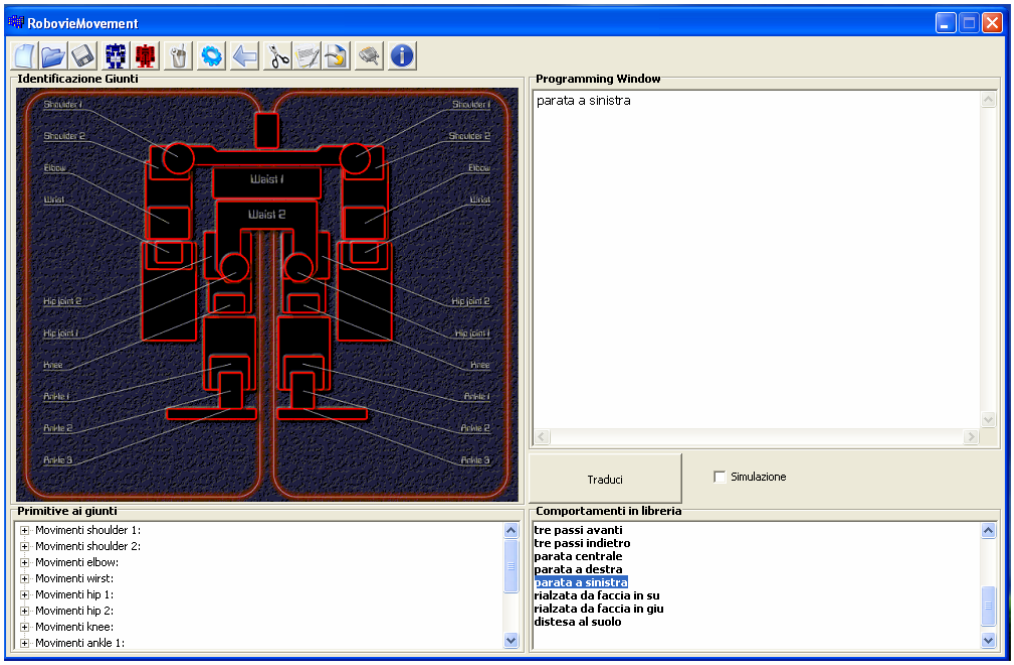

Figura 7.10: La programmazione del movimento parata a sinistra come comportamento di libreria.

7.5 Il movimento di distesa al suolo

Tra i vari comportamenti sviluppati e presenti nella libreria, c'è una particolare categoria che comprende tutta quella serie di movimenti che permettono di cambiare la base d'appoggio del robot.

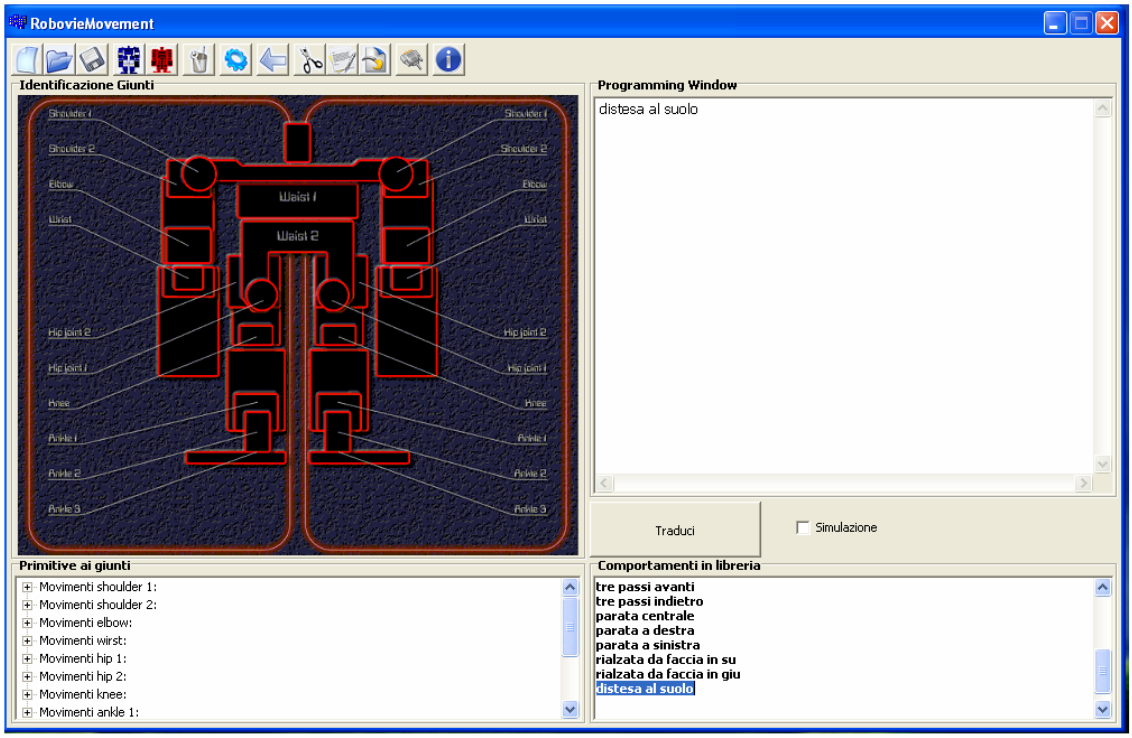

Figura 7.11: La programmazione della distesa al suolo come comportamento di libreria.

Il comportamento che porta il robot da posizione eretta a posizione distesa può essere utile, per esempio, nel momento in cui il controllore si accorge che la carica rimasta non è più sufficiente e, per evitare rovinose cadute causata dall'insufficienza dell'alimentazione ai giunti, il robot si distende dolcemente al suolo, pronto per l'inserimento di una batteria carica.

Come tutti i comportamenti di libreria, anche questo può essere programmato direttamente richiamandolo dalla programming window, come visibile in figura 7.11.

Come è possibile osservare nella figura 7.12, il movimento ha richiesto la coordinazione di molti giunti contemporaneamente.

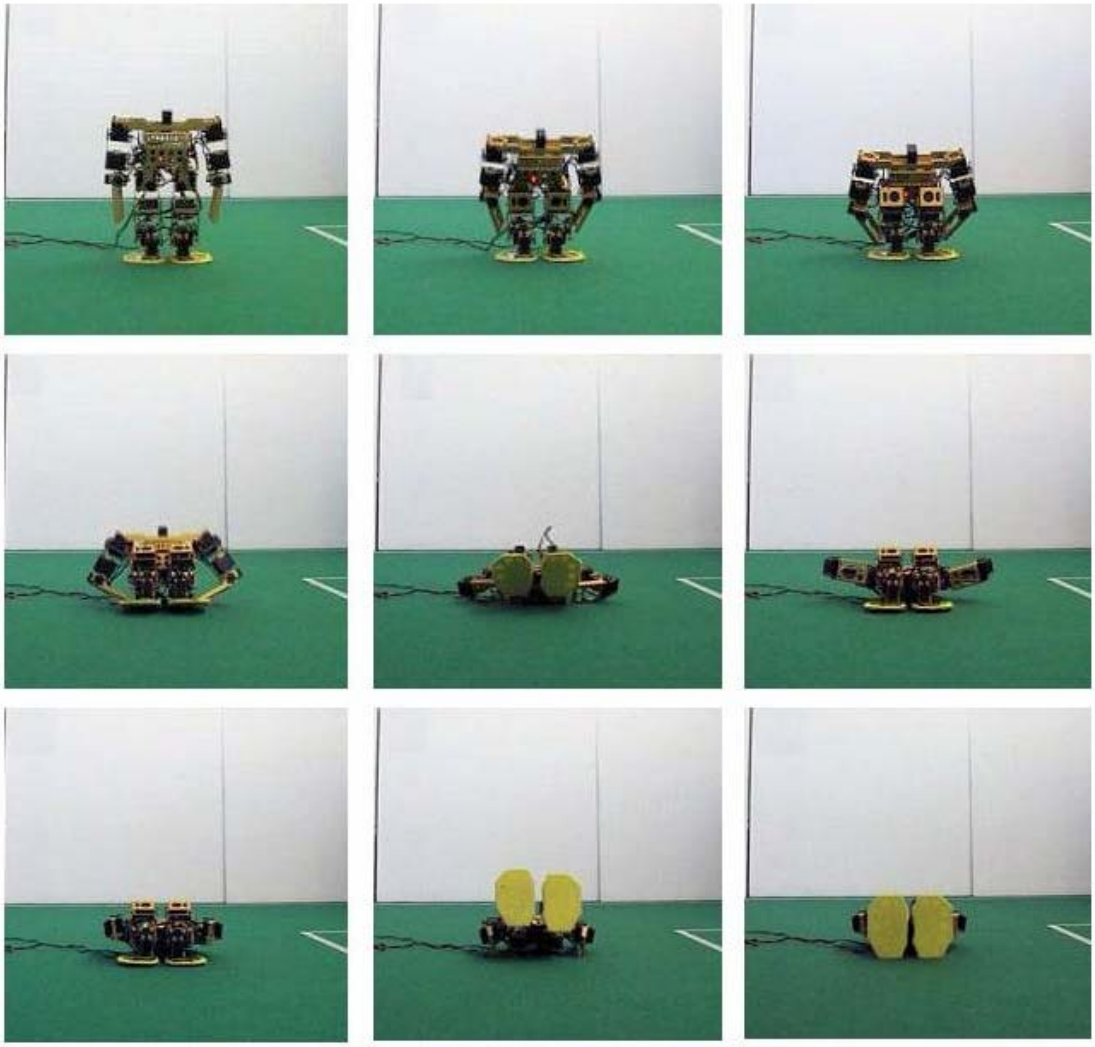

Figura 7.12: Le fasi principali del comportamento osservato da una posizione frontale.

In particolare i giunti delle spalle e delle braccia hanno consentito di sviluppare il movimento per il supporto durante la distensione, e la movimentazione dei giunti delle ginocchia ha consentito al robot di abbassarsi lentamente, evitando brusche collisioni col terreno.

Anche dall' angolazione che è possibile osservare nella figura 7.13, risulta evidente il massimo sforzo dei giunti delle ginocchia, che consentono al robot di non cadere, e l'aiuto che viene fornito al movimento dai giunti delle braccia.

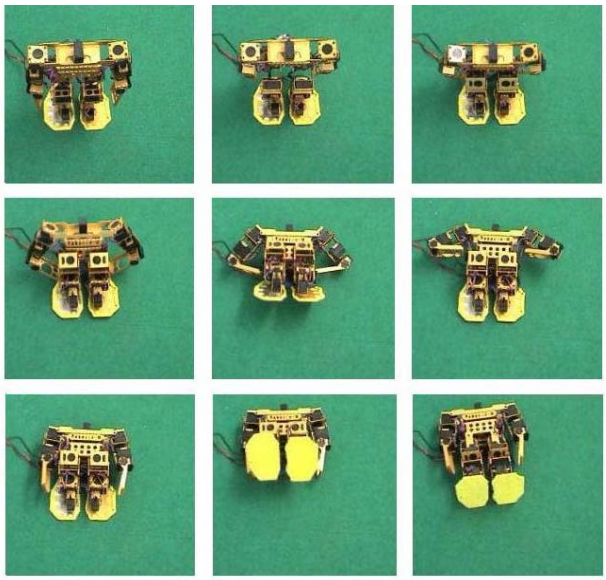

Figura 7.13: Il comportamento di distesa al suolo visto dall'alto.

Il comportamento, che è possibile osservare in simulazione nella figura 7.14, evidenzia la fase iniziale del movimento, fase in cui si ha il passaggio da double support a total support.

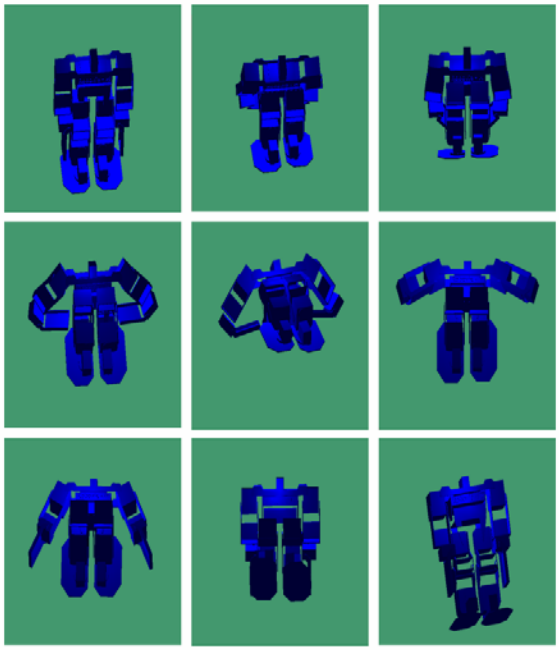

Figura 7.14: Il comportamento di distesa al suolo osservato in simulazione.

7.6 Il sollevamento da terra da faccia in su

Di grande importanza risultano essere, in ambito RoboCup, le rialzate da terra. Capita non di rado, infatti, che il robot a causa di collisioni con altri robot o ostacoli non evitati, cade per terra. A seconda della posizione in cui si può trovare il robot in seguito ad una caduta, sono stati sviluppati due comportamenti che permettessero di farlo ritornare nella posizione eretta.

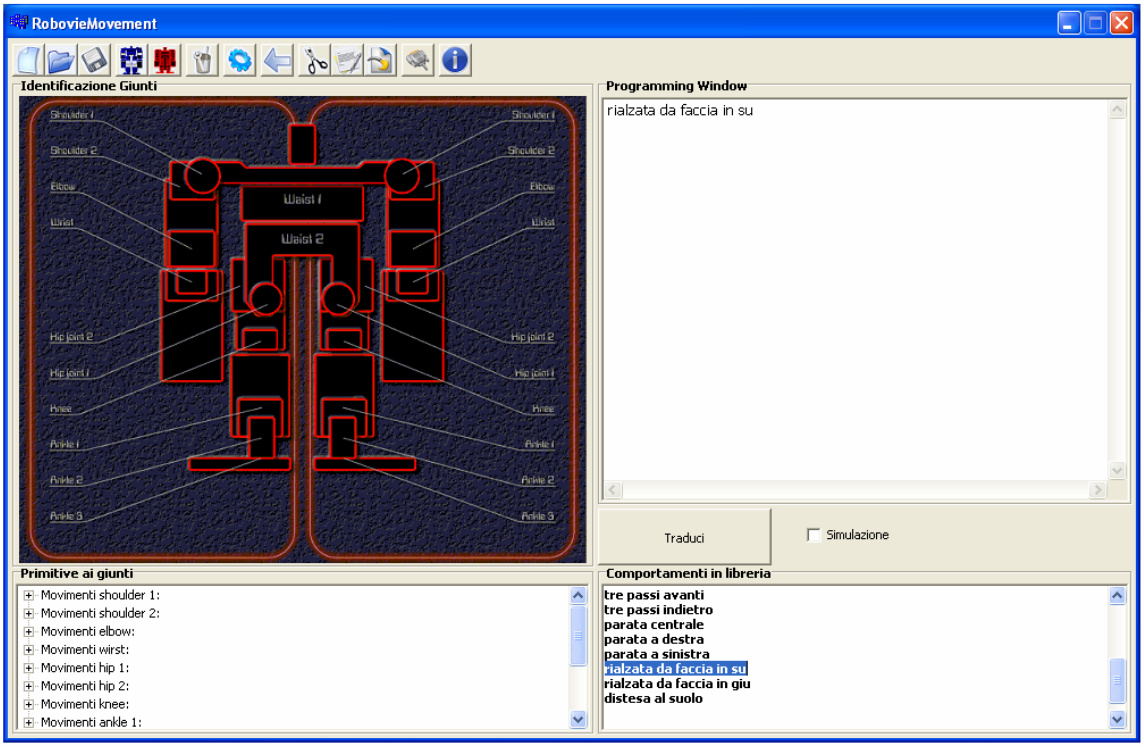

Figura 7.14: La programmazione del movimento di rialzata da faccia in su come comportamento di libreria.

Nella figura 7.14 viene messo in evidenza il fatto che sia possibile programmare il movimento come comportamento di libreria mediante "RobovieMovement". Come è possibile vedere nella figura 7.15 il robot si rialza da una posizione in cui si trova con la schiena a terra.

Particolare importanza assume, durante l'esecuzione del movimento, l'ausilio che i gomiti e le mani, forniscono durante la fase di transizione al double support.

Come è meglio evidenziato nella figura 7.15, in questo frangente sono proprio i giunti montati sulle braccia, quelli che maggiormente sono sottoposti a sforzo.

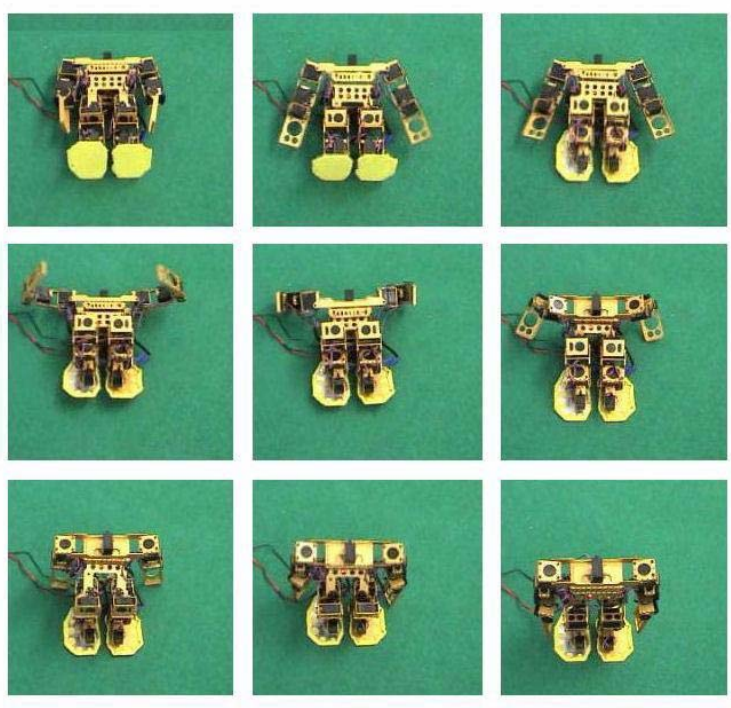

Figura 7.15: La rialzata da terra osservata dall'alto.

Nella figura 7.16, è evidenziato il modo con il quale il robot riesce a passare in una condizione in cui l'equilibrio è su entrambi i piedi.

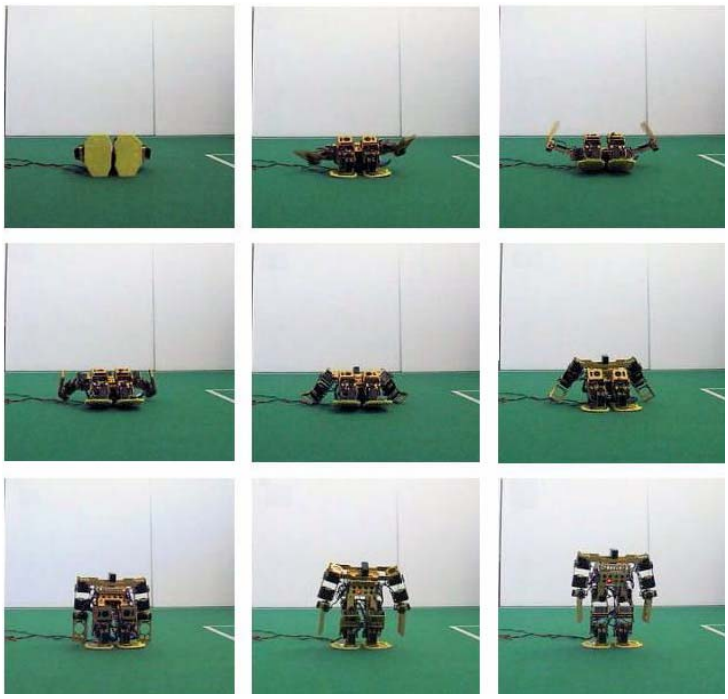

Figura 7.16: La rialzata da terra osservata di fronte.

Per completezza anche del suddetto comportamento viene risultato il risultato sperimentale ottenuto in simulazione nella figura 7.17.

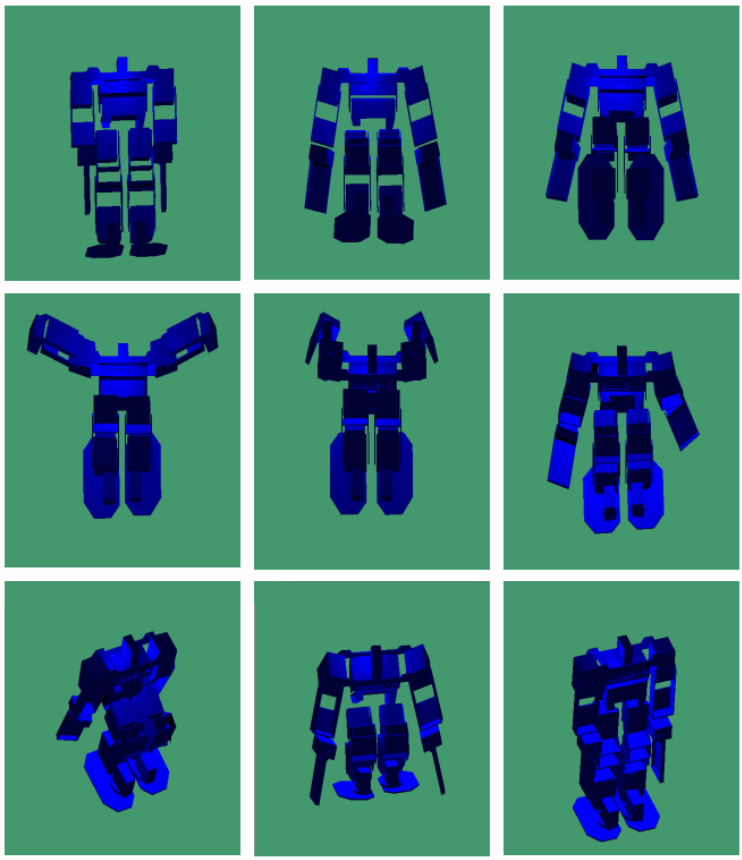

Figura 7.17: La rialzata da terra osservata in simulazione.

7.7 Il sollevamento da terra da faccia in giù

L'ultimo comportamento realizzato è quello che permette al robot, disteso per terra con la parte anteriore del corpo poggiata al suolo, di rialzarsi e assumere la posizione eretta. Anche questo comportamento figura tra i comportamenti sviluppati ed inseriti in libreria.

Questo comportamento viene eseguito dal robot in maniera fluida ed elegante, con la contemporaneità del movimento di numerosi giunti.

Il movimento consta di una prima fase in cui il robot arriva ad una posa in cui i giunti dei gomiti sono completamente chiusi ed i piedi toccano terra.

La movimentazione contemporanea dei giunti sulle caviglie e l'apertura delle braccia, che è possibile osservare in figura 7.18, consente al robot di distribuire equamente il suo peso su entrambi gli attuatori, in modo tale da non sollecitarne solamente uno.

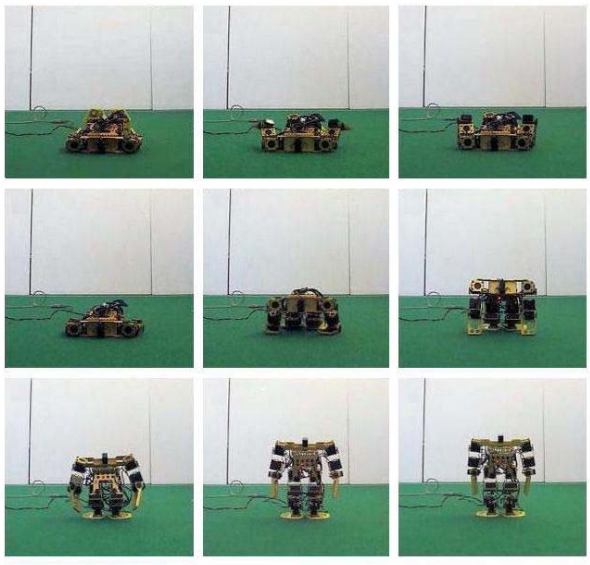

Figura 7.18: Una vista frontale del movimento di rialzata da faccia in giu.

Importante è anche il movimento di ausilio con le braccia durante la fase di rialzata che è meglio evidenziato nella figura 7.19, ed il bilanciamento finale effettuato dalle movimentazioni dei giunti degli shoulder.

Figura 7.20: La rialzata da terra a faccia in giù vista dall'alto.

7.8 Conclusioni ed evoluzioni future

Il lavoro svolto durante la realizzazione della presente tesi, ha permesso a fronte degli obiettivi prefissati, di raggiungere dei risultati soddisfacenti.

E' stato sviluppato un software per la creazione di movimenti complessi mediante linguaggio proprietario, che ha permesso di programmare una vastità di comportamenti complessi, utilizzabili in ambito RoboCup.

Tali comportamenti possono essere sia simulati, mediante opportuno software di visualizzazione 3D, che eseguiti direttamente dal robot.

I comportamenti sviluppati sono inoltre stati inseriti in una libreria all'interno della quale, in qualsiasi momento, è possibile avere accesso.

Interessante sarebbe lo sviluppo di un simulatore 3D che implementasse la fisica del sistema, prendendo in considerazione i momenti di inerzia, le forze che si generano durante l'esecuzione del movimento e che prevedesse il settaggio dei coefficienti di attrito, in modo da potere simulare perfettamente il comportamento prima di farlo eseguire.

Uno sviluppo del linguaggio di programmazione creato, potrebbe essere quello di renderlo più sintetico e con una migliore gestione degli errori.

Un obiettivo futuro potrebbe inoltre essere l'arricchimento della libreria dei comportamenti, con l'aggiunta di nuovi più complessi che riguardino anche diversi ambiti.

Bibliografia

[1] Folgheraiter M., (2006) "*Corso di robotica avanzata, Lezioni di Robotica Umanoide*", Politecnico di Torino, Torino;

[2] Gini G., (2005) "*Corso di Robotica 2, Lezioni di Robotica Bipede*", Università degli Studi di Bologna, Bologna;

[3] Kazuo Tanie, (2003) "*Humanoid Robot and its Application Possibility*", IEEE Conference on Multisensor Fusion and Integration for Intelligent Systems, 213-214;

[4] Kazuo Tanie and Kazuhito Yokoi (2003), "*Humanoid Robot and its Potential Applications*", ICIT 2003 – Maribor, Slovenia, 1-6;

[5] Lesk, M. E. and E. Schmidt (1975). "*Lex – A Lexical Analyzer Generator*". Computing Science Technical Report No. 39, Bell Laboratories, Murray Hill, New Jersey;

[6] Johnson, Stephen C. (1975). "*Yacc: Yet Another Compiler Compiler*". Computing Science Technical Report No. 32, Bell Laboratories, Murray hill, New Jersey;

[7] Niemann T., "*A Guide To Lex And Yacc*". E-papers Information OnLine, Portland, Oregon;

[8] Tad McGeer, (1990) "*Passive dynamic walking*", The International Journal of Robotics Research 9, no.2 62+;

[9] Tad McGeer, (1990) "*Passive walking with knees*", IEEE Robotics & Automation Conference (1990), 1640+;

[10] D. Juricic M. Vukobratovic, (1969) "*Contribution to the synthesis of biped gait*", IEEE Transactions on Bio-Medical Engineering, no. 1, 1-6;

[11] Shingo Iida et al., "*Humanoid Robot Control Based on Reinforcement Learning*", Dept. of Intelligence and Computer Science, Nagoya Institute of Technology Gokiso-cho, Showa-ku, Nagoya 466-8555, Japan;

[12] Wang G. et al. (2003), "*Cooperation of Dynamic Patterns and Sensorial Reflex for Humanoid walking*", Proceedings of the 2003 IEEE International Conference on Robotics & Automation Taipei, Taiwan, September 14-19, 2003 3421-3427;

[13] Napoleon, Shigeki Nakaura and Misuri Sampei (2003), "*Bilance Control Analysis of Humanoid Robot based on ZMP Feedback Control*", Proceeding of the 2002 IEEE/RSJ, Intl.Conference on Intelligent Robots and Systems EPFL, Lausanne, Switzerland. October 2002;

[14] Hirukawa H. et al. (2003), "*Experimental Evaluation of the Dynamic Simulation of Biped Walking of Humanoid Robots*", Proceedings of the 2003 IEEE International Conference on Robotics & Automation Taipei, Taiwan, September 14-19, 2003 1640-1645;

[15] Tsai-Yen Li et al. (2003), "*Motion Planning for Humanoid Walking in a Layered Environment*", Proceedings of the 2003 IEEE International Conference on Robotics & Automation Taipei, Taiwan, September 14-19, 2003 3421-3427;

[16] T. Guseo (2005), "*Architettura software per robot umanoide autonomo, Tesi di Laurea*", Università degli Studi di Padova, Padova;

[17] T.Raimondi, "*Principi di Controlli Automatici*", Medical Books, Palermo;

[18] http://www.fzi.de/divisions/ipt/WMC/preface/walking-machines-catalogue.html.

Appendice A

A.1 Il sorgente Lex

```
%{
/*****************************************************************************
mylexer.l
                                        LEX file.

Progetto: Linguaggio x la gestione dei movimenti di "Robovie-M"
Date: 01/08/2006
*****************************************************************************/

#include "myparser_tab.h"
#include <string.h>


char* crea_stringa(char* s, int n);

%}

A [aAà]
B [bB]
C [cC]
D [dD]
E [eEèé]
F [fF]
G [gG]
H [hH]
I [iI]
J [jJ]
K [kK]
L [lL]
M [mM]
N [nN]
O [oO]
P [pP]
Q [qQ]
R [rR]
S [sS]
T [tT]
U [uUù]
V [vV]
W [wW]
X [xX]
Y [yY]
Z [zZ]

%%

[ \t\n]*{R}{I}{P}{E}{T}{I}[ \t\n]*
                                  |
[ \t\n]*{R}{I}{P}{E}{T}{I}[ \t\n]*{A}{Z}{I}{O}{N}({E}|{I})[ \t\n]*
                                |
[ \t\n]*{R}{E}{P}{E}{A}{T}[ \t\n]*
                                              { return(RIPETI); }

[ \n\t]*{S}{T}{O}{P}[ \n\t]*         |
[ \n\t]*{E}{N}{D}[ \n\t]*            |
[ \n\t]*{F}{I}{N}{E}[ \n\t]*         { return(STOP);}

[ \n\t]*{B}{E}{G}{I}{N}[ \n\t]*      |
[ \n\t]*{I}{N}{I}{Z}{I}{O}[ \n\t]*   |
[ \n\t]*{S}{T}{A}{R}{T}[ \n\t]*            {return(INIZIO);}
```

```
{A}{L}{Z}{A}(({R}{E})?)                          { return(ALZA);}
{A}{B}{B}{A}{S}{S}{A}(({R}{E})?)                {return(ABBASSA);}
{M}{U}{O}{V}{I}                                  |
{R}{U}{O}{T}{A}(({R}{E})?)                      {return(RUOTA);}
{S}{P}{A}{L}{L}{A}                                 |
{S}{H}{O}{U}{L}{D}{E}{R}[ \t]*({I}|"1")[ \t]*      { return(SHOULDER_I); }
{B}{R}{A}{C}{C}{I}{O}                          |
{S}{H}{O}{U}{L}{D}{E}{R}[ \t]*{I}{I}[ \t]*  |
{S}{H}{O}{U}{L}{D}{E}{R}[ \t]*"2"[ \t]*            { return(SHOULDER_II); }
{P}{O}{L}{S}{O}                                  |
{E}{L}{B}{O}{W}                                  {return(ELBOW);}
{M}{A}{N}{O}                                     |
{W}{I}{R}{S}{T}                                  {return(WIRST);}

{V}{E}{R}{S}{O}[ \t]*{L}[ \t]*(("'"[ \t]*)?){A}{L}{T}{O}   |
{I}{N}[ \t]*{A}{L}{T}{O}                            {return(VERSO_ALTO);}

{V}{E}{R}{S}{O}[ \t]*{I}{L}[ \t]*{B}{A}{S}{S}{O}|
{I}{N}[ \t]*{B}{A}{S}{S}{O}                    {return(VERSO_BASSO);}

{V}{E}{R}{S}{O}[ \t]*{L}[ \t]*(("'"[ \t]*)?){I}{N}{T}{E}{R}{N}{O}        |
{A}{L}{L}[ \t]*(("'"[ \t]*)?){I}{N}{T}{E}{R}{N}{O}                       |
(({I}{N})|({V}{E}{R}{S}{O}))[ \t]*{D}{E}{N}{T}{R}{O} {return(VERSO_DENTRO);}

{V}{E}{R}{S}{O}[ \t]*{L}[ \t]*(("'"[ \t]*)?){E}{S}{T}{E}{R}{N}{O}        |
{A}{L}{L}[ \t]*(("'"[ \t]*)?){E}{S}{T}{E}{R}{N}{O}                       |
(({I}{N})|({V}{E}{R}{S}{O}))[ \t]*{F}{U}{O}{R}{I}    {return(VERSO_FUORI);}

{D}{E}{S}{T}{R}({A}|{O})     |
{D}{X}                       {yylval.str=crea_stringa("Destra",6);
                             return(DX_O_SX);}

(({A})|({V}{E}{R}{S}{O}))[ \t]*{D}{E}{S}{T}{R}{A}     {return(VERSO_DESTRA); }

(({A})|({V}{E}{R}{S}{O}))[ \t]*{S}{I}{N}{I}{S}{T}{R}{A}
                                                 { return(VERSO_SINISTRA);}

{S}{I}{N}{I}{S}{T}{R}({O}|{A})      |
{S}{X}                              {yylval.str=crea_stringa("Sinistra",8);
                                     return(DX_O_SX);     }

{I}{N}[ \t*]{S}{E}{N}{S}{O}[ \t]*{O}{R}{A}{R}{I}{O}                                  |
[ \t]*(({A})|({V}{E}{R}{S}{O}))[ \t]*(({S}{I}{N}{I}{S}{T}{R}{A})|({S}{X}))
[ \t]*                                          {return (IN_SENSO_ORARIO);}

{I}{N}[ \t*]{S}{E}{N}{S}{O}[ \t]*{A}{N}{T}{I}{O}{R}{A}{R}{I}{O}          |
[ \t]*(({A})|({V}{E}{R}{S}{O}))[ \t]*(({D}{E}{S}{T}{R}{A})|({D}{X}))[ \t]*     {return
(IN_SENSO_ANTIORARIO);}

{D}{I}                                  {return(DI);      }
{G}{R}{A}{D}({O}|{I})                {return(GRADI);}
{S}{P}{O}{S}{T}{A}(({R}{E})?)           {return (SPOSTARE);}
{T}{O}{R}{A}{C}{E}                      |
{W}{A}{I}{S}{T}[ \t]*{I}{I}[ \t]*       |
{W}{A}{I}{S}{T}[ \t]*"2"[ \t]*          {return (WAIST_II);}
{V}{I}{T}{A}                            |
{W}{A}{I}{S}{T}[ \t]*{I}[ \t]*          |
{W}{A}{I}{S}{T}[ \t]*"1"[ \t]*              {return (WAIST_I);}
{A}{N}{C}{A}                                |
{H}{I}{P}[ \t]*{I}[ \t]*                    |
{H}{I}{P}[ \t]*"1"[ \t]*                    {return (HIP_I);}
{C}{O}{S}{C}{I}{A}                          |
{H}{I}{P}[ \t]*{I}{I}[ \t]*                 |
{H}{I}{P}[ \t]*"2"[ \t]*                    {return (HIP_II);}
{K}{N}{E}{E}                                  |
{G}{I}{N}{O}{C}{C}{H}{I}{O}                 {return(KNEE);}
{G}{A}{M}{B}{A}                             |
{A}{N}({C}|{K}){L}{E}[ \t]*{I}[ \t]*        |
{A}{N}({C}|{K}){L}{E}[ \t]*"1"[ \t]*        {return (ANKLE_I);}
{C}{A}{V}{I}{G}{L}{I}{A}                     |
{A}{N}({C}|{K}){L}{E}[ \t]*{I}{I}[ \t]*      |
{A}{N}({C}|{K}){L}{E}[ \t]*"2"[ \t]*        {return (ANKLE_II);}
{P}{I}{E}{D}{E}                              |
{A}{N}({C}|{K}){L}{E}[ \t]*{I}{I}{I}[ \t]*   |
```

```
{A}{N}({C}|{K}){L}{E}[ \t]*"3"[ \t]*        {return (ANKLE_III);}
{I}{N}[\t* ]*{A}{V}{A}{N}{T}{I}             {return (AVANTI);}
{I}{N}{D}{I}{E}{T}{R}{O}                    {return (INDIETRO);}
{V}{O}{L}{T}{E}[ \t]*                    {return(VOLTE);}

","                          |
{V}{I}{R}{G}{O}{L}{A} |
"\n"                         |
{N}{U}{O}{V}{A}[ \t]*(({A}{Z}{I}{O}{N}{E})?){return(NUOVA_AZIONE);}
{E} {return(E);}
{C}{O}{N}[ \t]*{V}{E}{L}{O}{C}{I}{T}{A}[ \t]*(({D}{I})?)   |
{A}{L}{L}{A}[ \t]*{V}{E}{L}{O}{C}{I}{T}{A}[ \t]*{D}{I} {return(SET_VELOCITA);}

[0-9]+                      {yylval.str=crea_stringa(yytext,yyleng);
                                    return(INTEGER);   }
{U}{N}[ \t]*   |
{U}{N}{O}[ \t]*                          { yylval.num=1;return (NUMEROU); }
{D}{U}{E}[ \t]*                    {yylval.num=2;return (NUMEROU);}
{T}{R}{E}[ \t]*                    { yylval.num=3;return (NUMEROU); }
{Q}{U}{A}{T}{T}{R}{O}[ \t]*        |
{Q}{U}{A}{T}{T}{R}[ \t]*                 { yylval.num=4;return (NUMEROU);}
{C}{I}{N}{Q}{U}{E}[ \t]*                 |
{C}{I}{N}{Q}{U}[ \t]*       { yylval.num=5;return (NUMEROU);}
{S}{E}{I}[ \t]*                          { yylval.num=6;return (NUMEROU);}
{S}{E}{T}{T}{E}[ \t]*              |
{S}{E}{T}{T}[ \t]*                       { yylval.num=7;return (NUMEROU); }
{O}{T}{T}{O}[ \t]*                       |
{O}{T}{T}[ \t]*                          { yylval.num=8;return (NUMEROU); }


{N}{O}{V}{E}[ \t]*                       |
{N}{O}{V}[ \t]*                       {yylval.num=9;return (NUMEROU); }
{D}{I}{E}{C}{I}[ \t]*              {yylval.num=10;return (NUMEROD);}
{U}{N}{D}{I}{C}[ \t]*              |
{U}{N}{D}{I}{C}{I}[ \t]*                 { yylval.num=11;return (NUMEROD); }
{D}{O}{D}{I}{C}[ \t]*              |
{D}{O}{D}{I}{C}{I}[ \t]*                 {yylval.num=12;return (NUMEROD); }
{T}{R}{E}{D}{I}{C}[ \t]*                 |
{T}{R}{E}{D}{I}{C}{I}[ \t]*           { yylval.num=13;return (NUMEROD); }
{Q}{U}{A}{T}{T}{O}{R}{D}{I}{C}[ \t]*  |
{Q}{U}{A}{T}{T}{O}{R}{D}{I}{C}{I}[ \t]*    {yylval.num=14;return(NUMEROD);}
{Q}{U}{I}{N}{D}{I}{C}[ \t]*        |
{Q}{U}{I}{N}{D}{I}{C}{I}[ \t]*             {yylval.num=15;return (NUMEROD);}
{S}{E}{D}{I}{C}[ \t]*              |
{S}{E}{D}{I}{C}{I}[ \t]*                 {yylval.num=16;return (NUMEROD);}
{D}{I}{C}{I}{A}{S}{S}{E}{T}{T}{E}[ \t]*      {yylval.num=17;return (NUMEROD);}
{D}{I}{C}{I}{O}{T}{T}[ \t]*           |
{D}{I}{C}{I}{O}{T}{T}{O}[ \t]*           { yylval.num=18; return (NUMEROD); }
{D}{I}{C}{I}{A}{N}{N}{O}{V}{E}[ \t]*  {yylval.num=19;return (NUMEROD); }
{V}{E}{N}{T}[ \t]*                       |
{V}{E}{N}{T}{I}[ \t]*                {yylval.num=20;return (NUMEROD); }
{T}{R}{E}{N}{T}[ \t]*                |
{T}{R}{E}{N}{T}{A}[ \t]*                 {yylval.num=30;return (NUMEROD); }
{Q}{U}{A}{R}{A}{N}{T}[ \t]*          |
{Q}{U}{A}{R}{A}{N}{T}{A}[ \t]*           {yylval.num=40;return (NUMEROD); }
{C}{I}{N}{Q}{U}{A}{N}{T}[ \t]*           |
{C}{I}{N}{Q}{U}{A}{N}{T}{A}[ \t]*    {yylval.num=50;return (NUMEROD); }
{S}{E}{S}{S}{A}{N}{T}[ \t]*          |
{S}{E}{S}{S}{A}{N}{T}{A}[ \t]*           {yylval.num=60;return (NUMEROD); }
{S}{E}{T}{T}{A}{N}{T}[ \t]*          |
{S}{E}{T}{T}{A}{N}{T}{A}[ \t]*           {yylval.num=70;return (NUMEROD); }
{O}{T}{T}{A}{N}{T}[ \t]*                 |
{O}{T}{T}{A}{N}{T}{A}[ \t]*          {yylval.num=80;return (NUMEROD); }
{N}{O}{V}{A}{N}{T}[ \t]*                 |
{N}{O}{V}{A}{N}{T}{A}[ \t]*          {yylval.num=90;return (NUMEROD); }
{C}{E}{N}{T}[ \t]*                       |
{C}{E}{N}{T}{O}[ \t]*                {yylval.num=100;return (NUMEROC); }
{M}{I}{L}{L}{E}[ \t]*                |
{M}{I}{L}{A}[ \t]*                       {yylval.num=1000;return (NUMEROML);}
{M}{I}{L}{I}{O}{N}{I}[ \t]*          |
{M}{I}{L}{I}{O}{N}{E}[ \t]*          {yylval.num=1000000;return (NUMEROMN);}
{M}{I}{L}{I}{A}{R}{D}[ \t]*          |
{M}{I}{L}{I}{A}{R}{D}{I}[ \t]*           |
{M}{I}{L}{I}{A}{R}{D}{O}[ \t]*        {yylval.num=1000000000;return  NUMEROMD);}
```

```
%%

char* crea_stringa(char* s, int n)
        {
           char* t =malloc(n+1);
           strncpy(t, s, n);
           t[n]='\0';
           return t;

}
```

A.2 Il sorgente Yacc

```
%{
/*************************************************************************
myparser.y
                                    YACC file.

Progetto: Linguaggio x la gestione dei movimenti di "Robovie-M"
Date: 01/08/2006
*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <malloc.h>
#include <math.h>

#define TOP_DANCER 100

int VELOCITA=1;
int SHOULDER_I_SX=52;
int SHOULDER_I_DX=22;
int SHOULDER_II_SX=55;
int SHOULDER_II_DX=25;
int ELBOW_SX=58;
int ELBOW_DX=28;
int WIRST_SX=61;
int WIRST_DX=31;

int WAIST_I_C=67;
int WAIST_II_C=64;
int HIP_I_DX=4;
int HIP_I_SX=34;
int HIP_II_DX=7;
int HIP_II_SX=37;
int KNEE_DX=10;
int KNEE_SX=40;
int ANKLE_I_DX=13;
int ANKLE_I_SX=43;
int ANKLE_II_DX=16;
int ANKLE_II_SX=46;
int ANKLE_III_DX=19;
int ANKLE_III_SX=49;
int LOOP=70;

char FILE_OUTPUT[]="tradotta.txt";
// Contiene il nome del File di output
char FILE_INPUT[]="start.txt";
// File di Input
char FILE_FINETRADUZIONE[]="semaforo.txt";
// File che segnala la fine della traduzione
char FILE_TIPOTRADUZIONE[]="translation";
// File contenente le informazioni necessarie alla determinazione del tipo di
```

```
traduzione
 //da effettuare

char posizione_topdancer[]="@21A7fB8eC1bD80E7aF80G1eHa4I10Je0K80L71Me4N7fO85P7fQe1R5bSe
fT1fUb2V80,+1!00";

char posizione_iniziale_topdancer[]="@21A7fB8eC1bD80E7aF80G1eHa4I10Je0K80L71Me4N7fO85P
7fQe1R5bSefT1fUb2V80,+1!00";

char posizione_finale_topdancer[]="@21A7fB8eC1bD80E7aF80G1eHa4I10Je0K80L71Me4N7fO85P7fQ
e1R5bSefT1fUb2V80,+0!00";

char posizione_iniziale[]="@21A7fB8eC1bD80E7aF80G1eHa4I10Je0K80L71Me4N7fO85P7fQe1R5bSe
fT1fUb2V80,+1!00";

char posizione_finale[]=  "@21A7fB8eC1bD80E7aF80G1eHa4I10Je0K80L71Me4N7fO85P7fQe1R5
bSefT1fUb2V80,+0!00";

char
posizione_attuale[]="@21A7fB8eC1bD80E7aF80G1eHa4I10Je0K80L71Me4N7fO85P7fQe1R5bSefT1
fUb2V80,+1!00";
int  posizione_motori[75]={0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

char Nexa[2]={'0','0'};

int numero_ripetizioni=0;
int PrimaAzioneDaRipetere=0;
int NumeroAzioni=0;
int flag_ripeti_n_volte=0;
int flag_ripeti=0;
//flag che quando posto a 1 indica che sono all'interno di un loop
int flag_dopo_loop=0;
int NumeroAzioniDaRipetere=0;
//Per la determinazione del numero si passi di cui si deve
//tornare indietro al fine di creare dei loop
int TipoTraduzione=0;
// Per la determinazione del tipo di traduzione ke devo effettuare
//(Per il Software Robostage e RoboveiMaker)

//Struttura utilizzata per la creazione di una lista. Ogni elemento di tale lista
//contiene una riga del FILE_OUTPUT
struct riga {
       char informazione[105];
// Di conseguenza la lista contiene le righe del file: FILE_OUTPUT
       struct riga *next;
};
struct riga *testa=NULL;
//Puntatore alla testa della lista


//************************************************
//Prototipo delle funzioni utilizzate
//************************************************
void Dec2Esa(int,char*);
//Conversione da Decimale a Esadecimale relativamente ai motori della parte destra
void Dec2Esa_SX(int,char*);
//Conversione da Decimale a Esadecimale relativamente ai motori della parte sinistra
int Esa2Dec(char*);
//Conversione da Esadecimale a Decimale relativamente ai motori della parte destra
char ValoreEsadecimale(int);
char ValoreEsadecimaleParteSinistra(int);
int ValoreDecimale(char);
void MemorizzaMovimento(void);
//Memorizza il movimento da eseguire all'interno di una lista
void CreaRipetizioni(int);
//funzione utilizzata per la creazione dei cicli di azioni
void inizializza(void);
//Inizializza la posizione dei motori
void AggiornaMotori(char*,int);
//aggiorna la posizione dei motori in seguito ad un azione

void CreaNRipetizioni(int,int);
```

```
struct riga* CreaListaDaRipetere(int);
struct riga * ConcatenaListe(struct riga *,struct riga *);
void VisualizzaLista(struct riga*);

int Arrotonda(float f);

void FineTraduzione(void);
void yyerror(char*);
//***********************************************

%}

%token ALZA, ABBASSA, RUOTA, IN_SENSO_ORARIO, IN_SENSO_ANTIORARIO, SPOSTARE, AVANTI,
INDIETRO
%token VERSO_ALTO, VERSO_BASSO, VERSO_DENTRO, VERSO_FUORI, VERSO_DESTRA, VERSO_SINISTRA
%token SHOULDER_I, SHOULDER_II, ELBOW, WIRST, WAIST_I, WAIST_II, HIP_I, HIP_II, KNEE,
ANKLE_I,ANKLE_II,ANKLE_III
%token SET_VELOCITA, NUOVA_AZIONE, E, DI, GRADI
%token RIPETI, VOLTE, STOP, INIZIO
%token <num> NUMEROU,NUMEROD,NUMEROC,NUMEROML,NUMEROMN,NUMEROMD
%token <str> DX_O_SX
%token <str> INTEGER

%type <str> intero
%type <num> numero_letterale,centinaia,migliaia,miliardi,milioni

%union {
        int num;
        char* str;
}

%%

statement: azione{ FILE *fp;
//memorizzo tutti gli elementi della lista nel file FILE_OUTPUT
                      fp=fopen(FILE_OUTPUT,"a");
                             while(testa!=NULL) {
                                       fprintf(fp,"%s\n",testa->informazione);
                                              testa=testa->next;
                                                }
                                     if(TipoTraduzione==TOP_DANCER)
                                        fprintf(fp,"%s\n",posizione_finale_topdancer);
                                     //deve essere  diversificata asseconda dell'uscita
                                     else
                                        fprintf(fp,"%s\n",posizione_finale);
                                              fclose(fp);
                                }
                               ;

azione: azione NUOVA_AZIONE movimenti {NumeroAzioni++;
                                            if(flag_dopo_loop==1) {
               //Nel caso si prosegua con l'esecuzione di azioni dopo un loop
                                                                    struct riga* pointer;
                                                          int index,position;
                                                          pointer=testa;

        for(index=0;index<(PrimaAzioneDaRipetere-1);index++) pointer=pointer->next;
                                                    Dec2Esa_SX(NumeroAzioni,Nexa);

        position=strlen(pointer>informazione);
        pointer->informazione[position-2]=Nexa[0];
        pointer->informazione[position-1]=Nexa[1]
               posizione_motori[74]=NumeroAzioni+1;
        flag_dopo_loop=0;
                        }
        if(flag_ripeti==1) NumeroAzioniDaRipetere++;
        MemorizzaMovimento(); }


        |azione RIPETI movimenti{flag_ripeti=1; NumeroAzioniDaRipetere++;
                                                    NumeroAzioni++;
                                                    PrimaAzioneDaRipetere=NumeroAzioni;
                                                    MemorizzaMovimento(); }
```

```
        |azione RIPETI INIZIO movimenti{flag_ripeti=1; NumeroAzioniDaRipetere++;
                                        NumeroAzioni++;
                                        PrimaAzioneDaRipetere=NumeroAzioni;
                                        MemorizzaMovimento();}

        |azione RIPETI movimenti E
                                  {flag_ripeti=1;PrimaAzioneDaRipetere=NumeroAzioni+1;}

        |azione RIPETI INIZIO movimenti E
                             {flag_ripeti=1;PrimaAzioneDaRipetere=NumeroAzioni+1; }

        |RIPETI movimenti{flag_ripeti=1; NumeroAzioniDaRipetere++;
                                           NumeroAzioni++;
                                           PrimaAzioneDaRipetere=NumeroAzioni;
                                           MemorizzaMovimento(); }

        |RIPETI INIZIO movimenti{flag_ripeti=1; NumeroAzioniDaRipetere++;
                                                 NumeroAzioni++;
                                                 PrimaAzioneDaRipetere=NumeroAzioni;
                                                 MemorizzaMovimento(); }

        |RIPETI movimenti E{ flag_ripeti=1;PrimaAzioneDaRipetere=NumeroAzioni+1;  }

        |RIPETI INIZIO movimenti E{ flag_ripeti=1;PrimaAzioneDaRipetere=NumeroAzioni+1;}

        |azione RIPETI intero VOLTE movimenti {     flag_ripeti_n_volte=1;
                                                    numero_ripetizioni=atoi($3);
                                                    NumeroAzioni++;
                                                    PrimaAzioneDaRipetere=NumeroAzioni;
                                                    MemorizzaMovimento(); }

        |azione RIPETI intero VOLTE movimenti E {   flag_ripeti_n_volte=1;
                                                    numero_ripetizioni=atoi($3);

PrimaAzioneDaRipetere=NumeroAzioni+1;}

        |azione RIPETI intero VOLTE INIZIO movimenti {flag_ripeti_n_volte=1;
                                                          numero_ripetizioni=atoi($3);
                                                    NumeroAzioni++;
                                                    PrimaAzioneDaRipetere=NumeroAzioni;
                                                    MemorizzaMovimento(); }

        |azione RIPETI intero VOLTE INIZIO movimenti E {flag_ripeti_n_volte=1;
                                                    numero_ripetizioni=atoi($3);

PrimaAzioneDaRipetere=NumeroAzioni+1;}

        |RIPETI intero VOLTE movimenti {     flag_ripeti_n_volte=1;
                                             numero_ripetizioni=atoi($2);
                                             NumeroAzioni++;
                                             PrimaAzioneDaRipetere=NumeroAzioni;
                                                          MemorizzaMovimento(); }

        |RIPETI intero VOLTE movimenti E {   flag_ripeti_n_volte=1;
                                             numero_ripetizioni=atoi($2);
                                             PrimaAzioneDaRipetere=NumeroAzioni+1; }

        |RIPETI intero VOLTE INIZIO movimenti {flag_ripeti_n_volte=1;
                                               numero_ripetizioni=atoi($2);
                                               NumeroAzioni++;
                                             PrimaAzioneDaRipetere=NumeroAzioni;
                                             MemorizzaMovimento(); }

        |RIPETI intero VOLTE INIZIO movimenti E {flag_ripeti_n_volte=1;
                                             numero_ripetizioni=atoi($2);
                                             PrimaAzioneDaRipetere=NumeroAzioni+1; }

        |azione STOP
                  {if (flag_ripeti_n_volte==1)

{                                CreaNRipetizioni(PrimaAzioneDaRipetere,numero_ripetizi
oni);
                            flag_ripeti_n_volte=0; }
                else { CreaRipetizioni(NumeroAzioniDaRipetere); flag_dopo_loop=1; }
```

```
                                    }

        |azione NUOVA_AZIONE movimenti E {}

        |azione movimenti     {       if(flag_ripeti==1) NumeroAzioniDaRipetere++;
                                              NumeroAzioni++;
                                              MemorizzaMovimento(); }

        |azione movimenti E   {}
        |movimenti E   {}
        |movimenti     {       if(flag_ripeti==1) NumeroAzioniDaRipetere++;
                                      NumeroAzioni++;
                                      MemorizzaMovimento(); }
        ;

movimenti:
      ALZA SHOULDER_II DX_O_SX DI intero GRADI      {
                                      if (!strcmp($3,"Destra")) {
                                              posizione_motori[SHOULDER_II_DX]+=atoi($5);
                                              if (posizione_motori[SHOULDER_II_DX]>255)
                                            posizione_motori[SHOULDER_II_DX]=255;
              //Controllo ke il valore non sia fuori dal range
                                              AggiornaMotori($3,SHOULDER_II_DX);}
                      //Aggiorno la posizione dei motori
                                      else {
                                              posizione_motori[SHOULDER_II_SX]+=atoi($5);
                                              if (posizione_motori[SHOULDER_II_SX]>255)
                                              posizione_motori[SHOULDER_II_SX]=255;
                                              AggiornaMotori($3,SHOULDER_II_SX); }
                                             }
        |RUOTA SHOULDER_II DX_O_SX DI intero GRADI VERSO_ALTO      {
                                      if (!strcmp($3,"Destra")) {
                                              posizione_motori[SHOULDER_II_DX]+=atoi($5);

                                              if (posizione_motori[SHOULDER_II_DX]>255)
                                               posizione_motori[SHOULDER_II_DX]=255;
                       //Controllo ke il valore non sia fuori dal range
                                              AggiornaMotori($3,SHOULDER_II_DX);}
                             //Aggiorno la posizione dei motori
                                      else {
                                              posizione_motori[SHOULDER_II_SX]+=atoi($5);
                                              if (posizione_motori[SHOULDER_II_SX]>255)
                                              posizione_motori[SHOULDER_II_SX]=255;
                                              AggiornaMotori($3,SHOULDER_II_SX); }
                                      }
        |ABBASSA SHOULDER_II DX_O_SX DI intero GRADI{
                                      if (!strcmp($3,"Destra")) {
                                      posizione_motori[SHOULDER_II_DX]-=atoi($5);
                                             if(posizione_motori[SHOULDER_II_DX]<0)
                                              posizione_motori[SHOULDER_II_DX]=0;
                                              AggiornaMotori($3,SHOULDER_II_DX);}
                                      else {
                                              posizione_motori[SHOULDER_II_SX]-=atoi($5);
                                              if (posizione_motori[SHOULDER_II_SX]<0)
                                              posizione_motori[SHOULDER_II_SX]=0;
                                              AggiornaMotori($3,SHOULDER_II_SX); }
                                      }
        |RUOTA SHOULDER_II DX_O_SX DI intero GRADI VERSO_BASSO     {
                                      if (!strcmp($3,"Destra")) {
                                              posizione_motori[SHOULDER_II_DX]-=atoi($5);
                                              if (posizione_motori[SHOULDER_II_DX]<0)
                                              posizione_motori[SHOULDER_II_DX]=0;
                                              AggiornaMotori($3,SHOULDER_II_DX);}
                                      else {
                                              posizione_motori[SHOULDER_II_SX]-=atoi($5);
                                              if (posizione_motori[SHOULDER_II_SX]<0)
                                              posizione_motori[SHOULDER_II_SX]=0;
                                             AggiornaMotori($3,SHOULDER_II_SX); }
                                      }
        |RUOTA SHOULDER_I DX_O_SX DI intero GRADI IN_SENSO_ORARIO  {
                                      if (!strcmp($3,"Destra")) {
                                              posizione_motori[SHOULDER_I_DX]-=atoi($5);
                                              if (posizione_motori[SHOULDER_I_DX]<0)
                                              posizione_motori[SHOULDER_I_DX]=0;
```

```
                                        AggiornaMotori($3,SHOULDER_I_DX); }
                                else {
                                        posizione_motori[SHOULDER_I_SX]+=atoi($5);
                                        if (posizione_motori[SHOULDER_I_SX]>255)
                                        posizione_motori[SHOULDER_I_SX]=255;
                                        AggiornaMotori($3,SHOULDER_I_SX);}
                                }
        |RUOTA SHOULDER_I DX_O_SX DI intero GRADI VERSO_DENTRO           {
                                if (!strcmp($3,"Destra")) {
                                        posizione_motori[SHOULDER_I_DX]+=atoi($5);

                                        if (posizione_motori[SHOULDER_I_DX]>255)
                                         posizione_motori[SHOULDER_I_DX]=255;
//Controllo ke il valore non sia fuori dal range
                                        AggiornaMotori($3,SHOULDER_I_DX);}
                        //Aggiorno la posizione dei motori
                                else {
                                        posizione_motori[SHOULDER_I_SX]+=atoi($5);
                                        if (posizione_motori[SHOULDER_I_SX]>255)
                                        posizione_motori[SHOULDER_I_SX]=255;
                                        AggiornaMotori($3,SHOULDER_I_SX); }
                                }
        |RUOTA SHOULDER_I DX_O_SX DI intero GRADI VERSO_FUORI            {
                                if (!strcmp($3,"Destra")) {
                                        posizione_motori[SHOULDER_I_DX]-=atoi($5);

                                        if (posizione_motori[SHOULDER_I_DX]>255);
                                        posizione_motori[SHOULDER_I_DX]=255
//Controllo ke il valore non sia fuori dal range
                                        AggiornaMotori($3,SHOULDER_I_DX);
//Aggiorno la posizione dei motori
                                else {
                                        posizione_motori[SHOULDER_I_SX]-=atoi($5);
                                        if (posizione_motori[SHOULDER_I_SX]>255)
                                        posizione_motori[SHOULDER_I_SX]=255;
                                        AggiornaMotori($3,SHOULDER_I_SX); }
                                }
        |RUOTA SHOULDER_I DX_O_SX DI intero GRADI IN_SENSO_ANTIORARIO    {
                                if (!strcmp($3,"Destra")) {
                                        posizione_motori[SHOULDER_I_DX]+=atoi($5);
                                        if (posizione_motori[SHOULDER_I_DX]>255)
                                         posizione_motori[SHOULDER_I_DX]=255;
                                        AggiornaMotori($3,SHOULDER_I_DX);}
                                else {
                                        posizione_motori[SHOULDER_I_SX]-=atoi($5);
                                        if (posizione_motori[SHOULDER_I_SX]<0)
                                        posizione_motori[SHOULDER_I_SX]=0;
                                        AggiornaMotori($3,SHOULDER_I_SX);}
                                }
        |RUOTA ELBOW DX_O_SX DI intero GRADI IN_SENSO_ANTIORARIO   {
                                if (!strcmp($3,"Destra")) {
                                        posizione_motori[ELBOW_DX]-=atoi($5);
                                        if (posizione_motori[ELBOW_DX]<0)
                                        posizione_motori[ELBOW_DX]=0;
                                        AggiornaMotori($3,ELBOW_DX);}
                                else {
                                        posizione_motori[ELBOW_SX]+=atoi($5);
                                        if (posizione_motori[ELBOW_SX]>255)
                                        posizione_motori[ELBOW_SX]=255;
                                        AggiornaMotori($3,ELBOW_SX);}

                                }
        |RUOTA ELBOW DX_O_SX DI intero GRADI IN_SENSO_ORARIO{
                                if (!strcmp($3,"Destra")) {
                                        posizione_motori[ELBOW_DX]+=atoi($5);
                                        if (posizione_motori[ELBOW_DX]>255)
                                         posizione_motori[ELBOW_DX]=255;
                                        AggiornaMotori($3,ELBOW_DX);}
                                else {
                                        posizione_motori[ELBOW_SX]-=atoi($5);
                                        if (posizione_motori[ELBOW_SX]<0)
                                          posizione_motori[ELBOW_SX]=0;
                                        AggiornaMotori($3,ELBOW_SX);}
```

```
                                }
        |RUOTA WIRST DX_O_SX DI intero GRADI IN_SENSO_ORARIO       {
                                if (!strcmp($3,"Destra")) {
                                        posizione_motori[WIRST_DX]-=atoi($5);
                                        if (posizione_motori[WIRST_DX]<0)
                                         posizione_motori[WIRST_DX]=0;
                                        AggiornaMotori($3,WIRST_DX);}
                                else {
                                        posizione_motori[WIRST_SX]+=atoi($5);
                                        if (posizione_motori[WIRST_SX]>255)
                                        posizione_motori[WIRST_SX]=255;
                                        AggiornaMotori($3,WIRST_SX);}
                                }
        |RUOTA WIRST DX_O_SX DI intero GRADI IN_SENSO_ANTIORARIO   {
                                if (!strcmp($3,"Destra")) {
                                        posizione_motori[WIRST_DX]+=atoi($5);
                                        if (posizione_motori[WIRST_DX]>255)
                                        posizione_motori[WIRST_DX]=255;
                                        AggiornaMotori($3,WIRST_DX);}
                                else {
                                        posizione_motori[WIRST_SX]-=atoi($5);
                                        if (posizione_motori[WIRST_SX]<0)
                                         posizione_motori[WIRST_SX]=0;
                                        AggiornaMotori($3,WIRST_SX);}
                                        }
        |RUOTA WIRST DX_O_SX DI intero GRADI VERSO_DENTRO   {
                                if (!strcmp($3,"Destra")) {
                                        posizione_motori[WIRST_DX]+=atoi($5);
                                        if (posizione_motori[WIRST_DX]>255);
                                        posizione_motori[WIRST_DX]=255
                                        AggiornaMotori($3,WIRST_DX);}
                                else {
                                        posizione_motori[WIRST_SX]+=atoi($5);
                                        if (posizione_motori[WIRST_SX]>255)
                                        posizione_motori[WIRST_SX]=255;
                                       AggiornaMotori($3,WIRST_SX);}
                                }
        |RUOTA WIRST DX_O_SX DI intero GRADI VERSO_FUORI    {
                                if (!strcmp($3,"Destra")) {
                                        posizione_motori[WIRST_DX]-=atoi($5);
                                        if (posizione_motori[WIRST_DX]<0)
                                        posizione_motori[WIRST_DX]=0;
                                        AggiornaMotori($3,WIRST_DX);}
                                else {
                                        posizione_motori[WIRST_SX]-=atoi($5);
                                        if (posizione_motori[WIRST_SX]<0)
                                         posizione_motori[WIRST_SX]=0;
                                        AggiornaMotori($3,WIRST_SX);}
                                }

        |RUOTA HIP_I DX_O_SX DI intero GRADI IN_SENSO_ORARIO       {
                                if (!strcmp($3,"Destra")) {
                                        posizione_motori[HIP_I_DX]-=atoi($5);
                                        if (posizione_motori[HIP_I_DX]<0)
                                        posizione_motori[HIP_I_DX]=0;
                                        AggiornaMotori($3,HIP_I_DX); }
                                else {
                                        posizione_motori[HIP_I_SX]+=atoi($5);
                                        if (posizione_motori[HIP_I_SX]>255)
                                          posizione_motori[HIP_I_SX]=255;
                                        AggiornaMotori($3,HIP_I_SX);}
                                }
        | RUOTA HIP_I DX_O_SX DI intero GRADI IN_SENSO_ANTIORARIO  {
                                if (!strcmp($3,"Destra")) {
                                        posizione_motori[HIP_I_DX]+=atoi($5);
                                        if (posizione_motori[HIP_I_DX]>255)
                                         posizione_motori[HIP_I_DX]=255;
                                        AggiornaMotori($3,HIP_I_DX);}
                                else {
                                        posizione_motori[HIP_I_SX]-=atoi($5);
                                        if (posizione_motori[HIP_I_SX]<0)
                                         posizione_motori[HIP_I_SX]=0;
                                        AggiornaMotori($3,HIP_I_SX);  }
                                }
```

```
        |RUOTA HIP_I DX_O_SX DI intero GRADI VERSO_FUORI          {
                                        if (!strcmp($3,"Destra")) {
                                                posizione_motori[HIP_I_DX]-=atoi($5);
                                                if (posizione_motori[HIP_I_DX]<0)
                                                posizione_motori[HIP_I_DX]=0;
                                                AggiornaMotori($3,HIP_I_DX); }
                                        else {
                                                posizione_motori[HIP_I_SX]-=atoi($5);
                                                if (posizione_motori[HIP_I_SX]<0)
                                                posizione_motori[HIP_I_SX]=0;
                                                AggiornaMotori($3,HIP_I_SX);}
                                        }
        |RUOTA HIP_I DX_O_SX DI intero GRADI VERSO_DENTRO   {
                                        if (!strcmp($3,"Destra")) {
                                                posizione_motori[HIP_I_DX]+=atoi($5);
                                                if (posizione_motori[HIP_I_DX]>255)
                                                 posizione_motori[HIP_I_DX]=255;
                                                 AggiornaMotori($3,HIP_I_DX);}
                                        else {
                                                posizione_motori[HIP_I_SX]+=atoi($5);
                                                if (posizione_motori[HIP_I_SX]>255)
                                                posizione_motori[HIP_I_SX]=255;
                                                AggiornaMotori($3,HIP_I_SX);  }
                                        }
.
.
.
.
.
.
.
.
.


//---------------------------------------------------------------------------

/*      Settaggio della VELOCITA con cui eseguire i movimenti  */
velocita: SET_VELOCITA intero {
                    if(TipoTraduzione==TOP_DANCER){
                                        int vel1;
                                        int vel2;
                                        int vel;
                                        float kkk;
                                        vel1=atoi($2);
                                        printf("\nVel1= %d",vel1);
                                        vel2= vel1 * 255 ;
                                        printf("\nVel2 = %d",vel2);
                                        kkk= (float)(vel2)/211;
                                        printf("\nkkk vale %f",kkk);
                                        getchar();
                                        vel= Arrotonda(kkk);
                                        printf("\nvel vale %d",vel);
                                        getchar();
                                        if (vel<0 || vel>100)
                                                vel=27;
                                        Dec2Esa_SX(vel,Nexa);
                                        posizione_topdancer[VELOCITA]=Nexa[0];
                                        posizione_topdancer[VELOCITA+1]=Nexa[1];

        }
                                else{
                                        int vel;
                                        vel=atoi($2);
                                        if (vel<0 || vel>100)
        //Controllo ke il valore della velocita sia nel range : 0 < velocita < 100
                                                vel=33;
                                        Dec2Esa_SX(vel,Nexa);
                                        posizione_attuale[VELOCITA]=Nexa[0];
                                        posizione_attuale[VELOCITA+1]=Nexa[1];
                                }
                                }
        ;
```

```
intero: numero_letterale     {      itoa($1,$$,10);       }
       |      INTEGER        {      $$=$1;         }
       ;

numero_letterale:        miliardi            {  $$=$1;  }
                       |   centinaia             {  $$=$1;  }
                       |   migliaia              {  $$=$1;  }
                       |   milioni               {  $$=$1;}
                       ;

centinaia: NUMEROU                                  {$$=$1; }
          |NUMEROD                                  {$$=$1; }
           |NUMEROD NUMEROU                 {$$=$1+$2;}
           |NUMEROC                                  {$$=$1; }
           |NUMEROC NUMEROD                 {$$=$1+$2;}
           |NUMEROC NUMEROU                 {$$=$1+$2;}
           |NUMEROC NUMEROD NUMEROU           {$$=$1+$2+$3;}
           |NUMEROU NUMEROC                {$$=((100*$1)-100)+$2;}
           |NUMEROU NUMEROC NUMEROD          {$$=((100*$1)-100)+$2+$3;}
           |NUMEROU NUMEROC NUMEROU          {$$=((100*$1)-100)+$2+$3;}
           |NUMEROU NUMEROC NUMEROD NUMEROU{$$=((100*$1)100)+$2+$3+$4;}
                     ;

migliaia:      NUMEROML                             {$$=$1;}
            |centinaia NUMEROML              {$$=$1*$2;}
            |NUMEROML centinaia                     {$$=$1+$2;}
            |centinaia NUMEROML centinaia      {$$=$1*$2+$3;}
             ;

milioni:       NUMEROMN                                    {$$=$1;}
               |      migliaia NUMEROMN             {$$=$1*$2;}
               |      NUMEROMN migliaia                   {$$=$1+$2;}
               |      migliaia NUMEROMN migliaia    {$$=$1*$2+$3;}
               |      centinaia NUMEROMN            {$$=$1*$2;}
               |      NUMEROMN centinaia            {$$=$1+$2;}
               |      centinaia NUMEROMN centinaia     {$$=$1*$2+$3;}
               |      centinaia NUMEROMN migliaia   {$$=$1*$2+$3;}
               |      migliaia NUMEROMN centinaia   {$$=$1*$2+$3;}
             ;

miliardi:      NUMEROMD                                  {$$=$1;}
              |     milioni NUMEROMD                {$$=$1*$2;}
              |NUMEROMD milioni                     {$$=$1+$2;}
              |milioni NUMEROMD milioni     {$$=$1*$2+$3;}
              |milioni NUMEROMD centinaia   {$$=$1*$2+$3;}
              |milioni NUMEROMD migliaia    {$$=$1*$2+$3;}
              |migliaia NUMEROMD            {$$=$1*$2;}
       |     NUMEROMD migliaia              {$$=$1+$2;}
       |      migliaia NUMEROMD migliaia    {$$=$1*$2+$3;}
       |      centinaia NUMEROMD            {$$=$1*$2;}
       |     NUMEROMD centinaia             {$$=$1+$2;}
       |      centinaia NUMEROMD centinaia  {$$=$1*$2+$3;}
       |      centinaia NUMEROMD migliaia   {$$=$1*$2+$3;}
       |      centinaia NUMEROMD milioni    {$$=$1*$2+$3;}
       |      migliaia NUMEROMD milioni     {$$=$1*$2+$3;}
       |      migliaia NUMEROMD centinaia   {$$=$1*$2+$3;}
       ;

%%

extern FILE *yyin;

//---------------------------------------------------------------------

int main(void)
{
FILE *fp;
       yyin=fopen(FILE_INPUT,"r");
       if (yyin==NULL) {
              printf("ERRORE NON APRE IL FILE %s \n",FILE_INPUT);
             exit(0); }

       fp=fopen(FILE_TIPOTRADUZIONE,"r");
       //Leggo dal FILE_TIPOTRADUZIONE il valore ke mi permette di
```

```
        fscanf(fp,"%d",&TipoTraduzione);
        // capire se effettuare una traduzione per TOP DANCER o SLIDER
        fclose(fp);

        inizializza();

        while (!feof(yyin)){
                yyparse();     }
FineTraduzione();
return 0;
}
//---------------------------------------------------------------------

/*  Funzione x l'aggiornamento della posizione dei motori.Riceve in ingresso: TipoArto:
il tipo di arto ke devo muovere (Elbow, wirst, etc...)PosizioneArto: la posizione
dell'arto (ovvero se è un arto destro o sinistro)*/

void AggiornaMotori(char* PosizioneArto,int TipoArto)
{

if(!strcmp(PosizioneArto,"Destra")) {
//Effettuo la conversione in esadecimale relativamente agli arti della //parte destra
                if(TipoTraduzione==TOP_DANCER)
Dec2Esa_SX(posizione_motori[TipoArto],Nexa);
                else Dec2Esa(posizione_motori[TipoArto],Nexa);
                }
else {
//Effettuo la conversione in esadecimale relativamente agli arti della //parte sinistra
                if(TipoTraduzione==TOP_DANCER) Dec2Esa(posizione_motori[TipoArto],Nexa);
                else Dec2Esa_SX(posizione_motori[TipoArto],Nexa);
        }
//Chiudi else



// A questo punto non mi resta ke aggiornare la posizione dei motori
if(TipoTraduzione==TOP_DANCER){                     //Se il file d'uscita deve essere
utilizzato dal Software TOP DANCER
        posizione_topdancer[TipoArto]=Nexa[0];
//allora aggiorno il vettore contente la forma leggibile x TOP DANCER
        posizione_topdancer[TipoArto+1]=Nexa[1];}
else {
 // Se deve essere utilizzato con il Simulatore Slider
        posizione_attuale[TipoArto]=Nexa[0];
//allora aggiorno il vettore contenente la riga leggibile x SLIDER
        posizione_attuale[TipoArto+1]=Nexa[1];}
}
//---------------------------------------------------------------------

/*Funzione che memorizza in una lista la riga contenente la posizione dei motori
corrispondente ad un movimento
In questo modo viene creata una lista contenente le righe del file d'output!
Tale lista verrà memorizzata in un file solo alla fine
Non memorizzo direttamente le righe perchè queste devono essere modificate quando ci
Loop
*/
void MemorizzaMovimento()
{
int lungh;
struct riga *p, *temp;

p=(struct riga *)malloc(sizeof(struct riga));

//alloco la quantita di spazio necessaria x creare una cella contenente l'informazione
if(TipoTraduzione==TOP_DANCER) strcpy(p->informazione,posizione_topdancer);
// Se la traduzione deve essere x TOP DANCER allora 'informazione' conterr Euna riga
leggibile a TOP DANCER
else strcpy(p->informazione,posizione_attuale);

// altrimenti la riga contenuta in informazione sar Eleggibile al Simulatore SLIDER
p->next=NULL;

if(posizione_motori[74]!=0) {
```

```
//questa condizione risulta vera quando si esce da un LOOP
        Dec2Esa_SX(posizione_motori[74],Nexa);
        //Permette di proseguire l'esecuzione delle azioni subito dopo un LOOP
        lungh=strlen(p->informazione);
        p->informazione[lungh-2]=Nexa[0];
        p->informazione[lungh-1]=Nexa[1];
        posizione_motori[74]++;
        }

// Adesso devo inserire la cella appena creata in coda alla lista

if (testa==NULL) testa=p;
// Se la lista è vuota allora inserisco la cella appena creata in testa alla lista
else {                                  //altrimenti
        temp=testa;
        while(temp->next!=NULL){
        //scorro la lista fino a determinare l'ultimo elemento
                temp=temp->next;
                //e inserisco la cella in coda alla lista
                }
        temp->next=p;
        }
}

//---------------------------------------------------------------------

/*
Funzione di Inizializzazione:
Si occupa di memorizzare nella lista dei movimenti da effettuare la posizione iniziale
assunta dal Robot e di memorizzare il valore iniziale dei motori nel vettore
posizione_motori*/

void inizializza()
{
FILE *fp;                               //Creo il FILE_OUTPUT
fp=fopen(FILE_OUTPUT,"w");    // se già esiste ne cancello il contenuto
fclose(fp);

MemorizzaMovimento();
//In questo modo inserisco in testa alla lista dei movimenti
//la posizione iniziale ke il robot deve assumere

if(TipoTraduzione==TOP_DANCER) {
//Se la traduzione deve essere relativa a TOP DANCER modifico
//opportunamente i valori assunti dai vari arti
        printf("\n Traduzione x Top Dancer\n");
        VELOCITA=1;
        SHOULDER_I_SX=22;
        SHOULDER_I_DX=52;
        SHOULDER_II_SX=25;
        SHOULDER_II_DX=55;
        ELBOW_SX=28;
        ELBOW_DX=58;
        WIRST_SX=31;
        WIRST_DX=61;
        WAIST_I_C=67;
        WAIST_II_C=64;
        HIP_I_DX=37;
        HIP_I_SX=7;
        HIP_II_DX=34;
        HIP_II_SX=4;
        KNEE_DX=40;
        KNEE_SX=10;
        ANKLE_I_DX=43;
        ANKLE_I_SX=13;
        ANKLE_II_DX=46;
        ANKLE_II_SX=16;
        ANKLE_III_DX=49;
        ANKLE_III_SX=19;
        LOOP=70;
        posizione_motori[SHOULDER_I_DX]=225;
        posizione_motori[SHOULDER_I_SX]=225;
        posizione_motori[SHOULDER_II_DX]=91;
        posizione_motori[SHOULDER_II_SX]=91;
```

```
        posizione_motori[ELBOW_DX]=239;
        posizione_motori[ELBOW_SX]=239;
        posizione_motori[WIRST_DX]=31;
        posizione_motori[WIRST_SX]=31;
        posizione_motori[WAIST_I_C]=128;
        posizione_motori[WAIST_II_C]=178;
        posizione_motori[HIP_I_DX]=113;
        posizione_motori[HIP_I_SX]=113;
        posizione_motori[HIP_II_DX]=128;
        posizione_motori[HIP_II_SX]=128;
        posizione_motori[KNEE_DX]=228;
        posizione_motori[KNEE_SX]=228;
        posizione_motori[ANKLE_I_DX]=127;
        posizione_motori[ANKLE_I_SX]=127;
        posizione_motori[ANKLE_II_DX]=133;
        posizione_motori[ANKLE_II_SX]=133;
        posizione_motori[ANKLE_III_DX]=127;
        posizione_motori[ANKLE_III_SX]=127;

        }

        //Inizializzo la posizione iniziale dei motori
        else{
        posizione_motori[SHOULDER_I_DX]=225;
        posizione_motori[SHOULDER_I_SX]=225;
        posizione_motori[SHOULDER_II_DX]=91;
        posizione_motori[SHOULDER_II_SX]=91;
        posizione_motori[ELBOW_DX]=239;
        posizione_motori[ELBOW_SX]=239;
        posizione_motori[WIRST_DX]=31;
        posizione_motori[WIRST_SX]=31;

        posizione_motori[WAIST_I_C]=127;
        posizione_motori[WAIST_II_C]=77;
        posizione_motori[HIP_I_DX]=128;
        posizione_motori[HIP_I_SX]=128;
        posizione_motori[HIP_II_DX]=113;
        posizione_motori[HIP_II_SX]=113;
        posizione_motori[KNEE_DX]=228;
        posizione_motori[KNEE_SX]=228;
        posizione_motori[ANKLE_I_DX]=127;
        posizione_motori[ANKLE_I_SX]=127;
        posizione_motori[ANKLE_II_DX]=133;
        posizione_motori[ANKLE_II_SX]=133;
        posizione_motori[ANKLE_III_DX]=127;
        posizione_motori[ANKLE_III_SX]=127;
        }

}  //Chiudi Funzione Inizializza

//-----------------------------------------------------------------------

/*Funzione che converte il numero Decimale in esadecimale
 relativamente ai motori della parte destra ; Riceve in ingresso:    ndec:numero
decimale da convertire;      nesad: è un vettore di caratteri contenente il
n°esadecimale convertito*/

void Dec2Esa(int ndec, char* nesad)
{
int temp;
char n_esad[2];

        temp=ndec%16;                //resto della divisione
        nesad[1]=ValoreEsadecimale(temp);
        temp=ndec/16;                //quoziente della divisione
        nesad[0]=ValoreEsadecimale(temp);
}

//-----------------------------------------------------------------------

/* Funzione che converte il numero Decimale in esadecimale relativamente ai motori
della parte Sinistra
 Riceve in ingresso:  ndec:numero decimale da convertire;
nesad: è un vettore di caratteri contenente il n°esadecimale convertito */
```

```
void Dec2Esa_SX(int ndec, char* nesad)
{
int temp;
char n_esad[2];

       temp=ndec%16;                //resto della divisione
       nesad[1]=ValoreEsadecimaleParteSinistra(temp);
       temp=ndec/16;                //quoziente della divisione
       nesad[0]=ValoreEsadecimaleParteSinistra(temp);
}

//---------------------------------------------------------------------

/*
 Funzione che converte il numero esadecimale in in decimale
 Ingresso: ne: numero esadecimale da convertire
 Uscita:       ritorna il numero intero decimale corrispondente
*/
int Esa2Dec(char* ne)
{
int risultato=0;
int temp;

       temp=ValoreDecimale(ne[0]);
       risultato=risultato + temp*16;
       temp=ValoreDecimale(ne[1]);
       risultato=risultato + temp;

return (risultato);
}

//---------------------------------------------------------------------

/*Funzioni d'appoggio alla conversione*/
char ValoreEsadecimaleParteSinistra(int index)
{
char vettore_valori[16]={'0','1','2','3','4','5','6','7'
                        ,'8','9','a','b','c','d','e','f'};

return (vettore_valori[index]);

}

char ValoreEsadecimale(int index)
{
char vettore_valori[16]={'f','e','d','c','b','a','9','8','7',
                         '6','5','4','3','2','1','0'};

return (vettore_valori[index]);
}


int ValoreDecimale(char caratt)
{
int h;
char vettore_valori[16]={'f','e','d','c','b','a','9','8','7',
                         '6','5','4','3','2','1','0'};
int index;

for(h=0;h<16;h++)
{
       if(vettore_valori[h]==caratt) {
                      index=h;
                      h=16;
                      }
}

return (index);
}

//---------------------------------------------------------------------

/*
```

```
Funzione x la creazione di cicli ke permettono la ripetizione di movimenti Riceve in
Ingresso: NumRipetizioni - Numero di azioni da ripetere */

void CreaRipetizioni(int NumRipetizioni)
{
char Int2Char[31]={'0','1','2','3','4','5','6','7','8','9'};
struct riga *pointer;

pointer=testa;
while(pointer->next!=NULL) pointer=pointer->next;
//scorro la lista delle azioni fino a determinare l'ultima
pointer->informazione[LOOP]='-';
//aggiorno l'ultima azione in modo ke terminata l'esecuzione della stessa
//torno indietro x ripetere il ciclo voluto
if (NumRipetizioni<10){
       pointer->informazione[LOOP+1]=Int2Char[NumRipetizioni];}

else if (NumRipetizioni==10) {//Nel caso in cui il numero di azioni da ripetere sia 10
       pointer->informazione[LOOP+1]='1';
       pointer->informazione[LOOP+2]='0';
       pointer->informazione[LOOP+3]='!';
       pointer->informazione[LOOP+4]='0';
       pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==11) {//Nel caso in cui il numero di azioni da ripetere sia 11
       pointer->informazione[LOOP+1]='1';
       pointer->informazione[LOOP+2]='1';
       pointer->informazione[LOOP+3]='!';
       pointer->informazione[LOOP+4]='0';
       pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==12) {//Nel caso in cui il numero di azioni da ripetere sia 12
       pointer->informazione[LOOP+1]='1';
       pointer->informazione[LOOP+2]='2';
       pointer->informazione[LOOP+3]='!';
       pointer->informazione[LOOP+4]='0';
       pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==13) {//Nel caso in cui il numero di azioni da ripetere sia 13
       pointer->informazione[LOOP+1]='1';
       pointer->informazione[LOOP+2]='3';
       pointer->informazione[LOOP+3]='!';
       pointer->informazione[LOOP+4]='0';
       pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==14) {//Nel caso in cui il numero di azioni da ripetere sia 14
       pointer->informazione[LOOP+1]='1';
       pointer->informazione[LOOP+2]='4';
       pointer->informazione[LOOP+3]='!';
       pointer->informazione[LOOP+4]='0';
       pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==15) {
       pointer->informazione[LOOP+1]='1';
       pointer->informazione[LOOP+2]='5';
       pointer->informazione[LOOP+3]='!';
       pointer->informazione[LOOP+4]='0';
       pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==16) {
       pointer->informazione[LOOP+1]='1';
       pointer->informazione[LOOP+2]='6';
       pointer->informazione[LOOP+3]='!';
       pointer->informazione[LOOP+4]='0';
       pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==17) {
       pointer->informazione[LOOP+1]='1';
       pointer->informazione[LOOP+2]='7';
       pointer->informazione[LOOP+3]='!';
       pointer->informazione[LOOP+4]='0';
       pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==18) {
       pointer->informazione[LOOP+1]='1';
       pointer->informazione[LOOP+2]='8';
       pointer->informazione[LOOP+3]='!';
       pointer->informazione[LOOP+4]='0';
       pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==19) {
       pointer->informazione[LOOP+1]='1';
       pointer->informazione[LOOP+2]='9';
```

```
        pointer->informazione[LOOP+3]='!';
        pointer->informazione[LOOP+4]='0';
        pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==20) {
        pointer->informazione[LOOP+1]='2';
        pointer->informazione[LOOP+2]='0';
        pointer->informazione[LOOP+3]='!';
        pointer->informazione[LOOP+4]='0';
        pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==21) {
        pointer->informazione[LOOP+1]='2';
        pointer->informazione[LOOP+2]='1';
        pointer->informazione[LOOP+3]='!';
        pointer->informazione[LOOP+4]='0';
        pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==22) {
        pointer->informazione[LOOP+1]='2';
        pointer->informazione[LOOP+2]='2';
        pointer->informazione[LOOP+3]='!';
        pointer->informazione[LOOP+4]='0';
        pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==23) {
        pointer->informazione[LOOP+1]='2';
        pointer->informazione[LOOP+2]='3';
        pointer->informazione[LOOP+3]='!';
        pointer->informazione[LOOP+4]='0';
        pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==24) {
        pointer->informazione[LOOP+1]='2';
        pointer->informazione[LOOP+2]='4';
        pointer->informazione[LOOP+3]='!';
        pointer->informazione[LOOP+4]='0';
        pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==25) {
        pointer->informazione[LOOP+1]='2';
        pointer->informazione[LOOP+2]='5';
        pointer->informazione[LOOP+3]='!';
        pointer->informazione[LOOP+4]='0';
        pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==26) {
        pointer->informazione[LOOP+1]='2';
        pointer->informazione[LOOP+2]='6';
        pointer->informazione[LOOP+3]='!';
        pointer->informazione[LOOP+4]='0';
        pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==27) {
        pointer->informazione[LOOP+1]='2';
        pointer->informazione[LOOP+2]='7';
        pointer->informazione[LOOP+3]='!';
        pointer->informazione[LOOP+4]='0';
        pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==28) {
        pointer->informazione[LOOP+1]='2';
        pointer->informazione[LOOP+2]='8';
        pointer->informazione[LOOP+3]='!';
        pointer->informazione[LOOP+4]='0';
        pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==29) {
        pointer->informazione[LOOP+1]='2';
        pointer->informazione[LOOP+2]='9';
        pointer->informazione[LOOP+3]='!';
        pointer->informazione[LOOP+4]='0';
        pointer->informazione[LOOP+5]='0';}
else if (NumRipetizioni==30) {
        pointer->informazione[LOOP+1]='3';
        pointer->informazione[LOOP+2]='0';
        pointer->informazione[LOOP+3]='!';
        pointer->informazione[LOOP+4]='0';
        pointer->informazione[LOOP+5]='0';}

}


//-----------------------------------------------------------------
```

```
/* Funzione x la creazione di un numero prestabilito di azioni da
ripetere ;*Riceve in Ingresso: Num(Numero di azioni da ripetere) Prim(Primo elemento da
ripetere) */

void CreaNRipetizioni(int Prim,int Num)
{
int indice;
//struct riga *primo;
struct riga *temp;
struct riga *p;
//struct riga *Lista1;
struct riga *ScorriLista;
struct riga *nuova_lista;

nuova_lista=CreaListaDaRipetere(Prim);
//crea una lista contenente le azioni ke voglio siano ripetute

//Aggiungo alla lista principale le righe ke mi permettono la ripetizione delle azioni
for(indice=0;indice<Num;indice++) {
                      temp=nuova_lista;
                      while (temp!=NULL) {
                                    p=(struct riga *)malloc(sizeof(struct riga));

                                    //alloco la quantita di spazio necessaria x creare
una cella contenente l'informazione
                                    strcpy(p->informazione,temp->informazione);
                                    p->next=NULL;

                                    ScorriLista=testa;
                                    while(ScorriLista->next!=NULL)
ScorriLista=ScorriLista->next;
                                    //scorro la lista fino a determinare l'ultimo
elemento
                                    ScorriLista->next=p;

                                    //e inserisco la cella in coda alla lista
                                    NumeroAzioni++;

                                    //Incremento il numero di azioni eseguite
                                    temp=temp->next;
                                    }// Chiude il while
                      } //Chiude il for

}



//---------------------------------------------------------------------

/* Crea la lista contenente gli elementi da ripetere*/

struct riga* CreaListaDaRipetere(int PrimoElementoDaRipetere)
{
int i;
struct riga *primo;
struct riga *Lista=NULL;
struct riga *p;
struct riga *temp;

primo=testa;
for (i=0;i<PrimoElementoDaRipetere;i++) //Determino il I elemento da ripetere
                              primo=primo->next;

//Creo una lista i cui elementi sono lo righe da ripetere
while (primo!=NULL) {
       p=(struct riga *)malloc(sizeof(struct riga));
       //alloco la quantita di spazio necessaria x creare una cella contenente
l'informazione
       strcpy(p->informazione,primo->informazione);
       p->next=NULL;

       if (Lista==NULL) Lista=p;
```

```
       // Se la lista  Evuota allora inserisco la cella appena creata in testa alla
lista
       else {                                  //altrimenti
              temp=Lista;
              while(temp->next!=NULL) temp=temp->next;
       //scorro la lista fino a determinare l'ultimo elemento
       temp->next=p;                 //e inserisco la cella in coda alla lista
              }
       primo=primo->next;
       }//chiudi while

return(Lista);
}

//---------------------------------------------------------------------

/*Funzione ke visualizza gli elementi di una lista*/
void VisualizzaLista(struct riga* Lista)
{
while(Lista!=NULL) {
              printf("\n %s",Lista->informazione);
              Lista=Lista->next;

       }
}
//---------------------------------------------------------------------
/* Funzione ke concatena due liste */
struct riga* ConcatenaListe(struct riga *PrimaLista,struct riga *SecondaLista)
{
struct riga *temporan;
temporan=PrimaLista;
while(temporan->next!=NULL) temporan=temporan->next;
temporan->next=SecondaLista;
return(PrimaLista);
}
//---------------------------------------------------------------------
void FineTraduzione()
{
FILE *fp;
fp=fopen(FILE_FINETRADUZIONE,"w");
fclose(fp);
}
//---------------------------------------------------------------------
void yyerror(s)
       char* s;
{
printf("Errore");
//FineTraduzione();
}
//---------------------------------------------------------------------
int yywrap(void)
{
return 1;
}
//---------------------------------------------------------------------
int Arrotonda(float f)
{
int in;
float res;
in= (int)(f);
res= f-in;
if(res>0.5){
return(in+1);
}
else{
return(in);
}

}
```

Appendice B.

Appendice B.1: Istallazione "RobovieMovement"

L'installazione di "RobovieMovement" viene effettuata eseguendo il file *RobovieMovement_Setup.exe,* presente all'interno del CD in dotazione alla presente documentazione. In seguito, sarà sufficiente seguire le semplici istruzioni che compariranno nelle varie schermate.
Sarà possibile, inoltre, creare automaticamente un collegamento sul desktop al file *RobovieMovement.exe* al fine di eseguire direttamente da desktop il programma.
Per il corretto funzionamento del programma, sono necessari i seguenti requisiti minimi di sistema:

- Sistema operativo: Windows 95 o superiori;
- Processore: Pentium a 166 Mhz o equivalente;
- Lettore cd-rom;
- Hard Disk: 20 Mb liberi;
- Memoria RAM: 64 Mb;
- Installazione di "RoboStage.
- Istallazione di "RobovieMaker"

Il programma viene di default installato nella cartella *C:\Programmi\RobovieMovement* e può essere eseguito anche facendo doppio click sul file *RobovieMovement.exe* all'interno della suddetta cartella. *RobovieMovement* viene anche inserito nella barra di avvio veloce presente in basso al desktop e seguendo il percorso *Start->Programmi -> RobovieMovement*.
La disinstallazione può essere effettuata da pannello di controllo o mediante l'apposito unistall in dotazione al programma.