

UNIVERSITÀ DEGLI STUDI DI PADOVA

---

FACOLTÀ DI INGEGNERIA  
DIPARTIMENTO DI INGEGNERIA DELL' INFORMAZIONE

---

LAUREA TRIENNALE IN INGEGNERIA INFORMATICA

ELABORATO

CONTROLLO IN TEMPO REALE DEL  
ROBOT UMANOIDE ROBOVIE-X CON  
LINGUAGGIO PSEUDO-NATURALE

RELATORE: PROF. MICHELE MORO

LAUREANDO: NICOLÒ FREZZATO

ANNO ACCADEMICO 2010-2011



# Indice

<b>Sommario</b>	<b>VII</b>
<b>Introduzione</b>	<b>IX</b>
<b>1 Robovie-X</b>	<b>3</b>
1.1 Descrizione . . . . .	3
1.2 Caratteristiche principali . . . . .	4
1.3 I servomotori . . . . .	6
1.4 La scheda di controllo . . . . .	6
<b>2 RobovieMaker2</b>	<b>9</b>
2.1 Introduzione . . . . .	9
2.2 Creare i movimenti . . . . .	10
2.2.1 Schema della <i>Pose Area</i> . . . . .	10
2.2.2 Creare una posa . . . . .	11
2.2.3 Montaggio delle pose . . . . .	12
2.3 Analisi dei file di output . . . . .	16
2.3.1 Blocco Posa . . . . .	17
2.3.2 Blocco di ciclo . . . . .	18
2.3.3 Blocco di calcolo . . . . .	18
2.3.4 Blocco di salto condizionato . . . . .	19
2.3.5 Blocco di inizio e blocco di fine . . . . .	19
<b>3 Flex &amp; Bison</b>	<b>21</b>
3.1 Introduzione . . . . .	21
3.2 Flex: Analisi Lessicale . . . . .	21
3.3 Bison: Analisi Sintattica . . . . .	24
<b>4 Interpretazione del linguaggio pseudo-naturale</b>	<b>27</b>
4.1 Introduzione . . . . .	27
4.2 I comandi . . . . .	27

4.3	Suoni . . . . .	30
4.4	Velocità dei movimenti . . . . .	31
4.5	Più azioni contemporaneamente . . . . .	31
4.6	I cicli . . . . .	31
4.7	Creare comandi personalizzati . . . . .	32
4.7.1	Un esempio di “macro” . . . . .	32
4.8	Movimenti Predefiniti . . . . .	33
4.9	Attivare la modalità “real-time” . . . . .	33
4.10	Un esempio . . . . .	34
4.11	Tutti i comandi . . . . .	36
<b>5</b>	<b>Controllo in tempo reale</b>	<b>39</b>
5.1	Introduzione . . . . .	39
5.2	Analisi del protocollo di comunicazione . . . . .	39
5.3	Connessione al Robovie-X . . . . .	40
5.4	Scambiare informazioni con l’umanoide . . . . .	40
5.5	Struttura dei pacchetti . . . . .	41
5.5.1	Inizializzazione . . . . .	41
5.5.2	Accendere e spegnere i servo-motori . . . . .	42
5.5.3	Esecuzione di una posa . . . . .	43
<b>6</b>	<b>Realizzazione del traduttore - Manuale Tecnico</b>	<b>47</b>
6.1	Mylexer.l . . . . .	47
6.1.1	Definizioni . . . . .	47
6.1.2	Regole . . . . .	48
6.2	Myparser.y . . . . .	50
6.2.1	Prologo . . . . .	50
6.2.2	Dichiarazioni . . . . .	55
6.2.3	Regole grammaticali . . . . .	55
6.2.4	Macro . . . . .	58
6.3	Metodo <code>main()</code> . . . . .	59
6.4	Implementazione del movimento in tempo reale nel Traduttore . . . . .	60
6.5	Connessione e scambio dati . . . . .	60
6.5.1	La funzione <code>rcOpen()</code> . . . . .	60
6.5.2	Le funzioni <code>rcWrite()</code> e <code>rcRead()</code> . . . . .	61
6.5.3	Libreria <code>move</code> . . . . .	62
6.6	Creazione dei pacchetti . . . . .	63
6.6.1	La metodo <code>execPose()</code> . . . . .	63
6.7	Istruzioni per la compilazione . . . . .	64

<b>Conclusioni</b>	<b>65</b>
<b>A Flex &amp; Bison</b>	<b>67</b>
A.1 Flex . . . . .	67
A.2 Bison . . . . .	69
A.3 OnlineMode & rcCom . . . . .	85



# Sommario

Questo progetto ha lo scopo di realizzare uno strumento semplice ed efficace per permettere a un qualsiasi utente l'interazione con il robot umanoide *Robovie-X*. Data la sua facilità di utilizzo può essere impiegato nella didattica scolastica come primo approccio al mondo dell'informatica.

Il risultato di questo progetto è un programma eseguibile che riceve come input delle istruzioni, elementari o complesse, in linguaggio pseudo-naturale, ovvero delle frasi di senso compiuto in lingua inglese ma con una determinata struttura.

Il programma opera in 2 diverse modalità producendo risultati differenti:

- genera un “file di comandi” per l'umanoide, che è possibile eseguire utilizzando il software *RobovieMaker 2*
- rende possibile il controllo in tempo reale del *Robovie-X*

Un lavoro di reverse-engineering di *RobovieMaker 2*, software proprietario dell'umanoide, ha reso possibile la comprensione delle codifiche, dei parametri e del protocollo utilizzato successivamente nel programma realizzato in questo progetto.

La parte di programma che interpreta l'input e traduce le istruzioni rendendole eseguibili è stata realizzata utilizzando un analizzatore lessicale e grammaticale generato da *Flex & Bison*,

Le istruzioni previste che il traduttore è in grado di riconoscere sono:

- la modifica della posizione di un giunto o del valore di un sensore;
- la possibilità di muovere più di un giunto contemporaneamente;
- la specifica di un tempo di transizione tra due pose consecutive;
- l'assegnazione di un suono da riprodurre durante ogni posa;
- l'inserimento di movimenti predefiniti all'interno del movimento che si sta realizzando.

Inoltre si dà la possibilità all'utente di definire delle “macro”, ovvero delle sequenze di istruzioni da poter chiamare in un secondo momento.





# Introduzione

Negli ultimi anni la ricerca nel campo della robotica sta riscuotendo sempre più successo e si sta diffondendo sempre più anche nell'ambito della didattica: si è pensato, quindi, di sviluppare un progetto da mettere al servizio dell'insegnamento.

Il traduttore è stato creato per ridurre la distanza tra didattica e robotica e per permettere a persone che non possiedono delle competenze specifiche di utilizzare senza difficoltà un robot umanoide di ultima generazione.

La prima riflessione riguarda la necessità di far capire al robot un linguaggio naturale, cioè di alto livello, utilizzato normalmente per impartire comandi: si è reso quindi necessario comprendere come tradurre un'istruzione nei corrispondenti comandi di basso livello. Per la realizzazione sono stati utilizzati gli strumenti *Flex* e *Bison* attraverso i quali è stato possibile procedere con l'analisi sintattica e con l'analisi semantica delle istruzioni di alto livello, in modo da tradurle in istruzioni per il robot umanoide Robovie-X.

Le istruzioni che il traduttore è in grado di riconoscere, e successivamente di elaborare, riguardano il movimento di uno o più parti del corpo, sia in contemporanea che non; nella frase può essere specificato il tempo di transizione tra la posizione iniziale e quella finale e anche il suono da riprodurre durante il movimento; inoltre è possibile impartire dei comandi predefiniti ed è previsto anche l'utilizzo di cicli annidati.

La struttura del file generato dal traduttore è stata analizzata mediante un lavoro di reverse-engineering nel quale sono stati studiati i singoli blocchi che compongono una posa o un'operazione, al fine di capire meglio la funzione di ogni singolo parametro.

Questo progetto può essere una base per ulteriori sviluppi, sia ampliando la gamma dei movimenti attuabili che implementando nuove funzioni. Infatti, nonostante vengano già forniti circa 50 movimenti predefiniti per il robot, si è voluto creare un traduttore più versatile in grado di tradurre delle sequenze di istruzioni in un file di comandi che possano essere semplici, complesse oppure che contengano movimenti predefiniti, a seconda delle preferenze di chi utilizza il programma.

L'utente ha anche possibilità di creare dei comandi associando a una sequenza

di istruzioni una parola chiave.

Infine è stata aggiunta la possibilità di controllare l'umanoide in tempo reale, ovvero, il traduttore è in grado di interpretare i comandi ricevuti in ingresso e, attraverso il collegamento USB, impartire le istruzioni al Robovie-X.

Il risultato della tesi è un programma eseguibile che riconosce delle istruzioni sia elementari che complesse in lingua inglese e le trasforma in un file di comandi in uscita che rappresenta le stesse istruzioni inserite in ingresso, ma in un formato che sia decodificabile dal robot.

Contemporaneamente è possibile vedere in real-time gli effetti dei comandi sul robot.

Il robot umanoide Vstone Robovie-X ha delle ottime caratteristiche strutturali e meccaniche che, unite al prezzo non eccessivamente elevato, lo rendono un interessante candidato per un progetto di robotica educativa. Nonostante ciò, l'ambiente di programmazione fornito, il software RobovieMaker2, oltre al problema che non può essere studiato e modificato per scopi didattici presenta un ulteriore inconveniente: per creare un semplice movimento è necessaria una buona conoscenza del software che si sta utilizzando.

Al fine di risolvere questi due problemi si è deciso di realizzare un traduttore che permetta di trasformare un'istruzione in linguaggio pseudo-naturale nel corrispondente file di comandi nel modo più semplice e veloce possibile. Il file che si ottiene in output, se caricato con il programma RobovieMaker, fa eseguire al robot la sequenza di pose specificate precedentemente attraverso il linguaggio di alto livello.





# Capitolo 1

## Robovie-X

### 1.1 Descrizione

Il *Robovie*<sup>1</sup>-X è un robot umanoide di piccola taglia in grado di compiere movimenti complessi e sequenze di pose programmabili dall'utente. Vengono fornite in dotazione più di 50 azioni predefinite diverse tra cui camminare, fare la capriola e alzarsi da solo. Il robot è costituito da 13 o 17 motori programmabili più 2 led per gli occhi ed un altoparlante per riprodurre i suoni; può anche essere equipaggiato con schede di espansione.

---

<sup>1</sup>Robovie è un marchio registrato di proprietà di ATR



Figura 1.1: Robot umanoide Robovie-X

Nel nome *Robovie-X* la “X” sta per *flexible*, flessibile in italiano, che indica il fatto che lo stesso prodotto è disponibile in tre modelli: “X”, “X-Lite” e “X-PRO”. Il robot è in grado di ricevere input analogici e può produrre in uscita movimenti come la rotazione degli arti avanti e indietro, è in grado di riprodurre musica e ballare, inoltre può avere un comportamento espressivo come addormentarsi o essere comico.

## 1.2 Caratteristiche principali

Il modello che è stato utilizzato per la realizzazione di questo elaborato è il *Robovie-X*.

- Dimensioni : 343 x 180 x 71 mm
- Peso : 1,3 Kg
- Scheletro interno : alluminio
- Scheletro esterno : guscio in plastica
- Gradi di libertà :
  - Testa : 1
  - Braccia : 6
  - Gambe : 10

Il kit di montaggio comprende:

- 1 Robot da assemblare
- 17 Servomotori
- 2 LED per occhi
- 1 Batteria a 6 V/1400mA a 5 celle NiMH
- 1 Caricabatteria
- Altoparlanti 2 W
- Cavo USB
- 1 CD-ROM con il software RobovieMaker2 e la guida per l'uso
- Altro : viti



(a) Batteria Ni-MH da 6 V



(b) Altoparlante 2 W

Figura 1.2: Accessori principali

## 1.3 I servomotori

I servomotori di cui è composto il *Robovie-X* sono di tipo VS-S092J: nel kit di montaggio ne sono presenti 10 con il cavo di collegamento lungo 400 mm e 7 con il cavo di collegamento corto di 150 mm. La velocità di rotazione e la capacità di torsione, insieme alla leggerezza dello scheletro del robot, permettono di eseguire ogni tipo di movimento dai più semplici ai più complessi. Le caratteristiche dei servomotori sono:

- Dimensioni : 38 x 19 x 38.5 mm
- Torsoine : 9.2 kg·cm
- Velocità : 60 gradi / 0.11 s
- Peso : 42 g
- Massimo Angolo Operazionale : 180 gradi
- Voltaggio Operativo: 4 V - 9 V
- Sistema di Controllo: PWM



Figura 1.3: I servomotori

## 1.4 La scheda di controllo

La scheda montata sul *Robovie-X* è di tipo VS-RC003HV, monta una CPU ARM LPC2148FBD64 ed ha la possibilità di essere espansa aggiungendo delle schede di espansione o dei controller aggiuntivi.

- Tipo : VS-RC003HV
- CPU: LPC2148FBD64 (60MHz clock)



- Memoria: 512 kB (ROM), 64 kB (RAM)
- Porte: 30 x PWM (analogiche), 1 x USB
- Altre caratteristiche:
  - Supporto di controllo: compatibile con game pad e controllori ProBo.
  - Porta di espansione: include 1 porta IXPBUS.
  - Audio: 2 W output.
  - Sincronizzazione con il suono: l'audio e i movimenti possono essere sincronizzati.



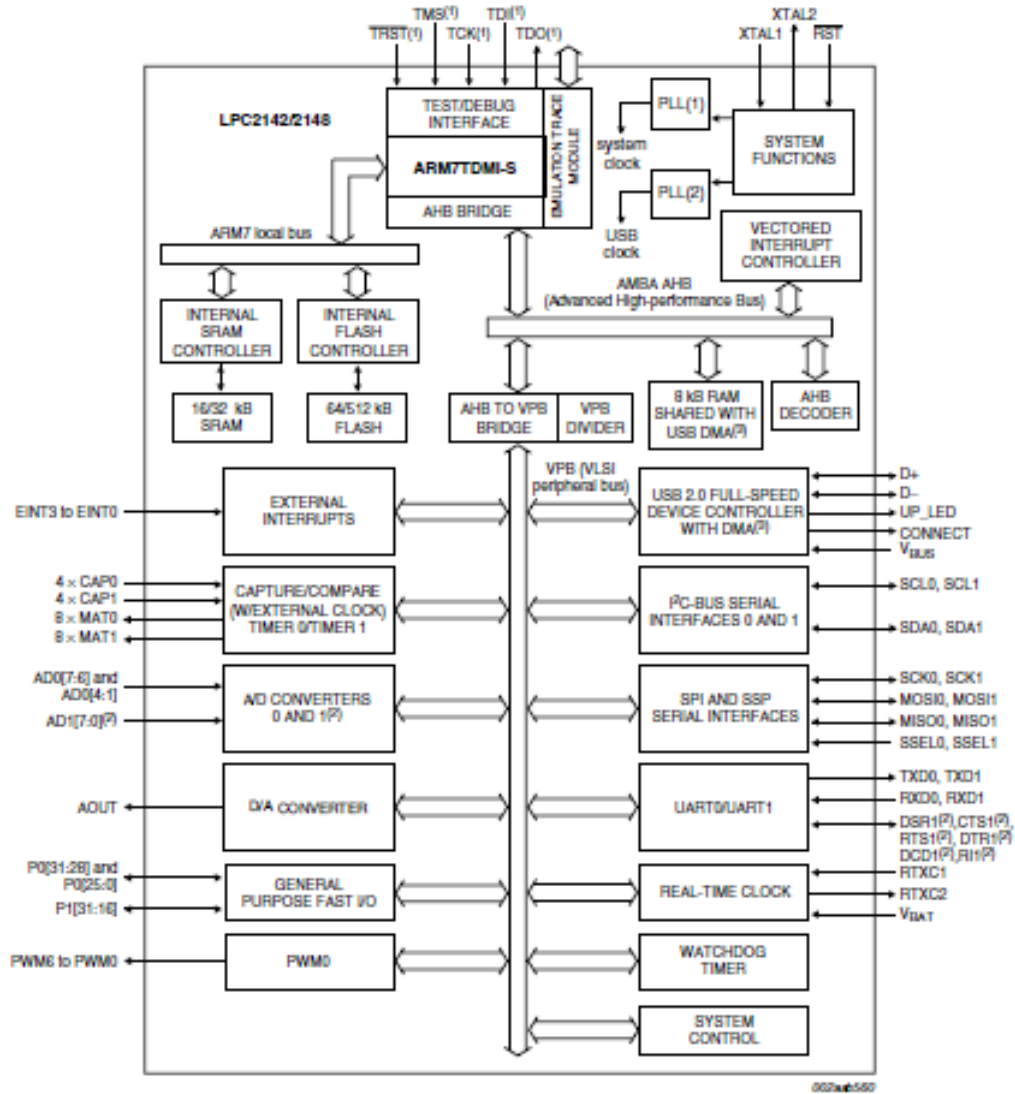
Figura 1.4: La scheda VS-RC003HV

Il processore montato sulla CPU è l'ARM7TDMI: un microprocessore general-purpose a 32 bit che offre alte prestazioni ed un consumo energetico molto basso. L'architettura ARM è basata sul principio *Reduced Instruction Set Computer* (RISC) e il set di istruzioni utilizza un meccanismo di decodifica molto più semplice di quello microprogrammato. Il risultato è la semplicità di un throughput elevato nell'esecuzione delle istruzioni e una notevole prontezza nella gestione degli interrupt, tutto questo a fronte di un core dalle dimensioni ridotte e dal prezzo contenuto.

La CPU LPC2148 incorpora 512 kB di memoria flash che può essere usata per contenere sia dati che codice. La memoria flash può essere programmata in diversi modi, ad esempio tramite la porta seriale. A fronte delle scelte effettuate nella progettazione, la memoria realmente disponibile per contenere il codice dell'utente è di 500 kB. La memoria flash in dotazione con la CPU garantisce un numero minimo di cicli di scrittura pari a 100000; inoltre garantisce vent'anni di durata per i dati contenuti in essa.

La memoria RAM disponibile è pari a 32 kB (altri 32 kB sono messi a disposizione dalla scheda) e può essere indirizzata a 8, 16 e 32 bit.

LPC2148FBD64 è equipaggiato inoltre con un USB device controller che permette lo scambio di informazioni a 12 Mbit/s con un USB host controller.



(1) Pins shared with GPIO.

(2) LPC2148 only.

(3) USB DMA controller with 8 kB of RAM accessible as general purpose RAM and/or DMA is available in LPC2148 only.

Figura 1.5: Schema della scheda LPC2148

# Capitolo 2

## RobovieMaker2

### 2.1 Introduzione

RobovieMaker2<sup>1</sup> è il software dedicato alla gestione della scheda VS-RC003 per il controllo del robot che permette di scegliere numerose impostazioni e preferenze come l'utilizzo di servomotori, controller e schede di espansione. Questo software include anche una interfaccia grafica (GUI) di pannelli utili per creare i movimenti utilizzando la scheda VS-RC003.

Le specifiche minime richieste per l'installazione sono:

- Sistema Operativo : Microsoft Windows 2000/XP/Vista/7
- Processore : Pentium-III (almeno 1 GHz) o superiore
- Memoria : 128 MB RAM o superiore
- Interfaccia : porta USB
- Display : XGA (1024x768) o superiore

---

<sup>1</sup>RobovieMaker è un marchio registrato di proprietà di ATR

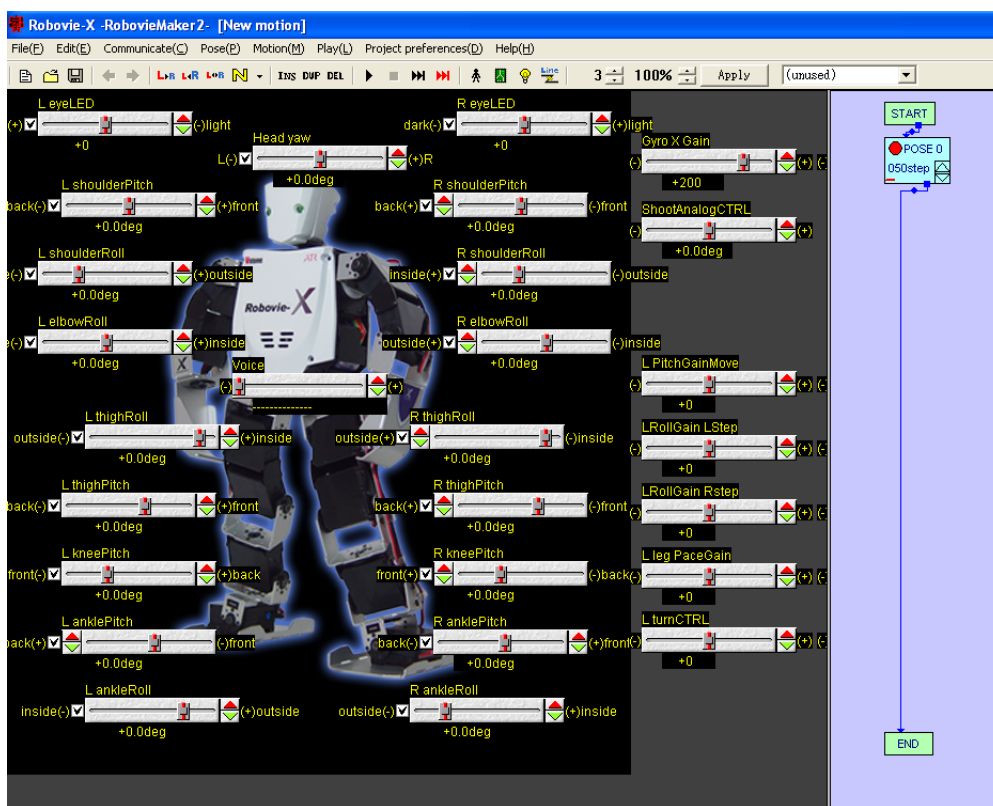


Figura 2.1: RobovieMaker2

## 2.2 Creare i movimenti

Creare i movimenti del robot, come la “camminata bipede” e “alzarsi da terra”, è lo scopo principale per cui questo programma è stato sviluppato.

### 2.2.1 Schema della *Pose Area*

È possibile creare movimenti nella *Pose Area* e nella *Motion Area* che sono le aree importanti per utilizzare questo software. La *Pose Area* è quella parte del programma dove si possono impostare le posizioni dei singoli servomotori, mentre la parte destra dello schermo, dov'è presente lo schema a blocchi, prende il nome di *Motion Area*. Per comprendere la struttura di un movimento basta pensare alle animazioni dal momento che i movimenti sono costituiti da pose ordinate in ordine temporale. Un robot esegue le pose in periodi fissi di tempo.

Si crea una posa in *Pose Area*, e la si registra nella *Motion Area*. Si ripete questo procedimento per creare tutte le pose di cui è composto un movimento. La *Motion Area* ha la funzione di modificare l'ordine di registrazione delle pose e per impostare il tempo di transizione tra due pose consecutive (che decide la velocità di movimen-

to); inoltre ha anche la funzione per registrare blocchi di condizione e di calcolo di variabili.

Nella *Pose Area* sono presenti diversi cursori per il controllo dei dispositivi collegati alla scheda CPU come i servomotori, i LED, ed i guadagni dei sensori. Questi si utilizzano ad esempio per creare le pose del robot impostando gli angoli dei servomotori. La *Pose Area* configura automaticamente la posizione e il tipo dei cursori a seconda del robot selezionato. La *Pose Area* può visualizzare fino a 62 cursori di scorrimento numerati dallo 0-esimo al 61-esimo, che hanno il ruolo di controllare rispettivamente l'assegnato servomotore. Ad esempio, dallo 0-esimo al 29-esimo cursore controllano gli angoli dei servomotori, il 30-esimo cursore è assegnato per la selezione del numero di suono da riprodurre e dal 31-esimo in poi i cursori servono per le impostazioni della scheda di espansione come il guadagno dei sensori del giroscopio.

Questo software usa la 62-esima e la 63-esima variabile, che dovrebbero essere utilizzate come cursori, come funzioni libere in RobovieMaker per VS-RC003. Pertanto, non è possibile utilizzarle come cursori in questo software. È possibile modificare i valori dei cursori usando le frecce e la “manopola” utilizzando il mouse per impostare le pose del robot. La cella di controllo dal cursore 0-esimo al 29-esimo hanno la funzione di controllare se gli interruttori dei servomotori sono accesi o spenti.



Figura 2.2: Il cursore per il controllo della testa del robot

### 2.2.2 Creare una posa

Si crea una posa accanto alla prima che prende il nome di posa di riferimento. La duplicazione di una posa ne crea un'altra modificabile: in questo modo si può cominciare a crearne un'altra dalla posizione finale della posa precedente. Il segnale visualizzato in colore rosso indica quale blocco posa si sta modificando. Per passare al blocco successivo e salvare il blocco posa corrente basta fare clic su un altro blocco posa. Mantenere in collegamento il proprio PC con la scheda CPU del robot con i servomotori accesi diventa conveniente per muovere il robot durante la fase di creazione dei movimenti.

Per collegare la scheda CPU al PC si utilizza il cavo incluso tra gli accessori. Successivamente, fare clic su “Comunicazione” - “Online” dalla barra dei menù o utilizzando il pulsante nella barra degli strumenti. In questo modo il PC avvia la

comunicazione con la scheda CPU e, finché la comunicazione sarà attiva, il pulsante viene mantenuto come premuto. Per terminare la connessione bisogna premere sullo stesso pulsante per passare allo stato offline.

Ora, si è pronti per creare la nuova posa utilizzando i cursori. È bene ricordare che il primo blocco deve essere collegato dal blocco *START* e che l'ultimo blocco deve avere un collegamento verso il blocco *END*.

Dopo aver modificato le pose, il passo successivo consiste nell'esecuzione del movimento creato. Per eseguire il movimento, si seleziona "Play" - "Play Motions" dal menù o dalla barra degli strumenti. Dopo il clic, il robot comincia ad eseguire il movimento dal blocco di partenza e continua secondo l'ordine indicato dalla freccia. Nel RobovieMaker le frecce che indicano l'esecuzione ordinata delle pose prendono il nome di flusso. Ci sono due tipi di flussi: "flusso normale" che parte da un blocco posa (di colore blu di default) e "flusso condizionale" usato per il blocco di ciclo o per il blocco di salto condizionato. Il flusso condizionale seleziona una freccia tra le due disponibili a seconda che sia verificata una condizione (che è di colore rosso di default se la condizione è vera). Il movimento si conclude quando il blocco posa è il blocco *END* oppure un qualsiasi blocco posa senza flusso in uscita.

Si noti che non è consentita alcuna operazione nel corso dell'esecuzione di un movimento, ad eccezione del comando di stop e dell'accensione o spegnimento dei servomotori. Per interrompere la riproduzione di un movimento si può selezionare "Play - Stop Motion" dal menù oppure premere il relativo pulsante dalla barra degli strumenti. Quando il movimento raggiunge la fine il software torna automaticamente dalla modalità di esecuzione di un movimento alla modalità di modifica. Durante l'esecuzione di un movimento il segnale nei blocchi si sposta per indicare in quale blocco si trova l'esecuzione in quell'istante.

### 2.2.3 Montaggio delle pose

Un movimento consiste in una sequenza di pose ordinate. La velocità di esecuzione di un blocco posa e la posizione che occupa all'interno di un movimento possono essere modificate nella *Motion Area*. I blocchi di salto condizionato, blocchi di ciclo, e blocchi di calcolo servono per la creazione di movimenti avanzati, come ripetere le stesse pose o l'esecuzione di pose diverse a causa della ricezione di informazioni da un sensore.

In un **blocco posa** è registrata una posa e le impostazioni già viste precedentemente. Il blocco posa non consiste solo nel segnale che indica quale blocco si sta modificando, ma anche nel settare il tempo della transizione e nel flusso normale

per indicare quale sarà il blocco successivo. È anche possibile cambiare nome alla posa e possono esistere più blocchi posa con la stessa etichetta anche se aumenta la confusione quando il numero di blocchi posa diventa elevato. Con un doppio clic sul blocco di posa è possibile configurare le impostazioni avanzate che sono il nome della posa e la posizione del blocco nella finestra.



Figura 2.3: Blocco posa

Il registro di posa ha la funzione di cambiare la velocità di movimento, cioè in quanto tempo la posa deve eseguire la trasformazione dalla posa precedente; questo viene chiamato tempo di transizione e si può cambiare con i pulsanti presenti sul blocco. Queste impostazioni, in ogni posa, consentono il montaggio di movimenti avanzati acrobatici come la capriola in avanti e la ruota laterale. Un tempo di transizione di 60 corrisponde circa ad 1 secondo poiché 1 unità di tempo di transizione corrisponde a 60 Hz (pari a circa 0,01666 s). È comunque possibile cambiare l'unità di transizione nelle preferenze di bordo della CPU. Per decelerare (o accelerare) un intero movimento si può modificare la percentuale di tempo di movimento nella barra degli strumenti.

I blocchi posa e un qualsiasi altro blocco visualizzato nella *Motion Area* è dotato di una freccia chiamata flusso che indica quale sarà la posa successiva durante l'esecuzione. Un blocco posa ha un flusso di tipo normale. Le connessioni tra i blocchi possono essere cambiate cliccando sulla radice o sulla punta della freccia e trascinandola in un blocco diverso.

Utilizzando il **blocco di ciclo** è possibile creare un ciclo di ripetizioni di pose fino al valore impostato. Una struttura che ripete delle pose viene chiamata struttura di ciclo, mentre una struttura che ripete pose per sempre prende il nome di struttura di ciclo infinita. Per inserire un blocco di ciclo fare clic su "Motion - "Aggiungi blocco del ciclo" dal menù. I blocchi di ciclo possono cambiare nome e il flusso di connessione come i blocchi posa avendo un flusso di tipo normale e una condizione per interrompere il ciclo. Il contatore del ciclo in RobovieMaker viene impostato dal bottone "loop counter" posizionato nella parte alta a destra. I blocchi di ciclo contano il numero delle ripetizioni eseguite ed escono quando il conteggio arriva al numero di cicli impostati nel contatore del ciclo. Pertanto, se il ciclo contiene diversi blocchi di ciclo, un ciclo viene contato diverse volte.



Figura 2.4: Blocco di ciclo

In generale, un programma realizza una struttura di ciclo utilizzando una variabile contatore. Durante la ripetizione di una parte del programma, il contatore viene decrementato e confrontato con lo zero. Se l'uguaglianza è verificata il programma esce dal ciclo. Questo software utilizza una struttura di ciclo in cui si può cambiare il valore della variabile contatore e dei salti condizionali. In questo modo si ha il vantaggio che gli utenti possono risolvere le loro esigenze in un unico momento senza dover preoccuparsi delle inizializzazioni dei contatori o dei confronti. D'altra parte, in questo software non è possibile utilizzare diversi cicli in un movimento finché il primo ciclo non termina cioè fino a quando il suo contatore è uguale a zero. Il ciclo successivo non ripeterà le pose ma salterà sul ramo di salto. Di conseguenza, non possono essere realizzati cicli nidificati.

Il **blocco di calcolo** serve per eseguire le operazioni con le variabili durante la riproduzione del movimento. Si possono realizzare parecchi movimenti avanzati inserendo i blocchi di calcolo. Per esempio, assegnando un contatore ad un blocco di calcolo diventa possibile creare cicli nidificati. Dalla combinazione di un salto condizionato diventa possibile creare diversi cicli in un movimento.

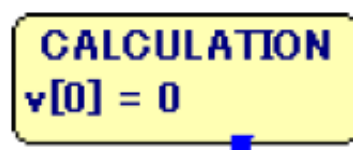


Figura 2.5: Blocco di calcolo

Per inserire un blocco di calcolo si seleziona “Motion” - “Add operation block” dal menù: questo verrà aggiunto nella *Motion Area*. Si può modificare il nome del blocco ed il flusso in uscita alla stessa maniera di un blocco posa. Nel blocco di calcolo viene visualizzata l'operazione che viene eseguita durante il movimento. Con un doppio click sul blocco di calcolo si apre la finestra di dialogo dove si possono modificare le preferenze dell'operazione da eseguire. Diversi calcoli sono disponibili ed in aggiunta è possibile assegnare un valore costante ad una variabile risultato in sostituzione di un calcolo. Le operazioni disponibili sono l'assegnazione di un valore ad una variabile, la somma, la differenza, la moltiplicazione e la divisione tra due



variabili o tra una variabile ed una costante.

Il **blocco di salto condizionato** ha la funzione di creare flussi di salto basati su condizioni definite dall'utente. Questo blocco possiede una funzione simile a quella di un blocco di salto nel senso che esegue un salto basandosi su una condizione. Tuttavia, la condizione non è solo per uscire da un ciclo ma anche per verificare condizioni composte da variabili arbitrarie.



Figura 2.6: Blocco di salto condizionato

Per aggiungere un blocco di salto condizionato si seleziona "Motion" - "Add condition branch block" dal menù, questo verrà aggiunto nella *Motion Area*. È possibile la modifica del nome del blocco e del flusso in uscita alla stessa maniera di un blocco posa; all'interno del blocco è visualizzata l'operazione che il programma compie durante l'esecuzione. Un blocco di salto condizionale salta quando la variabile utilizzata verifica la condizione. Nelle impostazioni di salto si può scegliere sia la variabile di confronto sia il tipo di condizione da verificare: maggiore del valore di soglia, minore del valore di soglia, controllo dei bit, uguale al valore di soglia, diverso dal valore di soglia, sempre vero e sempre falso.

Il **blocco di inizio** indica da dove comincia il movimento, ne può esistere solo uno e non può essere la destinazione del flusso di altri blocchi. Il blocco di inizio viene aggiunto automaticamente quando si crea un movimento e non può essere né aggiunto né eliminato.



Figura 2.7: Blocco di inizio e blocco di fine

Un **blocco di fine** può essere aggiunto al movimento scegliendo "Motion" - "Add end block" dal menù. Il blocco sarà aggiunto nella *Motion Area* dove si potranno collegare gli altri blocchi al blocco appena creato quando si vuole far terminare il

movimento. Il blocco di fine non ha flussi in uscita e come per il blocco di inizio non può essere rinominato.

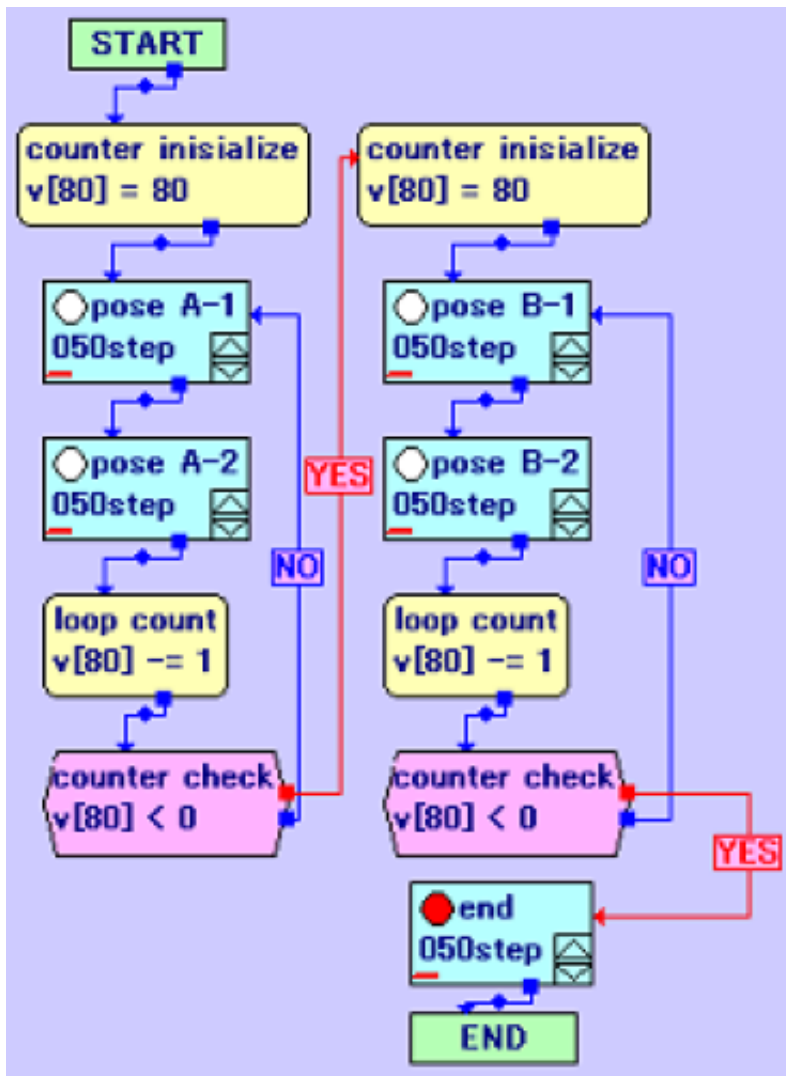


Figura 2.8: Esempio di schema a blocchi con due cicli

## 2.3 Analisi dei file di output

Quando un movimento viene creato con il RobovieMaker viene anche salvato in memoria. Il risultato di questa operazione è un file di testo costituito da un insieme di strutture, una per ogni blocco, con una particolare sintassi. Questa è formata da due parti principali che sono l'intestazione e la parte che contiene le informazioni: l'intestazione contiene l'etichetta del blocco, le sue coordinate all'interno della *Motion Area*, due campi di informazione per la freccia e due puntatori verso i blocchi successivi; se un blocco non ha un puntatore il campo contiene il valore -1. Questo primo pezzo di informazioni è comune per tutti i blocchi analizzati precedentemente

con l'aggiunta di un campo per indicare il tipo di flusso e di un campo puntatore per i blocchi di ciclo e di salto condizionato. Le informazioni contenute nella seconda parte del blocco sono diverse per ogni tipologia e verranno presentate di seguito.

```
_MOTION_INFO:[[Etichetta]]-[CoordinataX]-[CoordinataY]-[ParametroFreccia1]-[
ParametroFreccia2]-[Puntatore1]-[Puntatore2]
```

Il fatto che i movimenti siano registrati in un semplice file di testo dá la possibilità di poter modificare i valori dei parametri senza dover modificare il formato delle informazioni per creare dei blocchi con valori nuovi. Il software RobovieMaker è in grado di riconoscere la particolare sintassi con cui viene scritto il file ed è in grado di ricostruire il diagramma a blocchi associato. Questa funzione sarà il punto di partenza per creare nuove pose o interi movimenti senza utilizzare il RobovieMaker, ma usandolo come interprete tra il file di comandi per il robot e il robot stesso. In altre parole, con una applicazione esterna si crea un file di testo che verrà caricato nel RobovieMaker che a sua volta lo invierà al robot umanoide Robovie-X per l'esecuzione.

Ogni file salvato con il RobovieMaker contiene un'intestazione composta dalle istruzioni:

```
_MOTION_FORMAT:[ POSEDEF_V1R4 ]
_ENABLE_AXIS:[62]
```

dove la prima istruzione specifica il tipo di formato dei dati, mentre la seconda rappresenta il numero dei parametri del blocco posa. Bisogna tenere presente che i numeri che vengono assegnati ai blocchi sono in ordine di come questi vengono salvati sul file di output; inoltre la differenza tra la la singola parentesi quadra “[” o “]” e la doppia parentesi quadra “[[” o “[]” significa che nel secondo caso, se non è presente il dato, questo viene ignorato, mentre nel primo caso viene generato un errore.

### 2.3.1 Blocco Posa

Il blocco posa, oltre all'intestazione, contiene:

- un primo parametro che corrisponde al tempo di transizione che deve essere compreso tra 1 e 239;
- una lista di 62 valori esadecimali che stanno ad indicare la posizione dei cursori e di conseguenza la posizione dei servomotori o dei controlli sulla scheda di espansione. I parametri vanno dallo 0-esimo al 61-esimo, sono tutti indicati con 4 cifre, anche se queste dovessero essere tutte zero, e sono separati da una

virgola. È importante sottolineare una differenza tra i parametri che iniziano per 0x???? e 0X????: i primi sono valori modificabili in quanto rappresentano la posizione di un giunto o il valore di un sensore; i secondi sono valori non modificabili e quindi sempre costanti;

- un campo per inserire il percorso file audio da riprodurre;
- un numero binario a 30 cifre dalla 0-esima alla 29-esima che indica se un servomotore è acceso o spento dove 1 corrisponde a motore spento e 0 a motore acceso.

```
_MOTION_INFO: [[POSE 0]] - [50] - [101] - [6] - [6] - [3] - [-1]
_POSE: [50] - [0x0000,0x0aa5,0x3e97,0xf5c8,0x0000,0X0000,0x0000,0xf55b,0xc169,0x0a38,0
x0000,0X0000,0x0000,0X0000,0X0000,0x1d61,0x3680,0x0000,0x0000,0X0000,0X0000,0
xe29f,0xc980,0x0000,0X0000,0X0000,0x0000,0X0000,0X0000,0X0000,0x00c8,0
x0100,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0x0000,0
x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0X0000,0x0000,0
X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000] - [[]] - [00000000000000000000000000000000]
```

### 2.3.2 Blocco di ciclo

Il blocco contiene solo l'istruzione “\_OLDLOOP:” in aggiunta all'intestazione.

```
_MOTION_INFO: [[BREAK POINT]] - [50] - [165] - [16] - [78] - [5] - [2]
_OLDLOOP:
```

### 2.3.3 Blocco di calcolo

Il blocco di calcolo permette di eseguire delle operazioni tra una variabile ed una costante o tra due variabili. Nel primo caso l'istruzione sarà:

```
_VARCON: [Variabile che poi conterrà il risultato] - [Operazione] - [Costante]
```

mentre nel caso di due variabili si utilizza la seguente sintassi:

```
_VARVAR: [Variabile che poi conterrà il risultato] - [Operazione] - [Variabile di riferimento]
```

Le operazioni disponibili sono 5 e sono associate ad un identificativo che sarà quello che viene riportato nel file:

```
_MOTION_INFO: [[ CALCULATION ]] - [58] - [347] - [6] - [6] - [1] - [-1]
_VARCON: [1] - [0] - [3]
```

Identificativo	Operazione
0	Assegnazione
1	Addizione
2	Sottrazione
3	Moltiplicazione
4	Divisione

Tabella 2.1: Corrispondenza tra identificativo e operazione

### 2.3.4 Blocco di salto condizionato

Le variabili utilizzabili nel blocco di salto condizionato vanno dalla 0 alla 255. L'istruzione che va a completare l'intestazione del blocco è la seguente:

```
_MIF: - [Condizione] - [Variabile di riferimento] - [Valore di riferimento]
```

Nel campo [Variabile di riferimento] è possibile specificare delle opzioni avanzate selezionabili dalle proprietà del blocco in RobovieMaker.

Identificativo	Condizione
0	Non Definita
1	Maggiore
2	Minore
3	Controllo dei bit
4	Uguale
5	Diverso
6	Sempre Vero
7	Sempre Falso

Tabella 2.2: Corrispondenza tra identificativo e condizione

```
_MOTION_INFO: [[IF]] - [52] - [243] - [16] - [46] - [3] - [1]
_MIF: [2] - [1] - [3]
```

### 2.3.5 Blocco di inizio e blocco di fine

Per i blocchi di inizio e di fine vengono riportate le definizioni comprensive dell'intestazione. Si ricorda che il blocco di inizio non può essere puntato da nessun altro blocco e che il blocco di fine non può avere dei flussi in uscita.

```
_MOTION_INFO:[[]]-[50]-[10]-[6]-[0]-[1]-[-1]  
_STARTMOT:
```

```
_MOTION_INFO:[[ END ]]-[50]-[600]-[16]-[46]-[4]-[-1]  
_ENDMOT:
```

# Capitolo 3

## Flex & Bison

### 3.1 Introduzione

Per la realizzazione del traduttore dal linguaggio pseudo-naturale al file di comandi per il robot umanoide Robovie-X si è scelto di utilizzare gli strumenti Flex & Bison. Questi, solitamente usati in coppia, sono due strumenti che consentono di generare automaticamente il parser data la grammatica di un linguaggio e le espressioni regolari dei token. Flex e Bison non sono in grado di generare un compilatore completo: richiedono codice C/C++ di supporto per le parti non strettamente sintattiche. Entrambi sono dei generatori di codice ad alto livello: normalmente generano codice C che poi deve essere compilato insieme al codice di supporto scritto esplicitamente.

### 3.2 Flex: Analisi Lessicale

Flex genera programmi scritti in codice C atti ad elaborare sequenze di caratteri in ingresso. Questo strumento 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 quali sono specificate dal programmatore nel sorgente delle specifiche di Flex. Il codice prodotto, quindi, segmenta una sequenza di caratteri fornita in ingresso producendo un elemento per ogni sottosequenza che costituisce una stringa. Al riconoscimento di una stringa, viene eseguita una porzione di codice, definita 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 elemento terminale o token. I token sono la rappresentazione numerica delle parole chiave riconosciute che 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 Bison.

Il file di input in Flex è costituito da tre sezioni separate da una linea contenente solo “%%”:

```
definizioni
%%
regole
%%
codice utente
```

La sezione delle definizioni contiene le dichiarazioni delle definizioni di nomi semplici per semplificare le specifiche dello scanner e le dichiarazioni delle condizioni di inizio. Le definizioni dei nomi hanno la forma:

```
nome definizione
```

Il nome è una parola che inizia con una lettera o con un trattino basso (“\_”) seguiti da zero o da più lettere, cifre, “\_”, o “-” (trattino). L’inizio di una definizione viene considerato come il carattere che non sia uno spazio bianco mentre la fine corrisponde alla fine della riga. La definizione può essere successivamente essere riferita utilizzando il nome che in questo modo diventa una definizione. Per esempio,

```
DIGIT [0-9]
```

$\{\text{DIGIT}\}+“.”$   $\{\text{DIGIT}\}^*$  è equivalente a  $([0-9]+“.”$   $([0-9])^*$

La sezione delle regole di ingresso di Flex contiene una serie di regole della forma: lo schema (pattern), che non deve essere indentato, e l’azione (action), che deve iniziare sulla stessa linea.

Infine, la sezione del codice utente viene semplicemente copiata nel file `lex.yy.c`. La presenza di questa sezione è facoltativa e, se è mancante, le seconde %% nel file di input possono anche essere ignorate.

Gli schemi per le sequenze di ingresso vengono scritti utilizzando un insieme esteso di espressioni regolari.



Schema	Descrizione
x	riconosce il carattere 'x'
.	qualsiasi carattere (byte) ad eccezione di nuova riga
[xyz]	una classe di caratteri; questo schema riconosce 'x' o 'y' o 'z'
[abj-oZ]	una classe di caratteri con un intervallo; riconosce una 'a', una 'b', qualsiasi lettera da 'j' a 'o' oppure la 'Z'
[^A-Z]	una classe di caratteri negata; in questo caso riconosce ogni carattere escluse le lettere maiuscole
[^A-Z\n]	una classe di caratteri negata; in questo caso riconosce ogni carattere escluse le lettere maiuscole e del carattere di nuova riga
r*	riconosce 0 o più r dove r è un'espressione regolare
r+	riconosce una o più r
r?	riconosce zero o più r (r opzionale)
r{2,5}	riconosce qualsiasi numero di r da 2 a 5
r{2,}	riconosce 2 o più r
r{4}	riconosce esattamente 4 r
{name}	espansione della definizione di nome
"[xyz]\\"foo"	riconosce la stringa [xyz]"foo"
\0	riconosce il carattere NULL (codice ASCII 0)
\123	riconosce il carattere con valore in base ottale 123
\x2a	riconosce il carattere con valore in base esadecimale 2a
rs	riconosce l'espressione regolare r seguita dall'espressione regolare s (conocatenazione)
r s	riconosce r oppure s
r/s	riconosce r solo se è seguita da s
^r	riconosce una r solo se questa compare all'inizio di una linea
r\$	riconosce una r solo se questa compare alla fine di una linea
«EOF»	riconosce end-of-file

Tabella 3.1: Tipi di schema e descrizione

Le espressioni regolari riportate nella tabella qui sopra sono riportate secondo un ordine di precedenza da quella con maggiore precedenza all'inizio a quella con la minore alla fine. Si ricorda che all'interno di una classe di caratteri, tutte le espressioni regolari perdono il loro significato speciale ad eccezione di '\', '-', ']' e all'inizio della classe '^'.

Quando lo scanner appena generato è in esecuzione, analizza gli ingressi andando a cercare le stringhe che potrebbero corrispondere ad uno dei suoi schemi. Se vengono trovati più schemi a cui corrisponde una stringa, il riconoscitore sceglie quello in cui c'è più testo che coincide. Nel caso in cui ci siano più corrispondenze con la stessa lunghezza di testo allora lo scanner sceglie la prima che incontra nella lista delle regole. Una volta che la corrispondenza è stata determinata, il token è disponibile nel puntatore di caratteri generale `yytext` e la sua lunghezza nella variabile globale intera `yytext`. L'azione corrispondente allo schema selezionato viene eseguita e lo scanner è pronto ad analizzare la prossima stringa. Nel caso in cui nessuno schema sia stato scelto, viene eseguita la regola di default e il procedimento continua.

Ogni schema, in una regola, ha un'azione corrispondente che può essere una qualsiasi istruzione del linguaggio C. Lo schema termina con il primo carattere di spazio bianco che non sia di escape. Se non è prevista un'azione, quando lo schema è stato riconosciuto la stringa in ingresso viene semplicemente scartata. Nel caso che un'azione cominci con il carattere `'{'` questa continua fino a quando non si trova il carattere `'}'` corrispondente. Un'azione che consiste solamente in una barra verticale indica che l'azione corrispondente è la medesima della regola riportata successivamente.

L'output restituito da Flex è il file `lex.yy.c` che contiene la routine di scanner `yylex()`, un numero arbitrario di tabelle usate dalla routine per il riconoscimento dei token, un numero di routine ausiliarie e di macro. Ogniqualvolta `yylex()` viene invocato, esso scandisce i token dal file di input globale `yyin` (di default è impostato lo `stdin`). Il metodo continua finché viene raggiunto la fine-del-file o se qualche azione esegue una istruzione di `return`.

Tra i vari scopi per cui è stato progettato Flex il più importante è il suo utilizzo insieme all'analizzatore sintattico Bison che si aspetta di chiamare una routine chiamata `yylex()` per trovare il prossimo token da analizzare. Per utilizzare Flex insieme a Bison bisogna specificare l'opzione `-d` sulla riga di comando quando si compila il file dell'analizzatore sintattico per generare il file `y.tab.h` che contiene la definizione di tutti i token che possono apparire in input a Bison.

### 3.3 Bison: Analisi Sintattica

Bison utilizza regole grammaticali fornite dal programmatore allo scopo di analizzare, nella fase di analisi sintattica, i token restituiti da Flex. Questi devono essere organizzati secondo le regole di struttura dell'ingresso che prendono il nome di regole grammaticali; quando una di queste regole viene usata per ridurre l'ingresso viene invocata un'azione, cioè viene invocato il codice utente relativo alla regola.

Una struttura riconosciuta dall'analizzatore lessicale è chiamata simbolo terminale, mentre una struttura riconosciuta dal parser è chiamata simbolo non-terminale. Nella grammatica formale di Bison, un simbolo non terminale viene rappresentato in ingresso come un identificatore e per convenzione viene scritto in minuscolo mentre la rappresentazione in Bison per un simbolo terminale viene detta token. Per convenzione, questi identificatori devono essere scritti in maiuscolo per distinguerli da quelli non-terminali.

```
statement: RETURN expression ';'
        ;
```

Una grammatica formale seleziona i token solo per la loro classificazione: ad esempio, se una regola cita il simbolo terminale “costante intera”, significa che qualsiasi costante intera è grammaticalmente valida in quella posizione. Il valore esatto della costante è irrilevante per come viene analizzato l'input. Ma il valore preciso è molto importante per ciò che significa l'ingresso una volta che viene analizzato. Un compilatore è inutile se non riesce a distinguere tra il 4, 1 e 3.989 come costanti nel programma! Pertanto, ogni sequenza in ingresso valida in una grammatica Bison ha sia un tipo di token che un valore semantico.

Per essere utile, un programma deve fare di più che la semplice analisi di un ingresso, ma deve anche produrre un output in base all'input. In una grammatica Bison, una regola può contenere un'azione costituita da istruzioni in linguaggio C. Ogni volta che il parser riconosce una corrispondenza di tale regola l'azione viene eseguita. Ad esempio, questa è una regola che dice che un'espressione può essere la somma di due sottoespressioni:

```
expression: expression '+' expression { $$ = $1 + $3; }
        ;
```

L'azione dice come produrre il valore semantico dell'espressione somma dai valori delle due sottoespressioni.

Il file di input in Bison è un file di grammatica Bison la cui forma generale è la seguente:

```
%{
prologo
}%
dichiarazioni
%%
regole grammaticali
%%
epilogo
```

I simboli `'%%'`, `'%{'` e `'%}'` sono la punteggiatura che compare in ogni file di grammatica Bison per separare le sezioni. Il prologo può definire i tipi e le variabili utilizzate nelle azioni. È inoltre possibile utilizzare comandi per il preprocessore per definire le macro utilizzate, e di utilizzare pseudo-istruzioni `#include` per includere header file. Sempre nel prologo è necessario dichiarare l'analizzatore lessicale `yylex` e la stampante per gli errori `yyerror`. Le dichiarazioni dichiarano i nomi dei simboli terminali e non terminali, e possono anche descrivere le precedenze tra gli operatori e i tipi di dati dei valori semantici dei vari simboli. Le regole di grammatica definiscono le modalità per la costruzione di ogni simbolo non terminale a partire dalle sue parti. L'epilogo può contenere un codice che si desidera utilizzare. Spesso le definizioni di funzioni dichiarate nel prologo vanno messe qui. In un programma semplice, tutto il resto del programma può essere scritto qui.

# Capitolo 4

## Interpretazione del linguaggio pseudo-naturale

### 4.1 Introduzione

La scelta di realizzare un traduttore per i movimenti del robot umanoide Robovie-X nasce da due riflessioni principali: la prima riguarda la difficoltà di creare movimenti complessi utilizzando il software RobovieMaker mentre la seconda verte sempre sui problemi del programma fornito dall Vstone, ma riguarda la difficoltà di poter operare per un utente poco esperto. Per questi due motivi si è scelto di realizzare un traduttore che sia il più semplice possibile, ma allo stesso tempo modulare in modo da poter creare le sequenze di pose più complesse. A sostegno di queste scelte si è deciso di optare per l'uso della lingua inglese.

Gli altri vantaggi che discendono da questa realizzazione sono la possibilità di far effettuare al robot movimenti precisi, cioè di muovere gli arti ad angolazioni inserite dall'utente in entrambi i sensi senza dover prestare molta attenzione alla posizione dei cursori di RobovieMaker. Inoltre, la possibilità di creare cicli annidati non diventa più un problema di contatori e blocchi di calcolo in quanto l'inserimento di questi ultimi viene implementato direttamente dal traduttore. Come ultima considerazione, ma forse più importante di tutte le precedenti, si può affermare che l'applicazione realizzata non porta a delle limitazioni delle potenzialità del software RobovieMaker, anzi, punta ad essere la base per uno sviluppo futuro ad esempio verso un riconoscitore vocale del linguaggio naturale.

### 4.2 I comandi

Prima di procedere con la definizione dei comandi che il traduttore è in grado di conoscere diventa fondamentale conoscere quali parole chiave costituiscono un input

valido: il nome delle parti del corpo del robot prima di tutto e poi tutte le parole chiave di controllo.

Il robot umanoide Robovie-X viene scomposto in 17 giunti in cui il primo corrisponde alla testa, mentre i rimanenti 16 sono costituiti da due blocchi da 8 perché simmetrici tra destra e sinistra. Qui sotto vengono riportati i nomi specifici per ogni servomotore con la relativa traduzione in lingua inglese poiché sarà quella utilizzata per l'implementazione.

testa	:	head
spalla	:	shoulder
braccio	:	arm
polso	:	elbow
anca	:	hip
coscia	:	thigh
ginocchio	:	knee
polpaccio	:	calf
caviglia	:	ankle

Di norma, la prima parola con cui inizia una frase, che il traduttore è in grado di decodificare, indica quale azione deve eseguire il robot oppure se viene richiesta la modifica di un parametro. Nella pratica, il comando “muovi”, in inglese “move”, e il comando “ruota” o “gira”, in inglese “turn” coincidono nel senso che comunque utilizzando una oppure l'altra il risultato non cambia. Altre parole chiave sono “aumenta”, “diminuisci” e “imposta” le quali sono usate prevalentemente per l'impostazione dei parametri, ad esempio il guadagno dei giroscopi, ed in via del tutto eccezionale anche per la luminosità dei led del Robovie-X.

muovi	:	move
ruota	:	turn
<hr/>		
aumenta	:	increase
diminuisci	:	decrease
imposta	:	set

Un ultimo aspetto, ma non meno importante, riguarda la direzione verso il quale un arto del Robovie-X deve spostarsi: verso l'interno, verso l'esterno, in avanti e indietro. Seguendo la sintassi della lingua inglese questa specifica di movimento viene inserita come ultimo parametro della frase. Pertanto i vocaboli utilizzati sono:

dentro	:	inside
verso l'interno	:	inward
<hr/>		
fuori	:	outside
verso l'esterno	:	outward
<hr/>		
avanti	:	front
in avanti	:	forward
<hr/>		
dietro	:	back
indietro	:	backward
<hr/>		
destra	:	right
verso destra	:	rightward
<hr/>		
sinistra	:	left
verso sinistra	:	leftward

Per unire le parole chiave finora analizzate in modo da produrre una frase di senso compiuto sono necessarie delle ulteriori specifiche:

- Lato: il robot è formato da 16 servomotori simmetrici tra loro posizionati 8 sul lato destro e altrettanti sul lato sinistro. Diventa di fondamentale importanza, al fine di identificare univocamente ogni singolo giunto, determinare su quale lato si trova il servomotore che si intende azionare.
- Preposizione: si è scelto di effettuare una distinzione tra la preposizione “di” e la preposizione “a”, che sono le uniche utilizzate. Questo serve per poter capire se l’utente vuole posizionare l’arto ad una determinata angolazione partendo dalla posizione attuale oppure se l’arto deve essere spostato ad un angolo preciso indipendentemente dalla posizione corrente che occupa.
- Numero: rappresenta il valore numerico all’interno di un comando.
- Gradi: indica l’unità di misura del valore inserito nel campo Numero. Di norma rappresenta i gradi sessagesimali, ma per quanto riguarda la taratura della luminosità dei LED, questo viene sostituito dal simbolo di percentuale, %.

Ora, mettendo insieme tutte le parole chiave sopra citate si può generare un semplice comando:

```
azione lato arto preposizione numero gradi direzione
```

Quanto detto finora non vale per la testa e per i due occhi del robot umanoide Robovie-X per due motivi: il primo riguarda la testa e banalmente si nota che non

avrebbe senso specificare una testa “destra” piuttosto che una testa “sinistra”; il secondo, invece, si riferisce agli occhi relativamente al concetto di luminosità.

Per risolvere il primo problema si è dovuto eliminare il parametro lato andando di fatti a modificare la sintassi della frase di comando che così diventa:

```
azione arto preposizione numero gradi direzione
```

Nella frase prevista per inviare un comando al robot relativamente agli occhi è stato inserito il concetto di luminosità, *brightness* in inglese, e di illuminazione degli occhi in percentuale: se l’utente imposta la luminosità al valore zero il led sarà spento, mentre al valore di 100% corrisponde la massima luminosità. Da queste osservazioni la sintassi sviluppata è:

```
azione lato arto luminosita preposizione numero %
```

Infine, viene fatta un’analisi sulle istruzioni per l’impostazione dei valori dei guadagni dei giroscopi e dei controlli. Questi vengono utilizzati principalmente per lo sviluppo di movimenti complessi ed avanzati dove i semplici comandi non sono sufficienti.

```
gyrox
gyroy
shootanalog
pitchmove
rollstep
rollrstep
legpace
turnctr
```

Come si è fatto per gli arti simmetrici e non simmetrici sono state implementate due sintassi: una con il parametro lato e una senza, rispettivamente; nella frase che rappresenta il comando si sono utilizzati i medesimi comandi per l’azione che sono stati usati per gli occhi, cioè aumenta, diminuisci e imposta, e un ultimo parametro che indica se la modifica riguarda un controller oppure se è un guadagno.

```
azione (lato) sensore tipo preposizione numero
```

### 4.3 Suoni

Una ulteriore analisi è stata fatta sulla possibilità di riprodurre i suoni, caricati nella memoria ROM del robot utilizzando il traduttore. Utilizzando la particolare sintassi che viene riportata qui sotto, l’utente può scegliere quale traccia audio far riprodurre



al robot durante una posa indicando la cartella che contiene il file audio e il nome del file audio stesso dopo aver specificato eventualmente altre azioni utilizzando la parola chiave “playing”.

```
... playing cartella\file
```

## 4.4 Velocità dei movimenti

È anche possibile, come nel RobovieMaker, impostare la velocità di transizione per un blocco posa. Si ricorda che prima bisognava premere sulle frecce presenti sul blocco posa all'interno della *Motion Area* per impostare la velocità desiderata, accusando una perdita di tempo se il nuovo da impostare era molto lontano dal valore predefinito. Ora, invece, è solamente necessario specificare al termine di una istruzione di posa la seguente sintassi:

```
in integer milliseconds
```

Il codice all'interno del traduttore si preoccuperà del cambiamento di base per renderlo compatibile con il valore di clock della scheda che controlla il robot.

## 4.5 Più azioni contemporaneamente

Un altro aspetto interessante del nuovo software traduttore riguarda la possibilità di poter far muovere al robot più di un giunto contemporaneamente nello stesso blocco posa. Questa operazione risulta essere più facile rispetto a prima poiché basta elencare le istruzioni da compiere come un normale elenco: tra due azioni contemporanee è necessario inserire la parola AND oppure il simbolo della virgola ','. Dopo l'ultima istruzione però viene richiesto l'inserimento di un altro carattere speciale, il punto '.', in modo che il traduttore possa capire quando terminano le istruzioni relative ad un blocco in modo e quando iniziano quelle della posa successiva.

## 4.6 I cicli

La realizzazione delle ripetizioni di sequenze di blocchi posa cambia sostanzialmente con il traduttore rispetto al software RobovieMaker: l'utente non deve preoccuparsi né dei blocchi di calcolo, né dei blocchi di ciclo e tantomeno dei blocchi di salto condizionato. Queste faccende, infatti, vengono realizzate all'interno del traduttore stesso. È anche prevista la possibilità di avere più ripetizioni una dentro l'altra in modo da formare i cosiddetti cicli annidati. La sintassi prevede di digitare il comando REPEAT integer TIMES (ripeti intero volte) per creare il punto di inizio

della ripetizione, successivamente si inseriscono tutte le pose che si vuole far ripetere al Robovie-X o un qualsiasi altro comando ed infine con il comando STOP si termina il ciclo. Schematicamente si ottiene la seguente struttura:

```
repeat integer times
blocchi di istruzioni
stop
```

## 4.7 Creare comandi personalizzati

Si da all'utente anche la possibilità di creare comandi personalizzati ( "macro" ), associando a una parola chiave una sequenza di istruzioni dello stesso tipo di quelle descritte fino ad ora. Questa funzione si rivela molto utile in quanto rende il programma ancora più flessibile.

Facendo un esempio, con questa funzione l'utente può facilmente definire il comando *saluta*.

Per definire e realizzare una "macro" si usa il comando (al posto di macroname si sostituisce la parola chiave che da il nome alla "macro") :

```
save macro macroname
```

Successivamente si inseriscono le istruzioni che dovranno essere eseguite quando la "macro" viene richiamata. Per terminare l'input è sufficiente premere due volte INVIO lasciando una riga vuota.

L'istruzione che richiama una "macro" è la seguente:

```
execute macroname
```

### 4.7.1 Un esempio di "macro"

vuole creare il comando NO (muove la testa a sinistra e destra):

```
save macro as NO #Inizializzazione
```

Si definiscono le istruzioni che compongono la "macro":

```
repeat 3 times #Definizione
move head at 45 degrees rightward in 1500 ms.
move head at 45 degrees leftward in 1500 ms.
stop
move head of 45 degrees leftward in 1500 ms.
```

Esecuzione del comando appena definito:

```
execute macro NO #Esecuzione
```

## 4.8 Movimenti Predefiniti

Il traduttore è in grado di riconoscere anche dei comandi che non sono formati solo da singole istruzioni, ma riesce a decodificare anche delle sequenze di pose, ovvero dei movimenti. L'inserimento del comando viene gestito come una normale operazione: è dunque il programma che si preoccupa di elaborare e fornire in output i blocchi corrispondenti alla richiesta. La sintassi della frase con cui si richiede il comando dipende dalla richiesta che viene fatta: fare 3 passi avanti necessita di 4 parametri, mentre l'istruzione sbadiglia consta solo di una parola. Un aspetto che è degno di nota riguarda la possibilità di inserire i movimenti già inseriti all'interno del traduttore all'interno di singole istruzioni ottenendo come risultato l'unione di tutte le cose. I movimenti che sono già caricati nel traduttore sono i seguenti:

fai intero passi avanti	:	walk number step ahead
fai intero passi indietro	:	walk number step back
fai intero passi verso destra	:	walk number step rightward
fai intero passi verso sinistra	:	walk number step leftward
balla	:	dance
inchino	:	greetings
sbadiglia	:	yawn
alzati da pancia in su	:	getting up from face up
alzati da pancia in giù	:	getting up from face down
capirola avanti	:	front flip
capirola indietro	:	back flip

## 4.9 Attivare la modalità “real-time”

Attivando la modalità “real-time”, se il robot è connesso con il cavo USB, le istruzioni in input al traduttore vengo eseguite in tempo reale dall'umanoide.

Per avviare questa modalità operativa è sufficiente collegare il Robovie-X e assicurarsi che l'interruttore sia nella posizione ON e eseguire il comando:

```
online
```

Prima di chiudere il programma e scollegare l'umanoide è opportuno terminare la connessione con il comando:

```
offline
```

## 4.10 Un esempio

Qui sotto viene riportato un esempio complesso di input che racchiude alcune le funzionalità del traduttore e, nella pagina successiva, il relativo file di comandi generato.

```
turn right arm at 180 degrees outward playing Other\Hello.  
reset  
repeat 3 times  
move head at 45 degrees rightward in 1500 ms.  
move head of 90 degrees leftward in 1500 ms.  
stop  
repeat 2 times  
greetings  
stop  
exit
```

```

_MOTION_FORMAT: [ POSEDEF_V1R4 ]
_ENABLE_AXIS: [62]

_MOTION_INFO: [[]] - [50] - [10] - [6] - [0] - [1] - [-1]
_STARTMOT:

_MOTION_INFO: [[POSE1]] - [50] - [43] - [6] - [6] - [2] - [-1]
_POSE: [50] - [0x0000, 0x0aa5, 0x3e97, 0xf5c8, 0x0000, 0X0000, 0x0000, 0xf55b, 0xc169, 0x0a38, 0
x0000, 0X0000, 0x0000, 0X0000, 0X0000, 0x1d61, 0x3680, 0x0000, 0x0000, 0X0000, 0X0000, 0
xe29f, 0xc980, 0x0000, 0X0000, 0X0000, 0x0000, 0X0000, 0X0000, 0X0000, 0x00c8, 0
x0100, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0x0000, 0
x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0X0000, 0x0000, 0
X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000
] - [[]] - [00000000000000000000000000000000]

_MOTION_INFO: [[POSE2]] - [50] - [100] - [6] - [6] - [3] - [-1]
_POSE: [50] - [0x0000, 0x0aa5, 0x3e97, 0xf5c8, 0x0000, 0X0000, 0x0000, 0xf55b, 0xc169, 0x0a38, 0
x0000, 0X0000, 0x0000, 0X0000, 0X0000, 0x1d61, 0x9d38, 0x0000, 0x0000, 0X0000, 0X0000, 0
xe29f, 0xc980, 0x0000, 0X0000, 0X0000, 0x0000, 0X0000, 0X0000, 0X0000, 0x00c8, 0
x0100, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0x0000, 0
x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0X0000, 0x0000, 0
X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000] - [[X Voice Files\Other\
Hello.wav]] - [00000000000000000000000000000000]

_MOTION_INFO: [[CALCULATION3]] - [50] - [150] - [6] - [6] - [4] - [-1]
_VARCON: [0] - [80] - [3]

_MOTION_INFO: [[POSE4]] - [50] - [200] - [6] - [6] - [5] - [-1]
_POSE: [89] - [0x0000, 0x0aa5, 0x3e97, 0xf5c8, 0x0000, 0X0000, 0x0000, 0xf55b, 0xc169, 0x0a38, 0
x0000, 0X0000, 0x0000, 0X0000, 0X0000, 0x1d61, 0x3680, 0x0000, 0x0000, 0X0000, 0X0000, 0
xe29f, 0xc980, 0x0000, 0X0000, 0X0000, 0x2652, 0X0000, 0X0000, 0X0000, 0x00c8, 0
x0100, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0x0000, 0
x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0
X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000
] - [[]] - [00000000000000000000000000000000]

_MOTION_INFO: [[POSE5]] - [50] - [250] - [6] - [6] - [6] - [-1]
_POSE: [89] - [0x0000, 0x0aa5, 0x3e97, 0xf5c8, 0x0000, 0X0000, 0x0000, 0xf55b, 0xc169, 0x0a38, 0
x0000, 0X0000, 0x0000, 0X0000, 0X0000, 0x1d61, 0x3680, 0x0000, 0x0000, 0X0000, 0X0000, 0
xe29f, 0xc980, 0x0000, 0X0000, 0X0000, 0xd9ae, 0X0000, 0X0000, 0X0000, 0x00c8, 0
x0100, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0x0000, 0
x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0X0000, 0x0000, 0
X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000
] - [[]] - [00000000000000000000000000000000]

_MOTION_INFO: [[CALCULATION6]] - [50] - [300] - [6] - [6] - [7] - [-1]
_VARCON: [2] - [80] - [1]

_MOTION_INFO: [[IF7]] - [50] - [350] - [6] - [6] - [4] - [8]
_MIF: [4] - [80] - [0]

_MOTION_INFO: [[CALCULATION8]] - [50] - [400] - [6] - [6] - [9] - [-1]
_VARCON: [0] - [80] - [2]

_MOTION_INFO: [[POSE9]] - [50] - [450] - [6] - [6] - [10] - [-1]
_POSE: [10] - [0x0000, 0x0aa5, 0x3e97, 0xf5c8, 0x0000, 0X0000, 0x0000, 0xf55b, 0xc169, 0x0a38, 0
x0000, 0X0000, 0x0000, 0X0000, 0X0000, 0x1d61, 0x3680, 0x0000, 0x0000, 0X0000, 0X0000, 0
xe29f, 0xc980, 0x0000, 0X0000, 0X0000, 0x0000, 0X0000, 0X0000, 0X0000, 0x00c8, 0

```

```

x0100,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0x0000,0
x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0X0000,0x0000,0
X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000
]- [[] - [00000000000000000000000000000000]

_MOTION_INFO: [[POSE10]] - [50] - [500] - [6] - [6] - [11] - [-1]
_POSE: [50] - [0x0000,0x0ee7,0x3e97,0xfa0a,0x0000,0X0000,0x0000,0xf119,0xc169,0x05f6,0
x0000,0X0000,0x0000,0X0000,0X0000,0x1d61,0x2fb0,0xd34b,0x0000,0X0000,0X0000,0
xe29f,0xd050,0x0000,0X0000,0X0000,0x0000,0X0000,0X0000,0X0000,0x0000,0x00c8,0
x0100,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0x0000,0
x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0X0000,0x0000,0
X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000
]- [[] - [00000000000000000000000000000000]

_MOTION_INFO: [[POSE11]] - [50] - [550] - [6] - [6] - [12] - [-1]
_POSE: [50] - [0x0000,0x00da,0x3e97,0xcce8,0x0000,0X0000,0x0000,0xff26,0xc169,0x3318,0
x0000,0X0000,0x0000,0X0000,0X0000,0xf99d,0x2fb0,0xc4d1,0x0000,0X0000,0X0000,0
xe29f,0xd050,0x0000,0X0000,0X0000,0x0000,0X0000,0X0000,0X0000,0x0000,0x00c8,0
x0100,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0x0000,0
x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0X0000,0x0000,0
X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000]- [[X Voice Files\Other\
Hello.wav]] - [00000000000000000000000000000000]

_MOTION_INFO: [[POSE12]] - [50] - [600] - [6] - [6] - [13] - [-1]
_POSE: [50] - [0x0000,0x0589,0x3e97,0xe158,0x0000,0X0000,0x0000,0xfa77,0xc169,0x1ea8,0
x0000,0X0000,0x0000,0X0000,0X0000,0x1b40,0x2fb0,0xc4d1,0x0000,0X0000,0X0000,0
xe29f,0xd050,0x0000,0X0000,0X0000,0x0000,0X0000,0X0000,0X0000,0x0000,0x00c8,0
x0100,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0x0000,0
x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0X0000,0x0000,0
X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000
]- [[] - [00000000000000000000000000000000]

_MOTION_INFO: [[POSE13]] - [50] - [650] - [6] - [6] - [14] - [-1]
_POSE: [50] - [0x0000,0x0aa5,0x3e97,0xf5c8,0x0000,0X0000,0x0000,0xf55b,0xc169,0x0a38,0
x0000,0X0000,0x0000,0X0000,0X0000,0x1d61,0x3680,0x0000,0x0000,0X0000,0X0000,0
xe29f,0xc980,0x0000,0X0000,0X0000,0x0000,0X0000,0X0000,0X0000,0x0000,0x00c8,0
x0100,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0x0000,0
x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0x0000,0X0000,0x0000,0
X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000,0X0000
]- [[] - [00000000000000000000000000000000]

_MOTION_INFO: [[CALCULATION14]] - [50] - [700] - [6] - [6] - [15] - [-1]
_VARCON: [2] - [80] - [1]

_MOTION_INFO: [[IF15]] - [50] - [750] - [6] - [6] - [9] - [16]
_MIF: [4] - [80] - [0]

_MOTION_INFO: [[ END ]] - [50] - [800] - [16] - [46] - [-1] - [-1]
_ENDMOT:

```

## 4.11 Tutti i comandi

In questa sottosezione sono riportati, in lingua inglese, tutti i comandi che il traduttore è in grado di riconoscere:

- Testa:  
move head of 30 degrees leftward playing Other\Explosion  
turn head at 60 degrees rightward in 3000 ms
- Occhio:  
set left eye brightness at 90 %  
increase right eye brightness of 10 % and decrease left eye brightness of 30%
- Spalla:  
repeat 5 times move left shoulder of 45 degrees forward  
turn right shoulder at 15 degrees backward stop
- Braccio:  
move left arm of 45 degrees inward  
turn right arm at 15 degrees outward
- Polso:  
move left elbow of 45 degrees inward  
turn right elbow at 15 degrees outward
- Anca:  
move left hip of 45 degrees inward  
turn right hip at 15 degrees outward
- Coscia:  
move left thigh of 45 degrees forward  
turn right thigh at 15 degrees backward
- Ginocchio:  
move left knee of 45 degrees forward  
turn right knee at 15 degrees backward
- Polpaccio:  
move left calf of 45 degrees forward  
turn right calf at 15 degrees backward
- Caviglia:  
move left ankle of 45 degrees inward  
turn right ankle at 15 degrees outward
- Giroscopio:  
set gyrox gain at 200  
increase gyroy gain of 10  
decrease gyrox of 30

- Altri Sensori:
  - set shootanalog control at 45
  - increase shootanalog control of 10
  - decrease left pitchmove gain of 545
  - set right pitchmove gain at 30411
  - increase left rollstep gain of 4578
  - decrease right rollstep gain of 545
  - set left rollstep gain at 30411
  - increase right rollstep gain of 4578
  - decrease left legpace gain of 545
  - set right legpace gain at 30411
  - increase left turnctrl control of 4578
  - decrease right turnctrl control of 545
  
- Movimenti:
  - walk number step ahead
  - walk number step back
  - walk number step rightward
  - walk number step leftward
  - dance
  - greetings
  - yawn
  - getting up from face up
  - getting up from face down
  - front flip
  - back flip
  
- Macro
  - save macro macroname
  - execute macro macroname
  
- Modalità real-time
  - online
  - offline
  
- Azzera:
  - reset
  
- Fine:
  - exit



# Capitolo 5

## Controllo in tempo reale

### 5.1 Introduzione

In seguito alla realizzazione del traduttore si è scelto di aggiungere la possibilità di controllare il robot in tempo reale, direttamente dal traduttore e mediante il linguaggio definito nel capitolo precedenti. Questa funzione fornisce diversi vantaggi: avendo la possibilità di vedere l'effetto delle istruzioni illustrate nel capitolo precedente, senza passare per il RobovieMaker 2, la realizzazione dei file di comandi risulta notevolmente facilitata.

### 5.2 Analisi del protocollo di comunicazione

Le funzioni utilizzate sono state ricavate mediante un processo di “reverse-engineering” disassemblando e studiando del codice della libreria rclib.dll utilizzata del programma Robovie Maker per il robot VS-RC003, umanoide della generazione precedente al Robovie-X.

Le funzioni ricavate per il VS-RC003, sono state adattate e rese compatibili al Robovie-X, rendendo possibile l'implementazione della parte traduttore che comunica con il robot (contenuta nel file rcCom.c). Tuttavia per capire cosa inviare al robot è stata necessaria una fase di sniffing e analisi dei dati scambiati tra il Robovie-X e il software RobovieMaker2.

L'analisi dei pacchetti catturati ha permesso di comprendere come sono strutturate le informazioni in questi ultimi e di conseguenza di sviluppare le funzioni che preparano i pacchetti in modo che il robot possa essere comandato. Questa parte di codice è stata implementata nel file OnlineMode.c.

Si elenca ora le quattro fasi principali in cui avviene la comunicazione/comando dell'umanoide:

- Apertura della connessione tra Windows e umanoide;
- Inizializzazione dell'umanoide e accensione dei servomotori;
- Creazione e invio dei pacchetti contenenti le istruzioni per il Robovie-x;
- Spegnimento e chiusura della connessione con il robot.

### 5.3 Connessione al Robovie-X

Windows rileva il dispositivo come HID (Human Interface Device). Gli header che contengono i prototipi delle funzioni necessarie per interagire con i dispositivi HID sono `hidsdi.h` e `setupapi.h`, le relative librerie statiche sono `libhid` e `libsetupapi`.

La connessione si effettua con la chiamata della funzione `BOOL rcOpen()` che ha il compito di individuare il robot tra i vari HID collegati al pc e di aprire la connessione creando un Handle.

### 5.4 Scambiare informazioni con l'umanoide

La comunicazione avviene attraverso dei blocchi composti da pacchetti della dimensione standard di 64 Byte, se il blocco da è più grande, si divide l'informazione in più pacchetti fino a un massimo di 5.

È importante far notare che i byte all'interno del pacchetto devono avere una codifica Little-Endian, quindi se si decide di inviare il valore esadecimale `0xXXYY` all'interno del pacchetto deve essere scritto nell'ordine `0xYY 0xXX` in due byte consecutivi.

00	01	02	03	04	05	06	07
08	09	0A	0B	0C	0D	0E	0F
10	11	12	13	14	15	16	17
18	19	1A	1B	1C	1D	1E	1F
20	21	22	23	24	25	26	27
28	29	2A	2B	2C	2D	2E	2F
30	31	32	33	34	35	36	37
38	39	3A	3B	3C	3D	3E	3F

Tabella 5.1: Pacchetto generico da 64 Byte, ogni cella contiene 8bit





### 5.5.3 Esecuzione di una posa

Per far muovere il robot deve essere inviata una sequenza di 3 pacchetti contenente informazioni sulla posizione in cui devono portarsi ogni motore rispetto a quella iniziale. Questo offset è il risultato delle conversione da gradi a esadecimale con la funzione *degToOffset()* descritta nel capitolo successivo.

0x55	0x03	0xE0	0x01	0x7C	0x00	<b>J17</b>	<b>J17</b>
<b>J15</b>	<b>J15</b>	<b>J13</b>	<b>J13</b>	<b>J11</b>	<b>J11</b>	<b>J9</b>	<b>J9</b>
0x00	0x00	<b>J18</b>	<b>J18</b>	<b>J16</b>	<b>J16</b>	<b>J14</b>	<b>J14</b>
<b>J12</b>	<b>J12</b>	<b>J10</b>	<b>J10</b>	0x00	0x00	<b>J2</b>	<b>J2</b>
0x00	0x00	0x00	0x00	<b>J7</b>	<b>J7</b>	<b>J5</b>	<b>J5</b>
<b>J3</b>	<b>J3</b>	<b>J1</b>	<b>J1</b>	0x00	0x00	0x00	0x00
<b>J8</b>	<b>J8</b>	<b>J6</b>	<b>J6</b>	<b>J4</b>	<b>J4</b>	0x00	0x00
0x00	0x00	<b>J0</b>	<b>J0</b>	0x00	0x00	0x00	0x00

Tabella 5.3: Primo pacchetto

0x55	0x13	0x00	0x00	<b>J20</b>	0x00	0xC8	0x00
0x00	0x01	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Tabella 5.4: Secondo pacchetto

0x55	0x23	0x00	0x00	0x00	0x00	0x80	0x00
0x04	0x00	0x00	0x00	0x00	0x80	0x00	0x02
<b>J21</b>	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Tabella 5.5: Terzo pacchetto

I byte alla posizione **JX** devono essere contere la posizione a cui si vuole impostare il servomotore (o altro sensore), rispetto allo zero, **JX** secondo lo schema riportato in figura 5.1 e secondo la legenda in tabella 5.6.

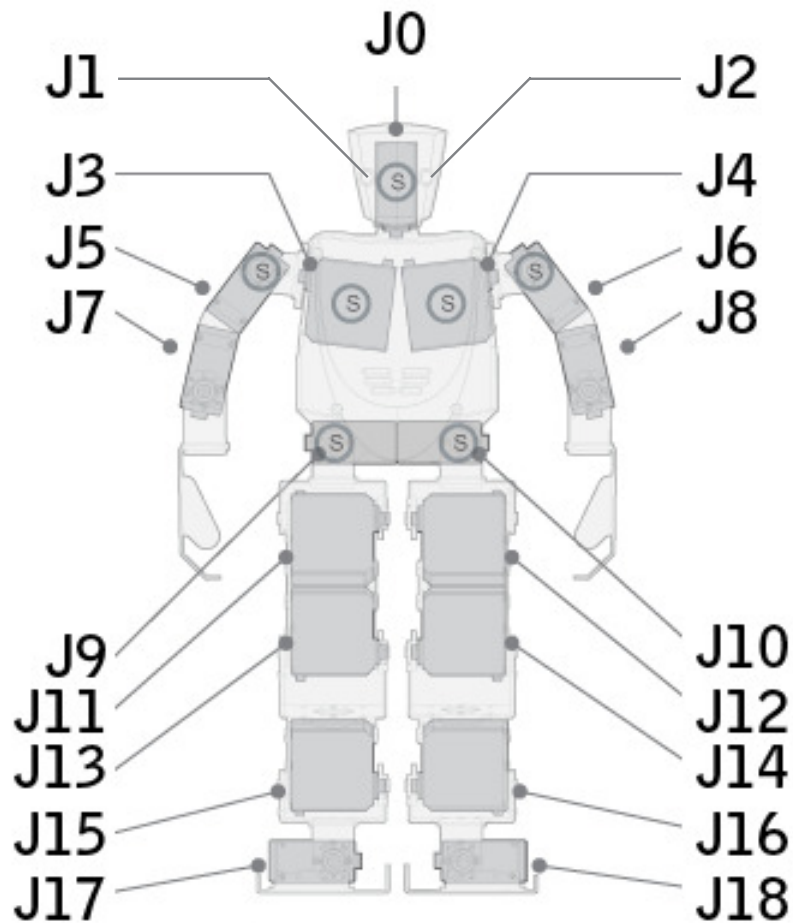


Figura 5.1: Mappatura dei servomotori

I valori da **J1** a **J18** vanno impostati con i gradi a cui si vuole portare ogni servomotore effettuando la conversione da gradi sessagesimali a esadecimale, con la formula che verrà spiegata nel capitolo successivo. Si fa notare che la posizione "zero" di alcuni servomotori non corrisponde alla posizione iniziale del robot.

La posizione **J21** contiene il tempo di esecuzione della posa. Il valore è espresso in cicli di clock del microcontrollore del robot.

Il suono da riprodurre durante una posa va indicato alla posizione **J20**. Il Robovie-X riproduce il suono memorizzato nella sua ROM a cui corrisponde il numero impostato.

J1	: Right Eye LED	J12	: Left Thigh Pitch
J2	: Left Eye LED	J13	: Right Knee Pitch
J3	: Right Shoulder Pitch	J14	: Left Knee Pitch
J4	: Left Shoulder Pitch	J15	: Right Ankle Pitch
J5	: Right Shoulder Roll	J16	: Left Ankle Pitch
J6	: Left Shoulder Roll	J17	: Right Ankle Roll
J7	: Right Elbow Roll	J18	: Left Ankle Roll
J8	: Left Elbow Roll		
J9	: Right Thigh Roll	J20	: Voice
J10	: Left Thigh Roll	J21	: Time
J11	: Right Thigh Pitch		

Tabella 5.6: Legenda mappatura

Si riporta ora come esempio i tre pacchetti che portano il robot nella posizione iniziale:

0x55	0x03	0xE0	0x01	0x00	0x7C	0x00	0x00
0xA5	0x0A	0x97	0x3E	0xC8	0xF5	0x00	0x00
0x00	0x00	0x00	0x00	0x5B	0xF5	0x69	0xC1
0x38	0x0A	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x61	0x1D	0x80	0x36
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x9F	0xE2	0x80	0xC9	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Tabella 5.7: Primo pacchetto

0x55	0x13	0x00	0x00	0x00	0x00	0xC8	0x00
0x00	0x01	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Tabella 5.8: Secondo pacchetto

0x55	0x23	0x00	0x00	0x00	0x00	0x80	0x00
0x04	0x00	0x00	0x00	0x00	0x80	0x00	0x02
0x01	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00
0x00	0x00	0x00	0x00	0x00	0x00	0x00	0x00

Tabella 5.9: Terzo pacchetto



# Capitolo 6

## Realizzazione del traduttore - Manuale Tecnico

Questo capitolo ha come obiettivo quello di fornire, a chi fosse interessato, le informazioni necessarie a comprendere il funzionamento del traduttore, al fine di modificarlo o espanderne le funzionalità. I requisiti necessari per tale scopo sono una buona conoscenza del linguaggio C e degli strumenti *Flex & Bison*.

### 6.1 Mylexer.l

Come è stato già spiegato nel capitolo precedente relativo allo strumento Flex il file `mylexer.l` ha la funzione di eseguire l'analisi lessicale del linguaggio naturale. La struttura del file è divisa in tre sezioni: definizioni, regole e codice utente, ma per nell'implementazione relizzata la parte relativa al codice utente non è stata utilizzata perché non necessaria.

#### 6.1.1 Definizioni

Il primo pezzo di codice riguarda la parte di definizione dei singoli caratteri che formano le parole del linguaggio utilizzato allo scopo di rendere più semplici le specifiche per lo scanner nelle sezioni successive. Seguendo la sintassi prevista per Flex viene indicato prima il nome della definizione e poi la definizione vera e propria:

```
A [aA]
B [bB]
...
```

### 6.1.2 Regole

Dopo le definizioni e il simbolo `%%` inizia la parte relativa alle regole, cioè dove le parole inserite dall'utente vengono riconosciute e, a seconda dell'input ricevuto, viene generato un token. Di seguito si riporta come viene mappato in uscita ogni parola lecita in ingresso al traduttore.

- Numero: nel caso in cui venga inserito un numero, la union `yylval` viene utilizzata per passare al parser il valore associato al token corrente ovvero `NUMBER`.

```
[0-9]+                                {yylval.number=atoi(ytext); return NUMBER;}
```

- Servomotori: anche per i servomotori viene associato un numero diverso che corrisponde ad un servomotore diverso. La scelta è dovuta dal fatto che successivamente, nel file Bison, viene utilizzato il costrutto `swtch-case` per riconoscere il giunto su cui si sta operando, cosa che non è possibile fare utilizzando i caratteri alfabetici. Inoltre, si osserva che i valori restituiti non sono sempre consecutivi: nel caso in cui un servomotore o un sensore sia simmetrico, l'incremento del valore restituito è di 2. L'utilità di questa scelta viene descritta successivamente nell'analisi del file `myparser.y`. Il token restituito è `JOINT` ad eccezione della testa, `HEAD`, e degli occhi, `LED`.

```
head                                {yylval.number=1; return HEAD;}
eyelld                              {yylval.number=2; return EYE;}
shoulder                            {yylval.number=4; return JOINT;}
arm                                  {yylval.number=6; return JOINT;}
elbow                                {yylval.number=8; return JOINT;}
hip                                  {yylval.number=10; return JOINT;}
thigh                                {yylval.number=12; return JOINT;}
knee                                  {yylval.number=14; return JOINT;}
calf                                  {yylval.number=16; return JOINT;}
ankle                                {yylval.number=18; return JOINT;}
gyrox                                {yylval.number=21; return JOINT;}
gyroy                                {yylval.number=22; return JOINT;}
shootanalog                          {yylval.number=23; return JOINT;}
pitchmove                            {yylval.number=24; return JOINT;}
rolllstep                            {yylval.number=26; return JOINT;}
rollrstep                            {yylval.number=28; return JOINT;}
legpace                              {yylval.number=30; return JOINT;}
turnctrl                             {yylval.number=32; return JOINT;}
```

- Lato: di fondamentale importanza risulta essere la posizione del servomotore o del sensore nel caso questi siano simmetrici. Ecco quindi che la distinzione viene fatta utilizzando la funzione `strcmp()` incluso nella libreria `string.h` di C: se l'oggetto appartiene alla parte destra viene restituito il valore numerico 1, altrimenti 0. Il token restituito in questo caso è `SECTION`.

```
right|left      {yyval.number=!strcmp(yytext,"right"); return SECTION;}
```

- Direzione: anche per la specifica del verso del movimento si è scelto di un token di tipo numerico. In questo caso però gli schemi delle regole risultano più complessi perché per un'azione possono corrispondere parole diverse: dentro e verso dentro (inside e inward) hanno lo stesso significato e restituiscono il token TO con lo stesso valore.

```
forward|front      {yyval.number=0; return TO;}
back((ward)?)      {yyval.number=1; return TO;}
in((ward)|(side))  {yyval.number=2; return TO;}
out((ward)|(side)) {yyval.number=3; return TO;}
rightward          {yyval.number=4; return TO;}
leftward           {yyval.number=5; return TO;}
```

- L'ultimo gruppo di parole per le quali si è scelto di utilizzare la "conversione" dalla parola ad un numero riguarda aumenta, decrementa e imposta. Questi tre vocaboli vengono restituiti con i valori 0, 1 e 2 nel token SIGN.

```
increase          {yyval.number=0; return SIGN;}
decrease          {yyval.number=1; return SIGN;}
set               {yyval.number=2; return SIGN;}
```

- Le altre regole hanno uno schema o l'azione corrispondente meno complesso di quelle appena presentate. Tuttavia meritano una particolare attenzione la regola che individua le preposizioni. Il pattern restituisce nel token OFFS uno quando riconosce la parola of e zero quando individua la parola at o to.

```
of|((at)|(to))    {yyval.number=!strcmp(yytext,"of"); return OFFS;}
```

Il token DEGREE viene utilizzato sia per contenere la parola degree stessa, al plurale e al singolare, sia per il simbolo di % utilizzato nel comando degli occhi.

```
degree((s)?)|"% " return DEGREE;
```

Infine, sono anche previste le regole per ignorare gli spazi bianchi tra le parole semplicemente lasciando vuota l'azione corrispondente al carattere di spazio e quella per riconoscere il percorso di un file audio in cui si rende necessario creare una stringa che possa contenere sia numeri che lettere ed in aggiunta anche il carattere di \. Quest'ultima istruzione deve essere necessariamente inserita come ultima regola, altrimenti tutti i successivi schemi e azioni non verrebbero mai eseguiti.

```
[ \t]+          /* ignore whitespace */;
[a-z0-9A-Z_\]+ {yyval.string=strdup(yytext); return AUDIO;}
```

Tutte le altre parole vengono restituite al file Bison mediante dei token che hanno lo stesso nome della parola riconosciuta.

## 6.2 Myparser.y

Il file `myparser.y`, diviso nelle sezioni prologo, dichiarazioni, regole grammaticali ed epilogo, realizza l'analisi sintattica del linguaggio naturale.

### 6.2.1 Prologo

Questa prima parte del file Bison contiene, dopo gli `#include`, le definizioni delle costanti che contengono i valori per la taratura del robot e le variabili globali che associano al nome di ogni giunto un valore numerico al fine di rendere più semplice e leggibile il codice del programma.

```
...
#define SHOULDER_ROLL_L          0xC980
#define SHOULDER_ROLL_R          0x3680
...
int knPitchL = 14;
int knPitchR = 15;
...
```

Successivamente vengono definiti ed inizializzati due array di tipo WORD (16 bit): `actualPosition` rappresenta la posizione corrente di tutti i servomotori e sensori del robot mentre `offs` contiene i valori che indicano la posizione iniziale.

```
WORD actualPosition[] = {TIME, HEAD_YAW, EYE_LED_L, EYE_LED_R, SHOULDER_PITCH_L,
    SHOULDER_PITCH_R, SHOULDER_ROLL_L, SHOULDER_ROLL_R, ELBOW_ROLL_L, ELBOW_ROLL_R,
    THIGH_ROLL_L, THIGH_ROLL_R, THIGH_PITCH_L, THIGH_PITCH_R, KNEE_PITCH_L,
    KNEE_PITCH_R, ANKLE_PITCH_L, ANKLE_PITCH_R, ANKLE_ROLL_L, ANKLE_ROLL_R, VOICE,
    GYROXGAIN, GYROYGAIN, CTRL_SHOOTANALOG, GAINMOVE_PITCH_L, GAINMOVE_PITCH_R,
    GAINLSTEP_ROLL_L, GAINLSTEP_ROLL_R, GAINRSTEP_ROLL_L, GAINRSTEP_ROLL_R,
    PACEGAIN_LEG_L, PACEGAIN_LEG_R, CTRL_TURN_L, CTRL_TURN_R};

WORD offs[] = {TIME, HEAD_YAW, EYE_LED_L, EYE_LED_R, SHOULDER_PITCH_L,
    SHOULDER_PITCH_R, SHOULDER_ROLL_L, SHOULDER_ROLL_R, ELBOW_ROLL_L, ELBOW_ROLL_R,
    THIGH_ROLL_L, THIGH_ROLL_R, THIGH_PITCH_L, THIGH_PITCH_R, KNEE_PITCH_L,
    KNEE_PITCH_R, ANKLE_PITCH_L, ANKLE_PITCH_R, ANKLE_ROLL_L, ANKLE_ROLL_R, VOICE,
    GYROXGAIN, GYROYGAIN, CTRL_SHOOTANALOG, GAINMOVE_PITCH_L, GAINMOVE_PITCH_R,
    GAINLSTEP_ROLL_L, GAINLSTEP_ROLL_R, GAINRSTEP_ROLL_L, GAINRSTEP_ROLL_R,
    PACEGAIN_LEG_L, PACEGAIN_LEG_R, CTRL_TURN_L, CTRL_TURN_R};
```

Dopo aver dichiarato la variabile `audio`, utile per memorizzare le informazioni relative ai suoni, si dichiara un array di `struct`, `motions`, di tipo `motionSequence`.

Questo è stato creato per poter memorizzare i blocchi che fanno parte di un movimento di qualunque tipo essi siano: nella *i*-esima cella dell'array viene salvato il blocco *i*-esimo così da ottenere la sequenza di blocchi che dà origine al movimento voluto. Il campo `type` ha la funzione di codificare quale tipo di blocco sia memorizzato (posa, calcolo o salto condizionato), `next1` e `next2` sono i puntatori ai blocchi successivi, l'array `info` contiene il valore dati che variano a seconda del tipo di blocco e `sound` indica il nome del percorso del file audio.

```
struct motionSequence
{
    int type; //1 : posa
    int next1;
    int next2;
    WORD info[INFO_NUM];
    char sound[100];
} motions[MAX_MOTIONS];
```

Una volta completate tutte le dichiarazioni e inizializzazioni delle variabili globali si procede all'implementazione dei metodi `yyerror` e `yywrap` previsti da Bison. La funzione `degToOffset` ha il compito di tradurre il valore dei gradi sessagesimali specificati dall'utente nella notazione esadecimale: questo è stato realizzato mediante una moltiplicazione per la costante `0x00da`.

```
int degToOffset(int deg)
{
    int offs = deg * 0xda;
    return offs;
}
```

`savePose()` è una funzione che salva la posizione attuale di tutti i parametri nell'array di `WORD info[]` che si trova nella prima posizione libera dell'array di struct `motions`. Al termine riporta il valore del tempo di esecuzione al valore di default e aggiorna il puntatore alla prossima cella di memoria libera.

```
void savePose()
{
    int i, succ;
    motions[current].type = 1;
    for (i = 0; i < INFO_NUM; i++)
        motions[current].info[i] = actualPosition[i];
    char audioPw[30] = "X Voice Files\\";
    if (strcmp(audio, "") != 0)
    {
        strcat(audioPw, audio);
        strcat(audioPw, ".wav");
        strcpy(motions[current].sound, audioPw);
    }
}
```

```

        strcpy(audio, "");
    }
    succ = current + 1;
    motions[current].next1 = succ;
    current++;
    actualPosition[time] = 50;
}

```

Le funzioni `saveCalculation()` e `saveIf()` sono stati realizzati per memorizzare rispettivamente i blocchi di calcolo e i blocchi di salto condizionato. Entrambe “salvano” nella prima cella dell’array `info` l’identificatore dell’operazione, nella seconda l’indice della variabile e nella terza il valore di confronto. In aggiunta nel metodo `saveIf()` viene salvato il valore del numero di blocco quando la condizione di salto è falsa.

```

void saveCalculation(int op, int index, int value)
{
    int succ;
    motions[current].type = 2;
    motions[current].info[1] = op;
    motions[current].info[2] = index;
    motions[current].info[3] = value;
    succ = current + 1;
    motions[current].next1 = succ;
    current++;
}

void saveIf(int op, int index, int value, int jump)
{
    int succ;
    motions[current].type = 3;
    motions[current].info[1] = op;
    motions[current].info[2] = index;
    motions[current].info[3] = value;
    motions[current].next2 = jump;
    succ = current + 1;
    motions[current].next1 = succ;
    current++;
}

```

Continuando con l’analisi del file `Bison`, la funzione `toString()` ha il compito di stampare su file l’intero movimento richiesto dall’utente. Sfruttando la struttura creata precedentemente, la stampa è stata realizzata riversando su file i valori contenuti all’interno di `motions` tenendo conto del tipo di blocco che si sta stampando e della sintassi prevista dal software `RobovieMaker`.

```

void toString()
{
    int i;

```

```

fprintf(fp, "_MOTION_FORMAT : [ POSEDEF_V1R4 ] \n _ENABLE_AXIS : [62] \n\n
_MOTION_INFO : [[]] - [50] - [10] - [6] - [0] - [1] - [-1] \n _STARTMOT : \n\n
");
fprintf(fp, "_MOTION_INFO : [[POSE1]] - [50] - [43] - [6] - [6] - [2] - [-1] \n
_POSE : [50] - [0x0000, 0x0aa5, 0x3e97, 0xf5c8, 0x0000, 0X0000, 0x0000, 0xf55b,
0xc169, 0x0a38, 0x0000, 0X0000, 0x0000, 0X0000, 0x0000, 0X0000, 0x1d61, 0x3680, 0x0000
, 0x0000, 0X0000, 0X0000, 0xe29f, 0xc980, 0x0000, 0X0000, 0X0000, 0x0000, 0
X0000, 0X0000, 0x0000, 0x00c8, 0x0100, 0X0000, 0X0000, 0X0000, 0X0000,
0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000,
0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0X0000, 0x0000, 0X0000, 0X0000, 0X0000
, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000] - [[]] -
[00000000000000000000000000000000] \n\n");

for (i = 2; i < current; i++)
{
switch (motions[i].type) {
case 1 :
fprintf(fp, "_MOTION_INFO: [[POSE%d]] - [50] - [%d] - [6] - [6] - [%d] - [-1] \n", i, i*50,
motions[i].next1);
fprintf(fp, "_POSE: [%d] - [0x%04x, 0x%04x, 0x%04x, 0x%04x, 0x%04x, 0X0000, 0x%04x
, 0x%04x, 0x%04x, 0x%04x, 0x%04x, 0X0000, 0x%04x, 0X0000, 0X0000, 0x%04x, 0
x%04x, 0x%04x, 0X0000, 0X0000, 0x%04x, 0x%04x, 0x%04x, 0X0000, 0
X0000, 0x%04x, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0x0000, 0x%04x, 0
X0000, 0x%04x, 0x%04x, 0x%04x, 0x%04x, 0x%04x, 0x%04x, 0x%04x, 0
X0000, 0x%04x, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0
X0000] - [%s] - [00000000000000000000000000000000] \n\n", motions[i].info[time],
motions[i].info[anRollR], motions[i].info[anPitchR], motions[i].info[
knPitchR], motions[i].info[thPitchR], motions[i].info[thRollR], motions[i].
info[anRollL], motions[i].info[anPitchL], motions[i].info[knPitchL],
motions[i].info[thPitchL], motions[i].info[thRollL], motions[i].info[eyeL],
motions[i].info[elRollR], motions[i].info[shRollR], motions[i].info[
shPitchR], motions[i].info[eyeR], motions[i].info[elRollL], motions[i].info
[shRollL], motions[i].info[shPitchL], motions[i].info[head], motions[i].
info[gyroX], motions[i].info[gyroY], motions[i].info[mvPitchL], motions[i].
info[mvPitchR], motions[i].info[lStepRollL], motions[i].info[lStepRollR],
motions[i].info[rStepRollL], motions[i].info[rStepRollR], motions[i].info[
paceLegL], motions[i].info[paceLegR], motions[i].info[turnL], motions[i].
info[turnR], motions[i].info[shootAn], motions[i].sound);
break;
case 2 :
fprintf(fp, "_MOTION_INFO: [[CALCULATION%d]] - [50] - [%d] - [6] - [6] - [%d] - [-1] \n", i,
i*50, motions[i].next1);
fprintf(fp, "_VARCON: [%d] - [%d] - [%d] \n\n", motions[i].info[1], motions[i].info
[2], motions[i].info[3]);
break;
case 3 :
fprintf(fp, "_MOTION_INFO: [[IF%d]] - [50] - [%d] - [6] - [6] - [%d] - [%d] \n", i, i*50,
motions[i].next2, motions[i].next1);
fprintf(fp, "_MIF: [%d] - [%d] - [%d] \n\n", motions[i].info[1], motions[i].info[2],
motions[i].info[3]);
break;
}
}

fprintf(fp, "_MOTION_INFO : [[ END ]] - [50] - [%d] - [16] - [46] - [-1] -
[-1] \n _ENDMOT :", 50*current);

```





```
}
}
```

## 6.2.2 Dichiarazioni

In questa sezione vengono dichiarati i nomi dei token, la union e, se necessario, il tipo a cui un token appartiene.

```
%token TURN STOP EXIT AND DOT BRIGHTNESS PARAMETER PLAY STEP FRONTFLIP BACKFLIP
%token DEGREE MILLISECOND IN TIMES REPEAT RESET GREETINGS YAWN WALK DANCE GUFFU
  GUFFD

%union
{
  int number;
  char * string;
}

%token <number> NUMBER
%token <number> HEAD
%token <number> EYE
%token <number> SECTION
%token <number> JOINT
%token <number> OFFS
%token <number> TO
%token <number> SIGN
%token <string> AUDIO
```

## 6.2.3 Regole grammaticali

L'analisi sintattica viene eseguita in questa sezione del file Bison: l'analizzatore sintattico, infatti, ricerca tra le regole grammaticali quale coincide con l'istruzione ricevuta dal linguaggio naturale. Per definire le azioni si parte dal generale per poi proseguire verso i dettagli: in questo modo diventa più facile il riconoscimento di frasi complesse che sono la concatenazione di più frasi semplici.

```
commands :
  | commands command
  ;

command :
stop | turn | exit | turn AND | turn DOT {savePose();}
| turn tempo | turn tempo AND | turn tempo DOT {savePose();}
| turn audio | turn audio AND | turn audio DOT {savePose();}
| turn tempo audio | turn tempo audio AND | turn tempo audio DOT {savePose();}
| turn audio tempo | turn audio tempo AND | turn audio tempo DOT {savePose();}
| repeat | reset | movements
| online | macro
```

;

L'azione repeat, in coppia con stop, permette di realizzare cicli, anche annidati, con l'ausilio di uno stack. Quando il parser riconosce la sequenza REPEAT NUMBER TIMES il traduttore crea un nuovo blocco di calcolo per inizializzare la variabile contatore e salva il numero del blocco corrente in modo da ricordare in quale posizione posizionarsi in caso di salto. Al termine del blocco di istruzioni che l'utente ha deciso di far ripetere STOP aggiunge un blocco di calcolo che decrementa la variabile contatore e un blocco di salto condizionato che punta al primo blocco della sequenza da ripetere.

```
repeat:
    REPEAT NUMBER TIMES
    {
        saveCalculation(0, 80 + counter, $2);
        loopStack[counter] = current;
        counter++;
    }
;

stop:
    STOP
    {
        counter--;
        saveCalculation(2, 80 + counter, 1);
        saveIf(4, 80 + counter, 0, loopStack[counter]);
    }
;
```

L'azione turn racchiude tutte le tipologie di movimenti e le impostazioni dei sensori: per ogni giunto, viene fatta la distinzione tra lato destro e lato sinistro, se necessario, e successivamente, a seconda della preposizione utilizzata nell'istruzione, si esegue un incremento o un decremento oppure la variabile modificata viene imposta ad un preciso valore.

```
turn:
    TURN SECTION JOINT OFFS NUMBER DEGREE TO
    {
        int side = $2;
        int joint = $3;
        if(side)
            {joint++;}
        switch(joint)
        {
            case 4:
            case 5:
                if ($4)
                {
```

```

        if ($7 == 0) {actualPosition[joint] -= degToOffset($5);}
        else if ($7 == 1) {actualPosition[joint] += degToOffset($5);}
    }
    else
    {
        if ($7 == 0) {actualPosition[joint] = degToOffset(-($5)) + offs[joint];}
        else if ($7 == 1) {actualPosition[joint] = degToOffset($5) + offs[joint];}
    }
    break;
}
}
;

```

A completamento di un'istruzione di movimento può essere prevista la specifica del tempo di esecuzione della posa e del suono da riprodurre. Nell'azione tempo vengono compiute le seguenti operazioni: all'inizio si converte il valore numerico da millisecondi a numero di cicli del processore del robot; successivamente si esegue un controllo sul valore ottenuto in quanto valori troppo bassi o troppo alti potrebbero causare dei danni ai servomotori del Robovie-X.

```

tempo:
    IN NUMBER MILLISECOND
    {
        actualPosition[time] = $2 * 0.06;
        if (actualPosition[0] < 1) actualPosition[0] = 1;
        else if (actualPosition[0] > 239) actualPosition[0] = 239;
    }
;

audio:
    PLAY AUDIO
    {
        strcpy(audio, $2);
    }
;

```

L'azione movements si occupa di caricare le sequenze di blocchi da un file esterno utilizzando fscanf() dalla libreria stdio.h di C. Ogni azione invoca il metodo loadFile() già analizzato precedentemente che si occupa della lettura e del trasferimento delle informazioni nell'array motions.

```

movements:
    GREETINGS
    {
        loadFile("greetings.txt");
    }
;

```

L'azione reset, quando invocata, riporta tutti i valori contenuti nell'array actualPosition a quelli di partenza. Questi, all'inizio, vengono memorizzati all'interno del vettore offs[], di conseguenza l'azione consiste nella copia di offs[] in actualPosition[].

```

reset:
    RESET
    {
        int i;
        for (i = 0; i < INFO_NUM; i++)
            actualPosition[i] = offs[i];
    }
;

```

Exit, infine, è l'azione che conclude l'inserimento di istruzioni: invocando il metodo toString() il traduttore riversa l'intero contenuto della struct motions sul file output.txt e, al termine, chiude il programma.

```

exit:
    EXIT
    {
        toString();
        return 0;
    }
;

```

## 6.2.4 Macro

Per permettere l'utilizzo di macro sono state implementate due regole grammaticali:

- SAVECMD TEXT
- EXEC CMD TEXT

La prima si occupa della definizione della macro, salva le istruzioni date in in input in un file di testo e rinomina il file con lo stesso nome della macro; la seconda istruzione si occupa dell'esecuzione della macro: imposta le due variabili d'ambiente state e macroName, terminando la funzione yyparse() e ritornando al main() che si occupa dell'esecuzione vera e propria della macro. L'implementazione del metodo main() è descritta nella sezione successiva.

```

macro:
    SAVECMD TEXT
    {
        FILE * fpmacro = fopen("$2", "w");
        printf("Insert command list \nHit enter to quit\n");
        char str[80];
    }
;

```

```

        do
        {
            gets(str);
            strcat(str, "\n"); /* add a newline */
            fputs(str, fpmacro);
        } while(*str!='\n');

        printf("Macro Saved\n");

        fclose(fpmacro);
    }

    | EXECCMD TEXT
    {
        state = 1;

        strcpy(macroName, $2);

        return;
    }
;

```

### 6.3 Metodo main()

Il metodo `main()` è quello che realizza il parsing delle istruzioni e che coordina le attività di apertura e chiusura del file di output.

La funzione principale è `yyparse()`, si occupa della lettura e del parsing dei token ricevuti in input attraverso stream o file a cui punta `yyin`.

Durante la normale esecuzione del traduttore `yyin` è un puntatore a `stdin` (standard input) e il parsing viene eseguito sulle istruzioni digitate nella finestra di dialogo. Quando viene richiamata una macro il `main` si occupa di creare un puntatore al file che la contiene, copiare il puntatore in `yyin`, reinizializzare il parser con il nuovo input e avviare il parsing con la chiamata a `yyparse()`.

```

int main(void)
{
    fp = fopen(FILE_OUTPUT, "w");

    while(state==0 )
    {
        yyin = stdin;

        yyrestart(yyin);

        yyparse();

        if ( state == 1)
        {
            yyin = fopen(macroName,"r");

            yyrestart(yyin);
        }
    }
}

```

```

        yyparse();

        state = 0;
        fclose(yyin);
    }
    yylex_destroy();
}

fclose(fp);
return 0;
}

```

## 6.4 Implementazione del movimento in tempo reale nel Traduttore

Per rendere possibile la connessione e comunicazione diretta tra robot e traduttore sono state aggiunte delle librerie contenenti le funzioni fondamentali, che permettono la connessione, l'inizializzazione, l'accensione, l'invio/ricezione di pacchetti e lo spegnimento.

## 6.5 Connessione e scambio dati

Le funzioni necessarie alla connessione e allo scambio di informazioni con il robot sono state implementate nel file **rcCom.c**.

### 6.5.1 La funzione rcOpen()

Questa funzione ha il compito di individuare il robot tra i vari HID collegati al pc e di aprire la connessione con esso. Generalmente quando un'applicazione che opera in user-mode e gestisce un HID viene avviata esegue la seguente sequenza di operazioni:

- Chiama HidD GetHidGuid per ottenere il GUID (Globally Unique Identifier) definito dal sistema per i dispositivi della classe HID:

```

static GUID Guid;
...
HidD GetHidGuid(&Guid);

```

- Chiama SetupDiGetClassDevs per ottenere un handle a un device information set che descrive le interfacce supportate dal sistema:

```

static HDEVINFO devInfoSet;
...
devInfoSet = SetupDiGetClassDevsA(&Guid,0,0,0x12);

```

- Chiama ripetutamente `SetupDiEnumDeviceInterfaces` per ottenere tutte le informazioni disponibili su un'interfaccia:

```
SP_DEVICE_INTERFACE_DATA interfaceData;
BOOL devDetail;
...
devDetail = SetupDiEnumDeviceInterfaces(devInfoSet,0,&Guid,ifaceCounter, &
    interfaceData);
```

- Chiama la funzione `SetupDiGetDeviceInterfaceDetail` affinché le informazioni ottenute sull'interfaccia vengano inserite in una struttura di tipo `SP_INTERFACE_DEVICE_DETAIL_DATA`:

```
SP_DEVICE_INTERFACE_DETAIL_DATA_A * interfaceDetailData;
..
devDetail = SetupDiGetDeviceInterfaceDetailA(devInfoSet,&interfaceData,NULL
    ,0,&reqSize,NULL);
interfaceDetailData = (struct SP_DEVICE_INTERFACE_DETAIL_DATA_A *)malloc(
    reqSize);
devDetail = SetupDiGetDeviceInterfaceDetailA( devInfoSet, &interfaceData,
    interfaceDetailData,reqSize,0,0);
```

Si effettuano due chiamate alla funzione `SetupDiGetDeviceInterfaceDetail` poiché non è nota a priori la dimensione necessaria per la struttura `SP_INTERFACE_DEVICE_DETAIL`:

- la prima chiamata termina con un errore, ma `reqSize` contiene la dimensione in byte richiesta per memorizzare la struttura.
- vengono riservati `reqSize` byte per mezzo della funzione `malloc`.
- con la seconda chiamata si ottiene effettivamente la struttura desiderata.

- Chiama la funzione `CreateFile` per ottenere un handle al HID:

```
HANDLE hObject;
...
hObject=CreateFile (interfaceDetailData->DevicePath,0x0c000000,3,0,3,0,0);
```

## 6.5.2 Le funzioni `rcWrite()` e `rcRead()`

Queste funzioni hanno il compito, rispettivamente, di scrivere e leggere il dispositivo, se il valore ritornato è nullo, allora viene invocata la funzione `GetLastError()`, che ritorna il codice dell'errore verificatosi.

```
bool rcWrite(BYTE *buffer, int bToWrite, DWORD *bWritten)
{
    //WriteFile(HANDLE, LPCVOID, DWORD, LPDWORD, LP_OVERLAPPED)
    bool returned = WriteFile(hObject, buffer, bToWrite, bWritten, 0);
    if(returned==0)
    {
```

```

    printf("errore in scrittura: %0x\n", GetLastError());
}
return returned;
}

```

```

bool rcRead(BYTE *buffer, int bToRead, DWORD *bRead)
{
    //ReadFile(HANDLE, LPCVOID, DWORD, LPDWORD, LPOVERLAPPED)
    bool returned = ReadFile(hObject, buffer, bToRead, bRead, 0);
    if(returned == 0)
    {
        printf("errore in lettura: %0x\n", GetLastError());
    }
    return returned;
}

```

### 6.5.3 Libreria move

La libreria move definita nell'header `move.h`, riportata nel listato, contiene la definizione di quattro buffer `toRobot` della dimensione di 64 byte che saranno utilizzati per la comunicazione con il robot.

Il metodo `init()` inizializza i quattro buffer con i valori di cui si è già parlato nel paragrafo 6.5, inizializza la connessione con il robot e invia il pacchetto che accende i servomotori portando l'umanoide nella posizione iniziale.

```

BYTE toRobot[MAX_BUF_SIZE], toRobot1[MAX_BUF_SIZE], toRobot2[MAX_BUF_SIZE],
    toRobot3[MAX_BUF_SIZE], fromRobot[MAX_BUF_SIZE];

bool init(void);
bool doMovements(void);
bool off(void);

```

Con la chiamata a `doMovements()` vengono inviati all'umanoide in sequenza i tre buffer `toRobot1`, `toRobot2` e `toRobot3`. La struttura dei tre buffer è stata descritta nel paragrafo 6.5.2.

```

bool doMovements()
{
    bool toReturn = false;

    toReturn = rcWrite(toRobot1, BYTES_TO_WRITE, &bytesWritten);
    toReturn = toReturn & rcWrite(toRobot2, BYTES_TO_WRITE, &bytesWritten);
    toReturn = toReturn & rcWrite(toRobot3, BYTES_TO_WRITE, &bytesWritten);

    return toReturn;
}

```

Il metodo `off()` invia pacchetto che comanda lo spegnimento del robot.



## 6.6 Creazione dei pacchetti

I pacchetti vengono creati e inviati durante l'esecuzione del metodo `savePose()` già descritto nel paragrafo 6.2. Ad ogni salvataggio di una posa nell'array di struct `motions`, se la connessione con l'umanoide è aperta, viene lanciato il metodo `execPose()`. Tale metodo riformatta le informazioni contenute nella struct `actualPosition` e lancia il metodo `doMovements()`.

### 6.6.1 La metodo `execPose()`

Questo metodo riceve in ingresso un puntatore al buffer che contiene gli offset dell'ultima posa salvata. Prima di lanciare il metodo `doMovements()` trasferisce i dati contenuti nel buffer ricevuto in ingresso nella posizione corretta dei tre array `toRobot`, scambiando sempre i byte più significativi con i meno significativi, ricordando quanto detto nel capitolo 5.

```
bool execPose(WORD * actualPosition)
{
    bool toReturn = false;

    //TIME
    toRobot3[17] = (BYTE) actualPosition[0];

    //HEAD_YAW
    toRobot1[59] = (BYTE) actualPosition[1];
    toRobot1[60] = actualPosition[1] >> 8;

    //EYE_LED_L
    toRobot1[31] = (BYTE) actualPosition[2];
    toRobot1[32] = actualPosition[2] >> 8;

    ....
    ....

    //ANKLE_ROLL_R
    toRobot1[7] = (BYTE) actualPosition[19];
    toRobot1[8] = actualPosition[19] >> 8;

    //VOICE
    toRobot2[5] = (BYTE) actualPosition[20];

    toReturn = doMovements();

    return toReturn;
}
```

## 6.7 Istruzioni per la compilazione

Per compilare i file `.l` e `.y` è necessario aver installato in Windows gli strumenti Flex, Bison e l'ambiente di sviluppo MinGW, che include il noto compilatore open source gcc e i necessari header file e librerie. Le istruzioni necessarie sono le seguenti:

```
flex mylexer.l
bison -d myparser.y
gcc lex.yy.c myparser.tab.c move.c onlineMode.c rcCom.c -o VX-Translator.exe
```

Il primo comando compila, mediante Flex, il file di specifiche `mylexer.l`. L'output di questa compilazione è il file `lex.yy.c` che è il file scritto nel linguaggio C all'interno del quale è contenuta la funzione `yylex()` che ha il compito di eseguire l'analisi lessicale a cui si appoggia il parser durante l'operazione di traduzione. Il secondo comando compila, mediante lo strumento Bison, il file `myparser.y`. L'opzione `-d` serve per generare la tabella in cui è presente la corrispondenza token-valore numerico. Tale tabella è presente nello header file `myparser.tab.h`. La compilazione genera inoltre un file `myparser.tab.c` dove è contenuta la funzione `yyparse()`, funzione che esegue l'analisi sintattica. Infine, il terzo ed ultimo comando compila, mediante gcc, i due file precedentemente generati e li unisce alle librerie che contengono i metodi per il controllo in tempo reale, creando in uscita un programma eseguibile `translator.exe` che è il vero e proprio traduttore.

# Conclusioni

Il risultato di questa tesi è VX-Translator, software che consente il controllo del robot umanoide Robovie-X.

Questa applicazione, grazie al linguaggio naturale che è stato sviluppato, risulta di semplice utilizzo per qualsiasi utente e allo stato attuale permette un controllo pressoché completo del movimento del Robovie-X anche senza l'utilizzo del software RobovieMaker.

Inoltre si è voluto costruire una base per sviluppi futuri: sarebbe interessante implementare la gestione dei sensori, giroscopio e accelerometro, nel software VX-Translator oltre che espandere il linguaggio e i movimenti attualmente riconosciuto e integrati nell'applicazione.

Per semplificare ulteriormente il programma si potrebbe pensare di sviluppare un'interfaccia grafica.



# Appendice A

## Flex & Bison

### A.1 Flex

Listing A.1: mylexer.l

```
%{
#include <stdio.h>
#include <string.h>
#include "myparser.tab.h"
%}

%option yylineno
A [aA+]
B [bB]
C [cC]
D [dD]
E [eEËË]
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 [uU~]
V [vV]
W [wW]
X [xX]
Y [yY]
Z [zZ]

%%
```

```

[0-9]+          {yyval.number=atoi(yytext); return NUMBER;}
stop           return STOP;
exit           return EXIT;
and|,          return AND;
"."           return DOT;
[ \t]+         /* ignore whitespace */;
move|turn      return(TURN);

increase       {yyval.number=0; return SIGN;}
decrease       {yyval.number=1; return SIGN;}
set            {yyval.number=2; return SIGN;}
right|left     {yyval.number=!strcmp(yytext,"right"); return SECTION;}
head           {yyval.number=1; return HEAD;}
eye|led        {yyval.number=2; return EYE;}
shoulder       {yyval.number=4; return JOINT;}
arm            {yyval.number=6; return JOINT;}
elbow          {yyval.number=8; return JOINT;}
hip            {yyval.number=10; return JOINT;}
thigh          {yyval.number=12; return JOINT;}
knee           {yyval.number=14; return JOINT;}
calf           {yyval.number=16; return JOINT;}
ankle          {yyval.number=18; return JOINT;}
gyrox          {yyval.number=21; return JOINT;}
gyroy          {yyval.number=22; return JOINT;}
shootanalog    {yyval.number=23; return JOINT;}
pitchmove      {yyval.number=24; return JOINT;}
rolllstep      {yyval.number=26; return JOINT;}
rollrstep      {yyval.number=28; return JOINT;}
legpace        {yyval.number=30; return JOINT;}
turnctrl       {yyval.number=32; return JOINT;}

degree((s)?|"%" return DEGREE;
millisecond((s)?|ms return MILLISECOND;
of|((at)|(to))     {yyval.number=!strcmp(yytext,"of"); return OFFS;}
in                return ININ;

forward|front|ahead      {yyval.number=0; return TO;}
back((ward)?)           {yyval.number=1; return TO;}
in((ward)|(side))       {yyval.number=2; return TO;}
out((ward)|(side))      {yyval.number=3; return TO;}
rightward                {yyval.number=4; return TO;}
leftward                 {yyval.number=5; return TO;}

reset                return RESET;

online|offline        {yyval.number=!strcmp(yytext,"online"); return ONLINE;}

brightness           return BRIGHTNESS;

repeat               return REPEAT;
time((s)?)           return TIMES;

gain|control         return PARAMETER;

play((ing)?)         return PLAY;

save[ \t]+macro[ \t] return SAVECMD;
exec[ \t]+macro[ \t] return EXECCMD;

```

```

greetings                return GREETINGS;
yawn                     return YAWN;
frontflip                return FRONTFLIP;
backflip                 return BACKFLIP;
dance                    return DANCE;
walk                     return WALK;
get[ \t]+up[ \t]+from[ \t]+face[ \t]+up    return GUFFU;
get[ \t]+up[ \t]+from[ \t]+face[ \t]+down  return GUFFD;

step((s)?)               return STEP;

[a-z0-9A-Z_\\]+         {yyval.string=strdup(yytext); return TEXT;}

.                        printf("invalid character");
%%

```

## A.2 Bison

Listing A.2: myparser.y

```

%{
#include <stdio.h>
#include <string.h>
#include "onlineMode.h"
#ifdef WIN32
    #include <windows.h>
#else
    typedef unsigned char WORD;
#endif

/*Initial Offset*/
#define HEAD_YAW                0x0000
#define EYE_LED_L                0x0000
#define EYE_LED_R                0x0000
#define SHOULDER_PITCH_L        0x0000
#define SHOULDER_PITCH_R        0x0000
#define SHOULDER_ROLL_L        0xC980
#define SHOULDER_ROLL_R        0x3680
#define ELBOW_ROLL_L            0xE29F
#define ELBOW_ROLL_R            0x1D61
#define THIGH_ROLL_L            0x0000
#define THIGH_ROLL_R            0x0000
#define THIGH_PITCH_L            0x0A38
#define THIGH_PITCH_R            0xF5C8
#define KNEE_PITCH_L            0xC169
#define KNEE_PITCH_R            0x3E97
#define ANKLE_PITCH_L            0xF55B
#define ANKLE_PITCH_R            0x0AA5
#define ANKLE_ROLL_L            0x0000
#define ANKLE_ROLL_R            0x0000
#define VOICE                    0x0000
#define TIME                    50
#define GYROXGAIN                200
#define GYROYGAIN                256

```

```

#define CTRL_SHOOTANALOG      0
#define GAINMOVE_PITCH_L     0
#define GAINMOVE_PITCH_R     0
#define GAINLSTEP_ROLL_L     0
#define GAINLSTEP_ROLL_R     0
#define GAINRSTEP_ROLL_L     0
#define GAINRSTEP_ROLL_R     0
#define PACEGAIN_LEG_L       0
#define PACEGAIN_LEG_R       0
#define CTRL_TURN_L          0
#define CTRL_TURN_R          0

/*Constants*/
#define MAX_MOTIONS          100
#define MAX_LOOPS            10
#define INFO_NUM              34

/*Global Variables*/
int time = 0;
int head = 1;
int eyeL = 2;
int eyeR = 3;
int shPitchL = 4;
int shPitchR = 5;
int shRollL = 6;
int shRollR = 7;
int elRollL = 8;
int elRollR = 9;
int thRollL = 10;
int thRollR = 11;
int thPitchL = 12;
int thPitchR = 13;
int knPitchL = 14;
int knPitchR = 15;
int anPitchL = 16;
int anPitchR = 17;
int anRollL = 18;
int anRollR = 19;
int voice = 20;
int gyroX = 21;
int gyroY = 22;
int shootAn = 23;
int mvPitchL = 24;
int mvPitchR = 25;
int lStepRollL = 26;
int lStepRollR = 27;
int rStepRollL = 28;
int rStepRollR = 29;
int paceLegL = 30;
int paceLegR = 31;
int turnL = 32;
int turnR = 33;

/*Array which contains the current value of joints*/
WORD actualPosition[] = {TIME, HEAD_YAW, EYE_LED_L, EYE_LED_R, SHOULDER_PITCH_L,
    SHOULDER_PITCH_R, SHOULDER_ROLL_L, SHOULDER_ROLL_R, ELBOW_ROLL_L, ELBOW_ROLL_R,
    THIGH_ROLL_L, THIGH_ROLL_R, THIGH_PITCH_L, THIGH_PITCH_R, KNEE_PITCH_L, KNEE_PITCH_R,

```



```

    ANKLE_PITCH_L, ANKLE_PITCH_R, ANKLE_ROLL_L, ANKLE_ROLL_R, VOICE, GYROXGAIN, GYROYGAIN,
    CTRL_SHOOTANALOG, GAINMOVE_PITCH_L, GAINMOVE_PITCH_R, GAINLSTEP_ROLL_L,
    GAINLSTEP_ROLL_R, GAINRSTEP_ROLL_L, GAINRSTEP_ROLL_R, PACEGAIN_LEG_L, PACEGAIN_LEG_R,
    CTRL_TURN_L, CTRL_TURN_R};

/*Array wich contains the initial offset*/
WORD offs[] = {TIME, HEAD_YAW, EYE_LED_L, EYE_LED_R, SHOULDER_PITCH_L, SHOULDER_PITCH_R,
    SHOULDER_ROLL_L, SHOULDER_ROLL_R, ELBOW_ROLL_L, ELBOW_ROLL_R, THIGH_ROLL_L,
    THIGH_ROLL_R, THIGH_PITCH_L, THIGH_PITCH_R, KNEE_PITCH_L, KNEE_PITCH_R, ANKLE_PITCH_L,
    ANKLE_PITCH_R, ANKLE_ROLL_L, ANKLE_ROLL_R, VOICE, GYROXGAIN, GYROYGAIN,
    CTRL_SHOOTANALOG, GAINMOVE_PITCH_L, GAINMOVE_PITCH_R, GAINLSTEP_ROLL_L,
    GAINLSTEP_ROLL_R, GAINRSTEP_ROLL_L, GAINRSTEP_ROLL_R, PACEGAIN_LEG_L, PACEGAIN_LEG_R,
    CTRL_TURN_L, CTRL_TURN_R};

char audio[100] = "";

/*Contains all the blocks which constitute the movement*/
struct motionSequence
{
    int type; //1: pose, 2: calculation, 3: if
    int next1;
    int next2;
    int info[INFO_NUM];
    char sound[100];
} motions[MAX_MOTIONS];

char FILE_OUTPUT[] = "output.txt";
FILE *fp;

/* state:
    0 -> OK
    1 -> call macro
    2 -> exit
*/
int state = 0;

char macroName[15];

extern FILE * yyin;
extern char *yylineno;
extern char *yytext;

int current = 2;
int counter = 0;
int loopStack[MAX_LOOPS];

/*Converts from degrees to hex*/
int degToOffset(int deg)
{
    int offs = deg * 0xda;
    return offs;
}

```

```

void yyerror(const char *str)
{
    fprintf(stdout, "%s: %s\n", str, yytext);
    return;
}

int yywrap()
{
    return 1;
}

/*Saves joint values into the struct*/
void savePose()
{
    int i, succ;
    motions[current].type = 1;
    for (i = 0; i < INFO_NUM; i++)
        motions[current].info[i] = actualPosition[i];
    char audioPwd[30] = "X Voice Files\\";
    if (strcmp(audio, "") != 0)
    {
        strcat(audioPwd, audio);
        strcat(audioPwd, ".wav");
        strcpy(motions[current].sound, audioPwd);
        strcpy(audio, "");
    }

    if (isOnline())
        execPose( actualPosition[i]);

    succ = current + 1;
    motions[current].next1 = succ;
    current++;
    actualPosition[time] = 50;
}

/*Saves a calculation block into the struct*/
void saveCalculation(int op, int index, int value)
{
    int succ;
    motions[current].type = 2;
    motions[current].info[1] = op;
    motions[current].info[2] = index;
    motions[current].info[3] = value;
    succ = current + 1;
    motions[current].next1 = succ;
    current++;
}

/*Saves an if block into the struct*/
void saveIf(int op, int index, int value, int jump)
{
    int succ;
    motions[current].type = 3;
    motions[current].info[1] = op;
    motions[current].info[2] = index;
    motions[current].info[3] = value;
    motions[current].next2 = jump;
}

```

```

succ = current + 1;
motions[current].next1 = succ;
current++;
}

/*Prints in text file output.txt the commands for Robovie-X*/
void toString()
{
    int i;

    fprintf(fp, "_MOTION_FORMAT:[ POSEDEF_V1R4 ]\n_ENABLE_AXIS:[62]\n\n_MOTION_INFO
: [[]] - [50] - [10] - [6] - [0] - [1] - [-1]\n_STARTMOT:\n\n");
    fprintf(fp, "_MOTION_INFO: [[POSE1]] - [50] - [43] - [6] - [6] - [2] - [-1]\n_POSE: [50] - [0x0000
, 0x0aa5, 0x3e97, 0xf5c8, 0x0000, 0X0000, 0x0000, 0xf55b, 0xc169, 0x0a38, 0x0000, 0X0000, 0
x0000, 0X0000, 0x0000, 0x1d61, 0x3680, 0x0000, 0x0000, 0X0000, 0X0000, 0xe29f, 0xc980, 0
x0000, 0X0000, 0X0000, 0x0000, 0X0000, 0X0000, 0X0000, 0x0000, 0x00c8, 0x0100, 0X0000, 0
X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0x0000, 0x0000, 0x0000, 0
x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0x0000, 0X0000, 0x0000, 0X0000, 0X0000, 0
X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0X0000
] - [[]] - [000000000000000000000000000000]\n\n");

    for (i = 2; i < current; i++)
    {
        switch (motions[i].type) {

            case 1 :
                fprintf(fp, "_MOTION_INFO: [[POSE%d]] - [50] - [%d] - [6] - [6] - [%d] - [-1]\n", i, i
*50, motions[i].next1);
                fprintf(fp, "_POSE: [%d] - [0x%04x, 0x%04x, 0x%04x, 0x%04x, 0x%04x, 0X0000, 0x%04x, 0
x%04x, 0x%04x, 0x%04x, 0x%04x, 0X0000, 0x%04x, 0X0000, 0X0000, 0x%04x, 0x%04x, 0x
%04x, 0x%04x, 0X0000, 0X0000, 0x%04x, 0x%04x, 0x%04x, 0X0000, 0X0000, 0x%04x, 0
X0000, 0X0000, 0X0000, 0x0000, 0x%04x, 0x%04x, 0X0000, 0X0000, 0X0000, 0X0000, 0
X0000, 0X0000, 0X0000, 0X0000, 0X0000, 0x%04x, 0x%04x, 0x%04x, 0x%04x, 0x%04x, 0x
%04x, 0x%04x, 0x%04x, 0x%04x, 0x%04x, 0X0000, 0x%04x, 0X0000, 0X0000, 0X0000, 0
X0000, 0X0000, 0X0000, 0X0000, 0X0000] - [%s
]] - [00000000000000000000000000000000]\n\n", motions[i].info[time], motions
[i].info[anRollR], motions[i].info[anPitchR], motions[i].info[knPitchR],
motions[i].info[thPitchR], motions[i].info[thRollR], motions[i].info[
anRollL], motions[i].info[anPitchL], motions[i].info[knPitchL], motions[
i].info[thPitchL], motions[i].info[thRollL], motions[i].info[eyeL],
motions[i].info[elRollR], motions[i].info[shRollR], motions[i].info[
shPitchR], motions[i].info[eyeR], motions[i].info[elRollL], motions[i].
info[shRollL], motions[i].info[shPitchL], motions[i].info[head], motions[
i].info[gyroX], motions[i].info[gyroY], motions[i].info[mvPitchL],
motions[i].info[mvPitchR], motions[i].info[lStepRollL], motions[i].info[
lStepRollR], motions[i].info[rStepRollL], motions[i].info[rStepRollR],
motions[i].info[paceLegL], motions[i].info[paceLegR], motions[i].info[
turnL], motions[i].info[turnR], motions[i].info[shootAn], motions[i].
sound);
                break;

            case 2 :
                fprintf(fp, "_MOTION_INFO: [[CALCULATION%d]] - [50] - [%d] - [6] - [6] - [%d] - [-1]\n",
i, i*50, motions[i].next1);
                fprintf(fp, "_VARCON: [%d] - [%d] - [%d]\n\n", motions[i].info[1], motions[i].
info[2], motions[i].info[3]);
                break;
        }
    }
}

```



```
    }
    else if (strcmp(op, "STOP") == 0)
    {
        counter--;
        saveCalculation(2, 80 + counter, 1);
        saveIf(4, 80 + counter, 0, loopStack[counter]);
    }
}

/*MAIN*/
int main(void)
{
    fp = fopen(FILE_OUTPUT, "w");

    while(state==0 )
    {
        yyin = stdin;

        yyrestart(yyin);

        yyparse();

        if ( state == 1)
        {
            yyin = fopen(macroName,"r");

            yyrestart(yyin);

            yyparse();

            state = 0;
            fclose(yyin);
        }
        yylex_destroy();
    }

    fclose(fp);
    return 0;
}

%}

%token TURN STOP EXIT AND DOT BRIGHTNESS PARAMETER PLAY STEP FRONTFLIP BACKFLIP
%token DEGREE MILLISECOND ININ TIMES REPEAT RESET GREETINGS YAWN WALK DANCE GUFFU
GUFFD
%token SAVECMD EXECCMD

%union
{
    int number;
    char * string;
}
```

```

%token <number> NUMBER
%token <number> HEAD
%token <number> EYE
%token <number> SECTION
%token <number> JOINT
%token <number> OFFS
%token <number> TO
%token <number> SIGN
%token <string> TEXT
%token <number> ONLINE

%%

commands:
    | commands command
    ;
errore: error { printf("Error ");};

command:
stop | turn | online | macro | exit | turn AND | turn DOT {savePose();}
| turn tempo | turn tempo AND | turn tempo DOT {savePose();}
| turn audio | turn audio AND | turn audio DOT {savePose();}
| turn tempo audio | turn tempo audio AND | turn tempo audio DOT {savePose();}
| turn audio tempo | turn audio tempo AND | turn audio tempo DOT {savePose();}
| repeat | reset | movements
;

repeat:
    REPEAT NUMBER TIMES
    {
        saveCalculation(0, 80 + counter, $2);
        loopStack[counter] = current;
        counter++;
    }
;

online:
    ONLINE
    {
        if ($1)
        {
            char check = goOnline();
        }
        else
        {
            printf("vado offline");
            goOffline();
        }
    }
;

stop:
    STOP
    {
        counter--;
        saveCalculation(2, 80 + counter, 1);
        saveIf(4, 80 + counter, 0, loopStack[counter]);
    }

```

```

    if (isOnline())
    {
        int k;

        for(k=loopStack[counter] ; k < current ; k++)
        {
            if(motions[k].type == 1)
                execPose(motions[k].info[0]);
        }
    }
}
;

macro:
SAVECMD TEXT
{
    FILE * fpmacro = fopen($2, "w");
    printf("Insert command list \nHit enter to quit\n");
    char str[80];
    do {
        //printf("Enter a string (CR to quit):\n");
        gets(str);
        strcat(str, "\n"); /* add a newline */
        fputs(str, fpmacro);
    } while(*str!='\n');

    printf("Macro Saved\n");

    fclose(fpmacro);
}

|EXEC CMD TEXT
{
    state = 1;

    strcpy(macroName, $2);

    return;
}
;

turn:
TURN HEAD OFFS NUMBER DEGREE TO
{
    int joint = $2;
    if ($3)
    {
        if ($6 == 4) {actualPosition[joint] += degToOffset($4);}
        else if ($6 == 5) {actualPosition[joint] -= degToOffset($4);}
    }
    else
    {
        if ($6 == 4) {actualPosition[joint] = degToOffset($4) + offs[joint];}
        else if ($6 == 5) {actualPosition[joint] = degToOffset(-($4)) + offs[joint];}
    }
}

```

```

|SIGN SECTION EYE BRIGHTNESS OFFS NUMBER DEGREE
{
    int side = $2;
    int joint = $3;

    if(side && joint != 1)
        {joint++;}

    if ($1 == 2)
        {actualPosition[joint] = ($6 * 0xffff / 0x64) - 0x8000 + offs[joint];}
    else
    {
        if ($1 == 0) {actualPosition[joint] += ($6 * 0xffff / 0x64);}
        else if ($1 == 1) {actualPosition[joint] -= ($6 * 0xffff / 0x64);}
    }
}

|TURN SECTION JOINT OFFS NUMBER DEGREE TO
{
    int side = $2;
    int joint = $3;

    if(side)
        {joint++;}

    switch(joint)
    {
    case 4:
        if ($4)
        {
            if ($7 == 0) {actualPosition[joint] += degToOffset($5);}
            else if ($7 == 1) {actualPosition[joint] -= degToOffset($5);}
        }
        else
        {
            if ($7 == 0) {actualPosition[joint] = degToOffset($5) + offs[joint];}
            else if ($7 == 1) {actualPosition[joint] = degToOffset(-($5)) + offs[
                joint];}
        }
        break;
    case 5:
        if ($4)
        {
            if ($7 == 0) {actualPosition[joint] -= degToOffset($5);}
            else if ($7 == 1) {actualPosition[joint] += degToOffset($5);}
        }
        else
        {
            if ($7 == 0) {actualPosition[joint] = degToOffset(-($5)) + offs[joint];}
            else if ($7 == 1) {actualPosition[joint] = degToOffset($5) + offs[joint
                ];}
        }
        break;
    case 6:
        if ($4)

```



```

    {
        if ($7 == 2) {actualPosition[joint] -= degToOffset($5);}
        else if ($7 == 3) {actualPosition[joint] += degToOffset($5);}
    }
    else
    {
        if ($7 == 2) {actualPosition[joint] = degToOffset(-($5)) + offs[joint];}
        else if ($7 == 3) {actualPosition[joint] = degToOffset($5) + offs[joint]
            };}
    }
    break;
case 7:
    if ($4)
    {
        if ($7 == 2) {actualPosition[joint] += degToOffset($5);}
        else if ($7 == 3) {actualPosition[joint] -= degToOffset($5);}
    }
    else
    {
        if ($7 == 2) {actualPosition[joint] = degToOffset($5) + offs[joint];}
        else if ($7 == 3) {actualPosition[joint] = degToOffset(-($5)) + offs[
            joint];}
    }
    break;

case 8:
    if ($4)
    {
        if ($7 == 2) {actualPosition[joint] += degToOffset($5);}
        else if ($7 == 3) {actualPosition[joint] -= degToOffset($5);}
    }
    else
    {
        if ($7 == 2) {actualPosition[joint] = degToOffset($5) + offs[joint];}
        else if ($7 == 3) {actualPosition[joint] = degToOffset(-($5)) + offs[
            joint];}
    }
    break;
case 9:
    if ($4)
    {
        if ($7 == 2) {actualPosition[joint] -= degToOffset($5);}
        else if ($7 == 3) {actualPosition[joint] += degToOffset($5);}
    }
    else
    {
        if ($7 == 2) {actualPosition[joint] = degToOffset(-($5)) + offs[joint];}
        else if ($7 == 3) {actualPosition[joint] = degToOffset($5) + offs[joint]
            };}
    }
    break;

case 10:
    if ($4)
    {
        if ($7 == 2) {actualPosition[joint] += degToOffset($5);}
        else if ($7 == 3) {actualPosition[joint] -= degToOffset($5);}
    }
}

```

```

else
{
    if ($7 == 2) {actualPosition[joint] = degToOffset($5) + offs[joint];}
    else if ($7 == 3) {actualPosition[joint] = degToOffset(-($5)) + offs[
        joint];}
}
break;

case 11:
if ($4)
{
    if ($7 == 2) {actualPosition[joint] -= degToOffset($5);}
    else if ($7 == 3) {actualPosition[joint] += degToOffset($5);}
}
else
{
    if ($7 == 2) {actualPosition[joint] = degToOffset(-($5)) + offs[joint];}
    else if ($7 == 3) {actualPosition[joint] = degToOffset($5) + offs[joint
        ];}
}
break;

case 12:
if ($4)
{
    if ($7 == 0) {actualPosition[joint] += degToOffset($5);}
    else if ($7 == 1) {actualPosition[joint] -= degToOffset($5);}
}
else
{
    if ($7 == 0) {actualPosition[joint] = degToOffset ($5)+ offs[joint];}
    else if ($7 == 1) {actualPosition[joint] = degToOffset(-($5)) + offs[
        joint];}
}
break;

case 13:
if ($4)
{
    if ($7 == 0) {actualPosition[joint] -= degToOffset($5);}
    else if ($7 == 1) {actualPosition[joint] += degToOffset($5);}
}
else
{
    if ($7 == 0) {actualPosition[joint] = degToOffset(-($5)) + offs[joint];}
    else if ($7 == 1) {actualPosition[joint] = degToOffset($5) + offs[joint
        ];}
}
break;

case 14:
if ($4)
{
    if ($7 == 0) {actualPosition[joint] -= degToOffset($5);}
    else if ($7 == 1) {actualPosition[joint] += degToOffset($5);}
}
else
{
    if ($7 == 0) {actualPosition[joint] = degToOffset(-($5)) + offs[joint];}

```

```
        else if ($7 == 1) {actualPosition[joint] = degToOffset($5) + offs[joint
        ];}
    }
    break;

case 15:
    if ($4)
    {
        if ($7 == 0) {actualPosition[joint] += degToOffset($5);}
        else if ($7 == 1) {actualPosition[joint] -= degToOffset($5);}
    }
    else
    {
        if ($7 == 0) {actualPosition[joint] = degToOffset($5) + offs[joint];}
        else if ($7 == 1) {actualPosition[joint] = degToOffset(-($5)) + offs[
        joint];}
    }
    break;

case 16:
    if ($4)
    {
        if ($7 == 0) {actualPosition[joint] -= degToOffset($5);}
        else if ($7 == 1) {actualPosition[joint] += degToOffset($5);}
    }
    else
    {
        if ($7 == 0) {actualPosition[joint] = degToOffset(-($5)) + offs[joint];}
        else if ($7 == 1) {actualPosition[joint] = degToOffset($5) + offs[joint
        ];}
    }
    break;

case 17:
    if ($4)
    {
        if ($7 == 0) {actualPosition[joint] += degToOffset($5);}
        else if ($7 == 1) {actualPosition[joint] -= degToOffset($5);}
    }
    else
    {
        if ($7 == 0) {actualPosition[joint] = degToOffset($5) + offs[joint];}
        else if ($7 == 1) {actualPosition[joint] = degToOffset(-($5)) + offs[
        joint];}
    }
    break;

case 18:
    if ($4)
    {
        if ($7 == 2) {actualPosition[joint] -= degToOffset($5);}
        else if ($7 == 3) {actualPosition[joint] += degToOffset($5);}
    }
    else
    {
        if ($7 == 2) {actualPosition[joint] = degToOffset(-($5)) + offs[joint];}
        else if ($7 == 3) {actualPosition[joint] = degToOffset($5) + offs[joint
        ];}
    }

```

```

    }
    break;

case 19:
    if ($4)
    {
        if ($7 == 2) {actualPosition[joint] += degToOffset($5);}
        else if ($7 == 3) {actualPosition[joint] -= degToOffset($5);}
    }
    else
    {
        if ($7 == 2) {actualPosition[joint] = degToOffset($5) + offs[joint];}
        else if ($7 == 3) {actualPosition[joint] = degToOffset(-($5)) + offs[
            joint];}
    }
    break;
}
}

|SIGN JOINT PARAMETER OFFS NUMBER
{
    int joint = $2;

    switch(joint)
    {
    case 21:
    case 22: if ($1 == 2)
        actualPosition[joint] = $5 ;
        else
        {
            if ($1 == 0) {actualPosition[joint] += $5;}
            else if ($1 == 1) {actualPosition[joint] -= $5;}
        }
        if (actualPosition[joint] > 256) actualPosition[joint] = 256;
        break;
    case 23: if ($1 == 2)
        actualPosition[joint] = $5 ;
        else
        {
            if ($1 == 0) {actualPosition[joint] += $5;}
            else if ($1 == 1) {actualPosition[joint] -= $5;}
        }
        break;
    }
}

|SIGN SECTION JOINT PARAMETER OFFS NUMBER
{
    int side = $2;
    int joint = $3;

    if(side && joint != 1)
        {joint++;}

    switch(joint)
    {
    case 23:
    case 24:

```

```
case 25:
case 26:
case 27:
case 28:
case 29:
case 30:
case 31:
case 32:
case 33: if ($1 == 2)
    actualPosition[joint] = $6 ;
    else
    {
        if ($1 == 0) {actualPosition[joint] += $6;}
        else if ($1 == 1) {actualPosition[joint] -= $6;}
    }
    break;
}
}
;

tempo:
    ININ NUMBER MILLISECOND
    {
        actualPosition[time] = $2 * 0.06;
        if (actualPosition[0] < 1) actualPosition[0] = 1;
        else if (actualPosition[0] > 239) actualPosition[0] = 239;
    }
;

audio:
    PLAY TEXT
    {
        strcpy(audio, $2);
    }
;

reset:
    RESET
    {
        int i;
        for (i = 0; i < INFO_NUM; i++)
            actualPosition[i] = offs[i];

        if (isOnline())
            execPose( & actualPosition[0]);
    }
;

movements:
    GREETINGS
    {
        loadFile("./\movs\greetings.txt");
    }

    |YAWN
    {
        loadFile("./\movs\yawn.txt");
    }
;
```

```

}

|FRONTFLIP
{
  loadFile(".\\movs\\frontflip.txt");
}

|BACKFLIP
{
  loadFile(".\\movs\\backflip.txt");
}

|DANCE
{
  loadFile(".\\movs\\dance.txt");
}

|GUFFU
{
  loadFile(".\\movs\\gettingupffup.txt");
}

|GUFFD
{
  loadFile(".\\movs\\gettingupffdown.txt");
}

|WALK NUMBER STEP TO
{
  switch($4) {
  case 0:
    loadFile(".\\movs\\goahead1.txt");
    saveCalculation(0, 80 + counter, ($2 - 1));
    loopStack[counter] = current;
    counter++;
    loadFile(".\\movs\\goahead2.txt");
    counter--;
    saveCalculation(2, 80 + counter, 1);
    saveIf(4, 80 + counter, 0, loopStack[counter]);
    loadFile(".\\movs\\goahead3.txt");
    break;
  case 1:
    loadFile(".\\movs\\goback1.txt");
    saveCalculation(0, 80 + counter, $2);
    loopStack[counter] = current;
    counter++;
    loadFile(".\\movs\\goback2.txt");
    counter--;
    saveCalculation(2, 80 + counter, 1);
    saveIf(4, 80 + counter, 0, loopStack[counter]);
    loadFile(".\\movs\\goback3.txt");
    break;
  case 4:
    loadFile(".\\movs\\goright1.txt");
    saveCalculation(0, 80 + counter, $2);
    loopStack[counter] = current;
    counter++;
    loadFile(".\\movs\\goright2.txt");

```

```

        counter--;
        saveCalculation(2, 80 + counter, 1);
        saveIf(4, 80 + counter, 0, loopStack[counter]);
        loadFile(".\\movs\\goright3.txt");
        break;
    case 5:
        loadFile(".\\movs\\goleft1.txt");
        saveCalculation(0, 80 + counter, $2);
        loopStack[counter] = current;
        counter++;
        loadFile(".\\movs\\goleft2.txt");
        counter--;
        saveCalculation(2, 80 + counter, 1);
        saveIf(4, 80 + counter, 0, loopStack[counter]);
        loadFile(".\\movs\\goleft3.txt");
        break;
    }
}
;

exit:
    EXIT
    {
        toString();

        state = 2;

        return;
    }
;
%%

```

## A.3 OnlineMode & rcCom

Listing A.3: rcCom.h

```

#include <windows.h>

#include <stdbool.h>

//i define se vono a definire la posizione iniziale
//una posizione iniziale caricata da file viene determinata
//a partire da questa (che È salvata nella rom del robot)
#define BYTES_TO_READ          65
#define BYTES_TO_WRITE        65
#define MAX_BUF_SIZE          64

#define HEAD_YAW                0x0000
#define EYE_LED_L               0X0000
#define EYE_LED_R               0X0000
#define SHOULDER_PITCH_L        0x0000
#define SHOULDER_PITCH_R        0x0000
#define SHOULDER_ROLL_L         0xC980
#define SHOULDER_ROLL_R         0x3680
#define ELBOW_ROLL_L            0XE29F

```

```

#define ELBOW_ROLL_R          0X1D61
#define THIGH_ROLL_L          0X0000
#define THIGH_ROLL_R          0X0000
#define THIGH_PITCH_L         0x0A38
#define THIGH_PITCH_R         0xF5C8
#define KNEE_PITCH_L          0xC169
#define KNEE_PITCH_R          0x3E97
#define ANKLE_PITCH_L         0xF55B
#define ANKLE_PITCH_R         0x0AA5
#define ANKLE_ROLL_L          0x0000
#define ANKLE_ROLL_R          0x0000
#define VOICE                  0x0000
#define TIME                   0x0001

//prototipi delle funzioni disponibili
void doMovements(void);
bool init(void);
void off(void);
bool rcOpen();
bool rcWrite(BYTE *buffer, int bToWrite, DWORD *bWritten);
bool rcRead(BYTE *buffer, int bToRead, DWORD *bRead);

```

Listing A.4: rcCom.c

```

#include <windows.h>
//includo le funzioni necessarie per gestire gli HID
#include <ddk\hidsdi.h>
#include <setupapi.h>

#include <stdbool.h>

#define MAX_BUF_SIZE 64

//specifiche del Robovie-X
#define VENDOR_ID 0x1962
#define PRODUCT_ID 0x1000
#define VERSION_NUMBER 0x0100

//GUID: Globally Unique Identifier, struttura che identifica univocamente un device
static GUID Guid;
static HDEVINFO devInfoSet;

//handle al file che permette di comunicare con in robot
HANDLE hObject;

//apre la connessione con il robovie-x se questo è collegato, altrimenti ritorna 0
bool rcOpen()
{
    SP_DEVICE_INTERFACE_DATA interfaceData;
    SP_DEVICE_INTERFACE_DETAIL_DATA_A *interfaceDetailData;
    interfaceData.cbSize=0x1c;
    DWORD ifaceCounter = 0;
    bool devDetail;
    DWORD reqSize = 0;
    HIDD_ATTRIBUTES attributes;
    DWORD error;

```



```

HidD_GetHidGuid(&Guid);
devInfoSet = SetupDiGetClassDevsA(&Guid, 0, 0, 0x12);
//questo ciclo permette di identificare il robot in presenza di altri HID
  collegati al pc
while(ifaceCounter<0x1000)
{
printf("ifaceCounter = %d\n", ifaceCounter);
hObject = NULL;
devDetail = SetupDiEnumDeviceInterfaces(devInfoSet, 0, &Guid, ifaceCounter, &
  interfaceData);
error = GetLastError();
printf("errore: %0x\n", error);
if(devDetail!=0)
{
  /* questa prima chiamata serve a ottenere la reqSize Corretta, è normale che
    termini con l'errore
      0x7A: ERROR_INSUFFICIENT_BUFFER
  */
  devDetail = SetupDiGetDeviceInterfaceDetailA(devInfoSet, &interfaceData, NULL
    , 0, &reqSize, NULL);
  error = GetLastError();
  printf("errore: %0x\n", error);
  printf("reqSize = %d\n", reqSize);
  //la chiamata malloc(reqSize) serve a riservare il giusto spazio in memoria
    per la struttura DEVICE_INTERFACE_DETAIL_DATA_A
  interfaceDetailData = (struct _SP_DEVICE_INTERFACE_DETAIL_DATA_A *)malloc(
    reqSize);
  //cbSize=5 dipende da un problema di windows, non deve essere cambiato
  interfaceDetailData->cbSize=5;
  //ottiene informazioni dettagliate sull'interfaccia
  devDetail = SetupDiGetDeviceInterfaceDetailA(devInfoSet, &interfaceData,
    interfaceDetailData, reqSize, 0, 0);
  error = GetLastError();
  printf("errore: %0x\n", error);
  if(devDetail!=0)
  {
    //crea il file che permette di comunicare col device
    hObject=CreateFile(interfaceDetailData->DevicePath, 0x0c000000, 3, 0, 3,
      0, 0);
    if(hObject!=INVALID_HANDLE_VALUE)
    {
      //mostra i dettagli del device e controlla che corrispondano a quelli del
        robovie-x in modo da distinguere
      //tra vari possibili hid collegati (quali mouse usb, tastiere usb
        joystick etc...)
      printf("file creato\n");
      devDetail = HidD_GetAttributes(hObject, &attributes);
      //mostra i dettagli del device
      printf("size = %0x\n", attributes.Size);
      printf("VendorID = %0x\n", attributes.VendorID);
      printf("ProductID = %0x\n", attributes.ProductID);
      printf("VersionNumber = %0x\n", attributes.VersionNumber);
      if(devDetail!=0)
      {
        //controlla che il device trovato sia quello giusto
        if((attributes.VendorID==VENDOR_ID) && (attributes.ProductID==
          PRODUCT_ID) && (attributes.VersionNumber>=VERSION_NUMBER))
        {

```

```

        //se sono verificate le condizioni dell'if precedente allora il
        device trovato è proprio il robovie-x
        devDetail=1;
        break;
    }
    else
    {
        devDetail=0;
        ifaceCounter++;
        continue;
    }
}
else
{
    ifaceCounter++;
    continue;
}
}
else
{
    ifaceCounter++;
    continue;
}
}
else
{
    ifaceCounter++;
    continue;
}
}
else
{
    //esce dal ciclo perchè non sono stati trovati device collegati
    break;
}
}
//se è stato riservato dello spazio per DEVICE_INTERFACE_DETAIL_DATA_A allora
questo viene liberato
//dal momento che la struttura non è più necessaria
if(interfaceDetailData->cbSize != 0)
{
    free(interfaceDetailData);
}
//elimina la lista dei device
SetupDiDestroyDeviceInfoList(devInfoSet);
return devDetail;
}

//invia bToWrite byte contenuti in memoria a partire dalla locazione puntata da
buffer
bool rcWrite(BYTE *buffer, int bToWrite, DWORD *bWritten)
{
    //WriteFile(HANDLE, LPCVOID, DWORD, LPDWORD, LP_OVRLAPPED)
    bool returned = WriteFile(hObject, buffer, bToWrite, bWritten, 0);
    if(returned==0)
    {
        printf("errore in scrittura: %0x\n", GetLastError());
    }
}

```

```

    return returned;
}
//riceve bToRead byte contenuti in memoria a partire dalla locazione puntata da
//buffer
bool rcRead(BYTE *buffer, int bToRead, DWORD *bRead)
{
    //ReadFile(HANDLE, LPCVOID, DWORD, LPDWORD, LPOVERLAPPED)
    bool returned = ReadFile(hObject, buffer, bToRead, bRead, 0);
    if(returned == 0)
    {
        printf("errore in lettura: %0x\n", GetLastError());
    }
    return returned;
}

```

Listing A.5: move.h

```

#include <windows.h>

//i define servono a definire la posizione iniziale
//una posizione iniziale caricata da file viene determinata
//a partire da questa (che è salvata nella rom del robot)
#define BYTES_TO_READ          65
#define BYTES_TO_WRITE        65
#define MAX_BUF_SIZE          64

#define HEAD_YAW               0x0000
#define EYE_LED_L              0x0000
#define EYE_LED_R              0x0000
#define SHOULDER_PITCH_L       0x0000
#define SHOULDER_PITCH_R       0x0000
#define SHOULDER_ROLL_L        0xC980
#define SHOULDER_ROLL_R        0x3680
#define ELBOW_ROLL_L           0xE29F
#define ELBOW_ROLL_R           0x1D61
#define THIGH_ROLL_L           0x0000
#define THIGH_ROLL_R           0x0000
#define THIGH_PITCH_L          0x0A38
#define THIGH_PITCH_R          0xF5C8
#define KNEE_PITCH_L           0xC169
#define KNEE_PITCH_R           0x3E97
#define ANKLE_PITCH_L          0xF55B
#define ANKLE_PITCH_R          0x0AA5
#define ANKLE_ROLL_L           0x0000
#define ANKLE_ROLL_R           0x0000
#define VOICE                   0x0000
#define TIME                    0x0001

BYTE toRobot[MAX_BUF_SIZE], toRobot1[MAX_BUF_SIZE], toRobot2[MAX_BUF_SIZE],
toRobot3[MAX_BUF_SIZE], fromRobot[MAX_BUF_SIZE];

//prototipi delle funzioni disponibili
bool init(void);
bool doMovements(void);
bool off(void);

```

Listing A.6: move.c

```

#include <windows.h>
#include <stdio.h>
#include <stdbool.h>

#include "move.h"

//struttura con la posizione di tutti i giunti
struct joints
{
    int head;
    int eyeL;
    int eyeR;
    int shPitchL;
    int shPitchR;
    int shRollL;
    int shRollR;
    int elRollL;
    int elRollR;
    int thRollL;
    int thRollR;
    int thPitchL;
    int thPitchR;
    int knPitchL;
    int knPitchR;
    int anPitchL;
    int anPitchR;
    int anRollL;
    int anRollR;
    int voice;
    int time;
};

struct joints initial = {HEAD_YAW, EYE_LED_L, EYE_LED_R, SHOULDER_PITCH_L,
    SHOULDER_PITCH_R, SHOULDER_ROLL_L, SHOULDER_ROLL_R,
    ELBOW_ROLL_L, ELBOW_ROLL_R, THIGH_ROLL_L, THIGH_ROLL_R, THIGH_PITCH_L,
    THIGH_PITCH_R, KNEE_PITCH_L,
    KNEE_PITCH_R, ANKLE_PITCH_L, ANKLE_PITCH_R, ANKLE_ROLL_L,
    ANKLE_ROLL_R, VOICE, TIME};

//struct joints actualPosition = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

DWORD bytesWritten;
DWORD bytesRead;

bool init()
{
    int j = 0;
    int i = 0;
    while(i<MAX_BUF_SIZE)
    {
        *(toRobot+i) = 0x00;
        i++;
    }
    toRobot[1] = 0x55;
    toRobot[2] = 0x01;
    toRobot[3] = 0xE0;
    toRobot[4] = 0x00;
}

```

```
toRobot [5] = 0x10;
toRobot [6] = 0x04;
toRobot [7] = 0x00;
toRobot [8] = 0x0b;
toRobot [9] = 0x20;
toRobot [10] = 0x00;
toRobot [11] = 0x8A;
toRobot [12] = 0xEC;
toRobot [13] = 0x08;
toRobot [14] = 0x09;
toRobot [15] = 0xE0;
toRobot [16] = 0x00;
toRobot [17] = 0x20;
toRobot [18] = 0x04;
toRobot [19] = 0x14;
toRobot [20] = 0x04;
rcWrite(toRobot, BYTES_TO_WRITE, &bytesWritten);
//prima di mandare un nuovo pacchetto legge le risposte del robot, queste sono in
    numero variabile, ma comunque
//sempre minore di 15
while(j<=15)
{
    rcRead(fromRobot, BYTES_TO_READ, &bytesRead);
    j++;
}
j=0;
toRobot [13] = 0x14;
toRobot [14] = 0x11;
toRobot [17] = 0x28;
toRobot [18] = 0x0E;
toRobot [19] = 0xC0;
toRobot [20] = 0x01;
toRobot [21] = 0x2E;
toRobot [22] = 0x02;
toRobot [23] = 0xE0;
toRobot [24] = 0x00;
toRobot [25] = 0x04;
toRobot [26] = 0x02;
toRobot [27] = 0x0E;
toRobot [28] = 0x04;
toRobot [29] = 0xC0;
toRobot [30] = 0x05;
toRobot [31] = 0x04;
toRobot [32] = 0x10;
rcWrite(toRobot, BYTES_TO_WRITE, &bytesWritten);
while(j<=15)
{
    rcRead(fromRobot, BYTES_TO_READ, &bytesRead);
    j++;
}
j=0;
toRobot [13] = 0x06;
toRobot [14] = 0x19;
toRobot [16] = 0x01;
toRobot [17] = 0xC0;
toRobot [18] = 0x40;
toRobot [19] = 0x00;
toRobot [20] = 0x00;
```

```

rcWrite(toRobot, BYTES_TO_WRITE, &bytesWritten);
while(j<=15)
{
    rcRead(fromRobot, BYTES_TO_READ, &bytesRead);
    j++;
}
j=0;
toRobot[14] = 0x21;
toRobot[16] = 0x02;
toRobot[17] = 0x00;
toRobot[18] = 0x10;
while(toRobot[14]<=0x09)
{
    toRobot[14] += 0x08;
    toRobot[17] += 0x10;
    if(toRobot[17] == 0x00)
    {
        toRobot[16]++;
    }
    rcWrite(toRobot, BYTES_TO_WRITE, &bytesWritten);
    while(j<=15)
    {
        rcRead(fromRobot, BYTES_TO_READ, &bytesRead);
        j++;
    }
    j=0;
}
toRobot[13] = 0x05;
toRobot[14] = 0x11;
toRobot[15] = 0x85;
toRobot[16] = 0x00;
toRobot[17] = 0xFD;
toRobot[18] = 0x00;
toRobot[19] = 0x00;
toRobot[20] = 0x00;
rcWrite(toRobot, BYTES_TO_WRITE, &bytesWritten);
while(j<=15)
{
    rcRead(fromRobot, BYTES_TO_READ, &bytesRead);
    j++;
}
j=0;
toRobot[13] = 0x08;
toRobot[14] = 0x19;
toRobot[15] = 0xE0;
toRobot[16] = 0x00;
toRobot[17] = 0x08;
toRobot[18] = 0x04;
toRobot[19] = 0x02;
toRobot[20] = 0x02;
toRobot[21] = 0x00;
toRobot[22] = 0x00;
rcWrite(toRobot, BYTES_TO_WRITE, &bytesWritten);
while(j<=15)
{
    rcRead(fromRobot, BYTES_TO_READ, &bytesRead);
    j++;
}

```

```
j=0;
toRobot [5] = 0x08;
toRobot [7] = 0x00;
toRobot [8] = 0x00;
toRobot [9] = 0x00;
toRobot [10] = 0x00;
toRobot [11] = 0x02;
toRobot [12] = 0x02;
toRobot [13] = 0x00;
toRobot [14] = 0x00;
toRobot [15] = 0x00;
toRobot [16] = 0x00;
rcWrite(toRobot, BYTES_TO_WRITE, &bytesWritten);
while(j<=15)
{
    rcRead(fromRobot, BYTES_TO_READ, &bytesRead);
    j++;
}
j=0;
toRobot [1] = 0x55;
toRobot [2] = 0x01;
toRobot [3] = 0xE0;
toRobot [4] = 0x00;
toRobot [5] = 0x10;
toRobot [6] = 0x04;
toRobot [7] = 0x00;
toRobot [8] = 0x0B;
toRobot [9] = 0x20;
toRobot [10] = 0x00;
toRobot [11] = 0x8A;
toRobot [12] = 0xEC;
toRobot [13] = 0x0F;
toRobot [14] = 0x21;
toRobot [15] = 0xE0;
toRobot [16] = 0x01;
toRobot [17] = 0x82;
toRobot [18] = 0x02;
toRobot [19] = 0x86;
toRobot [20] = 0x7C;
toRobot [21] = 0x80;
toRobot [22] = 0x81;
toRobot [23] = 0x5E;
toRobot [24] = 0x02;
toRobot [25] = 0x80;
toRobot [26] = 0x00;
toRobot [27] = 0x18;
toRobot [28] = 0x00;
toRobot [29] = 0x00;
toRobot [30] = 0x05;
toRobot [31] = 0x04;
toRobot [32] = 0x10;
rcWrite(toRobot, BYTES_TO_WRITE, &bytesWritten);
while(j<=15)
{
    rcRead(fromRobot, BYTES_TO_READ, &bytesRead);
    j++;
}
j=0;
```

```

/*}
//accende il robot portandolo alla posizione iniziale
BOOL initMotors()
{*/
    bool toReturn;
    //motorParameter contiene la posizione del motore come offset esadecimale
    WORD motorParameter;
    i = 0;
    while(i<MAX_BUF_SIZE)
    {
        *(toRobot+i) = 0x00;
        i++;
    }
    toRobot[1] = 0x55;
    toRobot[2] = 0x01;
    toRobot[3] = 0xE0;
    toRobot[4] = 0x09;
    toRobot[5] = 0xF8;
    toRobot[6] = 0x02;
    toRobot[7] = 0x01;
    //bastano questi parametri per accenderlo
    toRobot[10] = 0x8A;
    toRobot[11] = 0xEC;
    toRobot[12] = 0x0F;
    toRobot[13] = 0xC1;
    toRobot[14] = 0xE0;
    toRobot[15] = 0x01;
    toRobot[16] = 0x82;
    toRobot[17] = 0x02;
    toRobot[18] = 0x86;
    toRobot[19] = 0x7C;
    toRobot[20] = 0x80;
    toRobot[21] = 0x81;
    toRobot[22] = 0x5e;
    toRobot[23] = 0x02;
    toRobot[24] = 0x80;
    toRobot[26] = 0x18;
    toRobot[30] = 0xAE;
    toRobot[31] = 0xD9;
    toRobot[32] = 0xCE;
    toRobot[33] = 0x1D;
    toRobot[42] = 0x52;
    toRobot[43] = 0x26;
    toRobot[44] = 0x32;
    toRobot[45] = 0xE2;
    //accendo il robot
    toReturn = rcWrite(toRobot, BYTES_TO_WRITE, &bytesWritten);
    i = 0;
    while(i<MAX_BUF_SIZE)
    {
        *(toRobot1+i) = 0x00;
        i++;
    }
    //con toRobot1-3 comunico la posizione iniziale
    //ogni parametro per i motori È una WORD (16 bit), devo invertirne l'ordine
    // in modo che buffer[i] = lsB e buffer[i+1] = msB
    toRobot1[1] = 0x55;
    toRobot1[2] = 0x03;

```



```

toRobot1[3] = 0xE0;
toRobot1[4] = 0x01;
toRobot1[6] = 0x7C;
toRobot1[7] = (BYTE) initial.anRollR;
toRobot1[8] = (BYTE)(initial.anRollR >> 8);
toRobot1[9] = (BYTE)initial.anPitchR;
toRobot1[10] = (BYTE)(initial.anPitchR >> 8);
toRobot1[11] = (BYTE)initial.knPitchR;
toRobot1[12] = (BYTE)(initial.knPitchR >> 8);
toRobot1[13] = (BYTE)initial.thPitchR;
toRobot1[14] = (BYTE)(initial.thPitchR >> 8);
toRobot1[15] = (BYTE)initial.thRollR;
toRobot1[16] = (BYTE)(initial.thRollR >> 8);
toRobot1[19] = (BYTE)initial.anRollL;
toRobot1[20] = (BYTE)(initial.anRollL >> 8);
toRobot1[21] = (BYTE)initial.anPitchL;
toRobot1[22] = (BYTE)(initial.anPitchL >> 8);
toRobot1[23] = (BYTE)initial.knPitchL;
toRobot1[24] = (BYTE)(initial.knPitchL >> 8);
toRobot1[25] = (BYTE)initial.thPitchL;
toRobot1[26] = (BYTE)(initial.thPitchL >> 8);
toRobot1[27] = (BYTE)initial.thRollL;
toRobot1[28] = (BYTE)(initial.thRollL >> 8);
toRobot1[31] = (BYTE)initial.eyeL;
toRobot1[32] = (BYTE)(initial.eyeL >> 8);
toRobot1[37] = (BYTE)initial.elRollR;
toRobot1[38] = (BYTE)(initial.elRollR >> 8);
toRobot1[39] = (BYTE)initial.shRollR;
toRobot1[40] = (BYTE)(initial.shRollR >> 8);
toRobot1[41] = (BYTE)initial.shPitchR;
toRobot1[42] = (BYTE)(initial.shPitchR >> 8);
toRobot1[43] = (BYTE)initial.eyeR;
toRobot1[44] = (BYTE)(initial.eyeR >> 8);
toRobot1[49] = (BYTE)initial.elRollL;
toRobot1[50] = (BYTE)(initial.elRollL >> 8);
toRobot1[51] = (BYTE)initial.shRollL;
toRobot1[52] = (BYTE)(initial.shRollL >> 8);
toRobot1[53] = (BYTE)initial.shPitchL;
toRobot1[54] = (BYTE)(initial.shPitchL >> 8);
toRobot1[59] = (BYTE)initial.head;
toRobot1[60] = (BYTE)(initial.head >> 8);

toReturn = toReturn && rcWrite(toRobot1, BYTES_TO_WRITE, &bytesWritten);
i = 0;
while(i<MAX_BUF_SIZE)
{
    *(toRobot2+i) = 0x00;
    i++;
}
toRobot2[1] = 0x55;
toRobot2[2] = 0x13;
toRobot2[5] = (BYTE)initial.voice;
toRobot2[7] = 0xc8;
toRobot2[10] = 0x01;
toReturn = toReturn && rcWrite(toRobot2, BYTES_TO_WRITE, &bytesWritten);
i = 0;
while(i<MAX_BUF_SIZE)
{

```

```
        *(toRobot3+i) = 0x00;
        i++;
    }
    toRobot3[1] = 0x55;
    toRobot3[2] = 0x23;
    toRobot3[7] = 0x80;
    toRobot3[9] = 0x04;
    toRobot3[14] = 0x80;
    toRobot3[16] = 0x02;
    toRobot3[17] = (BYTE)initial.time;
    toReturn = toReturn && rcWrite(toRobot3, BYTES_TO_WRITE, &bytesWritten);
    return toReturn;
}

bool doMovements()
{
    bool toReturn = false;

    toReturn = rcWrite(toRobot1, BYTES_TO_WRITE, &bytesWritten);
    toReturn = toReturn & rcWrite(toRobot2, BYTES_TO_WRITE, &bytesWritten);
    toReturn = toReturn & rcWrite(toRobot3, BYTES_TO_WRITE, &bytesWritten);

    return toReturn;
}

bool off()
{
    bool toReturn = false;

    int i = 0;

    while(i<MAX_BUF_SIZE)
    {
        *(toRobot+i) = 0x00;
        i++;
    }
    toRobot[1] = 0x55;
    toRobot[2] = 0x01;
    toRobot[3] = 0xE0;
    toRobot[4] = 0x09;
    toRobot[5] = 0xF8;
    toRobot[6] = 0x02;
    toRobot[7] = 0x00;
    toRobot[8] = 0x00;
    toRobot[9] = 0x00;
    toRobot[10] = 0x00;
    toRobot[11] = 0x8A;
    toRobot[12] = 0xEC;
    toRobot[13] = 0x0F;
    toRobot[14] = 0x21;
    toRobot[15] = 0xE0;
    toRobot[16] = 0x01;
    toRobot[17] = 0x82;
    toRobot[18] = 0x02;
    toRobot[19] = 0x86;
    toRobot[20] = 0x7C;
    toRobot[21] = 0x80;
```

```
toRobot [22] = 0x81;
toRobot [23] = 0x5E;
toRobot [24] = 0x02;
toRobot [25] = 0x80;
toRobot [26] = 0x00;
toRobot [27] = 0x18;
toRobot [28] = 0x00;
toRobot [29] = 0x00;
toRobot [30] = 0x00;
toRobot [31] = 0xAE;
toRobot [32] = 0xD9;
toRobot [33] = 0xCE;
toRobot [34] = 0x1D;
toRobot [35] = 0x00;
toRobot [36] = 0x00;
toRobot [37] = 0x00;
toRobot [38] = 0x00;
toRobot [39] = 0x00;
toRobot [40] = 0x00;
toRobot [41] = 0x00;
toRobot [42] = 0x00;
toRobot [43] = 0x52;
toRobot [44] = 0x26;
toRobot [45] = 0x32;
toRobot [46] = 0xE2;

toReturn = rcWrite(toRobot, BYTES_TO_WRITE, &bytesWritten);

return toReturn;
}
```

Listing A.7: onlineMode.h

```
#include <windows.h>

#include <stdbool.h>

bool isOnline();

bool goOnline();

bool goOffline();

bool execPose(WORD * actualPosition);
```

Listing A.8: onlineMode.c

```
#include <stdio.h>
#include <string.h>

#include "onlineMode.h"
#include "move.h"

#ifdef WIN32
#include <windows.h>
#else

typedef unsigned short WORD;
```

```
typedef char          BYTE;
typedef unsigned long  DWORD;

#endif

char comState=0;

bool isOnline()
{
    return comState;
}

bool goOnline()
{
    rcOpen();
    init();
    comState = 1;

    return 1;
}

bool goOffline()
{
    bool toReturn = false;

    toReturn = off();
    comState = 0;

    return toReturn;
}

bool execPose(WORD * actualPosition)
{
    bool toReturn = false;

    //TIME
    toRobot3[17] = (BYTE) actualPosition[0];

    //HEAD_YAW
    toRobot1[59] = (BYTE) actualPosition[1];
    toRobot1[60] = actualPosition[1] >> 8;

    //EYE_LED_L
    toRobot1[31] = (BYTE) actualPosition[2];
    toRobot1[32] = actualPosition[2] >> 8;

    //EYE_LED_R
    toRobot1[43] = (BYTE) actualPosition[3];
    toRobot1[44] = actualPosition[3] >> 8;

    //SHOULDER_PITCH_L
    toRobot1[53] = (BYTE) actualPosition[4];
    toRobot1[54] = actualPosition[4] >> 8;

    //SHOULDER_PITCH_R
    toRobot1[41] = (BYTE) actualPosition[5];
    toRobot1[42] = actualPosition[5] >> 8;
```

```
//SHOULDER_ROLL_L
toRobot1[51] = (BYTE) actualPosition[6];
toRobot1[52] = actualPosition[6] >> 8;

//SHOULDER_ROLL_R
toRobot1[39] = (BYTE) actualPosition[7];
toRobot1[40] = actualPosition[7] >> 8;

//ELBOW_ROLL_L
toRobot1[49] = (BYTE) actualPosition[8];
toRobot1[50] = actualPosition[8] >> 8;

//ELBOW_ROLL_R
toRobot1[37] = (BYTE) actualPosition[9];
toRobot1[38] = actualPosition[9] >> 8;

//THIGH_ROLL_L
toRobot1[27] = (BYTE) actualPosition[10];
toRobot1[28] = actualPosition[10] >> 8;

//THIGH_ROLL_R
toRobot1[15] = (BYTE) actualPosition[11];
toRobot1[16] = actualPosition[11] >> 8;

//THIGH_PITCH_L
toRobot1[25] = (BYTE) actualPosition[12];
toRobot1[26] = actualPosition[12] >> 8;

//THIGH_PITCH_R
toRobot1[13] = (BYTE) actualPosition[13];
toRobot1[14] = actualPosition[13] >> 8;

//KNEE_PITCH_L
toRobot1[23] = (BYTE) actualPosition[14];
toRobot1[24] = actualPosition[14] >> 8;

//KNEE_PITCH_R
toRobot1[11] = (BYTE) actualPosition[15];
toRobot1[12] = actualPosition[15] >> 8;

//ANKLE_PITCH_L
toRobot1[21] = (BYTE) actualPosition[16];
toRobot1[22] = actualPosition[16] >> 8;

//ANKLE_PITCH_R
toRobot1[9] = (BYTE) actualPosition[17];
toRobot1[10] = actualPosition[17] >> 8;

//ANKLE_ROLL_L
toRobot1[19] = (BYTE) actualPosition[18];
toRobot1[20] = actualPosition[18] >> 8;

//ANKLE_ROLL_R
toRobot1[7] = (BYTE) actualPosition[19];
toRobot1[8] = actualPosition[19] >> 8;

//VOICE
toRobot2[5] = (BYTE) actualPosition[20];
```

```
//pose execution
toReturn = doMovements();

return toReturn;
}
```

# Bibliografia

- *RobovieMaker2 User Guide*, ATR Vstone Co., Ltd., 2008
- *Robovie-X Assembly Manual*, Vstone, ver. 1.11b
- *Lexical Analysis With Flex*, <http://flex.sourceforge.net/manual/>
- *Bison - GNU parser generator*, <http://www.gnu.org/software/bison/manual/>
- *C: A Reference Manual (5th Edition)*, Samuel P. Harbison, Guy L. Steele, Prentice Hall, 2002
- *Applicativo Open Source per il controllo del Robot RB200*, Riccardo Bonetto, Università degli studi di Padova, 2008
- *MSDN Library*, Microsoft Developer's Network, <http://www.microsoft.com>

