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

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica
(Prof. Riccardo Cassinis)

Libreria Experimenter per Saphira 8.2

Elaborato di esame di: **Marco Palini, Gianmarco Pugliese**

Consegnato il: **13 giugno 2003**

Sommario

Questo lavoro si propone di permettere una facile creazione, da parte dell'utente finale, di funzioni utilizzabili nel pacchetto software Saphira 8.2, mediante il linguaggio di programmazione Colbert.

In particolare, è stata sviluppata la libreria 'Experimenter', che offre delle funzionalità relative alla gestione dell'Experimenter module di un robot di tipo Pioneer 1. Questa libreria, sviluppata utilizzando il linguaggio di programmazione C++, offre delle funzioni che permettono di interagire con componenti hardware eterogenei: una pinza fornita di due polpastrelli, uno speaker ed un insieme di porte analogiche a cui possono essere collegati fino ad 8 RC Servo.

È stata sviluppata, inoltre, una activity basata su Colbert, denominata 'testexp', che ha costituito un utile mezzo per effettuare il testing delle funzioni offerte dalla libreria.

1. Introduzione

Questo elaborato ha lo scopo di documentare in modo esauriente i passi che l'utente del pacchetto software Saphira 8.2 deve compiere per realizzare una nuova libreria, utilizzando il linguaggio C++. È stata sviluppata, inoltre, una libreria denominata 'Experimenter', che offre all'utente finale un insieme di funzionalità utilzzabili nella programmazione di un robot mediante il linguaggio di programmazione Colbert, fornito come parte del pacchetto Saphira.

Il primo passo è costituito da una visione d'insieme delle caratteristiche del pacchetto software Saphira, facendo riferimento, in particolare, alle versioni 8.0 e successive. Verrà fornita, inoltre, una breve descrizione delle caratteristiche salienti del linguaggio di programmazione Colbert, utilizzabile dall'utente finale per realizzare programmi con cui interagire con un robot di tipo Pioneer.

Tale trattazione, lungi dall'essere esaustiva sull'argomento, costituisce solo un tentativo di fornire al lettore alcuni concetti necessari per la comprensione del lavoro svolto. Per un maggiore approfondimento, quindi, si rimanda ai documenti [1], [2], [3], [4], [5], [6] contenuti all'interno del pacchetto stesso.

1.1. Il pacchetto software Saphira 8.2

Il pacchetto software Saphira è stato sviluppato dalla SRI International's Artificial Intelligence Center, sotto la supervisione del Dr. Kurt Konolige ed ha lo scopo di fornire un ambiente di supporto allo sviluppo di applicazioni nel campo della robotica. Questo pacchetto trova applicazione, in particolare, nella programmazione della piattaforma per robot mobili Pioneer, sviluppata dallo stesso Dr. Konolige.

Saphira è basato su una architettura di tipo client/server, nella quale la parte server è collocata all'interno del robot mobile, mentre il processo client viene eseguito su di un PC ad esso collegato. Questo pacchetto permette all'utente di realizzare dei processi client, mediante una libreria di classi realizzata in linguaggio C++, con cui è possibile interfacciarsi in modo bidirezionale con un robot mobile. I metodi resi disponibili, infatti, costituiscono una Application Programmer's Interface (API) che permette sia di inviare dei comandi al server, sia di ricevere dati (come quelli relativi ai sensori) e di visualizzarli in una interfaccia grafica. Tale libreria offre, inoltre, la possibilità di gestire elementi chiave della programmazione del robot, quali il suo controllo, l'interpretazione dei dati provenienti dai sensori e la localizzazione dello stesso, ad un alto livello di astrazione. È il server, infatti, che si occupa in modo autonomo di gestire tutti i dettagli di basso livello, relativi, ad esempio, alla gestione dei motori.

1.2. Saphira + Aria = incapsulamento

Come già accennato, l'uso di Saphira 8.2 da parte dell'utente finale permette a quest'ultimo di programmare un robot mobile astraendo da tutti quei dettagli di basso livello, specifici della piattaforma robotica utilizzata. Per poter realizzare questa forma di incapsulamento, Saphira utilizza Aria, un sistema di controllo a basso livello per robot, realizzato dalla ActivMedia Robotics.

Questa libreria di classi costituisce una novità introdotta in Saphira a partire dalla versione 8.0. Precedentemente, infatti, la gestione di molti dettagli, ora delegata ad Aria, era parte integrante di Saphira. L'introduzione di questo nuovo insieme di classi ha permesso di aggiungere nuove funzionalità, quali, ad esempio, il controllo di più robot, nonché di gestire in modo più flessibile le caratteristiche di basso livello specifiche di un robot. L'architettura dell'insieme costituito da Saphira ed riarappresentata in Figura 1.

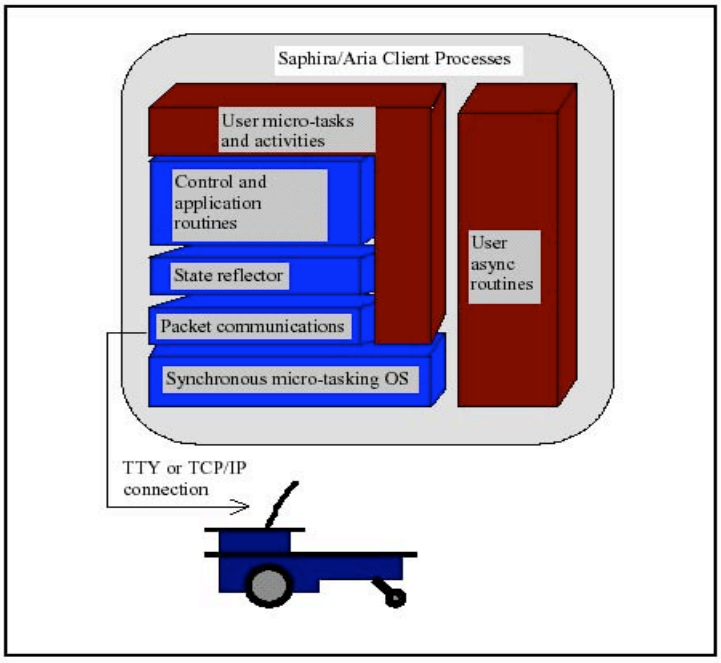

Figura 1 - Schema dell'architettura di Saphira/Aria

La Figura 1 mostra come l'utente finale abbia lo possibilità di realizzare dei programmi che offrono un maggiore controllo del robot, rinunciando al grado di astrazione offerto da Saphira ed utilizzando direttamente le classi messe a sua disposizione da Aria. Tuttavia, le nuove funzionalità così realizzate, indicate nella Figura 1 con il termine "User async routines", portano l'utente finale a compiere uno sforzo di programmazione maggiore, legato alla maggiore quantita di dettagli di cui deve tener conto personalmente. Questo, tuttavia, rappresenta un argomento avanzato della programmazione basata su Aria, quindi si rimanda il lettore interessato ad approfondire maggiormente, consultando il documento [4].

1.3. Il linguaggio Colbert

Colbert è un linguaggio di programmazione creato espressamente per la programmazione di piattaforme robotiche. Pensato come mezzo di interazione ad alto livello dell'utente finale con il pachetto Saphira/Aria, utilizza, mediante opportuni accorgimenti, le librerie di classi offerte da quest'ultimo, per dare agli utenti la possibilità di eseguire comandi di movimento del robot, di definire sequenze di azioni e di eseguire strategie di controllo del robot stesso.

Questo linguaggio, la cui grammatica è basata su un sottoinsieme del linguaggio C (come si evince leggendo i documenti [5], [6]), viene interpretato mediante un interprete incluso nel pacchetto Saphira. Quest'ultimo, infatti, visualizza una finestra di interazione, all'interno della quale l'utente può eseguire direttamente comandi, che l'interprete Colbert provvede ad eseguire immediatamente oppure può caricare ed eseguire strategie di controllo del robot più complesse. In particolare, questa seconda attività è basata sulla definizione delle cosiddette "activity", ovvero la creazione di schemi contenenti comandi Colbert e strutture di controllo.

1.3.1. La programmazione in pratica: Colbert ed il linguaggio C++

Definire una nuova activity in Colbert significa sostanzialmente creare un semplice file di testo, contenente un insieme di comandi riconoscibili dall'interprete. Questo insieme può essere costituito da:

- comandi Colbert relativi al controllo ed al movimento di un robot,
- istruzioni per la definizione, il caricamento o l'esecuzione di altre activity,
- istruzioni per definire variabili globali utilizzabili nell'activity stessa o nella finestra di interazione di Saphira,

Una caratteristica, che permette di definire Colbert un linguaggio di programmazione estensibile, è la possibilità di rendere visibili all'utente finale oggetti e funzioni attraverso l'interfaccia Colbert. Ciò si applica sia alle funzioni predefinite del pacchetto Saphira/Aria che ai nuovi oggetti ed alle nuove funzionalità realizzate dall'utente stesso. Conseguentemente, l'interazione tipica dell'utente finale con il complesso Saphira/Aria + Colbert nella creazione e nel debug di programmi per una piattaforma robotica può essere sintetizzata in due passi successivi:

- La creazione di programmi, mediante l'uso del linguaggio C++, che utilizzino le funzioni e gli oggetti definiti da Saphira/Aria, o che implementino funzionalità del tutto nuove,
- La definizione di activity Colbert che costituiscano l'interfaccia di tali elementi verso l'interprete dei comandi, permettendone così il caricamento, l'esecuzione ed il controllo dello stato.

2. Il problema affrontato

Riprendendo un fatto precedentemente accennato, si nota che la piattaforma per robot mobili denominata "Pioneer" è stata ideata e realizzata dal Dr. Kurt Konolige, come ben documentato in [7]. Quest'ultimo ha sviluppato con continuità la stessa, realizzandone versioni successive, i cui esponenti sono denominati rispettivamente "Pioneer 2" (sulle cui caratteristiche si possono trovare informazioni più accurate in [8], [9], [10]) e "Pioneer 3".

Il pacchetto software Saphira è stato sviluppato dalla SRI International's Artificial Intelligence Center, sotto la supervisione dello stesso Dr. Konolige e, come si può facilmente immaginare, le versioni progressivamente realizzate, rispecchiano le innovazioni introdotte nelle versioni 2 e 3 dei robot di tipo Pioneer. Conseguentemente, la versione 8.2 di Saphira, associata alla versione 1.2.0 di Aria, offre un insieme di funzionalità specificamente pensate per la gestione dell'hardware di un robot di tipo Piooner 2 o 3, che non sempre rispecchiano le caratteristiche dell'hardware presente sulla versione 1 della stessa piattaforma.

2.1. L'hardware: il "modulo Experimenter"

Il robot di tipo Pioneer 1 non disponeva, nella sua versione "base", di alcuni componenti hardware, quali, ad esempio, una pinza con la quale sollevare e trasportare oggetti. Era presente, tuttavia, la possibilità di acquistare separatamente un insieme aggiuntivo di elementi hardware, denominato "modulo Experimenter", che, montato sul robot, aggiungeva un insieme di nuovi componenti quali:

- Una pinza, composta da due polpastrelli controllati da un motore in grado di afferrare oggetti, trasportarli e fornita dei seguenti elementi aggiuntivi:
  - due sensori di contatto, posti sulle setremità anteriori dei polpestrelli, in grado di segnalare la collisione con un ostacolo,
  - due led verdi, visbili sulla parte superiore dei polpastrelli della pinza.
- Uno speaker in grado di emettere uno o più suoni.
- Un insieme di otto porte di ingresso di tipo analogico, in grado di pilotare fino a otto dispositivi RC Servo.

Lo scopo del presente lavoro, quindi, è costituito dall'offrire all'utente finale un modo semplice per controllare e testare lo stato dei componentti hardware contenuti nel modulo Experimenter. Al tempo stesso, nello sviluppo di questo elaborato, ci si è prefissa la finalità, coerente con la filosofia di fondo del pacchetto Saphira, di

minimizzare le conoscenze richieste all'utente finale, relativamente alle caratteristiche dei dispositivi hardware. Tuttavia, qualora il lettore sia interessato ad una visione particolareggiata sui dettagli tecnici necessari per installare ed utilizzare il modulo Experimenter, nonché per creare nuove funzionalità che abbiano un controllo a basso livello sui dispositivi stessi, si rimanda alla lettura dei documenti [1], [3], [4], [11].

2.2. Il software: compatibilità con Saphira 8.2

Il problema affrontato ha comportato la realizzazione di una libreria software con la quale concretizzare la gestione dell'hardware del modulo Experimenter. Al fine di raggiungere lo scopo prefisato, infatti, si è reso necessario fornire all'utente finale delle funzioni da poter eseguire direttamente all'interno dell'interprete Colbert contenuto in Saphira 8.2 o da integrare nei propri programmi.

La realizzazione di queste funzionalità ha richiesto di tenere costantemente in considerazione due aspetti principali del problema:

- La libreria che si occupa di gestire il modulo Experimenter offre delle funzionalità "nuove" in Saphira 8.x, dal momento che in alcuni casi non esistono classi predefinite, specifiche per la gestione dei componenti hardware in questione. Questo ha richiesto la definizione "ex novo", mediante il linguaggio C++, di classi non contenute nel pacchetto.
- In altri casi l'implementazione, offerta da Saphira, delle classi preposte al controllo delle versioni 2 e 3 della piattaforma robotica Pioneer, differiscono rispetto alle funzionalità, aventi lo stessa finalità, offerte dalle versioni precedenti del pacchetto per i robot di tipo Pioneer 1. Ciò, quindi, ha reso necessario creare delle classi il cui scopo è incapsulare le funzionalità, relative alla gestione dell'hardware presente sui robot di tipo Pioneer 2 e 3, rendendole così utilizzabili da parte di un programma che faccia riferimento ad un robot di tipo Pioneer 1.

3. La soluzione adottata

Per rendere disponibili le funzioni del modulo Experimenter nell'ambiente Colbert abbiamo realizzato una libreria condivisa chiamata Experimenter.so in linguaggio C, la quale si appoggia ad altre tre librerie ExPion1Gripper.so, ExPion1Speaker.so, ExPion1AnalohPort.so, tre classi in C++, che comunicano con il robot utilizzando le classi Saphira.

Abbiamo creato questi due livelli software perché in Colbert non si possono importare direttamente metodi di classi ma solo funzioni. Experimenter.so è l'interfaccia che rende disponibili i metodi all'ambiente Colbert attraverso funzioni C.

Mentre il livello più basso è realizzato per chi programma il robot in C++ appoggiandosi a Saphira, come rappresentato in Figura 2.

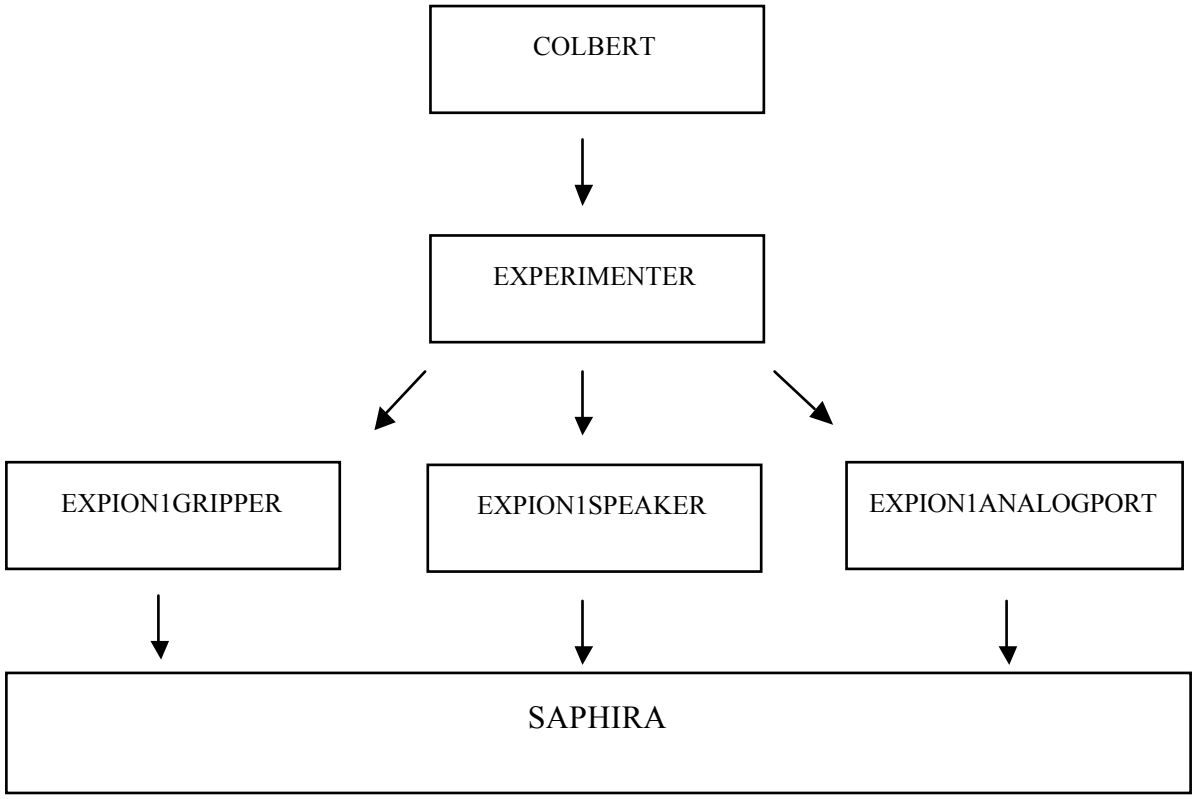

Figura 2 - Schema dei livelli dell'architettura software

3.1. Realizzazione della libreria Experimenter

Abbiamo preferito utilizzare funzioni, quando possibile, senza passaggio di parametri, quindi per settare o resettare un bit della porta di uscita, per esempio quello del led della pinza, sono state crete due funzioni, una per accenderlo una per spegnerlo.

Così l'utente che lavora sotto Colbert non deve preoccuparsi del parametro e del valore da passare alla funzione; infatti il nome stesso è sufficiente per comprenderne l'uso. Inoltre è stato messo a disposizione un help per il modulo Experimenter che elenca tutte le funzioni utilizzabili e un help per ciascuna delle funzioni.

Questo è il criterio generale adottato per la libreria Experimenter.so, diverso da quello delle classi per il livello più basso che hanno poche funzioni con passaggio di parametri.

Di seguito riportiamo il sorgente dell'interfaccia verso Colbert della libreria Experimenter.so:

```
/*
   ------------------------------------------------

   Experimenter.cpp

   Codice sorgente in linguaggio c++.
```

Questo file è stato realizzato da

Gianmarco Pugliese Matr. 19526
Marco Palini Matr. 25207

come parte del progetto

Experimenter

per il corso di

Robotica
Prof. Riccardo Cassinis
Università degli studi di Brescia
Facoltà di Ingegneria
A.A. 2002/03

Questo file fornisce l'interfaccia verso il linguaggio Colbert, contenuto nel pacchetto software Saphira, delle classi implementate dalla libreria Experimenter. Questa libreria permette di creare uno shared object, utilizzabile in Saphira 8.x, che si occupa della gestione della pinza, dello speaker e delle porte analogiche di un robot di tipo Pioneer 1 fornito dell'Experimenter module. Vengono definite, in particolare:

+ le variabili globali
  - gripper
  - speaker
  - analogport
+ le funzioni relative alla pinza
  - exGripperOpen
  - exGripperClose
  - exIsGripperTop
  - exIsGripperCarring
  - exIsGripperOpen
  - exIsInnerInGripper
  - exIsOuterInGripper
  - exIsBumpTouching
  - exTurnOnRightLed
  - exTurnOnLeftLed
  - exTurnOffRightLed
  - exTurnOffLeftLed
  - exIsRightLedOn
  - exIsLeftLedOn
+ le funzioni relative allo speaker

```
     - exPlayToneString
   + le funzioni relative alle porte analogiche
     - exGetAnalogPort
   + le funzioni di servizio
     - sfLoadInit
     - sfLoadExit

   ------------------------------------------------------
*/

#include "Saphira.h"
#include "ExArguments.h"
#include "ExPion1Gripper.h"
#include "ExPion1Speaker.h"
#include "ExPion1AnalogPort.h"

/* Definizione delle variabili globali relative alla
   pinza, allo speaker ed alle porte analogiche
   dell'Experimenter module di un robot di tipo Pioneer
   1. */
ExPion1Gripper    *gripper;
ExPion1Speaker    *speaker;
ExPion1AnalogPort *analogport;

/* Inizio dichiarazione funzioni globali relative alla
   gestione delle porte analogiche.
*/

/*
   Nome           : exGetAnalogPortValue
   Tipo restituito: int
                    È il valore che viene letto relativo
                    ad una delle otto porte analogiche.
   Parametri      : + int which
                      - È un intero che indica quale è
                        la porta di cui si vuole leggere
                        il valore.
   Scopo          : Permette di leggere il valore di una
                    porta analogica.
   Descrizione    : Legge il valore di una delle otto
                    porte analogiche offerte
                    dall'Experimenter module del Pioneer
                    1.
*/
int exGetAnalogPortValue( int which )
{
        return ( analogport->GetAnalogPortValue( which )
```

```
                );
};

/* ------------------------------------------------- */

/*
   Inizio dichiarazione funzioni globali relative alla
   gestione della pinza.
*/

/*
   Nome           : exGripperOpen
   Tipo restituito: void
   Parametri      : void
   Scopo          : Apre la pinza.
   Descrizione    : Aziona il motore che controlla la
                    pinza del Pioneer 1, posizionandola
                    in basso con i polpastrelli aperti.
*/
void exGripperOpen( void )
{
        gripper->SetGripperState( ExArguments::GRIPOPEN
                                  );
};

/*
   Nome           : exGripperClose
   Tipo restituito: void
   Parametri      : void
   Scopo          : Chiude la pinza.
   Descrizione    : Aziona il motore che controlla la
                    pinza del Pioneer 1, posizionandola
                    in alto con i polpastrelli chiusi.
*/
void exGripperClose( void )
{
        gripper->SetGripperState( ExArguments::GRIPCLOSE
                                  );
};

/*
   Nome           : exIsGripperTop
   Tipo restituito: bool
                    Può assumere il valore:
                    - True se la pinza è posizionata
                      all'estremo superiore
                    - False se la pinza si trova in
```

```
                        un'altra posizione.
   Parametri      : void
   Scopo          : Testa una particolare posizione della
                    pinza.
   Descrizione    : Controlla uno switch per sapere se la
                    pinza è arrivata all'estremo
                    superiore.
*/
bool exIsGripperTop( void )
{
        return !( gripper->GetGripperState() &
                  ExArguments::GRIPISTOP );
};

/*
   Nome           : exIsGripperCarring
   Tipo restituito: bool
                    Può assumere il valore:
                    - True se la pinza ha iniziato la sua
                      corsa verso il punto più alto e si
                      trova a metà corsa
                    - False se la pinza è ferma o si
                      trova in un punto diverso.
   Parametri      : void
   Scopo          : Testa una particolare posizione della
                    pinza.
   Descrizione    : Controlla uno switch per sapere se la
                    pinza ha iniziato a muoversi verso
                    l'alto e si trova a metà corsa.
*/
bool exIsGripperCarring( void )
{
        return !( gripper->GetGripperState() &
                  ExArguments::GRIPISCARRING );
};

/*
   Nome           : exIsGripperOpen
   Tipo restituito: bool
                    Può assumere il valore:
                    - True se la pinza si è aperta
                      completamente
                    - False se i polpastrelli sono in una
                      posizione diversa.
   Parametri      : void
   Scopo          : Testa una particolare posizione della
                    pinza.
```

```
   Descrizione    : Controlla uno switch per sapere se i
                    polpastrelli della pinza hanno
                    raggiunto la loro massima distanza
                    relativa.
*/
bool exIsGripperOpen( void )
{
        return !( gripper->GetGripperState() &
                  ExArguments::GRIPISOPEN );
};

/*
   Nome           : exIsInnerInGripper
   Tipo restituito: bool
                    Può assumere il valore:
                    - True se la fotocellula interna
                      della pinza vede un oggetto
                    - False se non viene rilevata la
                      presenza di nessun oggetto.
   Parametri      : void
   Scopo          : Testa la presenza di un oggetto nella
                    pinza.
   Descrizione    : Controlla se è presente un oggetto
                    nella parte più interna della pinza
                    quando è aperta e posizionata in
                    basso, mediante una fotocellula che
                    si trova nella parte interna dei
                    polpastrelli.
*/
bool exIsInnerInGripper( void )
{
        return ( gripper->IsBreakBeamObstructed(
                  ExArguments::GRIPINNERBB ) );
};

/*
   Nome           : exIsOuterInGripper
   Tipo restituito: bool
                    Può assumere il valore:
                    - True se la fotocellula esterna
                      della pinza vede un oggetto
                    - False se non viene rilevata la
                      presenza di nessun oggetto.
   Parametri      : void
   Scopo          : Testa la presenza di un oggetto nella
                    pinza.
   Descrizione    : Controlla se è presente un oggetto
```

```
                    nella parte più esterna della pinza
                    quando è aperta e posizionata in
                    basso, mediante una fotocellula che
                    si trova nella parte esterna dei
                    polpastrelli.
*/
bool exIsOuterInGripper( void )
{
        return ( gripper->IsBreakBeamObstructed(
                  ExArguments::GRIPOUTERBB ) );
};

/*
   Nome           : exIsBumpTouching
   Tipo restituito: bool
                    Può assumere il valore:
                    - True se uno dei sensori ha rilevato
                      una collisione con qualcosa
                    - False se non è stato rilevato
                      nessun contatto.
   Parametri      : void
   Scopo          : Testa la presenza del contatto con un
                    oggetto.
   Descrizione    : Controlla se il robot è entrato in
                    contatto con un ostacolo, mediante
                    sensori posti sulle estremità
                    anteriori dei polpastrelli.
*/
bool exIsBumpTouching( void )
{
        return ( gripper->IsBumpTouching() );
};

/*
   Nome           : exTurnOnRightLed
   Tipo restituito: void
   Parametri      : void
   Scopo          : Accende un led della pinza.
   Descrizione    : Accende il led posizionato
                    sull'estremita superiore del
                    polpastrello destro della pinza.
*/
void exTurnOnRightLed( void )
{
        gripper->SetLedState(
         ExArguments::RIGHTLED,ExArguments::RIGHTLED );
};
```

```
/*
   Nome           : exTurnOnLeftLed
   Tipo restituito: void
   Parametri      : void
   Scopo          : Accende un led della pinza.
   Descrizione    : Accende il led posizionato
                    sull'estremita superiore del
                    polpastrello sinistro della pinza.
*/
void exTurnOnLeftLed( void )
{
        gripper->SetLedState(
         ExArguments::LEFTLED,ExArguments::LEFTLED );
};

/*
   Nome           : exTurnOffRightLed
   Tipo restituito: void
   Parametri      : void
   Scopo          : Spegne un led della pinza.
   Descrizione    : Spegne il led posizionato
                    sull'estremita superiore del
                    polpastrello destro della pinza.
*/
void exTurnOffRightLed( void )
{
        gripper->SetLedState(
         ExArguments::RIGHTLED,ExArguments::LEDOFF );
};

/*
   Nome           : exTurnOffLeftLed
   Tipo restituito: void
   Parametri      : void
   Scopo          : Spegne un led della pinza.
   Descrizione    : Spegne il led posizionato
                    sull'estremita superiore del
                    polpastrello sinistro della pinza.
*/
void exTurnOffLeftLed( void )
{
        gripper->SetLedState(
         ExArguments::LEFTLED,ExArguments::LEDOFF );
};

/*
```

```
   Nome           : exIsRightLedOn
   Tipo restituito: bool
                    Può assumere il valore:
                    - True se il led destro è acceso
                    - False se il led destro è spento.
   Parametri      : void
   Scopo          : Testa lo stato di un led.
   Descrizione    : Controlla se il led destro è
                    effettivamente acceso.
*/
bool exIsRightLedOn( void )
{
        return ( gripper->GetLedState(
                  ExArguments::RIGHTLED ) );
};

/*
   Nome           : exIsLeftLedOn
   Tipo restituito: bool
                    Può assumere il valore:
                    - True se il led sinistro è acceso
                    - False se il led sinistro è spento.
   Parametri      : void
   Scopo          : Testa lo stato di un led.
   Descrizione    : Controlla se il led sinistro è
                    effettivamente acceso.
*/
bool exIsLeftLedOn( void )
{
        return ( gripper->GetLedState(
                  ExArguments::LEFTLED ) );
};

/* ------------------------------------------------- */

/* Inizio dichiarazione funzioni globali relative alla
   gestione dello speaker.
*/

/*
   Nome           : exPlayToneString
   Tipo restituito: void
   Parametri      : + const char *sequence
                      - È una stringa in cui è
                        memorizzata la sequenza delle
                        note da suonare, contenente al
                        massimo 20 note. Ad ogni
```

```
                        nota sono associati sette
                        caratteri della stringa,
                        indicanti rispettivamente:
                        * tre caratteri numerici riferiti
                          al byte della durata, cioè il
                          numero di incrementi che
                          moltiplicato per 20 ms da la
                          durata della nota
                        * il carattere '\' usato come
                          separatore
                        * tre caratteri numerici riferiti
                          al byte del tono, cioè il
                          numero di incrementi che
                          moltiplicato per 150
                          microsecondi definisce il tono
                          della nota.
                        È necessario che la durata ed il
                        tono occupino esattamente tre
                        caratteri.
                    + int n
                      - È il numero di byte di durata e
                        di tono effettivamente presenti
                        nella sequenza.
   Scopo          : Permette di suonare dallo speaker.
   Descrizione    : Emette una sequenza di note
                    attraverso lo speaker contenuto
                    nell'Experimenter module del Pioneer
                    1.
*/
void exPlayToneString( const char *sequence, int n )
{
        speaker->PlayToneString( sequence,n );
};

/* ------------------------------------------------- */

/* Inizio dichiarazione funzioni di servizio per
   l'interfaccia verso Colbert
*/

/*
   Nome           : sfLoadInit
   Tipo restituito: void
   Parametri      : void
   Scopo          : Definisce un'operazione di servizio.
   Descrizione    : Definisce quali sono le funzioni
                    della libreria Experimenter che
```

```
                    vengono rese disponibili in Colbert.
                    In particolare, per ognuna di esse,
                    crea una nuova funzione, che può
                    essere richiamata dalla finestra
                    comandi di Saphira o da una
                    activity Colbert, specificando:
                    - la stringa contenente il nome con
                      cui la funzione può essere chiamata
                    - il puntatore che esprime la
                      corrispondenza tra la nuova
                      funzione e quella della libreria
                    - il tipo restituito dalla nuova
                      funzione
                     -  il numero di parametri passati alla
                        nuova funzione
                    - il tipo di ogni parametro passato.
                    Definisce, inoltre, per ogni
                    funzione, una stringa di aiuto che
                    viene visualizzata nella finestra
                    comandi di Saphira quando l'utente
                    richiede un aiuto sulle funzioni
                    contenute nella libreria o sulla
                    singola funzione mediante il
                    comando help.
*/
SFEXPORT void sfLoadInit( void )
{
        sfMessage("
                  Experimenter loaded.
                  ");
        gripper = new ExPion1Gripper;
        speaker = new ExPion1Speaker;
        analogport = new ExPion1AnalogPort;

        /* Funzioni relative alla lettura del valore di
           una delle otto porte analogiche. */
        sfAddEvalFn( "GetAnalogPortValue",(void *)
                     exGetAnalogPortValue,sfVOID,1,sfINT
                   );

        /* Funzioni relative alla lettura ed al settaggio
           dello stato della pinza. */
        sfAddEvalFn( "GripperOpen",(void *)
                     exGripperOpen,sfVOID,0 );
        sfAddEvalFn( "GripperClose",(void *)
                     exGripperClose,sfVOID,0 );
        sfAddEvalFn( "IsGripperTop",(void *)
```

```
            exIsGripperTop,sfINT,0 );
sfAddEvalFn( "IsGripperCarring",(void *)
            exIsGripperCarring,sfINT,0 );
sfAddEvalFn( "IsGripperOpen",(void *)
            exIsGripperOpen,sfINT,0 );

/* Funzioni relative alla lettura dello stato
   delle fotocellule a infrarossi sui
   polpastrelli. */
sfAddEvalFn( "IsInnerInGripper",(void *)
            exIsInnerInGripper,sfINT,0 );
sfAddEvalFn( "IsOuterInGripper",(void *)
            exIsOuterInGripper,sfINT,0 );

/* Funzioni relative alla lettura dello stato del
   sensore di contatto. */
sfAddEvalFn( "IsBumpTouching",(void *)
            exIsBumpTouching,sfINT,0 );

/* Funzioni relative alla lettura ed al settaggio
   dello stato dei led. */
sfAddEvalFn( "TurnOnRightLed",(void *)
            exTurnOnRightLed,sfVOID,0 );
sfAddEvalFn( "TurnOnLeftLed",(void *)
            exTurnOnLeftLed,sfVOID,0 );
sfAddEvalFn( "TurnOffRightLed",(void *)
            exTurnOffRightLed,sfVOID,0 );
sfAddEvalFn( "TurnOffLeftLed",(void *)
            exTurnOffLeftLed,sfVOID,0 );
sfAddEvalFn( "IsRightLedOn",(void *)
            exIsRightLedOn,sfINT,0 );
sfAddEvalFn( "IsLeftLedOn",(void *)
            exIsLeftLedOn,sfINT,0 );

/* Funzioni relative all'emissione di suoni dallo
   speaker. */
sfAddEvalFn( "PlayToneString", (void *)
            exPlayToneString, sfVOID, 2,
            sfSTRING, sfINT );



/* Stringa di aiuto della libreria che mostra la
   lista di tutte le funzioni utilizzabili in
   Colbert. */
sfAddHelp( "Experimenter","
          Functions in the Experimenter library:
```

```
                * GetAnalogPortValue
                * GripperOpen, GripperClose,
                  IsGripperTop, IsGripperCarring,
                  IsGripperOpen,
                * IsInnerInGripper, IsOuterInGripper,
                * IsBumpTouching,
                * TurnOnRightLed, TurnOnLeftLed,
                  TurnOffRightLed, TurnOffLeftLed,
                  IsRightLedOn, IsLeftLedOn,
                * PlayToneString.

                Type 'help <fnction-name>' to get more
                help.
                " );

/* Stringhe di aiuto delle funzioni relative alle
   porte analogiche. */
sfAddHelp("GetAnalogPort",
          "
          int GetAnalogPort( int which )
          Gets the value from one of the eight
          analog ports in the Experimenter
          module.
          ");
/* Stringhe di aiuto delle funzioni relative alla
   pinza. */
sfAddHelp( "GripperOpen",
           "
           GripperOpen()
           Opens the gripper.
           " );
sfAddHelp( "GripperClose",
           "
           GripperClose()
           Closes the gripper.
           " );
sfAddHelp( "IsGripperTop",
           "
           int IsGripperTop()
           Tests whether the gripper is at its
           highest point or not.
           Returns a positive value in the first
           case, 0 otherwise.
           " );
sfAddHelp( "IsGripperCarring",
           "
           int IsGripperCarring()
```

```
            Tests whether the gripper is moving
            toward its highest point or not.
            Returns a positive value in the first
            case, 0 otherwise.
            " );
sfAddHelp( "IsGripperOpen",
            "
            int IsGripperTopOpen()
            Tests whether the gripper is at its
            lowest point with paddles open, or
            not.
            Returns a positive value in the first
            case, 0 otherwise.
            " );

/* Stringhe di aiuto delle funzioni relative alle
   fotocellule a infrarossi sui polpastrelli. */
sfAddHelp( "IsInnerInGripper",
            "
            int IsInnerInGripper()
            Tests whether the inner break beam on
            the paddles can see on abject, or not.
            Returns a positive value in the first
            case, 0 otherwise.
            " );
sfAddHelp( "IsOuterInGripper",
            "
            int IsOuterInGripper()
            Tests whether the outer break beam on
            the paddles can see on abject, or not.
            Returns a positive value in the first
            case, 0 otherwise.
            " );

/* Stringhe di aiuto delle funzioni relative al
   sensore di contatto. */
sfAddHelp( "IsBumpTouching",
            "
            int IsBumpTouching()
            Tests whether the bumpers on the
            paddles are touching on object, or
            not.
            Returns a positive value in the first
            case, 0 otherwise.
            " );

/* Stringhe di aiuto delle funzioni relative ai
```

```
   led. */
sfAddHelp( "TurnOnRightLed",
           "
           TurnOnRightLed()
           Turns the green led on the right
           paddle on.
           " );
sfAddHelp( "TurnOnLeftLed",
           "
           TurnOnLeftLed()
           Turns the green led on the left paddle
           on.
           " );
sfAddHelp( "TurnOffRightLed",
           "
           TurnOffRightLed()
           Turns the green led on the right
           paddle off.
           " );
sfAddHelp( "TurnOffLeftLed",
           "
           TurnOnRightLed()
           Turns the green led on the left paddle
           off.
           " );
sfAddHelp( "IsRightLedOn",
           "
           int IsRightLedOn()
           Tests whether the green led on the
           right paddle is turned on, or not.
           Returns a positive value in the first
           case, 0 otherwise.
           " );
sfAddHelp( "IsLeftLedOn",
           "
           int IsLeftLedOn()
           Tests whether the green led on the
           left paddle is turned on, or not.
           Returns a positive value in the first
           case, 0 otherwise.
           " );

/* Stringhe di aiuto delle funzioni relative allo
   speaker. */
sfAddHelp( "PlayToneString",
           "
           int PlayToneString( const char
```

```
                                        *sequence,int n )
                    Plays a sequence made up of n notes
                    through the speaker.
                    The sequence string is up to 20 notes
                    long. It is comprised of duration and
                    tone byte pairs in the form 'ddd/ttt'.
                    " );
};

/*
   Nome           : sfLoadExit
   Tipo restituito: void
   Parametri      : void
   Scopo          : Definisce un'operazione di servizio.
   Descrizione    : Dichiara quali sono le azioni da
                    eseguire quando la libreria
                    Experimenter viene scaricata dalla
                    memoria.
*/
SFEXPORT void sfLoadExit( void )
{
     sfMessage( "Unloading Experimenter." );
};

/* ------------------------------------------------- */
```

3.2. Realizzazione delle librerie ExPion1

Per gestire il modulo Experimenter del Pioneer 1 sono presenti delle porte di I/O, e la comunicazione con il robot si basa su un pacchetto di trasmissione chiamato SIP (Server Information Packet).

Quindi inserendo nel Client Command Number del SIP il valore sfCOMDIGOUT si effettua una richiesta di scrittura sulla porta dell'experimenter module. Inoltre è necessario inserire nel pacchetto la maschera dei bit e il valore di set o di reset. La lettura, invece, avviene leggendo il byte User Input, sempre appartenente al SIP.

Grazie alle classi di Saphira la comunicazione è diventata trasparente al programmatore, quindi con l'ausilio di pochi metodi siamo riusciti a gestire le porte di I/O senza preoccuparci del SIP.

Uno di questi metodi, il più importante, è Com2bytes. Riceve in ingresso tre parametri: il primo è il comando del pacchetto SIP, in questo caso DIGOUT, il secondo è il mascheramento della porta e il terzo il valore di set o di reset.

Oltre alle tre classi fondamentali ne è stata creata una quarta ExArguments, per definire le costanti utilizzate per il mascheramento e il settaggio delle porte di I/O.

Di seguito è presentato il codice delle quattro classi:

```
/*
   ------------------------------------------------------

   ExArguments.h

   Header file in linguaggio c++.

   Questo file è stato realizzato da

   Gianmarco Pugliese     Matr. 19526
   Marco Palini      Matr. 25207

   come parte del progetto

   Experimenter

   per il corso di

   Robotica
   Prof. Riccardo Cassinis
   Università degli studi di Brescia
   Facoltà di Ingegneria
   A.A. 2002/03

   Questo file fornisce la dichiarazione di costanti
   utilizzate nella libreria Experimenter, relative alla
   gestione dell'Experimenter module di un robot di tipo
   Pioneer 1.

   ------------------------------------------------------
*/

#ifndef EXARGUMENTS_H
#define EXARGUMENTS_H

class ExArguments
{

public:
        enum Arguments
        {
                /* Maschere dei bit della porta di input
                   DIGIN del robot di tipo Pioneer 1,
                   relative alla pinza dell'Experimenter
                   module. */
                GRIPSWITCH    = 0x07, /* Maschera dei
```

```
                                bit ID0-ID1-
                                ID2, per
                                leggere lo
                                stato della
                                pinza. */
GRIPISTOP      = 0x01, /* Maschera del
                          bit ID0,
                          relativo allo
                          switch per
                          sapere
                          se la pinza è
                          arrivata in
                          alto. */
GRIPISCARRING = 0x02, /* Maschera del
                          bit ID1,
                          relativo allo
                          switch per
                          sapere se la
                          pinza ha
                          iniziato a
                          sollevarsi. */
GRIPISOPEN     = 0x04, /* Maschera del
                          bit ID2,
                          relativo allo
                          switch per
                          sapere se i
                          polpastrelli
                          sono aperti. */
GRIPINNERBB    = 0x40, /* Maschera del
                          bit ID3,
                          relativo alla
                          fotocellula più
                          interna della
                          pinza. */
GRIPOUTERBB    = 0x80, /* Maschera del
                          bit ID4,
                          relativo alla
                          fotocellula più
                          esterna della
                          pinza. */
GRIPBUMP       = 0x20, /* Maschera del
                          bit ID5,
                          relativo ai
                          sensori di
                          contatto
                          della pinza. */
```

```
/* Maschere dei bit della porta di
   output DIGOUT del robot di tipo
   Pioneer 1, relative alla pinza
   dell'Experimenter module. */
GRIPPER       = 0x03, /* Maschera dei
                         bit OD0-OD1,
                         relativi
                         ai comandi
                         della pinza. */
GRIPOPEN      = 0x02, /* Valore
                         codificato
                         dei bit OD0-OD1
                         per abbassare e
                         aprire la
                         pinza. */
GRIPCLOSE     = 0x03, /* Valore
                         codificato
                         dei bit OD0-OD1
                         per alzare e
                         chiudere la
                         pinza. */
RIGHTLED      = 0x08, /* Maschera del
                         bit OD3,
                         relativo al
                         led destro. */
LEFTLED       = 0x10, /* Maschera del
                         bit OD4,
                         relativo al
                         led sinistro.
                      */
LEDOFF        = 0x00, /* Valore
                         codificato
                         dei bit OD3-OD4
                         per indicare lo
                         stato spento di
                         uno dei led
                         della pinza. */

/* Maschere dei bit della porta di
   output DIGOUT del robot di tipo
   Pioneer 1, relative allo speaker
   dell'Experimenter module. */
SPEAKER       = 0x0F  /* Maschera del
                         bit OD5,
                         relativo allo
                         speaker. */
```

```
                /* Maschere dei bit della porta di
                   output DIGOUT del robot di tipo
                   Pioneer 1, relative alle porte
                   analogiche dell'Experimenter module.
                */
                ANALOGPORT    = 0xD0, /* Maschera dei
                                         bit OD5-OD6-
                                         OD7, relativi
                                         alle porte
                                         analogiche. */
        };
};

#endif /* EXARGUMENTS_H */

/*
   ------------------------------------------------------

   ExPion1AnalogPort.h

   Header file in linguaggio c++.

   Questo file è stato realizzato da

   Gianmarco Pugliese     Matr. 19526
   Marco Palini          Matr. 25207

   come parte del progetto

   Experimenter

   per il corso di

   Robotica
   Prof. Riccardo Cassinis
   Università degli studi di Brescia
   Facoltà di Ingegneria
   A.A. 2002/03

   Questo file fornisce la dichiarazione della classe
   ExPion1AnalogPort, che offre delle funzionalità
   relative alla gestione di una delle otto porte
   analogiche fornite dall'Experimenter module di un
   robot di tipo Pioneer 1.

   ------------------------------------------------------
*/
```

```
#ifndef EXPION1ANALOGPORT_H
#define EXPION1ANALOGPORT_H

#include "export.h"

class ExPion1AnalogPort
{

public:
        /* Metodo che legge il valore di una delle otto
           porte analogiche. */
        SFEXPORT int GetAnalogPortValue( int which );
};

#endif /* EXPION1ANALOGPORT_H */

/*
   ------------------------------------------------------

   ExPion1Gripper.h

   Header file in linguaggio c++.

   Questo file è stato realizzato da

   Gianmarco Pugliese     Matr. 19526
   Marco Palini          Matr. 25207

   come parte del progetto

   Experimenter

   per il corso di

   Robotica
   Prof. Riccardo Cassinis
   Università degli studi di Brescia
   Facoltà di Ingegneria
   A.A. 2002/03

   Questo file fornisce la dichiarazione della classe
   ExPion1Gripper, che offre delle funzionalità relative
   alla gestione della pinza di un robot di tipo Pioneer
   1, fornito dell'Experimenter module.

   ------------------------------------------------------
```

```
*/

#ifndef EXPION1GRIPPER_H
#define EXPION1GRIPPER_H

#include "export.h"

class ExPion1Gripper
{

public:
        /* Metodo che cambia lo stato della pinza. */
        SFEXPORT int SetGripperState( int newstate );
        /* Metodo che legge lo stato della pinza. */
        SFEXPORT int GetGripperState( void );
        /* Metodo che testa la presenza di un oggetto
           nella pinza. */
        SFEXPORT bool IsBreakBeamObstructed( int which );
        /* Metodo che testa la presenza di un contatto
           con un oggetto. */
        SFEXPORT bool IsBumpTouching( void );
        /* Metodo che cambia lo stato di uno dei led
           della pinza. */
        SFEXPORT int SetLedState( int which, int newstate
                                  );
        /* Metodo che legge lo stato di uno dei led della
           pinza. */
        SFEXPORT int GetLedState( int which );
};

#endif /* EXPION1GRIPPER_H */

/*
   ------------------------------------------------------

   ExPion1Speaker.h

   Header file in linguaggio c++.

   Questo file è stato realizzato da

   Gianmarco Pugliese     Matr. 19526
   Marco Palini          Matr. 25207

   come parte del progetto

   Experimenter
```

```
   per il corso di

   Robotica
   Prof. Riccardo Cassinis
   Università degli studi di Brescia
   Facoltà di Ingegneria
   A.A. 2002/03

   Questo file fornisce la dichiarazione della classe
   ExPion1Speaker, che offre delle funzionalità relative
   alla gestione dello speaker di un robot di tipo
   Pioneer 1, fornito dell'Experimenter module.

   -----------------------------------------------------
*/

#ifndef EXPION1SPEAKER_H
#define EXPION1SPEAKER_H

#include "export.h"

class ExPion1Speaker
{
        /* Metodo che trasforma la sequenza di note dal
           formato "000/001/002/003" al formato "ABCD".
        */
        char * TransformToneString( const char *sequence,
                                    int n );

public:
        /* Metodo che permette di suonare dallo speaker.
        */
        SFEXPORT int PlayToneString( const char
                                     *sequence, int n ); };

#endif /* EXPION1SPEAKER_H */

/*
   -----------------------------------------------------

   ExPion1AnalogPort.cpp

   Codice sorgente in linguaggio c++.

   Questo file è stato realizzato da
```

```
   Gianmarco Pugliese     Matr. 19526
   Marco Palini          Matr. 25207

   come parte del progetto

   Experimenter

   per il corso di

   Robotica
   Prof. Riccardo Cassinis
   Università  degli studi di Brescia
   Facoltà di Ingegneria
   A.A. 2002/03

   Questo file fornisce l'implementazione dei metodi
   della classe ExPion1AnalogPort, che offre delle
   funzionalità relative alla gestione delle porte
   analogiche fornite dall'Experimenter module di un
   robot di tipo Pioneer 1. Vengono definiti, in
   particolare, i metodi:
   - GetAnalogPortValue

   ------------------------------------------------------
*/

#include "Saphira.h"
#include "ExArguments.h"
#include "ExPion1AnalogPort.h"
#include "export.h"

/*
   Nome           : GetAnalogPortValue
   Tipo restituito: int
                    È il valore che viene letto dalla
                    porta analogica.
   Parametri      : + int which
                      - È un intero che indica quale
                        delle otto porte analogiche si
                        vuole indirizzare.
   Scopo          : Legge il valore di una porta
                    analogica.
   Descrizione    : Interroga l'Experimenter module per
                    conoscere il valore attuale di una
                    delle otto porte analogiche.
*/
SFEXPORT int ExPion1AnalogPort::GetAnalogPortValue( int
```

```
                                                        which
                                                      )
{
        SfROBOT->com2Bytes(
         ArCommands::DIGOUT,ExArguments::ANALOGPORT,which
                          );
        return ( SfROBOT->getAnalog() );
};

/*
   ------------------------------------------------------

   ExPion1Gripper.cpp

   Codice sorgente in linguaggio c++.

   Questo file è stato realizzato da

   Gianmarco Pugliese     Matr. 19526
   Marco Palini      Matr. 25207

   come parte del progetto

   Experimenter

   per il corso di

   Robotica
   Prof. Riccardo Cassinis
   Università degli studi di Brescia
   Facoltà di Ingegneria
   A.A. 2002/03

   Questo file fornisce l'implementazione dei metodi
   della classe ExPion1Gripper, che offre delle
   funzionalità relative alla gestione della pinza di un
   robot di tipo Pioneer 1, fornito dell'Experimenter
   module. Vengono definiti, in particolare, i metodi:
   - SetGripperState
   - GetGripperState
   - IsBreakBeamObstructed
   - IsBumpTouching
   - SetLedState
   - GetLedState

   ------------------------------------------------------
*/
```

```
#include "Saphira.h"
#include "ExArguments.h"
#include "ExPion1Gripper.h"
#include "export.h"

/*
   Nome           : SetGripperState
   Tipo restituito: int
                    Può assumere il valore:
                    - 0 se la comunicazione con il robot
                       è avvenuta correttamente
                    - -1 se si è verificato un problema
                      di comunicazione.
   Parametri      : + int newstate
                      - È un intero che indica il nuovo
                        stato in cui si vuole che la
                        pinza si trovi. Può assumere il
                        valore:
                        * GRIPOPEN per posizionare la
                          pinza nel punto più basso con i
                          polpastrelli aperti
                        * GRIPCLOSE per posizionare la
                          pinza nel punto più alto con i
                          polpastrelli chiusi.
   Scopo          : Cambia lo stato della pinza.
   Descrizione    : Aziona il motore che controlla la
                    pinza del Pioneer 1, cambiando la
                    posizione della pinza e dei
                    polpastrelli.
*/
SFEXPORT int ExPion1Gripper::SetGripperState( int
                                              newstate );
{
        return ( SfROBOT->com2Bytes(
                  ArCommands::DIGOUT,
                  ExArguments::GRIPPER,newstate
                                    ) );
};

/*
   Nome           : GetGripperState
   Tipo restituito: int
                    Può assumere il valore:
                    - GRIPISOPEN se la pinza è
                      posizionata in basso con i
                      polpastrelli aperti
```

```
                    - GRIPISTOP se la pinza è posizionata
                      nel punto più alto
                    - GRIPISCARRING se la pinza è
                      attualmente in movimento e si trova
                      a metà corsa.
   Parametri      : void
   Scopo          : Legge la posizione della pinza.
   Descrizione    : Controlla uno switch per sapere quale
                    è la posizione attuale della piunza.
*/
SFEXPORT int ExPion1Gripper::GetGripperState( void )
{
        return ( SfROBOT->getDigIn() &
                 ExArguments::GRIPSWITCH );
};

/*
   Nome           : exIsBreakBeamObstructed
   Tipo restituito: bool
                    Può assumere il valore:
                    - True se la fotocellula which della
                      pinza vede un oggetto
                    - False se non viene rilevata la
                      presenza di nessun oggetto.
   Parametri      : int which
                    Può assumere il valore:
                    - GRIPINNERBB se la fotocellula che
                      si vuole controllare è quella
                      posizionata nella parte più interna
                      della pinza
                    - GRIPOUTERBB se si vuole controllare
                      l'ostruzione della fotocellula
                      posta nella parte più esterna della
                      pinza.
   Scopo          : Testa la presenza di un oggetto nella
                    pinza.
   Descrizione    : Controlla se è presente un oggetto
                    nella pinza quando è aperta e
                    posizionata in basso, mediante una
                    delle due fotocellule che si trovano
                    sui polpastrelli.
*/
SFEXPORT bool ExPion1Gripper::IsBreakBeamObstructed( int
                                                      Which
                                                     )
{
        return ( SfROBOT->getDigIn() & which );
```

```
};

/*
   Nome           : exIsBumpTouching
   Tipo restituito: bool
                    Può assumere il valore:
                    - True se uno dei sensori ha rilevato
                      una collisione con qualcosa
                    - False se non è stato rilevato
                      nessun contatto.
   Parametri      : void
   Scopo          : Testa la presenza del contatto con un
                    oggetto.
   Descrizione    : Controlla se il robot è entrato in
                    contatto con un ostacolo, mediante
                    sensori posti sulle estremità
                    anteriori dei polpastrelli.
*/
SFEXPORT bool ExPion1Gripper::IsBumpTouching( void )
{
        return !( SfROBOT->getDigIn() &
                  ExArguments::GRIPBUMP );
};

/*
   Nome           : SetLedState
   Tipo restituito: int
                    Può assumere il valore:
                    - 0 se la comunicazione con il robot
                      è avvenuta correttamente
                    - -1 se si è verificato un problema
                      di comunicazione.
   Parametri      : + int which
                      - È un intero che indica su quale
                        led deve essere eseguita
                        l'operazione. Può assumere il
                        valore:
                        * RIGHTLED per applicare il nuovo
                          stato al led destro
                        * LEFTLED per applicare il nuovo
                          stato al led sinistro.
                    + int newstate
                      - È un intero che indica il nuovo
                        stato in cui si vuole porre un
                        led. Può assumere il valore:
                        * LEDOFF per spegnere il led
                        * lo stesso valore del parametro
```

```
                            which per accendere il led.
   Scopo          : Cambia lo stato di uno dei led della
                    pinza.
   Descrizione    : Accende o spegne uno dei due led
                    posizionati sulle estremità superiori
                    dei polpastrelli della pinza.
*/
SFEXPORT int ExPion1Gripper::SetLedState( int which, int
                                          newstate )
{
        return ( SfROBOT->com2Bytes(
                  ArCommands::DIGOUT,which,newstate ) );
};

/*
   Nome           : GetLedState
   Tipo restituito: int
                    Può assumere il valore:
                    - LEDOFF se il led è attualmente
                      spento
                    - lo stesso valore del parametro
                      which se il led è acceso.
   Parametri      : + int which
                      - È un intero che indica su quale
                        led deve essere eseguita
                        l'operazione. Può assumere il
                        valore:
                        * RIGHTLED per applicare il nuovo
                          stato al led destro
                        * LEFTLED per applicare il nuovo
                          stato al led sinistro.
   Scopo          : Legge lo stato di uno dei led della
                    pinza.
   Descrizione    : Controlla in quale stato si trova uno
                    dei due led posizionati sulle
                    estremità superiori dei polpastrelli
                    della pinza.
*/
SFEXPORT int ExPion1Gripper::GetLedState( int which )
{
        return ( SfROBOT->getDigOut() & which );
};

/*
   ------------------------------------------------------
```

```
   ExPion1Speaker.cpp

   Codice sorgente in linguaggio c++.

   Questo file è stato realizzato da

   Gianmarco Pugliese     Matr. 19526
   Marco Palini      Matr. 25207

   come parte del progetto

   Experimenter

   per il corso di

   Robotica
   Prof. Riccardo Cassinis
   Università degli studi di Brescia
   Facoltà di Ingegneria
   A.A. 2002/03

   Questo file fornisce l'implementazione dei metodi
   della classe ExPion1Speaker, che offre delle
   funzionalità relative alla gestione dello speaker di
   un robot di tipo Pioneer 1, fornito dell'Experimenter
   module. Vengono definiti, in particolare, i metodi:
   - TransformToneString
   - PlayToneString

   ------------------------------------------------------
*/

#include "Saphira.h"
#include "ExArguments.h"
#include "ExPion1Speaker.h"
#include "export.h"

/*
   Nome           : TransformToneString
   Tipo restituito: char *
                    È la nuova stringa contenente la
                    sequenza trasformata, ove ad ogni
                    nota sono associati due caratteri, il
                    cui codice ASCII ha il valore pari
                    rispettivamente al byte della durata
                    e del tono della nota.
   Parametri      : + const char *sequence
```

```
                        - È una stringa in cui è
                          memorizzata la sequenza delle
                          note da suonare, contenente al
                          massimo 20 note. Ad ogni nota
                          sono associati sette caratteri
                          della stringa, indicanti
                          rispettivamente:
                          * tre caratteri numerici riferiti
                            al byte della durata, cioè il
                            numero di incrementi che
                            moltiplicato per 20 ms da la
                            durata della nota
                          * il carattere '\' usato come
                            separatore
                          * tre caratteri numerici riferiti
                            al byte del tono, cioè il
                            numero di incrementi che
                            moltiplicato per 150
                            microsecondi definisce il tono
                            della nota.
                          È necessario che la durata ed il
                          tono occupino esattamente tre
                          caratteri.
                      + int n
                        - È il numero di byte di durata e
                          di tono effettivamente presenti
                          nella sequenza.
   Scopo            : Trasforma la sequenza delle note.
   Descrizione      : Riceve in ingresso la stringa delle
                      note in formato
                      "durata\tono\durata\tono" ove durata
                      e tono sono valori numerici
                      compresi tra 0 e 255 e la trasforma
                      in una nuova stringa ove vengono
                      inseriti i codici ASCII di valore
                      durata e tono.
*/
char * ExPion1Speaker::TransformToneString( const char
                                            *sequence,
                                            int n )
{
        char *newsequence; /* Stringa in cui si inserisce
                              la nuova sequenza. */
        int notecounter;   /* Contatore delle note. */
        int srcpos;        /* Indicatore della posizione
                              del carattere letto nella
                              stringa di partenza. */
```

```
        int destpos;        /* Indicatore della posizione
                               del carattere scritto nella
                               stringa di destinazione. */
        char buffer[3];     /* Buffer temporaneo in cui
                               viene messa la durata o il
                               tono letto di una singola
                               nota. */
        int  noteval;       /* Intero contenente il valore
                               numerico convertito dei
                               caratteri del byte letto.
                            */

        newsequence = new char[n];
        for ( notecounter = 0, srcpos = 0, destpos = 0;
              notecounter < n; notecounter++ )
        {
                buffer[0] = *( sequence+srcpos++ );
                buffer[1] = *( sequence+srcpos++ );
                buffer[2] = *( sequence+srcpos++ );
                srcpos++; /* Si salta il terminatore per
                             andare al carattere
                             successivo. */
                noteval = atoi( buffer );
                *( newsequence+destpos++ ) = noteval;
        };
        return ( newsequence );
};

/*
   Nome           : PlayToneString
   Tipo restituito: int
                    Può assumere il valore:
                    - 0 se la comunicazione con il robot
                      è avvenuta correttamente
                    - -1 se si è verificato un problema
                      di comunicazione.
   Parametri      : + const char *sequence
                      - È una stringa in cui è
                        memorizzata la sequenza delle
                        note da suonare, contenente al
                        massimo 20 note. Ad ogni
                        nota sono associati sette
                        caratteri della stringa,
                        indicanti rispettivamente:
                        * tre caratteri numerici riferiti
                          al byte della durata, cioè il
                          numero di incrementi che
```

```
                          moltiplicato per 20 ms
                          da la durata della nota
                        * il carattere '\' usato come
                          separatore
                        * tre caratteri numerici riferiti
                          al byte del tono, cioè il
                          numero di incrementi che
                          moltiplicato per 150
                          microsecondi definisce il tono
                          della nota
                        È necessario che la durata ed il
                        tono occupino esattamente tre
                        caratteri.
                    + int n
                      - È il numero di byte di durata e
                        di tono effettivamente presenti
                        nella sequenza.
   Scopo          : Permette di suonare dallo speaker.
   Descrizione    : Emette una sequenza di note
                    attraverso lo speaker contenuto
                    nell'Experimenter module del Pioneer
                    1.
*/
SFEXPORT int ExPion1Speaker::PlayToneString( const char
                                             *sequence,
                                             int n )
{
        char* newsequence; /* Stringa in cui si inserisce
                              la nuova sequenza. */

        if ( n > 40 )    /* Il numero massimo dei byte */
                 n = 40; /* di durata\tono nella sequenza
                            è pari a 2*20. */
        newsequence = TransformToneString( sequence,n );
        return ( SfROBOT->comStrN(
                  ExArguments::SPEAKER,newsequence,n ) );
};
```

3.3. Testing delle librerie

Terminata la fase di sviluppo si è realizzata una fase di testing della libreria Experimenter mediante la realizzazione di una activity Colbert che potesse verificare la correttezza di tutte le funzionalità offerte.

La piattaforma robotica utilizzata è un robot di tipo Pioneer 1, collegato via radio modem ad un PC con sistema operativo Linux.

In particolare, sulla base delle esercitazioni tenute dal Prof. Cassinis durante il corso di robotica, si è cercato di realizzare una sorta di "rudimentale" robot sminatore. L'activity sviluppata, infatti, permette al robot di muoversi indefinitamente in linea retta sino al raggiungimento di un oggetto. Tale oggetto può essere un ostacolo che il robot non è in grado di trasportare. Ad esempio, questo è il caso in cui, durante il testing, il robot è entrato in collisione con la porta del Laboratorio di Robotica Avanzata. Questa situazione viene rilevata, all'interno dell'activity, mediante le funzioni di gestione dei sensori di contatto posti sulle parti anteriori dei polpastrelli della pinza e comporta uno stop del robot.

Al contrario, nel caso in cui l'oggetto sia trasportabile, la sua presenza viene rilevata mediante le funzioni di gestione delle fotocellule a raggi infrarossi poste sui polpastrelli della pinza. In questo caso viene eseguita una seconda activity denominata "StoreObject", la quale si occupa di spostare fisicamente l'oggetto segnalando con opportuni mezzi visivi e sonori il completamento dell'operazione.

Di seguito viene riportato il codice utilizzato per testare la libreria:

```
/*
   ------------------------------------------------------

   Testexp.act

   Activity file in linguaggio Colbert.

   Questo file è stato realizzato da

   Gianmarco Pugliese     Matr. 19526
   Marco Palini      Matr. 25207

   come parte del progetto

   Experimenter

   per il corso di

   Robotica
   Prof. Riccardo Cassinis
   Università degli studi di Brescia
   Facoltà  di Ingegneria
   A.A. 2002/03

   Questo file fornisce la definizione di una activity
   Colbert utilizzata per il testing della libreria
   Experimenter, relativa all'Experimenter module di un
   robot di tipo Pioneer 1. Nella activity definita il
   robot ricerca degli oggetti, muovendosi in linea retta
   ad una velocità fissata, fino a quando incontra un
```

```
   oggetto che può essere trasportato oppure un ostacolo.
   Nel caso incontri un oggetto trasportabile, stampa un
   messaggio nella finestra comandi di Saphira, lo
   solleva con la sua pinza e lo sposta dal suo percorso.
   In particolare, nel caso in cui l'oggetto sia stato
   rilevato mediante la fotocellula ad infrarossi
   posizionata nella parte più esterna
   della pinza, dopo averlo preso, viene acceso il led
   posizionato sull'estremità del polpastrello destro ed
   il robot si dirige verso destra, in direzione
   perpendicolare rispetto a quella del suo movimento.
   Invece, nel caso in cui sia la fotocellula posta nella
   parte più interna della pinza a rilevare l'oggetto,
   viene acceso il led sinistro ed il robot si dirige
   verso sinistra in una direzione perpendicolare
   rispetto a quella del movimento. In entrambe i casi,
   dopo aver percorso una distanza prefissata, deposita
   l'oggetto prelevato, emette due beep alti dallo
   speaker e fa lampeggiare due volte i led della pinza.
   Successivamente, il robot torna indietro e, dopo aver
   ruotato di 180 gradi rispetto alla direzione di
   movimento originaria, ricomincia la sua ricerca di
   nuovi oggetti. Nel caso in cui si incontri un
   ostacolo, viene stampato un opportuno messaggio nella
   finestra comandi di Saphira, viene emesso un beep di
   tono più grave e viene fermato il robot.

   ------------------------------------------------------
*/

loadlib Experimenter; /* Viene caricata in memoria la
                         libreria Experimenter. */

/*
   Nome           : StoreObject
   Tipo restituito: void
   Parametri      : void
   Scopo          : Sposta un oggetto.
   Descrizione    : Sposta un oggetto in una posizione
                    determinata. Controlla la presenza di
                    un oggetto mediante le fotocellule ad
                    infrarossi presenti sui polpastrelli
                    della pinza. Nel caso in cui venga
                    rilevato un oggetto dalla fotocellula
                    più esterna, il robot accende il led
                    sul polpastrello destro e gira verso
                    destra. Nel caso, invece, sia quella
```

```
                    più interna a rilevare un oggetto,
                    accende il led sinistro e gira a
                    sinistra. Successivamente si sposta,
                    lo deposita, emette due beep alti
                    dallo speaker, fa lampeggiare due
                    volte i led, torna indietro e gira di
                    180 gradi. Se invece incontra un
                    ostacolo segnala la situazione e
                    termina l'activity.
*/
act StoreObject( void )
{
        int counter; /* Contatore del numero di volte che
                        si fanno lampeggiare i led. */

        sfMessage( "Object found." );
        GripperClose();
        wait( 15 );
        if ( IsOuterInGripper() ) /* Se le fotocellule
                                     esterne hanno
                                     rilevato la presenza
                                     di un oggetto si
                                     eseguono delle
                                     azioni */
        {
                TurnOnLeftLed();
                turn( 90 );
        }
        if ( IsInnerInGripper() ) /* altrimenti, se
                                     l'oggetto è stato
                                     trovato da quelle
                                     interne, si eseguono
                                     azioni diverse. */
        {
                TurnOnRightLed();
                turn( -90 );
        }
        move( 500 );
        if ( !IsBumpTouching() ) /* Se non è stata
                                    rilevata una
                                    collisione, si esegue
                                    lo spostamento
                                    dell'oggetto, */
        {
                sfMessage( "Object stored." );
                GripperOpen();
                PlayToneString(
```

```
                 "005\003\005\000\005\003",6 );
                wait( 15 );
                move( -500 );
                if ( IsLeftLedOn() )
                        turn( 90 );
                if ( IsRightLedOn() )
                        turn( -90 );
                counter = 2;
                while ( counter>0  )
                {
                        counter = counter-1;
                        TurnOnRightLed();
                        TurnOnLeftLed();
                        wait( 5 );
                        TurnOffRightLed();
                        TurnOffLeftLed();
                        wait( 5 );
                }
                succeed;
        }
        else                    /* altrimenti si segnala
                                   opportunamente la
                                   situazione. */
        {
                sfMessage( "Motors stalled." );
                PlayToneString( "006\010",2 );
                fail;
        }
}

/*
   Nome           : FindObjects
   Tipo restituito: void
   Parametri      : void
   Scopo          : Trova degli oggetti.
   Descrizione    : Fa muovere il robot in linea retta
                    sino a quando trova un oggetto o
                    incontra un ostacolo. Nel primo caso
                    viene eseguita l'activity StoreObject
                    che sposta l'oggetto dal suo cammino
                    e fa invertire il senso di marcia.
                    Nella seconda situazione, invece, si
                    ferma il robot.
*/
act FindObjects( void )
{
        GripperOpen();
```

```
        speed( 150 );
        while ( 1 ) /* L'activity continua
                       indefinitamente, a meno che non
                       venga chiamata con il parametro
                       timeout <intervallo>. */
        {
                while ( !IsBumpTouching() &&
                        !IsInnerInGripper() &&
                        !IsOuterInGripper() )
                {}
                /* Se l'activity ha esaurito il tempo a
                   sua disposizione, si segnala il
                   fatto. */
                if ( sfTaskFinished( "FindObjects" ) )
            {
                        sfMessage( "Time is over." );
                         PlayToneString(
                          "005\003\005\000\005\003",6 );
                        succeed;
                }
                /* Nel caso sia stata rilevata una
                   collisione vengono eseguite azioni
                   opportune, */
                else if ( IsBumpTouching() )
            {
                        sfMessage( "Motors are
                                    stalled." );
                        PlayToneString( "006\010",2 );
                        stop;
                        move( -200 );
                        fail;
                }
                /* altrimenti, se è stato trovato un
                   oggetto, viene eseguita l'activity
                   StoreObject. */
                else if ( IsOuterInGripper() ||
                          IsInnerInGripper() )
                {
                        start StoreObject();
                        if ( sfTaskFinished(
                              "StoreObject" ) )
                                 speed( 150 );
                        else
                                 fail;
                }
        }
}
```

4. Modalità operative

La libreria Experimenter, come le librerie predefinite di Saphira ed Aria, hanno la possibilità di essere utilizzate sia su calcolatori provvisti dei sistemi operativi della famiglia Windows, sia su calcolatori con sistema operativo Linux.

I componenti software forniti, tuttavia, fanno riferimento al sistema operativo Linux. Qualora l'utente finale voglia utilizzare la libreria Experimenter in ambiente Windows, è necessario che faccia riferimento alle modalità operative riportate nel manuale del compilatore Microsoft Visual C++ v.6.x.

4.1. Componenti necessari

Per poter usare il software bisogna installare sotto Linux due pacchetti scaricabili dal sito della activmedia www.activmedia.com Aria-1.1-0.i386.rpm Saphira-8.1-0.i386.rpm, i quali verranno installati rispettivamente nelle directory /usr/local/Aria e /usr/local/Saphira. Inoltre devono essere create le seguenti variabili d'ambiente:

PATH=/usr/local/bin:$PATH

export SAPHIRA=/usr/local/Saphira

export ARIA=/usr/local/Aria

export LD_LIBRARY_PATH=${SAPHIRA}/lib:${ARIA}/lib

4.2. Modalità di installazione

Per usare sotto Colbert le nostre librerie è sufficiente copiarle in /usr/local/Saphira/lib e sono quelle elencate qui di seguito:

- Experimenter.so
- ExPion1Gripper.so
- ExPion1Speaker.so
- ExPion1AnalogPort.so

Però è necessario che la libreria Experimenter.so venga caricata in Colbert; per farlo basta eseguire il comando "loadlib Experimenter". A questo punto si possono utilizzare le funzione messe a disposizione. Per vedere l'elenco completo basta digitare "help Experimenter" e "help <nome funzione>" per un ulteriore aiuto.

Se altrimenti si preferisce ricompilare tutti i sorgenti basta copiare in /usr/local/Saphira/tutor la directory Experimenter con tutto il suo contenuto, eseguire il makefile in Experimenter/src e in /Experimenter e le librerie saranno automaticamente create nella directory /usr/local/Saphira/lib.

ATTENZIONE: Saphira necessita di una versione particolare di compilatore, la 2.95.3 senza la quale il comando loadlib non va a buon fine.

Di seguito riportiamo i makefile presenti nelle directory Experimenter/src e Experimenter.

```
############################################################
#                                                          #
# ExPion1AnalogPort - ExPion1Gripper - ExPion1Speaker      #
# makefile                                                 #
#                                                          #
# Makefile per le classi relative alle porte               #
# analogiche, alla pinza ed allo speaker contenuti         #
# nella libreria Experimenter.                             #
#                                                          #
# Questo file è stato realizzato da                        #
#                                                          #
# Gianmarco Pugliese     Matr. 19526                         #
# Marco Palini           Matr. 25207                       #
#                                                          #
# come parte del progetto                                  #
#                                                          #
# Experimenter                                             #
#                                                          #
# per il corso di                                          #
#                                                          #
# Robotica                                                 #
# Prof. Riccardo Cassinis                                  #
# Università degli studi di Brescia                        #
# Facoltà di Ingegneria                                    #
# A.A. 2002/03                                             #
#                                                          #
# Requisiti software necessari per la compilazione:        #
# + che siano stati installati i pacchetti:                #
#   - Saphira 8.x                                          #
#   - Aria 1.x                                             #
# + che siano definite le veriabili di ambiente            #
#   - SAPHIRA                                              #
#   - ARIA                                                 #
#   relative alle directory in cui sono stati              #
#   installati i rispettivi pacchetti (vedere il           #
#   file readme.txt nella directory di installazione di    #
#   Saphira per maggiori informazioni)                     #
# + che sia stata installata la libreria Experimenter      #
#   nella directory                                        #
#   - tutor/Experimenter                                     #
#   relativa alla directory di installazione del           #
#   pacchetto Saphira                                      #
# + la presenza del compilatore C++                        #
#   - GNU gcc-2.95.3 per il sistema operativo Linux        #
#   - Microsoft Visual C++ 6.x o successivi per i          #
#     sistemi operativi Windows 9x/ME/2000.                #
```

```
#                                                              #
################################################################

# Definizione delle directory necessarie per la
# compilazione.
EXPERIMENTER = $(SAPHIRA)/tutor/Experimenter
SRCD = ./
OBJD = $(EXPERIMENTER)/obj/
INCD = $(SAPHIRA)/ohandler/include/
BIND = $(SAPHIRA)/bin/
LIBD = $(SAPHIRA)/lib/
ARIAD = $(ARIA)

# Controlla quale è il sistema operativo installato.
include $(INCD)os.h

# Definizione della shell dei comandi.
SHELL = /bin/sh

# Definizione delle directory contenenti gli header file.
INCLUDE = -I$(INCD) -I$(ARIAD)/include
 -I$(EXPERIMENTER)/include/

################################################################

# Definizione dell'insieme di elementi di cui tenere
# traccia nella fase di compilazione.
all: $(OBJD)ExPion1AnalogPort.o $(OBJD)ExPion1Gripper.o
 $(OBJD)ExPion1Speaker.o
        touch all

# Tutti i file sorgente della libreria (con
# estensione.cpp) vengono compilati per ottenere i
# file oggetto corrispondenti (con estensione .o).
$(OBJD)ExPion1AnalogPort.o: $(SRCD)ExPion1AnalogPort.cpp
        $(CC) $(CFLAGS) -c $(SRCD)ExPion1AnalogPort.cpp
         $(INCLUDE) -o $(OBJD)ExPion1AnalogPort.o

$(OBJD)ExPion1Gripper.o: $(SRCD)ExPion1Gripper.cpp
        $(CC) $(CFLAGS) -c $(SRCD)ExPion1Gripper.cpp
         $(INCLUDE) -o $(OBJD)ExPion1Gripper.o

$(OBJD)ExPion1Speaker.o: $(SRCD)ExPion1Speaker.cpp
        $(CC) $(CFLAGS) -c $(SRCD)ExPion1Speaker.cpp
         $(INCLUDE) -o $(OBJD)ExPion1Speaker.o

################################################################
```

```
#                                                              #
# Experimenter makefile                                        #
#                                                              #
# Makefile per la libreria Experimenter.                       #
#                                                              #
# Questo file è stato realizzato da                            #
#                                                              #
# Gianmarco Pugliese Matr. 19526                        #
# Marco Palini           Matr. 25207                           #
#                                                              #
# come parte del progetto                                      #
#                                                              #
# Experimenter                                                 #
#                                                              #
# per il corso di                                              #
#                                                              #
# Robotica                                                     #
# Prof. Riccardo Cassinis                                      #
# Università degli studi di Brescia                            #
# Facoltà di Ingegneria                                        #
# A.A 2002/03                                                  #
#                                                              #
# Requisiti software necessari per la compilazione:            #
# + che siano stati installati i pacchetti:                    #
#   - Saphira 8.x                                              #
#   - Aria 1.x                                                 #
# + che siano definite le veriabili di ambiente                #
#   - SAPHIRA                                                  #
#   - ARIA                                                     #
#   relative alle directory in cui sono stati                  #
#   installati i rispettivi pacchetti (vedere il               #
#   file readme.txt nella directory di installazione di        #
#   Saphira per maggiori informazioni)                         #
# + che sia stata installata la libreria Experimenter          #
#   nella directory                                            #
#   - tutor/Experimenter                                       #
#   relativa alla directory di installazione del               #
#   pacchetto Saphira                                          #
# + la presenza del compilatore C++                            #
#   - GNU gcc-2.95.3 per il sistema operativo Linux            #
#   - Microsoft Visual C++ 6.x o successivi per i              #
#     sistemi operativi Windows 9x/ME/2000.                    #
#                                                              #
################################################################

# Definizione delle directory necessarie per la
# compilazione.
```

```
EXPERIMENTER = $(SAPHIRA)/tutor/Experimenter
SRCD = ./
OBJD = $(EXPERIMENTER)/obj/
INCD = $(SAPHIRA)/ohandler/include/
BIND = $(SAPHIRA)/bin/
LIBD = $(SAPHIRA)/lib/
ARIAD = $(ARIA)

# Controlla quale è il sistema operativo installato.
include $(INCD)os.h

# Definizione della shell dei comandi.
SHELL = /bin/sh

# Definizioni delle directory contenenti gli header file.
INCLUDE = -I$(INCD) -I$(ARIAD)/include
 -I$(EXPERIMENTER)/include/

##########################################################

# Definizione dell'insieme di elementi di cui tenere
# traccia nella fase di compilazione.
all: $(LIBD)Experimenter.so
        touch all

# Il file sorgente Experimenter.cpp viene compilato per
# ottenere il file oggetto corrispondente Experimenter.o.
$(OBJD)Experimenter.o: $(SRCD)Experimenter.cpp
        $(CC) $(CFLAGS) -c $(SRCD)Experimenter.cpp
         $(INCLUDE) -o $(OBJD)Experimenter.o

# Tutti i file oggetto precedentemente creati vengono
# linkati insieme per creare lo shared object
# Experimenter.so.
$(LIBD)Experimenter.so: $(OBJD)Experimenter.o
 $(OBJD)ExPion1AnalogPort.o $(OBJD)ExPion1Gripper.o
 $(OBJD)ExPion1Speaker.o
        $(LD) $(SHARED) -o $(LIBD)Experimenter.so
         $(OBJD)Experimenter.o $(OBJD)ExPion1AnalogPort.o
         $(OBJD)ExPion1Gripper.o $(OBJD)ExPion1Speaker.o
```

4.3. Avvertenze

Durante lo sviluppo della libreria Experimenter, si è fatto riferimento ad un particolare esponente della piattaforma robotica di tipo Pioneer 1, denominato “Tobor”. Questo robot, posseduto dal Laboratorio di Robotica Avanzata dell’Universita degli Studi di Brescia ha subito alcune opere di manutenzione da parte del responsabile del

laboratorio stesso, il Prof. Riccardo Cassinis. In particolare, quest'ultimo ha modificato la disposizione dei bit, all'interno della porta di input digitale, relativi alle fotocellule poste sui polpastrelli della pinza.

Questo lavoro inoltre ha messo in luce una incongruenza tra quanto riportato in [11] e il robot utilizzato, relativamente al bit della porta di output digitale che si occupa di gestire il verso di rotazione del motore che controlla la pinza.

Entrambe le modifiche effettuate sono riportate nella seguente tabella:

Uso	*Valore errato*	*Valore corretto*
Paddle inner break beam	ID3	ID6
Paddle outer break beam	ID4	ID7
Grippe motor rotation dir. (OD0)	1=down 0=up	0=down 1=up

5. Conclusioni e sviluppi futuri

È stata realizzata una libreria per comunicare con l'experimenter module del pioneer1 dall'interfaccia Colbert, appoggiandoci alle classi di Saphira 8.1.

Questo lavoro ha avuto come scopo la creazione di una libreria, utilizzabile all'interno del pacchetto software Saphira 8.2, in grado di fornire all'utente finale una gestione semplice dell'hardware contenuto nel modulo Experimenter. Alla fase di sviluppo vera e propria ha fatto seguito una fase di testing di quanto realizzato. Tuttavia, data l'assenza nel Laboratorio di Robotica Avanzata di dispositivi da poter collegare alle porte analogiche del modulo Experimenter, non sono state sviluppate funzionalità pensate specificamente per permettere all'utente di controllare lo stato di un dispositivo collegato.

Uno sviluppo possibile è costituito quindi dall'estensione della libreria Experimenter mediante funzionalità specifiche per la gestione di particolari classi di dispositivi collegati a tali porte, che possano riultare di interesse.

Bibliografia

[1] Konolige, K.: "Saphira Software Manual", Saphira Version 8 (Saphira/Aria integration), 2001

[2] Konolige, K.: "SAPHIRA TUTORIAL", Compiling, Loading and Debugging C++ Files: Unix Systems Sofware version 8.0 (Saphira/Aria), September 2001

[3] Doxygen 1.2.10: "Saphira Reference Manual 8.2.0", February 2003

[4] Doxygen 1.2.10: ”Aria Reference Manual 1.2.0”, February 2003

[5] Konolige, K.: ”Colbert User Manual”, Software version 8.0a (Saphira/Aria), September 2001

[6] Konolige, K.: ”COLBERT: A Language for Reactive Control in Saphira”, February 2003

[7] ActiveMEDIA ROBOTICS: “Pioneer Mobile Robot Operation Manual, Edition 2”, January 1998

[8] ActivMEDIA ROBOTICS:”Pioneer 2 Mobile Robot – Operation Manual”, January 1999

[9] ActivMEDIA ROBOTICS:”Pioneer 2 Gripper Manual v2”, February 2000

[10] ActivMEDIA ROBOTICS:”Pioneer 2 Arm Manual”, October 2002

[11] ActivMEDIA ROBOTICS: “Gripper & Experimenter’s Module Manual”, August 1997

Indice