



UNIVERSITÀ DEGLI STUDI DI TRIESTE

FACOLTÀ DI INGEGNERIA

Dipartimento di Elettrotecnica, Elettronica ed Informatica

Tesi di Laurea in
SISTEMI OPERATIVI

Sviluppo di un robot mobile a comando vocale

Laureando:

Lorenzo DAL COL

Relatore:

Prof. Ing. Enzo MUMOLO

Correlatore:

Dott. Ing. Massimiliano NOLICH

Anno Accademico 2006-2007

Ai miei genitori

CAPITOLO 1 INTRODUZIONE	1
1.1 Obiettivi.....	1
1.1 Architettura del sistema	3
1.2 Il TS-7250	3
1.3 Linguaggi e strumenti utilizzati.....	4
CAPITOLO 2 IL RICONOSCIMENTO VOCALE.....	6
2.1 Dragon Naturally Speaking	6
2.2 Sviluppo con le API	6
2.3 Il programma che effettua il riconoscimento	7
CAPITOLO 3 LINUX PER L'ARM	10
3.1 Requisiti per il sistema operativo.....	10
3.2 Il Cross-Compiler.....	11
3.3 Messa a punto del sistema Real-Time.....	12
3.4 Compilazione del kernel.....	13
3.5 Compilazione dei rtai-magma	14
3.6 Compilazione dei moduli audio (OSS)	15
3.7 Compilazione degli Alsa driver.....	15
3.8 Problemi riscontrati	16
CAPITOLO 4 REGISTRAZIONE DEI SUONI SU ARM.....	17
4.1 Compilazione di SOX.....	17
4.2 Registrazione tramite SOX e OSS	18
4.3 Ricampionamento del file Wave	20
CAPITOLO 5 COMUNICAZIONE TRA PROCESSI	21
5.1 L'Open Agent Architecture	21
5.2 Implementazione del riconoscitore vocale distribuito su OAA.....	22
5.3 Problemi riscontrati con OAA	23
5.4 Comunicazione tra processi tramite Socket.....	25
5.5 Esempio di client TCP per Linux	26
5.6 Esempio di server TCP per Linux	27
5.7 Compatibilità tra socket in Windows e in Linux.....	27
5.8 Implementazione del server in Windows	28
CAPITOLO 6 RICONOSCIMENTO DEI COMANDI VOCALI	30
6.1 Rilevamento delle parole	30
6.2 Riconoscimento del comando	30

CAPITOLO 7 MODULI REAL-TIME PER IL MOVIMENTO	33
7.1 I moduli del Kernel	33
7.2 Creazione di un modulo del Kernel.....	34
7.3 Esecuzione in Real-Time	35
CAPITOLO 8 IL MOVIMENTO DEL ROBOT	37
8.1 Le porte di I/O	37
8.2 Controllo dei motori.....	39
8.3 Lettura del sonar	40
8.4 Comunicazione tra modulo real-time e processo utente (RTAI IPC).....	43
8.5 Esecuzione dei comandi di movimento.....	45
CAPITOLO 9 CONCLUSIONI	46
9.1 Risultati raggiunti.....	46
9.2 Sviluppi futuri	47
APPENDICI	49
A. Red Boot: avvio del sistema operativo sull'ARM.....	49
B. Organizzazione dei sorgenti	50
C. Procedura completa di funzionamento.....	52
BIBLIOGRAFIA.....	54
RINGRAZIAMENTI	56
LICENZA D'USO.....	57
<i>Creative Commons Notice.....</i>	<i>60</i>

Capitolo 1

INTRODUZIONE

1.1 OBIETTIVI

L'obiettivo è quello di realizzare un robot mobile controllato a comando vocale, cioè un robot in grado di muoversi in ambienti domestici comandato tramite parole o brevi frasi pronunciate da una persona. Questo comporta la realizzazione di un software distribuito (su più processori) in modo che un calcolatore possa registrare delle parole o frasi pronunciate da una persona e occuparsi del movimento mentre un secondo calcolatore effettua il riconoscimento vocale sulla registrazione e invia il risultato testuale alla prima macchina. Il software deve occuparsi di tutte le fasi: la registrazione della voce, il riconoscimento delle parole, del comando, il controllo a basso livello dei motori e di un sonar per rilevare la distanza del robot dagli ostacoli.

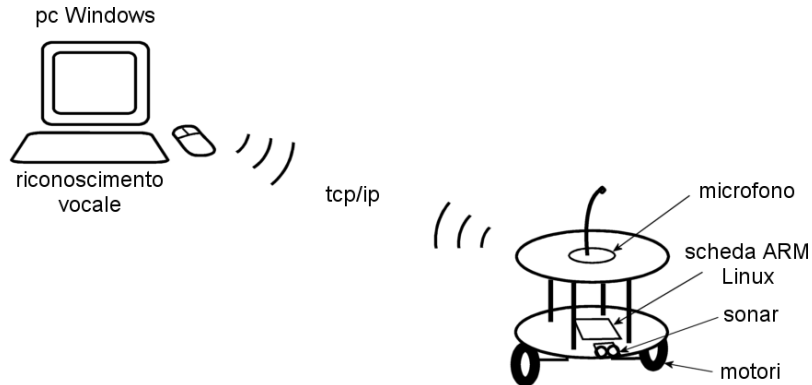


Figura 1: Il robot mobile

L'idea è di equipaggiare il robot con due ruote motrici e un sonar, collegati ad una scheda ARM con un sistema Linux che serve a coordinare tutto. La scheda ARM deve essere autosufficiente per quanto riguarda la registrazione dei comandi vocali, il controllo dei motori, del sonar e un minimo di logica per evitare di scontrare il robot sugli ostacoli, mentre solamente la fase di riconoscimento vocale viene assegnata a un sistema esterno Windows per il riconoscimento vocale.

Il funzionamento tipico da realizzare risulta essere il seguente:

1. la scheda ARM montata sul robot registra tramite il microfono USB la voce umana
2. viene rilevata una parola o una breve frase e inviata tramite ethernet in formato WAVE al computer con Windows che fa da "server"

3. il server riceve la registrazione ed esegue il riconoscimento vocale, restituendo al robot la frase trascritta
4. la frase viene decifrata per rilevare la presenza di un comando per il movimento
5. il comando viene eseguito dalla scheda ARM che si occupa anche di generare l'onda per muovere i motori
6. nel frattempo la scheda ARM sta leggendo un sensore ad ultrasuoni posizionato davanti al robot per controllare la distanza dagli ostacoli e fermare i motori in caso di pericolo.

Per supportare queste operazioni è necessario che il sistema che gira sulla scheda ARM sia allo stesso tempo ad alto livello di astrazione (per poter comunicare tramite socket in TCP, registrare in WAVE, supportare un microfono USB, ecc) e a basso livello, per poter controllare direttamente i motori elettrici e il sensore ad ultrasuoni. Per quanto riguarda la logica ad alto livello è sufficiente compilare e configurare appositamente la distribuzione Linux basata su Debian fornita insieme alla scheda ARM.

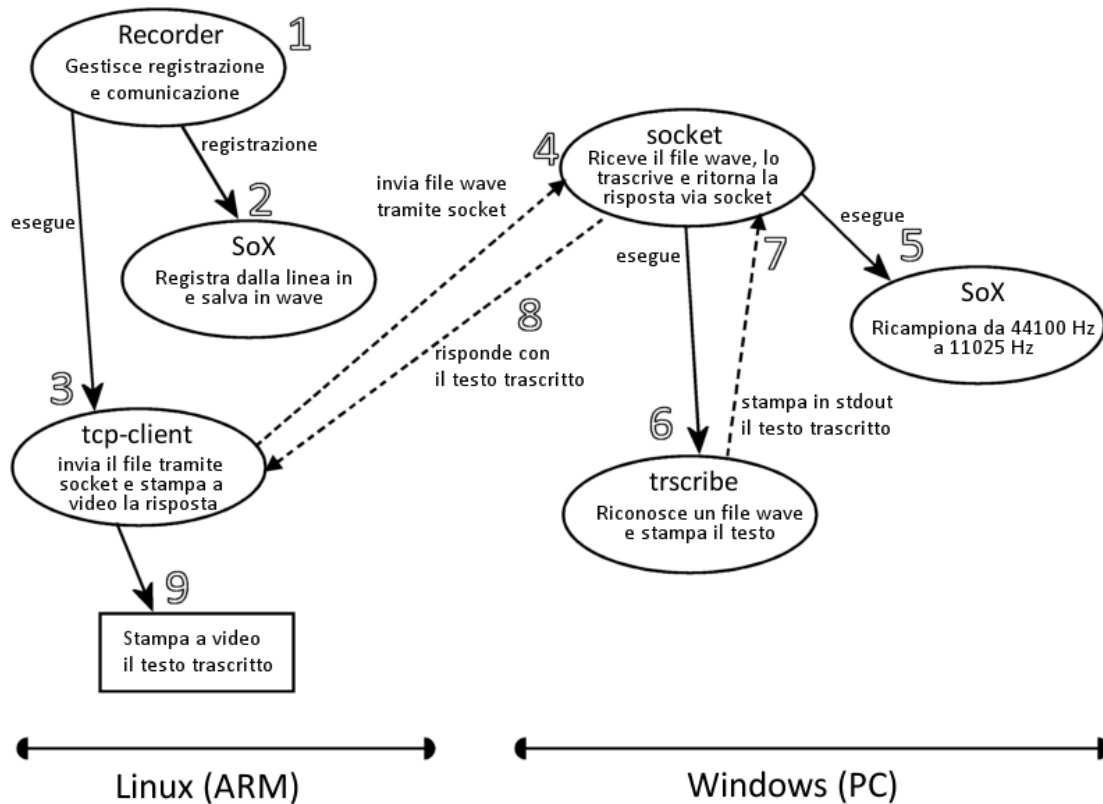


Figura 2: Schema dei processi per il riconoscimento vocale distribuito

Per poter controllare in tempo reale i motori e il sonar, che richiedono tempistiche precise e veloci, sarebbe necessario un vero sistema real-time, dove la parallelizzazione tra le routine viene controllata in modo preciso. Fortunatamente esiste la patch RTAI Magma da applicare al kernel di Linux e alcuni moduli da avviare nel caricamento che serve a questo scopo. Grazie a questa patch è infatti possibile eseguire dei moduli in tempo reale con priorità altissima, che non vengono schedati da Linux come i processi utente ma controllati direttamente a basso

livello. Questi moduli quindi possono essere programmati per eseguire alcune operazioni secondo tempistiche accurate e deterministiche, a differenza dei processi utente, quindi risultano particolarmente utili per le operazioni come la lettura del sonar e la generazione dell'onda quadra per controllare i motori. Si vuole realizzare un'architettura "orientata al processo", in modo da poter sostituire in futuro ogni singolo componente del sistema semplicemente sostituendo degli eseguibili.

1.1 ARCHITETTURA DEL SISTEMA

Il sistema è composto dalle seguenti componenti:

- Macchina Linux:
 - Scheda Arm TS-7250 prodotta da Technologic System
 - TS-Linux Embedded Operating System
- Macchina Windows:
 - Computer architettura x86
 - Microsoft Windows XP
 - Dragon Naturally Speaking 5 (software proprietario per il riconoscimento vocale)
- Telaio del robot: Max '99 Robot Kit prodotto da Zagros Robotics
- Sensore a ultrasuoni: Devantech SRF04/05 UltraSonic Ranger

I due computer comunicano tra loro sul layer IP (tcp/udp) grazie a una connessione Ethernet. Per la registrazione è stato utilizzato un microfono USB connesso alla board: Trust Digital USB Microphone MC-3200

- Impedenza: 2,2k Ohm
- Gamma frequenze: 50Hz - 20kHz
- Connessione USB

1.2 IL TS-7250

Il TS-7250 è un Single Board Computer compatto basato sul processore Cirrus EP9302 ARM9. Questo processore è un ARM920T a 200 MHz, con inclusa un'unità di gestione della memoria (MMU) che permette il supporto a sistemi operativi ad alto livello, quali Linux, Windows CE, ecc. In quanto processore general-purpose provvede inoltre un set di periferiche standard nella scheda e un set aggiuntivo di periferiche tramite il Bus standard PC/104. L'ARM920T ha un'architettura a 32bit, pipeline a 5 stadi, 16 KB di cache per le istruzioni e 16 KB per i dati.

Ecco le caratteristiche generali della scheda TS-7250:

- Sistema operativo embedded TS-Linux (basato su Debian)
- ARM9 CPU a 200 MHz con MMU
- Memoria Flash a 32 MB (o 64, 128 a seconda delle versioni)
- 32 MB di ram (o 64)
- 2 USB Compatibili con lo standard USB 2.0 (12 Mbit/s massimo)
- 2 porte seriali
- Ethernet 10/100

- 20 linee digitali di I/O
- convertitore A/D a 12 bit (5 canali)
- Watchdog Timer
- Bus di espansione PC/104
- Alimentazione +5VCD a 450 mA
- Dimensioni ridotte (9,7 x 11,5 cm)
- Temperatura di lavoro da -20 a +70 °C
- Consumi ridotti (massimo 2 Watt)

Per mantenere i costi ridotti la serie TS-7200 non ha controller video e interfaccia per la tastiera, quindi viene usata la porta seriale COM1 come interfaccia console verso un programma di emulazione terminale che gira su un computer esterno. Si può utilizzare Hyperterm su Windows, oppure minicom su Linux, inoltre si può utilizzare Putty, programma per terminali di svariati tipi, disponibile per entrambe le piattaforme. E' sufficiente un terminale ANSI o un emulatore per connettersi alla scheda ARM, utilizzando i seguenti parametri: 115200 baud, dati a 8 bit senza parità, 1 bit di stop (inoltre deve essere impostato il jumper JP2 nella scheda).

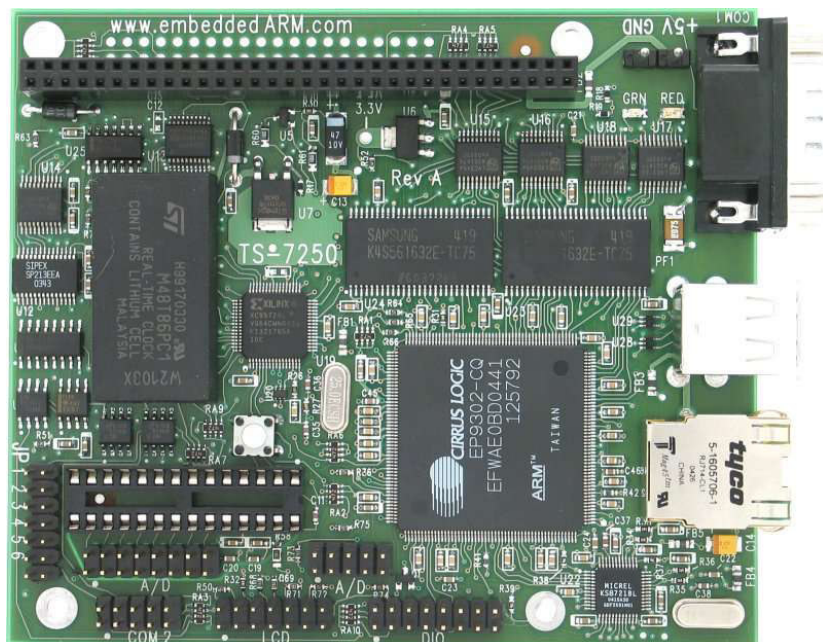


Figura 3: Scheda ARM 7250

1.3 LINGUAGGI E STRUMENTI UTILIZZATI

Per la realizzazione del programma è stato utilizzato principalmente il linguaggio C/C++, con il cross-compiler gcc-3.3.4-glibc-2.3.2 per Linux sull'ARM e il compilatore vcc di Microsoft Visual Studio 2005 per la parte di programma che gira su Windows XP. Per quanto riguarda

la registrazione su Linux è stato utilizzato il programma SOX appoggiato a Open Sound System (OSS) come driver audio.

Ho cercato di mantenere tutto quello che riguarda l'arm dentro la cartella /tools, sia il cross compilatore, sia i sorgenti, sia i binari compilati per arm.

Tutte le operazioni di compilazione sono state svolte in ambiente Kubuntu 7.04 e Windows XP Professional.

Capitolo 2

IL RICONOSCIMENTO VOCALE

2.1 DRAGON NATURALLY SPEAKING

Il software per il riconoscimento vocale è il Dragon Naturally Speaking 5, che è in grado di riconoscere le parole direttamente dalla linea in del computer in cui gira, oppure da un file wave (campionato a un canale, 11025 Hz, 16bit per campione mono).

Questo programma commerciale, di seguito chiamato Dragon 5, si appoggia alla Microsoft Speech Application Programming Interface (SAPI) e fornisce API per poter utilizzare le funzionalità di riconoscimento direttamente dal proprio software.

Il software Dragon Naturally Speaking 5 viene installato in Windows e si propone con una barra di strumenti sempre attiva che può interagire con moltissimi altri programmi che girano, come Word, il browser, il notepad, ecc. Inoltre è in grado di capire alcuni comandi ed eseguire le corrispondenti operazioni, per esempio eseguire un programma, chiudere una finestra ed altre.

Appena terminata l'installazione il Dragon 5 ha bisogno di essere istruito, per adattarsi al timbro, la velocità, gli accenti della voce dell'utente. Questo addestramento avviene leggendo alcuni brani proposti dal software, che adatta alcune variabili in modo da ridurre gli errori di riconoscimento vocale. Ne segue che ogni utente ha un proprio profilo personale all'interno di Dragon 5, e non è scontato che due persone che utilizzano lo stesso profilo possano essere capite correttamente dal software.

2.2 SVILUPPO CON LE API

Avendo a disposizione l'SDK (Software Development Kit) di Naturally Speaking è possibile sviluppare proprie applicazioni utilizzando le API di riconoscimento vocale. Le applicazioni si possono appoggiare direttamente alle SAPI di Microsoft oppure alle api proprietarie di Dragon: nel primo caso si accede ai metodi grezzi che si occupano del riconoscimento vocale, nel secondo caso si hanno a disposizione tutte le funzionalità per interagire con Dragon 5. Tramite le API è possibile gestire i profili degli utenti, accedere ai menù e alle finestre di Dragon, l'interfaccia per i comandi vocali, per il riconoscimento tramite microfono, e, soprattutto, alle funzioni per la trascrizione dei file wave.

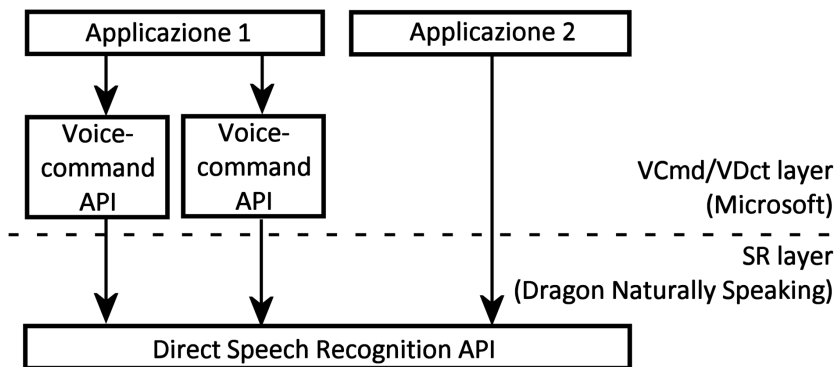


Figura 4: Architettura API di Naturally Speaking

Il metodo per trascrivere i file wave appartiene alla Custom Dictation Interface (IDgnDictCustom); di seguito riporto la documentazione a riguardo:

To transcribe dictation from a .WAV file into a standard Windows edit control
 Call the TranscribeFile method on the dictation-edit (or custom dictation) object:

```
pIDgnDictEdit->TranscribeFile("c:\\wavfiles\\openmail.wav");
```

You can use the TranscriptionStopped method of the IDgnDictEditNotifySink interface to notify your application when the transcription stops or is completed.

Purtroppo il metodo di trascrizione si può applicare solamente ad un oggetto pIDgnDictEdit, che è un controllo edit di MFC (Visual C++), in pratica una textarea. Questo significa che il metodo TranscribeFile non restituisce la stringa riconosciuta, ma è solamente in grado di scriverla dentro un controllo edit. Inoltre l'esecuzione di questo metodo termina quasi istantaneamente, mentre il riconoscimento viene fatto da un thread esterno di Dragon 5. Pertanto è necessario impostare una funzione di callback che viene richiamata quando Dragon 5 termina l'esecuzione per conoscere esattamente il momento in cui il riconoscimento è finito e poter utilizzare la stringa, recuperandola appositamente dal controllo edit. Questa limitazione comporta che è impossibile utilizzare le API di Dragon su un semplice programma C/C++ eseguito in console, ma è necessario un programma in finestra, eseguito in Windows.

2.3 IL PROGRAMMA CHE EFFETTUA IL RICONOSCIMENTO

Ho pensato di creare un programma in grado di effettuare il riconoscimento vocale di un file wave nel modo più semplice in assoluto per essere usato in un sistema ad agenti. I requisiti di questo programma sono:

- deve poter essere eseguito in console
- deve accettare come parametro di ingresso il percorso del file wave da trascrivere
- deve scrivere nello standard output la stringa testuale riconosciuta e chiudersi

In questo modo si guadagna la comodità di poter ridirezionare lo standard output del riconoscitore nello standard input di altri programmi, e allo stesso tempo fare testing in

console. Per realizzare questo, tenendo conto che le API di Dragon hanno bisogno di lavorare su dei controlli visuali, è stato scritto un Main classico che a sua volta esegue il WinMain che si occupa della creazione della finestra e la gestione dei messaggi di Windows.

```
int main(int argc, char *argv[]) {
    if (argc!=2)
        exit(1);
    filename = argv[1];
    WinMain(NULL,NULL,NULL,1);
}
```

Il WinMain contiene le routine per l'inizializzazione della finestra di dialog e il ciclo "infinito" che legge i messaggi di sistema. Nella funzione che controlla tutti i messaggi di sistema:

```
BOOL CALLBACK DialogProc( HWND hDlg, UINT uMsg, WPARAM wParam, LPARAM lParam )
```

se viene rilevato il messaggio WM_INITDIALOG si procede all'inizializzazione degli oggetti utilizzati dalle API di Dragon, si ridirigono gli standard output e input, e si chiama la procedura che effettua la trascrizione:

```
case WM_INITDIALOG:
// inizializzo gli oggetti per le API di Dragon
initAll(hDlg);
hStdout = GetStdHandle(STD_OUTPUT_HANDLE);
hStdin = GetStdHandle(STD_INPUT_HANDLE);
if ((hStdout == INVALID_HANDLE_VALUE) || (hStdin == INVALID_HANDLE_VALUE))
    ExitProcess(1);
transcribe(filename);
return TRUE;
```

La initAll sostanzialmente contiene i seguenti comandi, abbreviati qui eliminando il controllo degli errori:

```
// Cerco se è stato caricato un profilo utente di Dragon
bResult = TryUsers( hDlg );
// set up Dictation Edit
hRes = initDictEdit( hDlg );
```

La funzione che effettua la trascrizione del file wave è la seguente:

```
BOOL transcribe(char *filename) {
    SetWindowText(GetDlgItem( hDlg, IDC_EDITTEXT ), "");
    // get the filename
    char *szFilename = filename;
    if ( sizeof(szFilename) == 0 )    {
        // probably no filename entered in the box
        printf("Invalid Filename");
        return TRUE;
    }
    // transcribe the file
    assert( g_pIDgnDictEdit != NULL );
    HRESULT hRes = g_pIDgnDictEdit->TranscribeFile( szFilename );
    if( FAILED( hRes ))
```

```

{
    if( hRes == E_NOTIMPL )      {
        wsprintf( g_szError,
            _T("The version of Dragon NaturallySpeaking ")
            _T("currently running does not support ")
            _T("transcription.") );
    }
    else {
        wsprintf( g_szError,
            _T("IDgnDictEdit->TranscribeFile( %s ) ")
            _T("failed, hRes = 0x%X"), szFilename, hRes );
        if ( hRes == E_INVALIDARG )      {
            _tcscat( g_szError,
                _T("\n\nPlease make sure that the file ")
                _T("exists and the file format is ")
                _T("11.025 kHz, 16 bit mono.") );
        }
    }
    printf("%s",g_szError);
    PostQuitMessage( 0 );
    return TRUE;
}
}

```

La chiamata TranscribeFile non è bloccante, perché passa l'esecuzione a un thread della API di Dragon, pertanto quando termina l'esecuzione di Transcribe il riconoscimento non è terminato quasi certamente. Per questo viene impostata una funzione di callback (una sorta di evento) che viene chiamato da Dragon quando la trascrizione è terminata:

```

STDMETHODIMP TranscriptionStopped() {
    HWND m_edittext = GetDlgItem( m_hWndParent, IDC_EDITTEXT );
    char *s;
    GetWindowText(m_edittext, s, 256);
    s = strcat(s, " ");
    bool fSuccess = WriteFile(hStdout, s, strlen(s), &dwWritten, NULL);
    if (! fSuccess)
        exit(1);
    exit(0);
    return S_OK;
}

```

Quando viene eseguito il callback TranscriptionStopped questo recupera dal controllo edit il testo riportato dal riconoscitore e lo scrive nello standard output, già ridirezionato nel Main. Infine viene terminata l'esecuzione del programma.

Capitolo 3

LINUX PER L'ARM

3.1 REQUISITI PER IL SISTEMA OPERATIVO

Il dispositivo ts7250 fornito da Technologic System è provvisto di un sistema operativo ridotto, basato su kernel Linux. Questo sistema operativo gestisce tutte le funzionalità "general purpose" comunemente usate dai computer e tutte le funzionalità offerte dal processore EP9302 montato, detto anche ARM9. Il sistema operativo ufficialmente supportato dall'azienda è basato su un kernel 2.4.26 di Linux. Quest'ultimo, a sua volta, è ufficialmente supportato e modificato per la referenziazione delle zone di memoria dell'architettura, in particolare, queste sono dettate dal layout della scheda e dal processore Cirrus EP9302. Il sistema fornito da Technologic System caricato nella scheda è la versione 2.4.26-ts9, la sigla ts9 sta ad indicare che la versione di rilascio dalla casa costruttrice è la 9.

Dati gli obiettivi del progetto, che prevedono l'utilizzo di un sistema di gestione dei processi a priorità e la necessità di avere la massima sicurezza sulla durata di un processo, si è ricercata una soluzione praticabile. L'impedimento maggiore, in questo senso, è stato dato dal sistema operativo fornito dalla casa costruttrice, il quale rendeva impossibile la gestione dei processi in modalità realtime. Pertanto, si è dovuto provvedere alla modifica di un altro sistema operativo, sempre fornito dalla Technologic System, in modo da renderlo realtime.

Il primo passo è stato, quindi, di cercare una versione funzionante di sistema operativo fornita dalla Technologic System, l'unica versione che sembra funzionare correttamente e che crea meno difficoltà di compilazione e di allocazione di zone di memoria in avvio è la ts11. I sorgenti del sistema operativo si trovano nel sito www.embeddedarm.com, inclusi nel tarball `tskernel-2.4.26-ts11-src.tar.gz`.

Il secondo passo è stato quello di cercare una modifica possibile per il sistema operativo fornito dalla Technologic System, la quale risulta disponibile in molti siti web, tuttavia è consigliabile scaricarla dal sito della ditta stessa. Questo perché, la modifica integra parti che toccano seriamente la specificità del dispositivo, non solamente le API del sistema operativo che si desidera compilare.

La modifica si trova nel sito www.embeddedarm.com e si trova sotto il nome di `rt-tskernel-2.4.26-tsX-adeos.patch`. Questa modifica, tuttavia, è generica e prima di passare alla sua applicazione è bene tenerne conto. La genericità che questa ha, implica una modifica puntuale e ordinata per poter essere applicata.

Il terzo passo, uno tra i più delicati, è quello di recuperare un cross-compilatore in grado di compilare i file sorgente del sistema operativo in codifica ARM (la codifica del nostro dispositivo). Non è una cosa semplice recuperare dal web un cross-compilatore funzionante perché vi sono moltissime variabili che lo portano a funzionare in maniera ambigua e, i risultati ai quali può portare sono i più disparati. Gli errori che si potrebbero incontrare, in caso di scelta errata, portano sempre ad una notevole perdita di tempo e, in molti casi anche all'impossibilità di generare kernel Linux funzionanti.

3.2 IL CROSS-COMPILER

Il cross-compiler è un compilatore che genera file eseguibili per una piattaforma differente rispetto a quella in cui gira. I cross-compilatori vengono usati regolarmente per generare software per sistemi embedded, dove la capacità di calcolo è ridotta e la compilazione richiederebbe molto tempo o risulterebbe addirittura impossibile, come in microcontrollori con memoria molto ridotta. Gcc è una collezione di compilatori che può essere impostata per effettuare cross-compilazioni per diverse architetture e linguaggi. Gcc necessita di una copia delle binutils compilata per la piattaforma finale.

Per quanto riguarda i processori ARM esistono nel web moltissimi cross-compilatori differenti, per molte architetture (in particolare Cygwin e Linux x86), ma è necessario scegliere la versione adatta alla scheda utilizzata per il progetto. Il cross-compilatore usato si trova nel sito <ftp://oz.embeddedarm.com/yaffs> sotto il nome di `crosstool.tgz`, ed è l'unico cross-compilatore fornito dalla Technologic System che permette, in maniera relativamente semplice, una corretta compilazione del kernel preso in considerazione. Per corretta compilazione si intende non solo che il processo di compilazione che porta ad un'immagine del sistema operativo vada a buon fine, ma anche al corretto caricamento nel sistema ts7250 fornito.

Dopo queste operazioni preliminari si può passare alla prima fase di compilazione del kernel, che riguarda la creazione di una cartella utile per la decompressione del file `rt-tskernel-2.4.26-ts11-src.tar.gz`, l'applicazione della modifica al kernel, la compilazione dello stesso e lo stoccaggio delle immagini del kernel (risultato delle compilazioni fatte) in una directory apposita, per il caricamento nella scheda via rete, utilizzando il comune protocollo http supportato da RedBoot.

E' stato scaricato il cross compilatore "arm-9tdmi-linux-gnu" dall'indirizzo:

```
ftp://ftp.embeddedarm.com/ts-arm-linux-cd/cross-toolchains/crosstool-linux-gcc-4.0.1-glibc-2.3.5.tar.bz2
```

Successivamente è stato decompresso nella cartella `/tools`, in modo da avere il percorso `/tools/crosstool/gcc-4.0.1-glibc-2.3.5/arm-unknown-linux-gnu/bin`

```
$ tar xf crosstool-linux-gcc-4.0.1-glibc-2.3.5.tar.bz2
$ mv /tools/opt/crosstool /tools/crosstool
```

Ci sono diversi cross-compilers disponibili, ne sono stati provati alcuni, ma quello che ha dato meno problemi è `arm-9tdmi-linux-gnu`.

Per completare il cross-compiler arm-9tdmi-linux-gnu è stato necessario scaricare le binutils e le modutils. Si possono trovare facilmente i sorgenti per modutils-2.4.26, tuttavia, esistono strade più semplici e meno complicate per abilitare il cross-compiler a generare moduli inseribili nel kernel Linux. Uno dei modi è trovare un rpm che contenga le binutils compilate per la versione del nostro compilatore, in particolare è stato possibile recuperare le binutils contenute nel file modutils-2.4.26-1.i386.rpm. Aprendolo con un gestore di archivi qualsiasi si può dotare il proprio compilatore di tutte le funzionalità necessarie alla compilazione di moduli. Le modutils-2.4.26-1.i386.rpm devono essere decomprese nella cartella del cross-compiler che si desidera usare, in questo caso si dovranno decomprimere nella directory /tools/crosstool/arm-9tdmi-linux-gnu/gcc-3.3.4-glibc-2.3.2/sbin/. Si avrà, in questo modo, completato il cross-compiler con le seguenti cartelle : depmod, insmod, kallsyms, ksyms, modinfo, rmmmod.

3.3 MESSA A PUNTO DEL SISTEMA REAL-TIME

L'unica build compatibile con RTAI che è stata trovata, dopo numerosi tentativi è la ts11.

All'indirizzo <http://www.embeddedarm.com/linux/ARM.htm> è stato quindi scaricato il kernel ts11 (<ftp://ftp.embeddedarm.com/ts-arm-linux-cd/sources/tskernel-2.4.26-ts11-src.tar.gz>). Successivamente è stato decompresso in /tools, in modo da avere il percorso: /tools/linux24/. E' stata applicata la patch ADEOS scaricata all'indirizzo <ftp://ftp.embeddedarm.com/ts-arm-linux-cd/sources/rt-tskernel-2.4.26-tsX-adeos.patch>. Il file patch è stato posizionato in /tools/.

Prima di applicare la modifica Adeos ai sorgenti del kernel Linux 2.4.26-ts11 fornito dalla Technologic System è stato modificato il file rt-tskernel-2.4.26-tsX-adeos.patch, in particolare sono state sostituite le seguenti righe:

```
-EXTRAVERSION =-vrs1-cirrus-1-2-1-ts8
+EXTRAVERSION =-vrs1-cirrus-1-2-1-ts8-rt
```

Con le due righe corrispondenti alla versione dei sorgenti scaricati:

```
-EXTRAVERSION =-ts11
+EXTRAVERSION =-ts11-rt
```

Sono stati inoltre cambiati i percorsi del crosscompiler:

```
+CROSS_COMPILE = /usr/local/opt/crosstool/arm-linux/gcc-3.3.4-glibc-2.3.2/bin/arm-linux-
```

è diventato

```
+CROSS_COMPILE = /tools/crosstool/arm-9tdmi-linux-gnu/gcc-3.3.4-glibc-2.3.2/bin/arm-9tdmi-linux-gnu-
```

Per quanto riguarda depmod invece:

```
+DEPMOD = /usr/local/opt/crosstool/arm-linux/gcc-3.3.4-glibc-2.3.2/sbin/depmod
```

è diventato

```
+DEPMOD          = /tools/crosstool/arm-9tdmi-linux-gnu/gcc-3.3.4-glibc-
2.3.2/sbin/depmod
```

Dopo aver fatto questa sostituzione il file è stato rinominato in `rt-tskernel-2.4.26-ts11-adeos.patch`

Il comando usato per applicare la patch è:

```
$ cd /tools/linux24
$patch -p1 -b < ../../rt-tskernel-2.4.26-ts11-adeos.patch
```

3.4 COMPILAZIONE DEL KERNEL

Prima di eseguire i comandi per la compilazione è necessario apportare alcune modifiche al file Makefile contenuto nella directory `/tools/linux24/`. Queste modifiche sono molte e, di seguito, si dà una immagine di come dovrebbero apparire le due parti più importanti del file (se sono stati seguiti in maniera rigorosa i punti precedenti). Dopo aver applicato la patch con successo è opportuno controllare il makefile del kernel, devono esserci queste dichiarazioni:

```
ARCH := arm
CROSS_COMPILE = /tools/crosstool/arm-9tdmi-linux-gnu/gcc-3.3.4-glibc-
2.3.2/bin/arm-9tdmi-linux-gnu-
DEPMOD = /tools/crosstool/arm-9tdmi-linux-gnu/gcc-3.3.4-glibc-2.3.2/sbin/depmod
```

Per configurare le impostazioni base eseguire il comando:

```
make ts7250_2k_config
make oldconfig (attivando solamente ADEOS e ADEOS DOMAINS ARE THREAD)
make menuconfig
```

Ecco una lista di tutte le configurazioni che sono state effettuate ulteriormente:

```
Loadable module support --->
[*] Kernel module loader
ESC
System Type --->
EP9301 Options --->
[*] EP9301 Internal DMA Support
ESC
ESC
Plug and Play configuration --->
<*> Plug and Play support
<*> ISA Plug and Play support
ESC
Sound --->
<*> Sound support
<M> OSS sound modules (NEW)
[*] Verbose initialisation (NEW)
ESC
USB support --->
[*] USB verbose debug messages
```

```
<M>  USB Audio support
ESC
SAVE
```

Dopo aver completato la configurazione con successo è stato dato il comando che effettua la vera e propria compilazione del kernel e dei moduli, generando i files zImage e i moduli.

```
make
make zImage
make modules
make modules_install INSTALL_MOD_PATH=/tools/linux24
```

E' normale che compaiano errori del tipo:

```
depmod: ELF file /tools/linux24/lib/modules/2.4.26-ts11-
rt/kernel/net/packet/af_packet.o not for this architecture
```

Infatti i moduli non possono essere installati sulla macchina i386, ma sono già stati copiati nella cartella target utile.

I files compilati ora sono sotto le directory:

```
/tools/linux24/arch/arm/boot/zImage          (circa 700kb)
/tools/linux24/lib/modules/2.4.26-ts11-rt    (circa 600kb in tar.gz)
```

Il file zImage va copiato in un proprio http server, in modo da poterlo caricare runtime dall'arm. La cartella /tools/linux24/lib/modules/2.4.26-ts11-rt viene compressa (mantenendo i link sym) e decompressa nel file system dell'arm sotto /lib/modules/2.4.26-ts11-rt.

Grazie a queste operazioni si riesce a caricare nell'ARM il kernel appena compilato con tutte le impostazioni scelte da noi, ed abbiamo a disposizione gli ulteriori moduli compilati ma non inclusi nel kernel.

3.5 COMPILAZIONE DEI RTAI-MAGMA

RTAI è l'acronimo di Real Time Application Interface e, come si può intuire mette a disposizione una serie di API per la realizzazione di software Real Time. Il primo passo per la compilazione di RTAI è quello di scaricare dal sito <ftp://oz.embeddedarm.com> i sorgenti che si trovano sotto il nome di `rtai-3.2-magma-src.tar.gz` e decomprimerli nella cartella /tools/.

Una volta decompresso l'archivio si avranno i sorgenti nella cartella /tools/rtai-3.2, e a questo punto si può configurare RTAI con i comandi:

```
cd /tools/rtai-3.2
make ARCH=arm
```

Se compare un'errore del tipo " `mconf.c (...) static declaration of 'current_menu' follows non-static declaration`" è necessario modificare il file `realtivo` . In particolare, è necessario sostituire la dichiarazione `static struct menu *current_menu;` con `struct menu *current_menu;` . Una volta selezionate le componenti di RTAI da compilare si può passare alla modifica del file `makefile`, si deve infatti aggiungere la dichiarazione

```
CROSS_COMPILE=/tools/crosstool/arm-9tdmi-linux-gnu/gcc-3.3.4-glibc-
2.3.2/bin/arm-9tdmi-linux-gnu-
```

appena dopo `CC=gcc`. Una volta fatte tutte queste modifiche si può passare al processo di compilazione con i comandi

```
make
make install
```

A questo punto si disporrà della cartella `/usr/realtime` che dovrà essere copiata nel dispositivo `ts7250`.

3.6 COMPILAZIONE DEI MODULI AUDIO (OSS)

I driver per il suono, per l'usb e per audio usb risultano compilati durante la compilazione del kernel. Pertanto è sufficiente far partire con `insmod` i vari moduli richiesti.

Inserendo il microfono usb nell'arm viene creato il dispositivo:

```
/dev/sound/dsp1
```

Con il comando `cat /dev/sound/dsp1` si può avere una brutale ma efficace idea se il microfono usb funziona o meno. (si vedono stampati su schermo i campioni letti dal microfono in caratteri ASCII). Per fare un test più accurato della registrazione dei suoni con OSS è necessario usare `sox`, non disponibile per arm direttamente (vedere in seguito la fase di compilazione).

3.7 COMPILAZIONE DEGLI ALSA DRIVER

E' stato scaricato del sito della cirrus il pacchetto contenente i driver audio alsa:

```
alsa-driver-1.0.4.tar
```

E' stato scompattato nella cartella `/tools/alsa`, in modo da avere il percorso `/tools/alsa/alsa-driver-1.0.4/`. A causa di alcune parti del `makefile` di alsa driver mal configurate, è stato necessario creare il link simbolico `/tools/crosstool/arm-9tdmi-linux-gnu/gcc-3.3.4-glibc-2.3.2/bin/ld` al file `/tools/crosstool/arm-9tdmi-linux-gnu/gcc-3.3.4-glibc-2.3.2/bin/arm-9tdmi-linux-gnu-ld`

Il comando utilizzato per compilare gli alsa driver è:

```
$ ./configure --with-cards=all --with-debug=detect --with-kernel=/tools/linux24
--with-cross=/tools/crosstool/arm-9tdmi-linux-gnu/gcc-3.3.4-glibc-2.3.2/bin/ --
host=arm --target=/tools/alsa/alsa-driver-1.0.4 -with-
moddir=/tools/linux24/lib/modules
```

Con davanti la variabile `CC="..."` impostata al percorso del cross compilatore gcc che stiamo utilizzando. Durante la compilazione nel mio ambiente si sono verificati numerosi errori nel codice C: variabili dichiarate più volte, funzioni non esistenti, ecc. Ho provveduto a commentare le doppie dichiarazioni, facendo attenzione a non eliminare parti di codice, inoltre sono stati effettuati diversi cambiamenti, rinominando anche una funzione.

Dopo essere riusciti a compilare con successo nella cartella `/tools/alsa/alsa-driver-1.0.4/modules` finalmente troviamo finalmente tutti i link simbolici ai moduli compilati. E' stata copiata l'intera struttura `/tools/alsa/alsa-driver-1.0.4/` nel file system dell'arm e sono stati fatti partire i moduli tramite `insmod nomefile.o`, dopo esserci posizionati nella cartella `modules`. E' necessario seguire un ordine ben preciso nel far partire i moduli di alsa, ordine suggerito dai messaggi di errore di dipendenze non trovate man mano che si fanno partire i moduli.

3.8 PROBLEMI RISCONTRATI

Quando si lancia il comando `./configure` (con tutti i suoi parametri) si nota che non è stato trovato il supporto USB nel kernel (configurato invece correttamente), nè `usb-core` nè `usb-modules`. Questo comporta che non viene creato il file `snd-usb-audio.o` nella cartella `modules`. Ho provveduto a forzare la creazione di questo file, modificando lo script `configure` e sono riuscito a compilarlo con successo. Facendo partire tutti i moduli alsa sull'arm si riescono a eseguire con successo. Con il comando `lsmod` si riescono a vedere tutti i moduli in esecuzione, fra i quali una decina di moduli `snd_*`, compreso `snd_usb_audio`. Sebbene tutto questo al momento della connessione del cavo del microfono usb nell'arm si leggono a console diversi tentativi di configurazione del dispositivo, ma con diversi errori provenienti da `sounddriver.c`. Inoltre leggendo nel file di log sembrerebbe che alsa abbia rilevato il dispositivo con il solo rate di 48000, fatto strano e inaspettato, ma non affidabile.

Capitolo 4

REGISTRAZIONE DEI SUONI SU ARM

4.1 COMPILAZIONE DI SOX

Nel mondo Linux esiste una vasta varietà di programmi in grado di registrare (e manipolare) suoni in formato wave; alcuni tra i più usati sono:

- Audacity: editor avanzato di suoni per Linux, implementa moltissime funzionalità, tra cui la registrazione in numerosi formati
- GNUsound: editor di suoni con supporto per tracce multiple
- SoX: convertitore e registratore di suoni in praticamente tutti i formati esistenti (di Chris Bagwell)

Per questo progetto è stato utilizzato SoX, il più leggero dei tre ma con moltissime funzioni. Il suo pregio maggiore è che gira da console (a differenza di Audacity), inoltre tutti i comandi necessari vengono impartiti all'avvio tramite i parametri, mentre la registrazione viene conclusa semplicemente terminando il processo.

I sorgenti di SoX sono disponibili su Sourceforge, all'indirizzo:

```
http://prdownloads.sourceforge.net/sox/sox-14.0.0.tar.gz?download
```

La versione utilizzata nel progetto è la 13.0.0 (ora è disponibile è la 14.0.0), della quale si trovano anche i binary per numerose piattaforme (Redhat RPM package, Debian package, NetBSD, BeOS Binary, Win32 Binary (Win95/98/NT/XP), OS/2, Atari ST binary, DGJPP DOS Binary). Non sono disponibili i binary per TS-Linux (ARM), pertanto ho provveduto a compilare SoX mediante il cross-compiler. I sorgenti sono stati decompressi dentro /tools/sox-13.0.0/. La procedura utilizzata è la solita, è stato impostato il path dei binari del cross-compiler, e sono stati dati i comandi:

```
$ ./configure
$ make
$ make install
```

In particolare per completare la cross-compilazione con successo dentro Makefile ci devono essere:

```
host = arm-unknown-none
host_alias = arm
host_cpu = arm
host_os = none
host_vendor = unknown
```

```
CC = /tools/crosstool/arm-9tdmi-linux-gnu/gcc-3.3.4-glibc-2.3.2/bin/arm-9tdmi-
linux-gnu-gcc
CPP = /tools/crosstool/arm-9tdmi-linux-gnu/gcc-3.3.4-glibc-2.3.2/bin/arm-9tdmi-
linux-gnu-gcc -E
prefix = /tools/sox-13.0.0
```

Una volta completata la procedura di compilazione con successo si dispone nella directory /tools/sox-13.0.0/bin/ dei file binary compilati per arm: play, rec, sox.

Per controllare effettivamente se sox è stato compilato per l'arm è sufficiente eseguire il comando:

```
$ file /tools/sox-13.0.0/bin/sox
```

Per verificare il funzionamento sulla macchina Arm è sufficiente copiarlo in questa (via ftp) e provare il comando:

```
$ sox --help
```

dalla console emulata dell'arm.

4.2 REGISTRAZIONE TRAMITE SOX E OSS

SoX è un programma eseguibile da console che serve a convertire files nella maggior parte dei formati audio esistenti, eventualmente applicando un ricampionamento o applicando degli effetti al suono. E' in grado di trattare ogni tipo di formato, sia files autodescrittivi, cioè che contengono un header con tutte le informazioni sulla codifica (come il wave) sia files grezzi con all'interno campioni puri, senza meta-dati. Nel secondo caso è necessario fornire nella riga di comando tutti i parametri sulla codifica utilizzata.

Un'analisi accurata della distorsione e del rumore introdotti da SoX è disponibile nell'articolo <http://leute.server.de/wilde/resample.html>

Per descrivere totalmente i dati audio campionati si usano le seguenti caratteristiche:

- rate: è la frequenza di campionamento, espresso come campioni/secondo. Per esempio, il rate di un CD audio è di 44100.
- data size: è la precisione con cui sono salvati i dati (campioni), solitamente sono interi a 8 o 16 bit (con segno o senza)
- data encoding: che tipo di codifica è stata usata per i campioni. Esempi: u-law, ADPCM, dati lineari con segno, ecc.
- channels: quanti canali sono contenuti nel file audio, solitamente 1 o 2 (Mono o Stereo)

In Linux la linea di audio input (il microfono) scrive il proprio flusso in uno pseudo file gestito dai driver audio (OSS o ALSA solitamente), ad esempio in /dev/sound/dsp1 per quanto riguarda OSS. Ogni programma che vuole usufruire del flusso audio in input deve leggere questo file, codificato in un formato specifico dei driver sonori in uso, in generale si può considerare un formato grezzo e privo di informazioni sulla codifica). E' anche possibile stampare in diretta a video l'input del microfono tramite il comando cat.

SoX è in grado di maneggiare questo flusso audio come se fosse un file audio salvato su disco, a patto che si impostino tramite i parametri della riga di comando il formato e la codifica giusti. In questo modo SoX si occuperà di fare una trasformazione dal formato grezzo dei driver audio a un formato standard come il wave; in questo modo SoX è usato come un vero e proprio registratore di suoni. Poiché il flusso audio del microfono non termina mai, a differenza dei normali files salvati su disco, SoX continuerà a trasformare il flusso finché non verrà terminato tramite un segnale di Kill. Nell'header del file wave ci dovrebbe essere anche il numero totale dei campioni raccolti (quindi indirettamente la durata), ma nel caso della registrazione SoX non è in grado di sapere quando verrà terminato, e poiché l'header sta all'inizio del file il numero totale dei campioni viene preventivamente impostato a zero. Fortunatamente, anche se i file wave risultanti non sono propriamente standard, essi vengono utilizzati correttamente da tutte le applicazioni utili per questo progetto (SoX stesso, il trascrittore di Dragon 5, e anche Windows Media Player se necessario, usato per fare dei test; Audacity, più rigoroso, sembra avere dei problemi con questi file). Sarebbe comunque sufficiente alla fine della registrazione controllare la lunghezza del file escluso l'header e modificare il numero totale dei campioni appositamente.

Ecco trascritta la prima parte di "man sox", contenente la sintassi classica delle chiamate a questo programma:

NAME

```
sox - Sound eXchange : universal sound sample translator
```

SYNOPSIS

```
sox infile outfile
sox [ general options ] [ format options ] infile
    [ format options ] outfile
    [ effect [ effect options ] ... ]
soxmix infile1 infile2 outfile
soxmix [ general options ] [ format options ] infile1
    [ format options ] infile2
    [ format options ] outfile
    [ effect [ effect options ] ... ]
General options:
    [ -h ] [ -p ] [ -v volume ] [ -V ]
Format options:
    [ -t filetype ] [ -r rate ] [ -s/-u/-U/-A/-a/-i/-g/-f ]
    [ -b/-w/-l ]
    [ -c channels ] [ -x ] [ -e ]
```

In pratica la sintassi fondamentale è <sox infile outfile> dove sia infile che outfile vengono specificati con tutti i parametri che individuano il formato, il file, la codifica, ecc. Per registrare dalla linea in si utilizza come infile lo pseudo file prodotto dai driver della scheda audio (OSS), solitamente posizionato in /dev/sound/dsp1 oppure direttamente /dev/dsp1. E' necessario impostare il formato corretto del file in entrata, per esempio ossdsp nel caso di Open Sound System. Un esempio di comando per registrare dalla linea in ingresso (lo pseudo file ossdsp) a un file wave è:

```
$ /bin/sox -q -t ossdsp /dev/sound/dsp1 -t wav test.wav
```

In cui si sono tralasciati i parametri per impostare il rate, la dimensione dei campioni, ecc. poiché si accettano quelli di default. Un esempio di ricampionamento di un file wave è:

```
$ /bin/sox -t wav data.wav -t wav -r 11025 -w dataresampled.wav
```

Non è necessario in caso di file in ingresso in formato wave impostare il rate e la dimensione dei campioni poiché i wave contengono queste informazioni nell'header.

4.3 RICAMPIONAMENTO DEL FILE WAVE

Sulla scheda Arm tramite i driver OSS è risultato possibile effettuare registrazioni solamente alla frequenza di campionamento di 44100, 16 bit unsigned. Qualsiasi tentativo di impostare la linea di input di SoX ad altre frequenze risulta vano, poiché viene restituito un messaggio di warning che avvisa che il rate è stato impostato a 11025.

La versione 5 di Dragon Naturally Speaking è in grado di trascrivere solamente wave con rate a 11025, pertanto è risultato necessario effettuare un ricampionamento della registrazione. SoX è in grado di effettuare il ricampionamento in diretta, applicandolo al flusso di entrata e salvando il wave in uscita direttamente ricampionato – è infatti una delle tante trasformazioni che è in grado di effettuare al segnale in diretta. Purtroppo effettuare il ricampionamento sull'Arm è risultato particolarmente difficile, per via delle prestazioni veramente basse. Per esempio, l'Arm ha impiegato più di un minuto per ricampionare una registrazione di 3 secondi, a differenza del mio notebook (Intel Pentium M 2.26 Ghz) che ha completato l'operazione in pochi centesimi di secondo.

Per questo motivo il ricampionamento è stato assegnato come operazione da eseguire sul PC Windows, appena prima di passare il file wave al Dragon 5. Questo modo di agire è anche logicamente più corretto, dato che è il Dragon ad imporre la limitazione di 11025 Hz di campionamento.

Capitolo 5

COMUNICAZIONE TRA PROCESSI

5.1 L'OPEN AGENT ARCHITECTURE

La “Open Agent Architecture” dell'SRI International di Menlo Park è un framework che integra una comunità di agenti software in un ambiente distribuito. Essa facilita le iterazioni tra le componenti distribuite delegando tasks, data requests e triggers. Un computer non risolve un problema, ma si limita a inviare a una comunità di agenti software un certo numero di task. L'architettura dell'SRI Intl ha definito la personalità dei “facilitator” (gli intermediari di cui sopra), i quali diventano al tempo stesso indici degli agenti presenti nella comunità e “assistenti” capaci di smistare i task agli agenti più “competenti”.

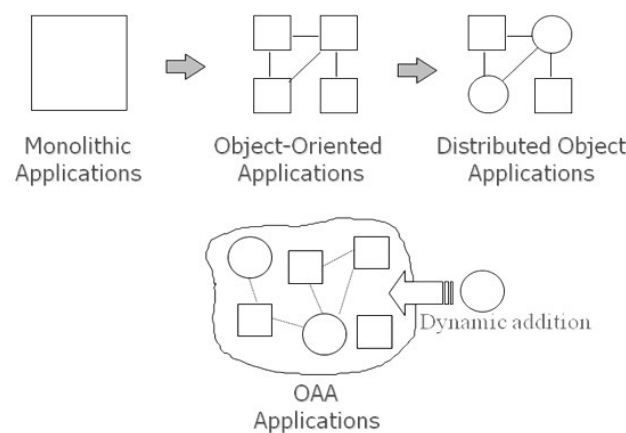


Figura 5: I vari tipi di applicazioni distribuite

In sostanza un ambiente OAA è generalmente configurato in questo modo:

- c'è un agente Facilitator che fa la parte del server e mantiene un registro di tutti gli agenti disponibili nella rete
- ci sono molti altri agenti distribuiti in altri client della rete configurati per interfacciarsi tutti allo stesso agente Facilitator
- gli agenti distribuiti possono essere in grado di fornire soluzioni a domande (in questo caso dichiarano la loro disponibilità al Facilitator) oppure solamente usufruire dei servizi della rete

- quando un agente fa una domanda al Facilitator questo cerca nel suo registro se esiste un agente in grado di rispondere, se così, delega il lavoro a questo agente; una volta trovata la soluzione viene ripetuto il percorso a ritroso e il primo agente riceve la soluzione
- gli agenti possono essere di diversi tipi: interfacce utente, applicazioni, riconoscitori di linguaggio naturale, ecc.
- tutte le comunicazioni avvengono tramite un meta linguaggio capibile da tutti gli agenti: Interagent Communication Language (ICL)
- a livello di rete le comunicazioni avvengono tramite socket su di una porta specificata

Il linguaggio di comunicazione tra agenti (ICL) serve a registrare presso il Facilitator le capacità risolutive di un agente, richiedere un servizio alla comunità cioè fare una domanda, ricevere risposte, compiere azioni su altri agenti, impostare dei trigger, trasmettere dati.

Non è necessario generare del codice di comunicazione tra agenti manualmente, poiché sono fornite le API per gran parte dei linguaggi utilizzati: C, C++, Visual Basic, Java, Delphi, Prolog, Lisp.

Caratteristiche dell'Open Agent Architecture:

- gli agenti possono essere programmati in diversi linguaggi e interfacciarsi con sistemi già esistenti
- c'è la possibilità di aggiungere o sostituire gli agenti dinamicamente
- gli agenti sono distribuiti in diversi computer, anche con sistemi operativi differenti
- esecuzione parallela di subtasks e processi
- nasconde le dipendenze hardware e software
- supporta il riconoscimento della scrittura manuale, dei gesti e della voce
- si possono combinare le soluzioni di diversi agenti
- supporta il riconoscimento dei linguaggi naturali
- sintassi di comunicazione tra agenti standard (ICL)
- è gratuita e open source (rilasciata sotto licenza GNU Lesser General Public License LGPL)

Ecco elencati alcuni tipici utilizzi dell'architettura ad agenti: uffici automatizzati, messaggistica unificata, mappe multimodali, motori di ricerca, collaborazione istantanea, controllo di robot, traduzione simultanea, strumenti video, ecc.

5.2 IMPLEMENTAZIONE DEL RICONOSCITORE VOCALE DISTRIBUITO SU OAA

Si è scelto di utilizzare l'architettura ad agenti per la realizzazione del software per il riconoscimento vocale distribuito.

L'implementazione è stata progettata in questo modo:

- il Facilitator gira sulla macchina Windows e sta in ascolto in TCP su una determinata porta
- nel pc Windows viene eseguito un agente in grado di risolvere il problema del riconoscimento vocale: questo agente "Transcriber" si registra presso il Facilitator

dichiarando la disponibilità di risolvere problemi del tipo Transcribe(IN <sequenza byte in formato wave>, OUT <testo riconosciuto e trascritto>)

- sulla macchina ARM viene eseguito un agente che comunica con il Facilitator tramite ethernet e si occupa delle registrazioni con il microfono: quando viene completata una registrazione invia al Facilitator la domanda Transcribe(<file registrato wave>)
- il Facilitator riconosce la domanda e la ridireziona verso l'agente Transcriber
- alla fine l'agente che registra, presso l'ARM, riceve una risposta testuale oppure un errore in caso di problemi

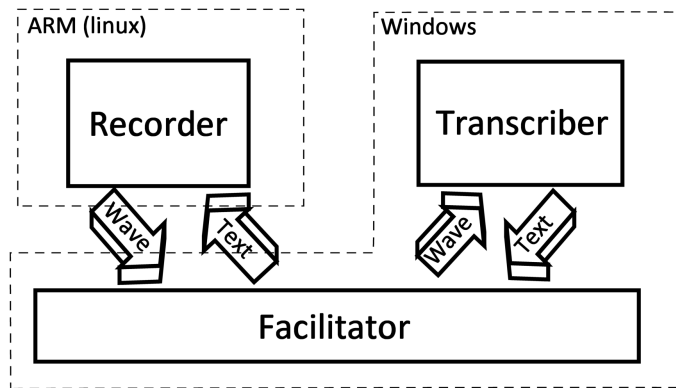


Figura 6: Schema del riconoscimento distribuito

Per quanto riguarda lo transcriber è stato realizzato con questa architettura:

1. il programma viene eseguito
2. vengono inizializzati gli oggetti utilizzati per la trascrizione (API di Dragon 5)
3. vengono inizializzati gli oggetti dell'Open Agent Architecture
4. viene inizializzato l'agente, dichiarandosi presso il Facilitator
5. parte un thread infinito che sta in ascolto per le richieste da parte di altri agenti
6. ogni volta che viene ricevuta una richiesta si chiama una funzione che si occupa della procedura di trascrizione (presso un thread differente)
7. quando la trascrizione è completata viene inviato il testo trascritto come risposta

E' stato effettivamente realizzato l'agente "Transcriber" (funzionante), testato solamente in ambito debug.

5.3 PROBLEMI RISCONTRATI CON OAA

Per quanto riguarda la realizzazione dell'agente "Recorder", che deve girare sulla macchina Linux è stato necessario compilare l'Open Agent Architecture per questo tipo di sistema.

E' stata scaricata la versione completa di tutti i sorgenti in C/C++ della libreria OAA (dal sito <http://www.ai.sri.com/~oaa/distribution/v2.3/2.3.2/>).

Nella libreria è presente un file README contenente questo avviso per quanto riguarda la compilazione:

Although source code is included for the facilitator, libraries and sample agents, the build procedure is not particularly well documented, and not as well-organized as it might be (especially under Windows). Therefore, compiling the source code may require some expertise.

I sorgenti completi della libreria si possono trovare al percorso oaa2.3.1\src\oaalib\c, ove è presente anche il Makefile, teoricamente già pronto per l'uso.

Purtroppo dopo diversi tentativi non è stato possibile compilare oaalib per l'ARM: sono stati fatti diversi tentativi di compilazione con cross-compiler differenti in ambienti diversi (Cygwin e Kubuntu), ma in entrambi i casi sono stati riscontrati numerosi errori di compilazione.

Nel caso di compilazione da cygwin è stato dato il comando:

```
/tools/oa2.3.1/src/oaalib/ $ make
```

Che ha completato con successo tutte le verifiche imposte dal Makefile, ma si è bloccato durante la compilazione con l'errore:

```
make[5]: Leaving directory
~/oa2.3.1/src/oaalib/c/external/pkgconfig/pkgconfig-0.14.0'
/bin/sh ./mkinstalldirs /oa2.3.1/src/oaalib/c/stow/pkgconfig-
0.14.0/share/aclocal
/usr/bin/install -c -m 644 ./pkg.m4 /oa2.3.1/src/oaalib/c/stow/pkgconfig-
0.14.0/share/aclocal/pkg.m4
make[4]: Leaving directory
~/oa2.3.1/src/oaalib/c/external/pkgconfig/pkgconfig-0.14.0'
make[3]: Leaving directory
~/oa2.3.1/src/oaalib/c/external/pkgconfig/pkgconfig-0.14.0'
make[2]: Leaving directory
~/oa2.3.1/src/oaalib/c/external/pkgconfig/pkgconfig-0.14.0'
/oa2.3.1/src/oaalib/c/stow/pkgconfig-0.14.0
/oa2.3.1/src/oaalib/c/stow/pkgconfig
/bin/bash: /oa2.3.1/src/oaalib/c/stow/pkgconfig-0.14.0: is a directory
make[1]: *** [pkgconfig-support] Error 126
make[1]: Leaving directory ~/oa2.3.1/src/oaalib/c'
make: *** [subdir_c] Error 2
```

Provando a compilare con il cross-compiler su Kubuntu si ricevono altri errori di compilazione su files differenti. Infine ho optato per evitare di utilizzare la libreria OAA, perdendo gran parte delle sue funzionalità. Fortunatamente l'architettura del sistema per il riconoscimento non è particolarmente complessa e comprende solamente due agenti più il facilitator, in sole due macchine, quindi ho deciso di realizzare il sistema distribuito basando le comunicazioni su socket TCP a livello di sistema operativo, senza altre librerie per gestire i messaggi. Questo approccio ha aumentato la complessità dei due agenti, poiché ora devono occuparsi anche della comunicazione tramite socket (azione del tutto trasparente nel caso di OAA), ma ha praticamente azzerato le incompatibilità tra librerie di Windows e Linux, poiché i socket vengono usati a basso livello.

5.4 COMUNICAZIONE TRA PROCESSI TRAMITE SOCKET

Tra le varie forme di IPC (Inter Process Communication - comunicazione tra processi), i socket sono di gran lunga la più popolare. Data una qualsiasi piattaforma, è probabile che ci siano altre forme di IPC più veloci, ma per la comunicazione tra piattaforme diverse i socket sono quasi una scelta obbligata. I socket (letteralmente presa) sono un'API originariamente sviluppata nel 1982 nel BSD UNIX 4.1c, chiamata anche Socket di Berkley, che permette a dei processi di trasmettere e ricevere dei dati bidirezionalmente su una rete, e questo senza bisogno di conoscere l'hardware, quindi indipendentemente dal tipo d'interfaccia di rete usato, sia esso Ethernet, Token Ring, seriale, o altro.

Il Socket è un punto di collegamento tra un client che richiede una risorsa ed un server che la fornisce, ed è costituito dai seguenti parametri: porta del client, porta del server, indirizzo IP del client, indirizzo IP del server e protocollo utilizzato. Il protocollo usato può essere TCP (un protocollo di tipo stream), UDP (un protocollo di tipo datagram, ICMP e IP) e la differenza essenziale tra TCP e UDP è che nel caso dell'UDP non è necessario stabilire una connessione client/server, e in pratica non c'è garanzia che i pacchetti arrivino a destinazione e nell'esatto ordine, quindi UDP non è affidabile. Dato che c'è la necessità di inviare file wave si è optato quindi per il protocollo TCP.

In Linux un Socket è un descrittore (tipo di file) che va inizializzato tramite la funzione `socket()`, la cui sintassi è `int socket(int domain, int type, int protocol);` che restituisce un valore uguale o maggiore di zero in caso di successo, o minore di zero in caso di errore, e si devono usare i file header `#include < sys/types.h >` e `#include < sys/socket.h >`.

Domain definisce un dominio di comunicazione, ovvero la famiglia del protocollo che si userà per la comunicazione; type indica il tipo di Socket da costruire; protocol indica il protocollo utilizzato dal Socket.

Per creare una semplice connessione lato client si deve:

1. creare un Socket handle;
2. riempire la struttura `sockaddr` dei valori necessari (definita nel `filessocket.h`);
3. connettersi tramite la funzione `connect()`;
4. inviare dei byte tramite `send()`;
5. rileggere i dati inviati tramite `recv()`;
6. chiudere la connessione tramite `close()`;

Invece il compito di un server è quello di rimanere in ascolto su una porta e gestire connessioni multiple da parte dei client. I passi iniziali da fare sono:

1. creare un Socket handle;
2. riempire la struttura `sockaddr` dei valori necessari;
3. assegnare un indirizzo e una porta locale al server tramite la funzione `bind()`;
4. mettersi in ascolto di connessioni client tramite la funzione `listen()`;
5. creare dei socket per ogni connessione client tramite `accept()`;

Le due principali cause per cui non si riesce a mettere un server in ascolto con una porta, cioè non si riesce ad effettuare un bind, sono che la porta sia già occupata o che un firewall ne blocchi l'accesso.

5.5 ESEMPIO DI CLIENT TCP PER LINUX

Di seguito vengono presentate le porzioni di un programma di esempio di un client che comunica tramite socket tcp. Nell'inizializzazione viene inclusa la libreria PracticalSocket.h, che non è altro che un wrapper per le librerie standard dei socket in Unix:

- <sys/socket.h> per socket(), connect(), send(), and recv()
- <netdb.h> per gethostbyname()
- <arpa/inet.h> per inet_addr()
- <unistd.h> per close()
- <netinet/in.h> per sockaddr_in

```
// Initialize
#include "PracticalSocket.h" // For Socket and SocketException
#include <iostream>          // For cerr and cout
const int RCVBUFSIZE = 32; // Size of receive buffer
```

Successivamente, viene stabilita la connessione con il server servAddress presso la porta echoServPort:

```
// Establish connection with the echo server
TCPSocket sock(servAddress, echoServPort);
```

In un unico comando viene inviata la stringa al server:

```
// Send the string to the echo server
sock.send(echoString, echoStringLen);
```

Dopo di che si predispose un ciclo che legge la risposta sullo stesso socket, e continua a leggerla finchè riceve byte:

```
char echoBuffer[RCVBUFSIZE + 1]; // Buffer for echo string + \0
int bytesReceived = 0;           // Bytes read on each recv()
int totalBytesReceived = 0;     // Total bytes read

// Receive the same string back from the server
while (totalBytesReceived < echoStringLen) {
    // Receive up to the buffer size bytes from the sender
    if ((bytesReceived = (sock.recv(echoBuffer, RCVBUFSIZE))) <= 0) {
        cerr << "Unable to read";
        exit(1);
    }

    totalBytesReceived += bytesReceived; // Keep tally of total bytes
    echoBuffer[bytesReceived] = '\0';   // Terminate the string!
    cout << echoBuffer;                 // Print the echo buffer
}
```

5.6 ESEMPIO DI SERVER TCP PER LINUX

Per quanto riguarda il server l'inizializzazione avviene allo stesso modo del client:

```
// Initialize
#include "PracticalSocket.h" // For Socket and SocketException
#include <iostream>          // For cerr and cout
const int RCVBUFSIZE = 32; // Size of receive buffer
```

Nel main c'è fondamentalmente l'inizializzazione del socket e un ciclo infinito che attende connessioni:

```
TCPServerSocket servSock(echoServPort); // Server Socket object
for (;;) {                               // Run forever
    HandleTCPClient(servSock.accept()); // Wait for a client to connect
}
```

Ecco la funzione che si occupa delle connessioni TCP:

```
// TCP client handling function
void HandleTCPClient(TCPsocket *sock) {
    cout << "Handling client ";
    cout << sock->getForeignAddress() << ":";
    cout << sock->getForeignPort();
    // Send received string and receive again until the end of transmission
    char echoBuffer[RCVBUFSIZE];
    int recvMsgSize;
    while ((recvMsgSize = sock->recv(echoBuffer, RCVBUFSIZE)) > 0) {
        // Zero means end of transmission
        // Echo message back to client
        sock->send(echoBuffer, recvMsgSize);
    }
    delete sock;
}
```

In sostanza viene stampato a video l'IP del client e la porta utilizzata dal client, poi viene ricevuto il messaggio dal client e allo stesso tempo viene rimandato indietro come echo.

5.7 COMPATIBILITÀ TRA SOCKET IN WINDOWS E IN LINUX

Fondamentalmente le comunicazioni tramite socket sono compatibili in Windows e Linux, in particolare i livelli di astrazione delle librerie in C non permettono il verificarsi di problemi quali *big endian/little endian*.

La differenza fondamentale tra la programmazione dei socket in Linux e in Windows sta nelle librerie. Per il primo si utilizzano le librerie nate con Unix: <sys/socket.h>, <netdb.h>, <arpa/inet.h>, mentre per i socket in Windows si utilizza "wsock32.lib".

L'inizializzazione del socket in Windows è differente (e un po' più macchinosa), eccone un esempio:

```

WORD wVersionRequested;
WSADATA wsaData;
wVersionRequested = MAKEWORD(2, 0); // Request WinSock v2.0
if (WSAStartup(wVersionRequested, &wsaData) != 0) {
    // Load WinSock DLL
    throw SocketException("Unable to load WinSock DLL");
}

```

Inoltre la sintassi della libreria winsock è differente rispetto a socket.h, ad esempio i comandi: WSACleanup, closesocket, differiscono rispetto ai corrispondenti di Linux. Fortunatamente la libreria “PracticalSocket” trovata in rete sopperisce quasi completamente a queste differenze, poichè è un wrapper che include tutte le funzioni per lavorare con i socket, definendo codice differente a seconda se viene compilato da Windows oppure Linux (tramite le direttive #ifdef WIN32, #else, #endif).

5.8 IMPLEMENTAZIONE DEL SERVER IN WINDOWS

Il server per Windows consiste in un programma che apre un socket e rimane in ascolto per eventuali connessioni da parte del client ARM. Quando viene ricevuta una connessione in ingresso viene salvato lo stream di byte ricevuto nel file data.wav. La ricezione avviene tramite un buffer di 100 byte. Man mano che viene riempito il buffer questo viene scritto direttamente nel file wave e svuotato. Quando viene ricevuta la sequenza “ENDOFFILEWAV” viene terminata la ricezione dei byte. Non è stato potuto utilizzare le classiche funzioni della libreria capaci di individuare la fine dello stream poiché questa veniva confusa con il carattere \0, presente comunque all’interno del file wave in più casi.

Il server Windows contiene inoltre un insieme di funzioni che permettono la creazione di processi figli e la comunicazione con essi tramite pipes. In pratica una volta finito di ricevere il file data.wav viene creato un processo figlio che esegue il programma trscribe.exe, in grado di trascrivere un file wave restituendo il risultato nello standard output. Quando viene creato questo processo gli viene “agganciato” un pipe in modo da ricevere l’output dentro il programma tramite il metodo ReadCharFromPipe. Per semplificare questa gestione del processo è stata creata una funzione wrapper doAction che non fa altro che eseguire un processo figlio, attendere la terminazione e ritornare tutti i caratteri trasmessi come standard output. Appena dopo la ricezione del file wave ma prima di chiamare il trascrittore è stato inserito il comando:

```
system("sox -t wav data.wav -t wav -r 11025 -w dataresampled.wav");
```

che ricampiona il file originale (a 44100 Hz) in wave a 11025 Hz – 16bit, come richiesto dalle API di Dragon 5. Il software SoX utilizzato in questo caso è la versione compilata per Windows, scaricabile direttamente dal sito ufficiale. La risposta testuale fornita dal processo “transcribe” viene restituita tramite socket al client che aveva aperto la connessione.

Dal punto di vista logico il server per Windows è realizzato in questo modo:

```
char *doAction(char * exeFile, char *wavefilename)
```

Prende gli handle degli standard output e input, crea due pipe e le associa a questi handle. Alla fine chiama il metodo `createChildProcess`, e nel caso questo termini con successo ritorna il risultato del metodo `ReadCharFromPipe`.

```
BOOL CreateChildProcess(char * exeFile, char * wavefilename)
```

Viene inizializzato un processo figlio con tutti i parametri necessari e con le pipes create precedentemente come standard input e output. Alla fine dell'esecuzione vengono distrutte le pipe.

```
char *ReadCharFromPipe(VOID)
```

Legge l'output del processo figlio e lo restituisce come una stringa terminata.

```
int main(int argc, char *argv[])
```

Crea il socket sulla porta richiesta e rimane in attesa di un client tramite il comando `HandleTCPClient(servSock.accept());`

```
void HandleTCPClient(TCPsocket *sock)
```

Controlla ed accetta la connessione del client, crea il file binario in scrittura `data.wav`, riceve tutti i byte dal client che vengono direttamente scritti nel file.

Terminata la ricezione esegue il comando esterno per ricampionare il file wave, poi esegue come processo figlio il software che esegue la trascrizione, e riceve l'output di questo. Infine risponde tramite socket al client trasmettendogli il testo trascritto:

```
sock->send(response, strlen(response));
```

Il socket viene infine distrutto.

Capitolo 6

RICONOSCIMENTO DEI COMANDI VOCALI

6.1 RILEVAMENTO DELLE PAROLE

Il riconoscimento delle parole all'interno del flusso audio viene fatto analizzando in tempo reale i campioni registrati da SoX. Innanzitutto scarto i primi 44 byte perché sono l'header del file wave. Finché i campioni si mantengono di poco distanti dal valore 127 che rappresenta lo zero significa che il microfono sta registrando silenzio, e in questo caso scarto i campioni. Appena supero questa soglia inizio a salvare in diretta i campioni sul file wave. Nel frattempo monitoro gli ultimi 15000 campioni (che sono circa 0,35 secondi), se sono tutti dentro la soglia del silenzio considero il comando vocale finito, quindi termino il salvataggio dei campioni e continuo l'esecuzione con il riconoscimento vocale.

6.2 RICONOSCIMENTO DEL COMANDO

Il Dragon 5 si occupa di fare il riconoscimento vocale e la trascrizione della parola registrata in una parola testuale. A questo punto però il sistema non è in grado di interpretare direttamente le parole "avanti", "indietro", "vai a destra", "vai a sinistra" come dei comandi eseguibili, deve essere fatto un riconoscimento. Ho deciso di creare un insieme di comandi che possono essere eseguiti dal robot, codificati tramite un carattere (1 byte), in modo da avere a disposizione 256 comandi possibili per ogni evenienza. A questo punto ho creato un vocabolario di parole con significato che il robot deve riconoscere, suddivise in classi a seconda del comando. Il riconoscimento deve essere fatto secondo una certa priorità, in modo da essere sicuri che in caso di errore il robot esegua il comando meno problematico (ad esempio fermarsi piuttosto che girare).

Carattere	Comando	Parole da riconoscere
'2','w'	avanti (veloce)	avanti, vai avanti, veloce, avanti veloce
'd'	destra	destra, vai a destra, gira a destra
'a'	sinistra	sinistra, vai a sinistra, gira a sinistra
's'	indietro	indietro, vai indietro, gira

'0'	fermo	fermo, stop, stai fermo
'1'	(avanti) piano	piano, vai piano, avanti piano

Per il comando avanti veloce in realtà è possibile usare un qualsiasi carattere non utilizzato per gli altri comandi. Purtroppo il riconoscimento non sempre produce il risultato atteso, ad esempio la parola "destra" potrebbe essere riconosciuta come "resta","testa", oppure la parola "avanti" potrebbe essere tagliata come "Ti" o addirittura "Di". Questo è dovuto a molti fattori, quali il rumore di fondo, un addestramento non ottimale del Dragon 5, un tono di voce diverso da quello usato inizialmente o altri problemi non prevedibili. Per riuscire a riconoscere gran parte dei comandi anche quando il riconoscimento non avviene in modo ottimale è stata usata la libreria standard per le espressioni regolari (regex.h), il riconoscimento dei comandi avviene facendo un match di una piccolissima parte della parola da rilevare, solitamente l'ultima sillaba. Fortunatamente le parole da rilevare (avanti, destra, sinistra, indietro, fermo, piano) hanno l'ultima sillaba abbastanza differente. Pertanto il riconoscimento avviene in questa maniera:

```
int res;
char cmd = 0;
// RILEVO AVANTI
res = 0;
res += match(output,"nti"); // Avanti, Vai avanti, Avanti veloce...
res += match(output,"[Vv]ai"); // Vai
res += match(output,"Di"); // Di
if (res) { cmd = 'w'; }
// RILEVO LA DESTRA
res = 0;
res += match(output,"ta"); // Testa, Resta...
res += match(output,"ra"); // Destra, Sinistra, Entra...
if (res) { cmd = 'd'; }
// RILEVO LA SINISTRA
res = 0;
res += match(output,"ist(r{0,1})a"); // Sinistra
if (res) { cmd = 'a'; }
// RILEVO INDIETRO
res = 0;
res += match(output,"tro"); // Indietro, Retro
res += match(output,"[Ll]o"); // Lo
res += match(output,"[Gg]ir."); // Gira, Girati, Giro
if (res) { cmd = 's'; }
// RILEVO AVANTI VELOCE
res = 0;
```

```
res += match(output,"[Vv]eloce");    // Veloce
if (res) { cmd = 'w'; }
// RILEVO STOP
res = 0;
res += match(output,"[Ss]to"); // Stop, Sto
res += match(output,"mo"); // Fermo
res += match(output,"0"); // 0
if (res) { cmd = '0'; }
// RILEVO PIANO
res = 0;
res += match(output,"no"); // Piano
if (res) { cmd = '1'; }
```

In questo modo vengono rilevati gran parte delle parole riconosciute da Dragon 5. E' da notare che i comandi sono messi in ordine di priorità e un comando rilevato correttamente va a sovrascrivere un precedente rilevamento, ad esempio un comando "vai avanti fermo" provoca il rilevamento del comando "avanti" che viene però sovrascritto dal comando "fermo". La parola "sinistra" passa il rilevamento del comando "destra" poichè finisce per "ra" ma poi viene eseguito anche il match con il pattern "istra" e quindi viene rilevato correttamente il comando. Anche una frase come "avanti piano" passa inizialmente il rilevamento del comando "avanti", che però viene sovrascritto dopo aver rilevato la parola "piano". Data la grande flessibilità nel rilevamento dei comandi bisogna fare attenzione con le parole non previste nel vocabolario perchè potrebbero essere rilevate erroneamente. Ad esempio la frase "vai avanti ancora" passerebbe correttamente il match con il pattern "ra" e quindi verrebbe rilevato come un comando "destra". E' possibile rendere il riconoscimento molto più preciso semplicemente affinando le espressioni regolari, ma si perderebbe un po' in flessibilità.

Capitolo 7

MODULI REAL-TIME PER IL MOVIMENTO

7.1 I MODULI DEL KERNEL

Un modulo linux è una porzione di software che può essere aggiunta al kernel quando questo è già in esercizio, ovvero a run time. Il vantaggio principale della struttura modulare consiste nel fatto che non è necessario ricompilare l'intero kernel per includere un modulo. E' importante sottolineare che Linux rimane un sistema operativo monolitico, sebbene il suo massiccio uso di moduli potrebbe suggerire il contrario.

Un Loadable Kernel Module (LKM) può implementare diverse tipologie di servizi. Tipicamente si distinguono in queste categorie:

1. Driver dei dispositivi: Per comunicare con i dispositivi hardware senza mostrare i dettagli sul loro funzionamento.
2. Driver per i file system: Per implementare l'accesso ad uno specifico file system implementando le procedure specifiche alla memorizzazione dei dati.
3. Driver di rete: Per l'implementazione di protocolli per le reti a diversi livelli.
4. Chiamate a sistema: Per specificare nuove system call o per ridefinire l'implementazione di una syscall esistente nel kernel.
5. Interpreti per eseguibili: Per caricare e lanciare eseguibili in diversi formati. Il kernel Linux può eseguire programmi con formati diversi, purché vi siano moduli che ne conoscano la struttura.

I moduli in Linux non possono:

- Usare funzioni della libreria standard C (per esempio quelle definite in math.h)
- Usare l'aritmetica floating point
- Avere un utente proprietario

I moduli sono generalmente file che terminano con l'estensione .o e si collocano al di sotto della directory /lib/modules/<versione>/, dove la versione si riferisce al kernel per il quale sono stati predisposti. Per esempio, /lib/modules/2.0.33/, si riferisce ai moduli del kernel 2.0.33.

7.2 CREAZIONE DI UN MODULO DEL KERNEL

La struttura base di un modulo deve includere fondamentalmente due metodi: il metodo di inizializzazione e quello di uscita. Il primo verrà invocato prima dell'esecuzione del codice presente all'interno del modulo stesso; il secondo verrà invocato prima della sua terminazione.

```
#include <linux/init.h>
#include <linux/module.h>
MODULE_LICENSE("Dual BSD/GPL");
static int my_init(void) {
    printk(KERN_ALERT "Module Inizialization\n");
    return 0;
}
static void my_exit(void) {
    printk(KERN_ALERT "Module Termination\n");
}
module_init(my_init);
module_exit(my_exit);
```

Si noti come si utilizza la funzione `printk()` al posto della classica `print()`, in quanto quest'ultima fa parte della standard lib di C, non eseguibile dal kernel space.

Per compilare un modulo è necessario utilizzare il comando `make`. Il programma `make` genera i comandi per espletare la sua funzione usando un file di descrizione noto come `makefile`; questi comandi sono poi eseguiti dalla shell. Il `makefile` è essenzialmente un insieme di regole che il `make` deve eseguire per aggiornare un programma o, come in questo caso, un modulo. Queste regole generalmente sono legate alla definizione delle dipendenze tra i file. Il `make` è in grado di accorgersi automaticamente se un file è stato modificato dalla data di modifica, ed eseguire solamente gli aggiornamenti necessari.

Il `makefile` che utilizzeremo è il seguente:

```
TARGET := rt_progetto
NOME    := 2.4.26-ts11-rt
INCLUDE := -I/lib/modules/$(NOME)/build/include -I/usr/realtime/include
CFLAGS  := -O2 -Wall -DMODULE -D__KERNEL__ -DLINUX -ffast-math
CC      := /tools/crosstool/arm-9tdmi-linux-gnu/gcc-3.3.4-glibc-2.3.2/bin/arm-9t
${TARGET}.o: ${TARGET}.c
    $(CC) $(CFLAGS) ${INCLUDE} -c ${TARGET}.c
clean::
    $(RM) .smi* *.cmd *.o *.ko *.mod.c
    $(RM) -R .tmp*
```

La compilazione da come risultato un file kernel object (estensione .o o .ko in base alla versione di Linux) che potrà essere inserito a run time senza ricompilare il kernel o riavviare il sistema operativo. Per gestire i moduli i comandi fondamentali sono tre:

- `lsmod` visualizza la lista dei moduli attualmente caricati.
- `insmod <modulo.o>` carica un nuovo modulo
- `rmmod <modulo.o>` rimuove un modulo caricato

Il programma `insmod` si preoccupa di chiamare le primitive `sys_init_module()` e `sys_create_module()` che rispettivamente inizializzano e creano il modulo. A sua volta saranno queste routine a chiamare la `module_init` che si occuperà di far eseguire il codice definito all'interno del modulo stesso, prima di procedere con l'esecuzione delle internals. Analogamente il programma `rmmod` chiama la primitiva `sys_delete_module()` che, specularmente alla create, rimuove il modulo dal kernel e si preoccupa di far eseguire il codice precedentemente definito nella `exit`.

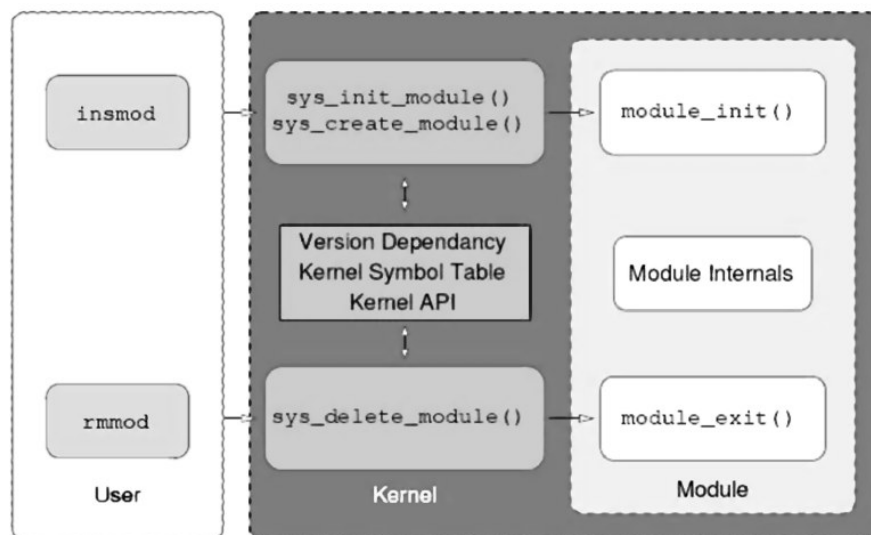


Figura 7: Architettura dei moduli del kernel

7.3 ESECUZIONE IN REAL-TIME

RTAI è una API che offre gli stessi servizi del core del kernel di Linux, aggiungendovi le caratteristiche tipiche di un sistema operativo real time. RTAI consiste sostanzialmente in un interrupt dispatcher: intercetta gli interrupt delle periferiche e, se necessario, le reindirizza a Linux. Non si tratta tuttavia di una modifica invasiva del kernel stesso, bensì RTAI utilizza il concetto di HAL (hardware abstraction layer) per avere informazioni da Linux e allo stesso tempo per intercettare le funzioni fondamentali. A tutti gli effetti RTAI considera Linux come un task ad elevata priorità eseguito in background. RTAI è completamente module oriented, e questo consente all'amministratore di scegliere quali moduli caricare per avere maggiori prestazioni. Nel nostro caso è stato scritto un piccolo script per caricare all'avvio del sistema operativo i moduli di RTAI necessari per far eseguire il driver del robot:

```
#!/bin/sh
/sbin/insmod /usr/realtime/modules/rtai_hal.o
/sbin/insmod /usr/realtime/modules/rtai_ksched.o
/sbin/insmod /usr/realtime/modules/rtai_fifos.o
```

E' importante sottolineare che alcuni moduli di RTAI sono mutuamente esclusivi, mentre altri per essere caricati esigono la presenza nel kernel di moduli base, come per esempio `rtai_hal`, che fornisce il livello di astrazione hardware sopra citato. Il modulo che è stato programmato quale driver per il robot, consiste di un insieme di task periodici che vengono schedulati da RTAI, e che sono stati preparati per lo scopo mediante l'utilizzo di alcuni metodi esposti dall'interfaccia real time. Tali metodi sono:

`rt_set_periodic_mode`

Prepara la schedulazione di RTAI in modalità periodica.

`rtf_create`

Crea una FIFO real time. La RT-FIFO è un meccanismo basato su caratteri che serve per far comunicare tra loro task real-time e processi ordinari di Linux.

`rt_task_wait_period`

Attende fino al sopraggiungere del prossimo periodo. Questa funzione è tipicamente posta alla fine della serie di istruzioni che vengono eseguite all'interno di uno stesso periodo.

`start_rt_timer`

Avvia un timer real time.

`rt_task_init`

Crea un nuovo task real time nello user space. La `rt_task_init` fornisce un buddy task (chiamato anche proxy task) al processo di Linux che vuole accedere ai servizi di schedulazione di RTAI.

`rt_task_make_periodic_relative_ns`

`rt_task_make_periodic` e `rt_task_make_periodic_relative_ns` marciano il task, precedentemente creato con la `rt_task_init`, come adatto a un'esecuzione periodica, specificando il periodo come parametro.

`stop_rt_timer`

Ferma il timer real time.

`rtf_destroy`

Distrugge la FIFO real time precedentemente creata.

`rt_task_delete`

Cancella il task creato mediante la `rt_task_init`

Capitolo 8

IL MOVIMENTO DEL ROBOT

8.1 LE PORTE DI I/O

Particolare importanza va data alla localizzazione delle porte di I/O sulla scheda utilizzata, infatti il microcontrollore Cirrus EP9302 presenta delle porte di ingresso-uscita chiamate General Purpose Input/Output (GPIO) che consentono una diretta comunicazione tra i vari dispositivi terminali (sensori e motori) e la scheda. Queste porte hanno la particolarità di permettere di impostare ogni singolo bit come ingresso o uscita. Il processore dispone di 7 porte I/O: porta A da 8 bit, B da 8 bit, C da 1 bit, E da 2 bit, F da 3 bit, G da 2 bit, H da 4 bit.

La porta E è già utilizzata dalla scheda per pilotare i due led, uno verde e uno rosso, presenti su di essa, mentre solo le porte A, B, C, F e H sono messe a disposizione dell'utente tramite due connettori, il DIO e il LCD, presenti sulla scheda ARM. Il connettore DIO è il principale mezzo di ingresso-uscita disponibile sulla scheda ARM utilizzata, la sua pedinatura è qui sotto riportata.

Table: DIO1 Header Pin Configuration

DIO1 Pin	Default Signal	TS-7200	TS-7300
1	DIO_0		
2	GND		
3	DIO_1		
4	Port_C0	ADC0	EXT_RESET
5	DIO_2		DIO_8
6	SPI_Frame	ADC4	
7	DIO_3		
8	DIO_8		ADC4
9	DIO_4		
10	SPI_MISO		
11	DIO_5		
12	SPI_MOSI		
13	DIO_6		
14	SPI_CLK		
15	DIO_7		
16	+3.3 V		

Figura 8: Piedinatura del connettore DIO

I pin, da DIO_0 a DIO_7, sono collegati alla porta B, il pin DIO_8 corrisponde al bit 0 della porta F ed infine il pin 4 del connettore corrisponde alla porta C.

Il connettore LCD è principalmente destinato al pilotaggio di una scheda equipaggiata con schermo LCD, ma può essere utilizzata come porta I/O generica e la sua pedinatura è descritta di seguito.

Table: LCD Header Pin Configuration

GND	Bias	LCD_WR	LCD_0	LCD_2	LCD_4	LCD_6
2	4	6	8	10	12	14
1	3	5	7	9	11	13
5V	LCD_RS	LCD_EN	LCD_1	LCD_3	LCD_5	LCD_7

Figura 9: Piedinatura del connettore LCD

I pin LCD_D0 a LCD_D7 corrispondono alla porta A, mentre i pin LCD_RS, LCD_EN e LCD_WR# corrispondono ai 3 bit della porta H.

Per accedere alle porte bisogna prima impostare il verso (ingresso o uscita) dei singoli bit del GPIO tramite determinati registri associati ad esse. Questi registri, chiamati Direction Data Register (DDR), hanno una determinata locazione in memoria. Se un bit è settato a 1 in un determinato DDR il pin della porta associata diventa un uscita, mentre se è settato a 0 questo corrisponde ad un ingresso. Una volta definito il verso si può accedere alle porte; questo avviene scrivendo o leggendo in una determinata locazione di memoria associata alla porta desiderata, questa viene chiamata Data Register (DR). Qui sono riportati gli indirizzi in esadecimale per le porte presenti sui connettori.

	DDR	DR
Porta A	0x80840010	0x80840000
Porta B	0x80840014	0x80840004
Porta C	0x80840018	0x80840008
Porta F	0x80840034	0x80840030
Porta H	0x80840044	0x80840040

Dal punto di vista del codice, per accedere ai registri delle porte, sono a disposizione due funzioni di linux: `outb_p()` per scrivere nel registro e `inb_p()` per leggere dal registro. La lettera b presente dopo "out" o "in" indica che si scrive o si legge un byte, mentre il suffisso _p indica che la funzione blocca l'esecuzione di un task fino a quando non viene stabilizzato il contenuto del registro. Le firme delle funzioni sono definite nel seguente modo:

```
void outb_p (int val, long port)
```

dove val sta per il valore in byte da scrivere nel registro e port sta per la locazione di memoria del registro.

```
int inb_p(long port)
```

dove port sta per la locazione di memoria del registro che si intende leggere.

Dato che ogni bit di una porta va definito come input o output e che le porte sono condivise da più dispositivi si rende necessario l'applicazione di filtri per settare i bit interessati senza modificare gli altri. Per applicare questi filtri si adotta il seguente metodo:

```
outb_p(inb_p(0x80840010) | 0x01, 0x80840010);
```

in questo esempio si setta come uscita il pin 0 della porta A, infatti viene prima letto il registro DDR associato, si esegue un'operazione or bit per bit e il risultato viene scritto nello stesso registro.

8.2 CONTROLLO DEI MOTORI

Per muoversi, la piattaforma dispone di due motori ma, dato che il robot è oloonomo, è importante regolare correttamente la velocità delle due ruote motrici in modo che si comporti correttamente. Infatti, per far sì che il robot vada dritto, le due velocità devono uguagliarsi, altrimenti la traiettoria risulterà curva. Per la regolazione della velocità dei motori viene utilizzata la scheda #203 della Wirz Electronics che fornisce in uscita una corrente ad entrambi gli attuatori, proporzionale al segnale PWM in ingresso.

La regolazione della velocità di un motore tramite segnale PWM consiste nel pilotare il circuito chiamato ponte ad H, presente nella scheda #203, con un segnale descritto in figura. Il segnale consiste in un'onda quadra periodica con frequenza dai 100Hz a 20 kHz, dove il tempo nel quale il segnale è a V_m all'interno del periodo (duty cycle) implica la velocità di rotazione del motore: maggiore è il tempo, maggiore è la velocità, fino ad arrivare ad un segnale costante V_m che corrisponde a velocità massima; di conseguenza se il tempo è nullo, il segnale è nullo e si ha il motore fermo.

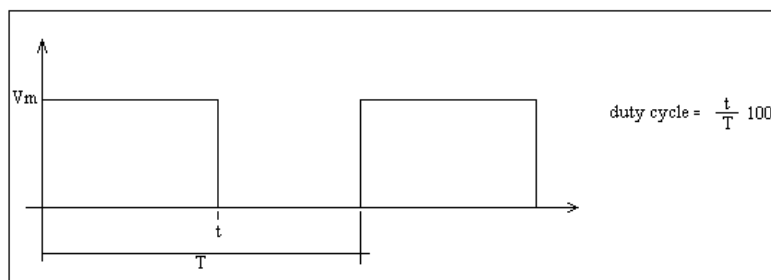


Figura 10: Segnale modulato in PWM

Unica cosa che rimane è stabilire la direzione di rotazione dei motori. Questo avviene sempre tramite la scheda #203 che in ingresso, oltre al PWM, richiede un segnale digitale che indichi il senso di rotazione per ogni motore. Particolare attenzione è da dare al segnale di direzione, in quanto questo può essere modificato solo e solo se prima il segnale PWM diviene nullo, altrimenti si ha un alto rischio di guastare il ponte ad H. Quindi prima si porta a 0 il duty cycle poi si può modificare la direzione.

Dato che la scheda PWM deve mettere a disposizione un segnale PWM al ponte ad H, all'interno del modulo sono presenti due task periodici, uno per motore, che generano questo segnale, con frequenza di 200Hz e con duty cycle assegnato tramite variabile globale. Questo

task opera nel seguente modo: ad ogni periodo, pone a livello alto il pin di uscita associato per poi chiamare la funzione `rt_sleep()` che attende per il tempo dato dal duty cycle permettendo ad altri task da eseguire; finito il tempo, riporta a zero il pin per il tempo rimanente per concludere il periodo. Visto che le operazioni effettuate all'interno del periodo sono due scritture sul registro DR della porta A, il fattore di utilizzazione del processore è molto basso, circa 10%. I segnali richiesti dal ponte ad H vengono forniti direttamente dalla porta B della scheda arm e sono collegati nel seguente modo:

- PWM_A => pin DIO_0
- Direction_A => pin DIO_1
- PWM_B => pin DIO_2
- Direction_B => pin DIO_3

Di conseguenza nella funzione `init_module()` abbiamo la riga di comando:

```
outb_p(inb_p(0x80840014) | 0x0F, 0x80840014);
```

dove i quattro bit meno significativi sono settati a 1 per rendere i pin della porta A delle uscite.

Qui viene riportato il codice della funzione che implementa il segnale PWM.

```
void rt_pwm_A () {
    while(1==1) {
        while (pwm_duty_cycle_A>0) {
            outb_p(inb_p(0x80840004) | 0x01, 0x80840004);
            rt_sleep(nano2count((pwm_time_A*pwm_duty_cycle_A)/100));
            outb_p(inb_p(0x80840004) & 0xFE, 0x80840004);
            rt_sleep(nano2count(((pwm_time_A*(100-
pwm_duty_cycle_A))/100)-10000));
        }
        outb_p(inb_p(0x80840004) & 0xFE, 0x80840004);
    }
}
```

Da notare la presenza di due `while` dovuti al fatto che con duty cycle a 0, il primo `outb_p` porta a 1 comunque il segnale anche per brevissimo periodo, ma se in quel istante viene cambiata la direzione di rotazione, c'è il rischio di danneggiare il ponte ad H.

8.3 LETTURA DEL SONAR

Il sonar, come si può vedere dalla figura, è composto da due sensori e un circuito di condizionamento. I sensori sono composti da un altoparlante ad ultra suoni e un microfono. La presenza di due sensori permette una misura binoculare la quale fa sì che l'errore di misura sia minimo, mentre il circuito di condizionamento permette di convertire i segnali dei sensori in segnali leggibili da dispositivi di acquisizione, nel nostro caso la scheda ts7250.

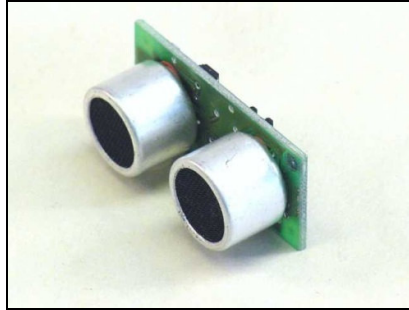


Figura 11: Sensore ad ultrasuoni SRF04

Il dispositivo sonar presenta quattro terminali di collegamento, tra i quali 2 sono per l'alimentazione da 5V e gli altri due per la comunicazione con la scheda di acquisizione. Il primo "trigger" è un ingresso da fornire dalla scheda; quest'ultimo deve mantenere a livello alto il segnale per circa $10\ \mu\text{s}$ dando il comando di iniziare la sequenza per la misurazione. Il secondo segnale "echo" viene posto alto dal dispositivo e letto dalla scheda per tutto il tempo in cui si aspetta il ritorno del segnale sonoro. La sequenza dei segnali è riportata nella figura. Di conseguenza il task che gestisce la sequenza di misurazione della distanza esegue i seguenti passi:

1. Invio del segnale "trigger" da $15\ \mu\text{s}$.
2. Attesa che "echo" diventi alto.
3. Inizio conteggio del tempo.
4. Attesa che "echo" diventi 0.
5. Fine conteggio e calcolo del valore.

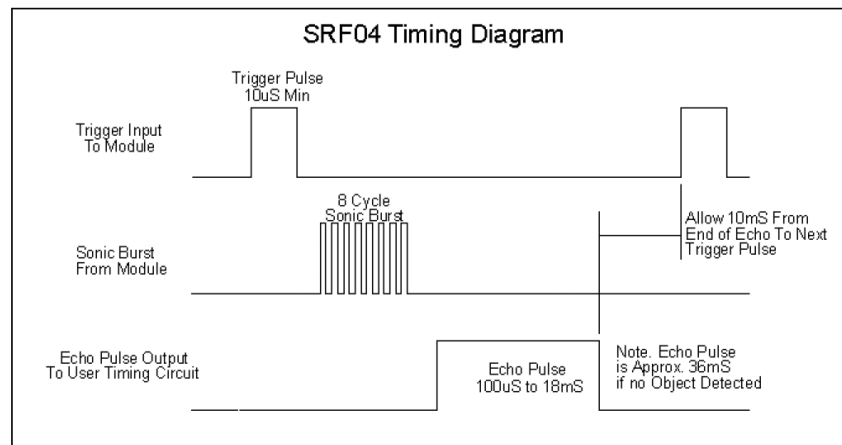


Figura 12: Diagramma dei tempi dei segnali del sensore

Dato che la sequenza va ripetuta periodicamente, il task è implementato come task periodico. A questo punto diventa importante la misurazione del tempo di esecuzione del task per stabilire il periodo. Questo è possibile utilizzando la funzione `rt_get_cpu_time_ns()` che fornisce in una variabile `RTIME` un valore temporale in nano secondi. Un modo per utilizzare la funzione è la seguente:

```

start = rt_get_cpu_time_ns();
/*.....Codice.....*/
printk(rt_get_cpu_time_ns()-start);

```

Il task del sonar varia il suo tempo di esecuzione a seconda dell'eco, quindi si misura il caso peggiore, si punta il sonar verso una direzione senza ostacoli. I risultati variano tra i 37 e 38 ms e il periodo è di 100ms, dando un fattore di utilizzazione del 38%.

I sonar sono collegati alla scheda nel seguente modo. L'alimentazione a 5V viene prelevata dal connettore LCD pin 1 e dal pin 2 la massa (GND), mentre il segnale digitale del dispositivo è collegato alla porta A sul connettore LCD e precisamente:

- Trigger_A => pin LCD_DO => output
- Echo_A => pin LCD_D1 => input

Dato che il codice per la misura della distanza è presente all'interno di un modulo, nella funzione `init_module()`, il registro DDR della porta A verrà settato tramite questa linea di codice.

```
outb_p((inb_p(0x80840010)|0x05) & 0xf5,0x80840010);
```

Di seguito è riportato il codice del task periodico per la gestione dei sonar.

```

static void rt_sonar_A(int t) {
RTIME start_eco_pulse = 0;      //Per contare il tempo
long int sonar_value_A;        //Il valore della distanza
// Task periodico che verrà schedulato da RTAI
while(1==1) {
    // Inizio la sequenza di misura
    // Imposto a 1 il pin 0 dell porta A
    // Aspetto 15 micro-sec
    // Imposto a 0 il pin 0 dell porta A
    outb_p(inb_p(0x80840000)| 0x01,0x80840000);
    rt_sleep(nano2count(15000LL));
    outb_p(inb_p(0x80840000)& 0xFE,0x80840000);
    //aspetto fino a che il secondo bit, collegato a "echo" diviene <> 0
    while ((inb_p(0x80840000) & 0x02) == 0x00) {}
    // conto quanto sta a 1 il secondo bit, cioè l'eco del segnale sonoro
    start_eco_pulse = rt_get_cpu_time_ns();
    while ((inb_p(0x80840000) & 0x02) == 0x02) {}
    sonar_value_A = (((rt_get_cpu_time_ns()-start_eco_pulse)/1000)/58);
    rt_task_wait_period();
} // endwhile
}

```

8.4 COMUNICAZIONE TRA MODULO REAL-TIME E PROCESSO UTENTE (RTAI IPC)

È necessario creare un canale di comunicazione tra il processo utente che si occupa del riconoscimento dei comandi e il modulo del kernel in real-time che si occupa del controllo dei motori e del sonar. L'idea è che il processo utente compia il riconoscimento e invii i comandi codificati in un byte (un carattere) al modulo real-time, il quale ogni volta che riceve un comando si occupa di controllare i motori appositamente.

Non è possibile usare i classici metodi di Inter Process Communication degli ambienti Linux poiché queste chiamate di sistema non sono compatibili con l'architettura real-time. La comunicazione tra processi utente e moduli real-time avviene quindi tramite apposite librerie in diversi modi:

- **FIFO**

È il primo meccanismo di comunicazione di cui RTAI fornisce una versione specializzata per i processi real-time. È un buffer di lettura/scrittura unidirezionale non bloccante che permette il trasferimento asincrono di dati, letti in maniera sequenziale dalla ipotetica coda in cui sono inseriti al momento della scrittura: nessun dato può essere sovrascritto. Corrisponde ad un file del tipo `/dev/rtn` e deve essere acceduto dai processi real-time tramite le API di RTAI. I processi standard vi accedono con le primitive previste da Linux per l'accesso ai file.

- **Shared Memory**

Si definisce un blocco di memoria che può essere letto e scritto da qualsiasi processo attivo nel sistema in modo asincrono. Si accede alla memoria direttamente tramite un puntatore al primo indirizzo allocato. I dati non sono accodati ma sovrascritti: occorre garantire accesso in mutua esclusione per mantenere l'integrità dei dati. Il meccanismo di memoria condivisa viene realizzato tramite un file (`/dev/rtai_shm`).

- **Message Passing e RPC**

È un semplice meccanismo di scambio messaggi punto-a-punto: mittente e destinatario devono conoscersi. Le code in cui sono depositati i messaggi in attesa di essere consumati sono ordinate secondo la priorità del messaggio (del mittente). L'invio di messaggi blocca per il mittente fino a quando il messaggio viene ricevuto. Nel caso di RPC il mittente attende fino al ricevimento della risposta del destinatario.

- **Mailbox**

È il meccanismo IPC più flessibile che RTAI mette a disposizione: permette ai processi di inviare messaggi che sono automaticamente memorizzati ordinati per priorità e letti al momento del bisogno. Più produttori e più consumatori di messaggi possono essere connessi ad una mailbox.

Questi metodi permettono la comunicazione sia tra processi real-time sia tra processi Linux standard. I meccanismi di comunicazione sono realizzati come moduli kernel indipendenti: devono essere caricati in memoria solo al momento in cui vi sono processi che ne fanno uso.

Inizialmente avevo pensato di utilizzare il metodo della Shared Memory, predisponendo una piccola area di memoria condivisa in cui salvare il comando da eseguire. Poiché il processo utente (che fa il riconoscimento) scrive solamente e il modulo real-time (che esegue il comando) legge solamente non ci sono neppure problemi di conflitto per la condivisione della risorsa. Purtroppo tentando di compilare il modulo che gestisce la memoria condivisa (rtai_ksm) ho riscontrato la totale assenza dei sorgenti di questo all'interno del pacchetto RTAI fornito dalla Technologic System.

Per questo motivo sono passato a realizzare l'IPC tramite FIFO, inteso come un tubo a senso unico dove il processo utente scrive un byte per volta (cioè un comando) che il modulo real-time legge man mano.

Nel modulo real-time è stata inclusa la libreria "rtai_fifos" e sono stati inseriti i seguenti comandi:

```
#define RTF_COMMAND 3
rtf_create(RTF_COMMAND, sizeof(char));
rtf_reset(RTF_COMMAND);
rtf_create_handler(RTF_COMMAND, fifo_handler);
```

La prima riga crea una FIFO identificata dal numero 3 delle dimensioni di un carattere, la seconda funzione la svuota, mentre il terzo comando è il più interessante: imposta un handler, cioè una specie di evento che si attiva ogni volta che viene scritto un carattere nella FIFO. L'handler impostato è una funzione creata appositamente per ricevere il carattere inviato e impostare una variabile globale che rappresenta l'ultimo comando vocale.

```
static int fifo_handler(unsigned int fifo) {
    if (rtf_get(RTF_COMMAND, &cmd, sizeof(cmd))) {
        printk("\n received command: %c \n", cmd);    }
    return 0;
}
```

In questo modo la lettura della FIFO è particolarmente comoda perché non è necessario che il modulo real-time controlli regolarmente la presenza di nuovi comandi, poiché questi vengono rilevati automaticamente dal modulo grazie all'evento.

Nel processo utente la scrittura dentro la FIFO è banale perché viene gestita direttamente da chiamate di sistema, infatti è sufficiente scrivere caratteri nello pseudo file /dev/rtf/3 con le funzioni IO classiche del C.

```
int fifo;
fifo = open("/dev/rtf/3", O_RDWR);
write(fifo, &cmd, sizeof(cmd));
close(fifo);
```

8.5 ESECUZIONE DEI COMANDI DI MOVIMENTO

Rimane solamente da considerare come vengono eseguiti i comandi all'interno del modulo real-time. L'algoritmo prende in considerazione l'ultimo comando ricevuto e la distanza rilevata dal sonar in questo modo:

- se il comando è uno fra "sinistra", "indietro", "destra" e semaforo è uguale a zero
 - azzero PWM e attendo 5ms
 - inverto la direzione di uno dei due motori (a seconda del comando ricevuto)
 - imposto PWM=50
 - imposto il semaforo a 1 nel caso di destra o sinistra, a 2 nel caso di indietro
 - imposto l'ultimo comando ricevuto a '0'
- altrimenti se l'ultimo comando ricevuto è '0' oppure la distanza del sonar è minore di 30 cm e il semaforo è uguale a zero
 - azzero PWM e attendo 5ms
- altrimenti se la distanza è minore di 80 cm e il semaforo è a zero
 - imposto i PWM dei motori in modo proporzionale alla distanza (più mi avvicino più rallento)
- altrimenti se la distanza è maggiore di 80 cm e il semaforo è a zero
 - imposto PWM al massimo (100) o a bassa velocità (20) nel caso in cui il comando sia '1' cioè "piano"
- altrimenti se il semaforo è uguale a 1 significa che sto voltando
 - attendo circa 1 secondo (il tempo necessario al robot per girarsi di 90°)
 - azzero PWM e attendo 5ms
 - imposto la direzione di entrambi i motori in avanti
 - imposto il semaforo a zero
- infine se il semaforo è uguale a 2 significa sta girando su sé stesso
 - attendo circa 2 secondi (il tempo necessario al robot per girarsi di 180°)
 - azzero PWM e attendo 5ms
 - imposto la direzione di entrambi i motori in avanti
 - imposto il semaforo a zero

Capitolo 9

CONCLUSIONI

9.1 RISULTATI RAGGIUNTI

La versione finale del robot risulta funzionante in ambienti controllati, infatti esso è in grado di comprendere varie parole o brevi frasi e di riconoscerle come comandi. Il robot è in grado di muoversi in avanti, destra, sinistra, girare su sé stesso, andare piano e veloce in base ai comandi vocali che gli vengono impartiti. Inoltre è presente un minimo di logica applicata al sensore ultrasuoni per evitare urti con gli ostacoli.

Un risultato interessante è essere riusciti a realizzare tutta la logica esclusivamente con la scheda ARM (eccetto il riconoscimento vocale), parallelizzando operazioni a bassissimo livello come la generazione del segnale ad onda quadra PWM e il controllo del sonar e operazioni ad alto livello come l'utilizzo del microfono USB, le connessioni tramite socket TCP su Ethernet, ecc.

Il sistema di riconoscimento vocale distribuito è funzionante e abbastanza flessibile. E' stato dedicato molto tempo all'apprendimento e alla pratica di tutte le funzionalità e le limitazioni della scheda ARM, un ambiente nuovo rispetto a quelli con cui avevo avuto a che fare finora.

La registrazione della voce sull'ARM viene eseguita con successo, anche se limitata esclusivamente a 44100 Hz. E' da notare l'assenza di scheda audio nella scheda ARM utilizzata, sostituita tramite il microfono USB. Questo ha richiesto una ricompilazione del kernel rispetto a quello originale fornito dalla casa costruttrice, operazione che ha richiesto molto tempo. I problemi maggiori si sono presentati nella ricerca della versione adatta del kernel, in modo da poter supportare anche l'esecuzione di processi in real-time. Non è stato facile neppure trovare la configurazione perfetta per far funzionare il cross-compiler, soprattutto a causa della scarsità di documentazione a riguardo.

Ottimi risultati sono stati raggiunti nella messa a punto globale della macchina ARM, a partire dalla riconfigurazione di Red Boot in modo da caricare il kernel tramite http e disporre della console anche attraverso la Ethernet oltre che la porta seriale. Il nuovo kernel inoltre include i moduli audio, usb e la patch RTAI per i moduli real-time. Non è stato possibile cross-compilare con successo i moduli ALSA per una gestione più avanzata del microfono USB.

Per quanto riguarda il riconoscimento vocale ci si è affidati a un software commerciale, quindi i risultati e gli errori riscontrati nella trascrizione dipendono da questo e dalla bontà dell'addestramento che è stato dato. E' sicuramente soddisfacente la procedura utilizzata per

interagire con le API di Dragon 5, disponibili solo in ambiente visuale normalmente, ma utilizzabili da console grazie al wrapper che è stato prodotto.

E' stata analizzata a lungo un'architettura ad agenti realizzata tramite la libreria Open Agent Architecture che alla fine non si è potuta realizzare a causa di alcuni errori di compilazione della stessa nell'ARM. Ho abbracciato comunque l'idea di realizzare un sistema totalmente orientato agli agenti, che comunicano tra loro tramite PIPE se girano sulla stessa macchina e tramite SOCKET per macchine diverse.

Ne è risultata un'architettura molto modulare ed eterogenea, dove ogni agente è rappresentato da un processo indipendente, correlato con gli altri solo tramite parametri di ingresso o ridirezionamenti degli standard output (in alcuni casi si fa riferimento a dei file globali, a cui accedono più processi).

Un risvolto interessante di questa architettura è che i singoli processi sono facilmente sostituibili, ad esempio se in seguito si volesse cambiare riconoscitore vocale è sufficiente creare un nuovo `trscribe.exe` che riconosce il wave passato come parametro e scrive il testo nello standard output. Se invece si volesse cambiare il programma che effettua la registrazione sarebbe sufficiente riconfigurare `recorder.arm` in modo da utilizzare il nuovo software al posto di `SoX`.

9.2 SVILUPPI FUTURI

Ora il software è realizzato in modo da riconoscere e completare un comando vocale alla volta. Nel momento in cui si finisce di pronunciare una parola, ad esempio "avanti", questo comando deve essere riconosciuto dal Dragon 5, e restituito all'ARM. Durante il tempo in cui viene effettuata questa operazione non è possibile dare altri comandi vocali al robot. Lo stato di attesa in cui il robot è in grado di "sentire" comandi è dato dal LED lampeggiante sul microfono USB, altrimenti fisso in caso in cui non stia registrando. Un effetto indesiderato è che se viene rilevato un rumore questo verrebbe preso come un comando, e passato al riconoscitore, che non restituirà alcun risultato; ciò non è un problema grave, solamente che durante questo tempo non è possibile dare altri comandi. Per ovviare a questo problema si potrebbe trasformare tutti i processi (il recorder, il tcp-client, il server Windows, il transcribe di Dragon 5) in multithreading, in modo da poter registrare di continuo tutti i comandi ed eseguire il riconoscimento in parallelo.

Per migliorare il riconoscimento dei comandi e soprattutto evitare che vengano "ascoltati" anche rumori di fondo come passi delle persone, colpi, brusio di sottofondo si dovrebbe inserire un filtro in grado di eliminare questi disturbi. Inoltre il rilevamento del silenzio dovrebbe essere fatto solamente per quanto riguarda la voce umana non su tutto lo spettro delle frequenze registrate dal microfono. Queste operazioni dovrebbero essere svolte da un circuito esterno all'ARM.

Per migliorare il riconoscimento delle parole e delle frasi si potrebbe provare a sostituire il software Dragon 5 con altri riconoscitori o con una versione più recente dello stesso, in grado di trascrivere file direttamente a 44100 Hz con un incremento di affidabilità. Si potrebbe

considerare anche di effettuare il riconoscimento direttamente sulla scheda ARM con un apposito software.

APPENDICI

A. RED BOOT: AVVIO DEL SISTEMA OPERATIVO SULL'ARM

La sequenza di boot della scheda TS-7250 si divide in quattro stadi:

1. TS-BOOTROM
2. RedBoot ROM
3. Kernel Linux
4. Prompt di login

Appena dopo l'accensione la scheda esegue il codice proprietario di avvio TSBOOTROM, che immediatamente esegue il RedBoot, un programma di avvio dalle caratteristiche avanzate, contenente strumenti di rete, tftp (trivial ftp), console via seriale, supporto per le immagini JFFS2/YAFFS2 su flash e capacità di caricare su ram ed eseguire il kernel di Linux.

Se RedBoot non viene interrotto con la combinazione di tasti Ctrl-C (trasmessi da terminale remoto sulla COM1) entro un secondo questo inizierà a caricare il precedente file system JFFS2/YAFFS2, e caricherà in memoria il kernel Linux di default.

Se si interrompe l'esecuzione di RedBoot con Ctrl-C si accede alla console di questo, di cui si può visualizzare la configurazione di default tramite il comando:

```
$ fconfig -l
```

La configurazione può essere cambiata completamente tramite il comando fconfig.

Lo script di default carica il kernel dalla flash e imposta Linux ad usare l'immagine JFFS2/YAFFS2 nella flash. Il kernel di Linux deve essere caricato all'indirizzo di memoria 0x00218000.

Il caricamento del kernel dalla flash viene effettuato automaticamente con il comando:

```
$ fis load vlinux
```

Dopo il caricamento del kernel viene eseguito automaticamente il comando:

```
$ exec -c "console=ttyAM0,115200 root=/dev/mtdblock1"
```

Avendo compilato un kernel personalizzato differente da quello di default, ma volendo evitare di sovrascrivere quello di default ho provveduto a modificare lo script di avvio di RedBoot in questo modo:

1. RedBoot attende prima di caricare lo script di default per 5 secondi

2. Il caricamento può essere interrotto con Ctrl-C su seriale oppure con una connessione via ethernet sulla porta 9000 (molto comodo se si è connessi tramite rete e non seriale)
3. Di default RedBoot non carica il kernel presente nella flash, ma lo scarica da un server web presente ad un ip fissato (la mia macchina) e lo carica nella ram al solito indirizzo
4. Esegue il kernel presente in ram

Per fare funzionare il caricamento via rete è stata impostata la configurazione di rete di RedBoot, fissando un ip, netmask e gateway.

Se si vuole provvedere al caricamento manuale del kernel via rete (serve un server web) è sufficiente digitare il comando:

```
$ load -v -r -b 0x00218000 -m http -h 140.105.63.206 /kernels/zImage
```

Per caricare il kernel via http all'indirizzo giusto.

```
$ exec -c "console=ttyAM0,115200 root=/dev/mtdblock1"
```

Per eseguire il kernel e impostare la console sulla seriale.

B. ORGANIZZAZIONE DEI SORGENTI

Nel sistema operativo Ubuntu è stata creata la cartella /tools che contiene tutto il software che è stato cross-compilato su questo ambiente:

```
+ tools
  + alsa
    + alsa-driver-1.0.4
  + arm
  + crosstool
    + arm-9tdmi-linux-gnu
      + gcc-3.3.4-glibc-2.3.2
  + linux24
  + oaa2.3.1
  + realtime
  + rtai-3.2
  + sox-13.0.0
```

Dentro crosstool si trova il cross-compiler per l'ARM che è stato utilizzato per tutte le compilazioni in Ubuntu. Tutti i makefile o i comandi utilizzati per la compilazione fanno riferimento a questa cartella.

Nella cartella linux24 ci sono i sorgenti completi del kernel di Linux a cui è stata aggiunta la patch RTAI-MAGMA. Questi sono stati compilati con una precisa configurazione descritta nel capitolo 3.4 .

In realtime ci sono i sorgenti dei moduli real-time (rtai_ksched, rtai_fifos, rtai_hal), compilati con un makefile standard modificato solo per quanto riguarda la cross-compilazione

(descritta nel capitolo 3.5). Il risultato della compilazione, cioè i moduli in binario per l'ARM si trova in rta-3.2.

In alsa ci sono i sorgenti degli alsa driver, dei quali è stata tentata senza successo una cross-compilazione, come pure per la cartella oaa2.3.1 che contiene i sorgenti della Open Agent Architecture.

In sox-13.0.0 si trovano i sorgenti del programma SoX con relativo makefile che è stato usato per compilare per l'ARM.

Dentro la cartella arm ci sono i sorgenti dei moduli real-time per il kernel che sono stati scritti per il controllo dei motori e del sonar. In particolare c'è il file rt_ferma.c che contiene i sorgenti per l'esecuzione dei comandi, il controllo dei motori e del sonar utilizzati nel progetto finale, con relativo makefile. Sono presenti anche altri sorgenti di altri programmi utilizzati per testing dei motori e del sonar.

Tutti i sorgenti di queste cartelle (i moduli real-time, il modulo per il controllo dei motori, il cross-compiler, ecc, sono direttamente collegati fra loro con riferimenti assoluti).

Una parte del software che è stato creato è stata compilata in Windows utilizzando l'emulatore Cygwin per comodità, anche se sarebbe stata equivalente una compilazione in Ubuntu. In Cygwin si usa lo stesso cross-compiler, solamente la versione apposita per Cygwin, fornita comunque dalla Technologic System.

Ecco l'albero delle cartelle del software compilato su Cygwin in Windows:

```

+ cygwin
  + fork
  + socket
  + tcpsocket
  + tools
    + crosstools
      + arm-unknown-linux-gnu
        + gcc-3.3.2-glibc-2.3.2
  + trscribe
  + docs

```

La cartella tools contiene sostanzialmente solamente il cross-compiler utilizzato; è necessario fare sempre riferimento a questa cartella quando si procede a compilare qualsiasi sorgente per l'ARM.

Nella cartella fork è presente il client tcp per l'ARM che si occupa dell'invio del file wave (tcp-client.cc) e il processo che effettua la registrazione, il rilevamento delle parole e il riconoscimento del comando (recorder.c).

E' stata una versione completamente testuale utilizzata per controllare il movimento (mautista.arm), questa versione simula in tutto e per tutto l'arrivo dei comandi vocali, solamente che questi vengono in realtà introdotti da tastiera.

Il comando usato per cross-compilare questi due sorgenti è semplicemente:

```

$ /tools/crosstools/arm-unknown-linux-gnu/gcc-3.3.2-glibc-2.3.2/bin/arm-unknown-linux-gnu-g++ tcp-client.cc -o tcp-client.arm

```

In sostanza si esegue il cross-compilatore g++ passando come parametro il file sorgente e il file di output. La stessa cosa vale per tutti gli altri sorgenti descritti di seguito.

Nella cartella tcpsocket sono presenti i sorgenti base usati per verificare e testare il funzionamento dei socket tra Windows e Linux, queste versioni non sono utilizzate nel software finale.

Infine nella cartella socket c'è il progetto per Visual Studio 2005 contenente i sorgenti del server tcp che gira su Windows, utilizzato nella soluzione finale. Questo progetto è impostato per compilare correttamente il file TCPEchoServer.cpp, comprese tutte le sue dipendenze (PracticalSocket). Il file risultante, socket.exe è il server che rimane in attesa di ricevere il file wave, e in automatico si occupa di richiamare il processo che fa la trascrizione, infine restituisce il risultato sempre tramite socket. E' quindi necessario che in questa cartella siano presenti anche i programmi sox.exe (per il ricampionamento del wave) e trscribe.exe (per il riconoscimento del file).

Rimane infine l'ultimo progetto (trscribe) compilato in Visual Studio 2005: il programma che esegue il riconoscimento vocale e la trascrizione del file wave. E' opportuno ricordare che il programma trscribe necessita dell'SDK di Naturally Speaking, opportunamente installato sulla macchina. Sono quindi presenti i riferimenti alle librerie (C:\Programmi\Dragon Systems\DNSTools\include). Potrebbe essere necessario inserire tale cartella nella variabile globale PATH in Windows.

Infine nella cartella docs sono presenti tutti gli articoli in formato digitale che sono stati utilizzati come riferimento e documentazione.

C. PROCEDURA COMPLETA DI FUNZIONAMENTO

Di seguito descrivo in pratica la procedura completa per avviare correttamente il robot.

- Inizializzazione del server Windows:
 1. Accensione del pc Windows.
 2. Controllare le impostazioni della rete locale (deve essere impostato a esempio con IP 192.168.0.77 e maschera 255.255.255.0)
 3. Apertura di Dragon Naturally Speaking 5.
 4. Apertura del profilo utente di Dragon (preventivamente addestrato).
 5. Esecuzione in console del programma socket.exe su una porta determinata (es: 2000).
 6. Nella stessa cartella di socket.exe devono essere presenti anche i programmi sox.exe e trscribe.exe, inoltre i files data.wav e dataresampled.wav devono essere scrivibili.
 7. A questo punto il server è in ascolto per connessioni tcp (il firewall deve permetterlo) ed è pronto per effettuare il riconoscimento vocale.
- Inizializzazione della scheda ARM montata sul robot:
 1. Verificare il corretto collegamento della scheda ARM ai motori, al sonar (tramite i connettori sulle porte DIO e LCD) e al server Windows tramite Ethernet.
 2. Accensione della scheda ARM tramite batteria

3. Connessione immediata alla scheda tramite porta seriale (o telnet su porta 9000) tramite il comando CTRL-C che ferma il Red Boot.
 4. Caricamento del kernel compilato con RTAI e supporto audio tramite i comandi visti nel capitolo A
 5. Inizializzazione di Linux
 6. Controllare la configurazione della network, eventualmente impostarla correttamente in modo che il server Windows sia raggiungibile (ping 192.168.0.77 ad esempio). Per riconfigurare l'ARM è sufficiente il comando:

```
$ fconfig eth0 192.168.0.2 netmask 255.255.255.0
```
 7. Inizializzazione di tutti i moduli esterni al kernel necessari per il funzionamento (audio, usb, usb media, usb audio, rtai_hal, rtai_ksched, rtai_fifos) mediante gli script /develop/initAll.sh, /develop/initRTAI.sh. E' necessario controllare che i moduli siano stati inizializzati con successo e che il microfono collegato alla porta USB sia riconosciuto correttamente
 8. Caricamento del modulo real-time per il controllo dei motori mediante il comando:

```
$ insmod /develop/rt_ferma.o
```
 9. Collegamento e accensione fisica dei motori tramite interruttore (a questo punto se il modulo rt_ferma.o funziona correttamente i motori dovrebbero restare fermi).
 10. Esecuzione del programma /develop/recorder.arm passando come argomento l'IP del server Windows (192.168.0.77) e la porta utilizzata (2000).
 11. Insieme a recorder.arm deve essere presente il programma tcp-client.arm e la cartella deve essere scrivibile.
- Tipico funzionamento:
 1. Con recorder.arm in esecuzione ogni volta che viene dato un comando vocale al microfono questo viene rilevato, registrato, inviato tramite socket al server Windows.
 2. Il processo socket.exe riceve il file wave, lo salva in data.wav e lo notifica a video
 3. Il file viene ricampionato da sox.exe e salvato in dataresampled.wav
 4. Viene chiamato il programma trscribe.exe che apre una piccola finestra e infine si chiude restituendo il testo trascritto
 5. recorder.arm riceve infine il testo e provvede a rilevare il comando
 6. recorder.arm scrive il comando riconosciuto (un byte) nella fifo del modulo real-time
 7. il modulo real-time rt_ferma riceve il comando e lo esegue
 8. ogni qualvolta il robot raggiunge la distanza di 30 cm da un ostacolo si ferma e accetta solamente comandi per girarsi, mai un "avanti".

BIBLIOGRAFIA

Articoli e manuali:

- La guida completa C++ - Herbert Schildt. - 2. ed. - Milano : McGraw-Hill, 1999
- Differences Between Windows and Unix Non-Blocking Sockets - Itamar, 2003
<http://www.advogato.org/article/672.html>
- TCP/IP Sockets in C: Practical Guide for Programmers - Michael J. Donahoo and Kenneth L. Calvert
<http://cs.ecs.baylor.edu/~donahoo/practical/C.Sockets/practical/>
- Getting started with ts-linux – Technologic Systems, 2006
- Linux for ARM on TS-7000 User's Guide – Technologic Systems, 2006
- TS-7250 Hardware Manual – Technologic Systems, 2006
- TS ARM Linux Developer's Manual – Technologic Systems, 2004

Siti internet di riferimento:

- WAVE PCM soundfile format
<http://ccrma.stanford.edu/CCRMA/Courses/422/projects/WaveFormat/>
- outb_p - Linux man page
http://linux.die.net/man/2/outb_p
- SRF04 - Ultra-Sonic Ranger
http://www.inertialsolutions.us/pdf_files/devantech-srf04-tech.pdf
- Low Current Motor Driver
<https://www.zagrosrobotics.com/files/wirz203.pdf>
- Socket Programming HOWTO - Gordon McMillan
<http://www.python.it/doc/howto/Socket/sockets-it/sockets-it.html>
- sox - Linux man page
<http://linux.die.net/man/1/sox>
<http://cf.ccmr.cornell.edu/cgi-bin/w3mman2html.cgi?sox>

Documentazione tecnica:

- Single Board Computers for Embedded Systems
http://www.embeddedarm.com/epc/prod_SBC.htm#arm
- TS-7250 ARM Single Board Computer
<http://www.embeddedarm.com/epc/ts7250-spec-h.html>
- Linux for ARM on TS-7000 Series SBC's
<http://www.embeddedarm.com/linux/ARM.htm>

- Real Time Application Interface for TS-7000 series SBC's
<http://www.embeddedarm.com/linux/rtai.htm>
- Open Sound System
<http://www.opensound.com/>
- Sound & MIDI Software For Linux
<http://linux-sound.org/one-page.html>
- The Open Agent Architecture
<http://www.ai.sri.com/oa/>
- The Open Agent Architecture™ 2.3.2 Distribution
<http://www.ai.sri.com/~oaa/distribution/v2.3/2.3.2/>
- RTAI API Documentation 3.0r5
<http://download.gna.org/rtai/documentation/kilauea/html/main.html>

RINGRAZIAMENTI

Ringrazio innanzitutto i miei genitori, per aver sempre creduto in me e per essersi sempre fidati delle mie capacità, dandomi la possibilità di fare sempre ciò che ho desiderato sostenendomi nelle decisioni.

Ringrazio mia sorella che mi ha sempre fornito un modello positivo con cui confrontarmi, con la quale ho sempre potuto condividere dubbi e problemi.

Un sentito ringraziamento va a tutti i miei amici, in particolare Lisa, Andrea, Carlo, Sara, che hanno sempre apprezzato ciò che ho fatto e hanno contribuito a creare in me la passione che ho per quello che studio.

Ringrazio anche gli amici di Trieste, in particolare Lorenzo, con i quali ho condiviso moltissime giornate “universitarie” e non.

Infine l'ultimo speciale ringraziamento va ad Ale, che ha illuminato gli ultimissimi mesi prima della laurea con i suoi occhi e il suo sorriso.

LICENZA D'USO

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. TO THE EXTENT THIS LICENSE MAY BE CONSIDERED TO BE A CONTRACT, THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

- a. **"Adaptation"** means a work based upon the Work, or upon the Work and other pre-existing works, such as a translation, adaptation, derivative work, arrangement of music or other alterations of a literary or artistic work, or phonogram or performance and includes cinematographic adaptations or any other form in which the Work may be recast, transformed, or adapted including in any form recognizably derived from the original, except that a work that constitutes a Collection will not be considered an Adaptation for the purpose of this License. For the avoidance of doubt, where the Work is a musical work, performance or phonogram, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered an Adaptation for the purpose of this License.
- b. **"Collection"** means a collection of literary or artistic works, such as encyclopedias and anthologies, or performances, phonograms or broadcasts, or other works or subject matter other than works listed in Section 1(f) below, which, by reason of the selection and arrangement of their contents, constitute intellectual creations, in which the Work is included in its entirety in unmodified form along with one or more other contributions, each constituting separate and independent works in themselves, which together are assembled into a collective whole. A work that constitutes a Collection will not be considered an Adaptation (as defined above) for the purposes of this License.
- c. **"Distribute"** means to make available to the public the original and copies of the Work or Adaptation, as appropriate, through sale or other transfer of ownership.
- d. **"Licensor"** means the individual, individuals, entity or entities that offer(s) the Work under the terms of this License.
- e. **"Original Author"** means, in the case of a literary or artistic work, the individual, individuals, entity or entities who created the Work or if no individual or entity can be identified, the publisher; and in addition (i) in the case of a performance the actors, singers, musicians, dancers, and other persons who act, sing, deliver, declaim, play in, interpret or otherwise perform literary or artistic works or expressions of folklore; (ii) in the case of a phonogram the producer being the person or legal entity who first fixes the sounds of a performance or other sounds; and, (iii) in the case of broadcasts, the organization that transmits the broadcast.
- f. **"Work"** means the literary and/or artistic work offered under the terms of this License including without limitation any production in the literary, scientific and artistic domain, whatever may be the mode or form of its expression including digital form, such as a book, pamphlet and other writing; a lecture, address, sermon or other work of the same nature; a dramatic or dramatico-musical work; a choreographic work or entertainment in dumb show; a musical composition with or without words; a cinematographic work to which are assimilated works expressed by a process analogous to cinematography; a work of drawing, painting, architecture, sculpture, engraving or lithography; a photographic work to which are assimilated works expressed by a process analogous to photography; a work of applied art; an illustration, map, plan, sketch or three-dimensional work relative to geography, topography, architecture or science; a performance; a broadcast; a phonogram; a compilation of data to the extent it is protected as a copyrightable work; or a work performed by a variety or circus performer to the extent it is not otherwise considered a literary or artistic work.
- g. **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

- h. **"Publicly Perform"** means to perform public recitations of the Work and to communicate to the public those public recitations, by any means or process, including by wire or wireless means or public digital performances; to make available to the public Works in such a way that members of the public may access these Works from a place and at a place individually chosen by them; to perform the Work to the public by any means or process and the communication to the public of the performances of the Work, including by public digital performance; to broadcast and rebroadcast the Work by any means including signs, sounds or images.
- i. **"Reproduce"** means to make copies of the Work by any means including without limitation by sound or visual recordings and the right of fixation and reproducing fixations of the Work, including storage of a protected performance or phonogram in digital form or other electronic medium.

2. Fair Dealing Rights. Nothing in this License is intended to reduce, limit, or restrict any uses free from copyright or rights arising from limitations or exceptions that are provided for in connection with the copyright protection under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

- a. to Reproduce the Work, to incorporate the Work into one or more Collections, and to Reproduce the Work as incorporated in the Collections;
- b. to create and Reproduce Adaptations provided that any such Adaptation, including any translation in any medium, takes reasonable steps to clearly label, demarcate or otherwise identify that changes were made to the original Work. For example, a translation could be marked "The original work was translated from English to Spanish," or a modification could indicate "The original work has been modified.";
- c. to Distribute and Publicly Perform the Work including as incorporated in Collections; and,
- d. to Distribute and Publicly Perform Adaptations.

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats. Subject to Section 8(f), all rights not expressly granted by Licensor are hereby reserved, including but not limited to the rights set forth in Section 4(d).

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

- a. You may Distribute or Publicly Perform the Work only under the terms of this License. You must include a copy of, or the Uniform Resource Identifier (URI) for, this License with every copy of the Work You Distribute or Publicly Perform. You may not offer or impose any terms on the Work that restrict the terms of this License or the ability of the recipient of the Work to exercise the rights granted to that recipient under the terms of the License. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties with every copy of the Work You Distribute or Publicly Perform. When You Distribute or Publicly Perform the Work, You may not impose any effective technological measures on the Work that restrict the ability of a recipient of the Work from You to exercise the rights granted to that recipient under the terms of the License. This Section 4(a) applies to the Work as incorporated in a Collection, but this does not require the Collection apart from the Work itself to be made subject to the terms of this License. If You create a Collection, upon notice from any Licensor You must, to the extent practicable, remove from the Collection any credit as required by Section 4(c), as requested. If You create an Adaptation, upon notice from any Licensor You must, to the extent practicable, remove from the Adaptation any credit as required by Section 4(c), as requested.
- b. You may not exercise any of the rights granted to You in Section 3 above in any manner that is primarily intended for or directed toward commercial advantage or private monetary compensation. The exchange of the Work for other copyrighted works by means of digital file-sharing or otherwise shall not be considered to be intended for or directed toward commercial advantage or private monetary compensation, provided there is no payment of any monetary compensation in connection with the exchange of copyrighted works.
- c. If You Distribute, or Publicly Perform the Work or any Adaptations or Collections, You must, unless a request has been made pursuant to Section 4(a), keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or if the Original Author and/or Licensor designate another party or parties (e.g., a sponsor institute, publishing entity, journal) for attribution ("Attribution Parties") in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or

- parties; (ii) the title of the Work if supplied; (iii) to the extent reasonably practicable, the URI, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work; and, (iv) consistent with Section 3(b), in the case of an Adaptation, a credit identifying the use of the Work in the Adaptation (e.g., "French translation of the Work by Original Author," or "Screenplay based on original Work by Original Author"). The credit required by this Section 4(c) may be implemented in any reasonable manner; provided, however, that in the case of a Adaptation or Collection, at a minimum such credit will appear, if a credit for all contributing authors of the Adaptation or Collection appears, then as part of these credits and in a manner at least as prominent as the credits for the other contributing authors. For the avoidance of doubt, You may only use the credit required by this Section for the purpose of attribution in the manner set out above and, by exercising Your rights under this License, You may not implicitly or explicitly assert or imply any connection with, sponsorship or endorsement by the Original Author, Licensor and/or Attribution Parties, as appropriate, of You or Your use of the Work, without the separate, express prior written permission of the Original Author, Licensor and/or Attribution Parties.
- d. For the avoidance of doubt:
- i. **Non-waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme cannot be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License;
 - ii. **Waivable Compulsory License Schemes.** In those jurisdictions in which the right to collect royalties through any statutory or compulsory licensing scheme can be waived, the Licensor reserves the exclusive right to collect such royalties for any exercise by You of the rights granted under this License if Your exercise of such rights is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(b) and otherwise waives the right to collect royalties through any statutory or compulsory licensing scheme; and,
 - iii. **Voluntary License Schemes.** The Licensor reserves the right to collect royalties, whether individually or, in the event that the Licensor is a member of a collecting society that administers voluntary licensing schemes, via that society, from any exercise by You of the rights granted under this License that is for a purpose or use which is otherwise than noncommercial as permitted under Section 4(c).
- e. Except as otherwise agreed in writing by the Licensor or as may be otherwise permitted by applicable law, if You Reproduce, Distribute or Publicly Perform the Work either by itself or as part of any Adaptations or Collections, You must not distort, mutilate, modify or take other derogatory action in relation to the Work which would be prejudicial to the Original Author's honor or reputation. Licensor agrees that in those jurisdictions (e.g. Japan), in which any exercise of the right granted in Section 3(b) of this License (the right to make Adaptations) would be deemed to be a distortion, mutilation, modification or other derogatory action prejudicial to the Original Author's honor and reputation, the Licensor will waive or not assert, as appropriate, this Section, to the fullest extent permitted by the applicable national law, to enable You to reasonably exercise Your right under Section 3(b) of this License (right to make Adaptations) but not otherwise.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE WORK, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NON-INFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

- a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Adaptations or Collections from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.

- b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

- a. Each time You Distribute or Publicly Perform the Work or a Collection, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
- b. Each time You Distribute or Publicly Perform an Adaptation, Licensor offers to the recipient a license to the original Work on the same terms and conditions as the license granted to You under this License.
- c. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
- d. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
- e. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.
- f. The rights granted under, and the subject matter referenced, in this License were drafted utilizing the terminology of the Berne Convention for the Protection of Literary and Artistic Works (as amended on September 28, 1979), the Rome Convention of 1961, the WIPO Copyright Treaty of 1996, the WIPO Performances and Phonograms Treaty of 1996 and the Universal Copyright Convention (as revised on July 24, 1971). These rights and subject matter take effect in the relevant jurisdiction in which the License terms are sought to be enforced according to the corresponding provisions of the implementation of those treaty provisions in the applicable national law. If the standard suite of rights granted under applicable copyright law includes additional rights not granted under this License, such additional rights are deemed to be included in the License; this License is not intended to restrict the license of any rights under applicable law.

CREATIVE COMMONS NOTICE

Creative Commons is not a party to this License, and makes no warranty whatsoever in connection with the Work. Creative Commons will not be liable to You or any party on any legal theory for any damages whatsoever, including without limitation any general, special, incidental or consequential damages arising in connection to this license. Notwithstanding the foregoing two (2) sentences, if Creative Commons has expressly identified itself as the Licensor hereunder, it shall have all rights and obligations of Licensor.

Except for the limited purpose of indicating to the public that the Work is licensed under the CCPL, Creative Commons does not authorize the use by either party of the trademark "Creative Commons" or any related trademark or logo of Creative Commons without the prior written consent of Creative Commons. Any permitted use will be in compliance with Creative Commons' then-current trademark usage guidelines, as may be published on its website or otherwise made available upon request from time to time. For the avoidance of doubt, this trademark restriction does not form part of the License.

Creative Commons may be contacted at <http://creativecommons.org/>.