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

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica Mobile
(Prof. Riccardo Cassinis)

Sviluppo del software di controllo per il marker attivo del robot Speedy

Elaborato di esame di:

Simone Borda, Guido Pollini, Sara Sabbadini

Consegnato il: 29 giugno 2007

Sommario

L'elaborato svolto fa parte di un più ampio progetto volto a sviluppare un sistema di posizionamento per il robot ActivMedia Pioneer 1 "Speedy". Per identificarne la posizione, il robot viene equipaggiato con un gruppo di LED ad alta intensità che lampeggiano ad una frequenza stabilita dall'utente. Elaborando le immagini acquisite da una telecamera è possibile riconoscere i lampeggi dei LED e quindi stabilire la posizione del robot. Un sistema di questo tipo è già stato realizzato per il robot ActivMedia Pioneer 3 "Morgul" ed ora si vuole realizzare un sistema analogo anche per il modello Pioneer1. Nasce il problema di creare il software in grado di comandare il lampeggio dei LED e posizionarli, tramite un motore a passo, in modo tale da poter essere rilevati da una telecamera. Il presente elaborato si propone di risolvere questo problema.

1. Introduzione

Questo elaborato si colloca all'interno di un progetto di più ampio respiro che si pone come obiettivo la realizzazione di un sistema di localizzazione per robot mobili. Per localizzare il robot, tale progetto prevede l'utilizzo di una telecamera il cui campo visivo riesca ad inquadrare tutto l'ambiente in cui il robot deve muoversi; l'idea è che, nel caso in cui il robot ne abbia bisogno, possa conoscere la propria posizione semplicemente facendo richiesta ad un calcolatore esterno connesso ad una telecamera fissa in grado di acquisire tutta la scena in cui si muove il robot stesso.

Per poter localizzare più facilmente ed in maniera più affidabile il robot mediante l'utilizzo della telecamera, è stato necessario applicare al robot un marker attivo, costituito da LED ad alta intensità che vengono fatti lampeggiare ad una frequenza ben definita e ben distinguibile da ogni comportamento di qualsiasi oggetto o fenomeno esistente in natura. Nelle immagini in figura 1.1 e figura 1.2 è possibile vedere rispettivamente i LED ad alta intensità utilizzati ed il sistema completo del marker applicato nella parte superiore del robot *Speedy*.

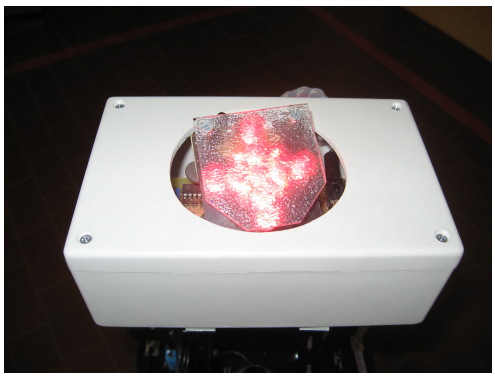

Figura 1.1 – LED ad alta intensità

Figura 1.2 – Marker installato sul robot *Speedy*

Lo scopo di questo elaborato è stato lo sviluppo di un software che consente all'utente di controllare il lampeggio dei LED montati sul robot *Speedy* e la rotazione degli stessi mediante un motore a passo.

2. Il problema affrontato

Si è dovuto affrontare il problema di rendere disponibili ad un calcolatore remoto le funzionalità offerte da un microcontrollore precedentemente installato e programmato sul robot *Speedy*. Tale microcontrollore si serve di un'interfaccia seriale per ricevere ed inviare dati al calcolatore a bordo di *Speedy* allo scopo di consentire il funzionamento del marker.

Il lavoro si è suddiviso in due fasi principali:

- verifica del corretto funzionamento del microcontrollore e del marker, al fine di individuare i comandi effettivamente disponibili e le relative risposte che il microcontrollore invia sulla seriale;
- implementazione del software che, ricevendo i comandi inviati dall'utente al marker tramite connessione TCP, li redirige al microcontrollore, ne riceve le risposte e le inoltra al client remoto.

3. La soluzione adottata

Nel seguito saranno descritte le attività svolte per risolvere il problema presentato.

3.1. Test dell'hardware preesistente

Per verificare il corretto funzionamento del marker sono stati testati tutti i comandi indicati dalla documentazione fornita per il microcontrollore [1]: tramite `echo [comando] > /dev/ttyUSB1` sono stati rediretti sull'interfaccia seriale (corrispondente in questo caso al dispositivo identificato con ttyUSB1) i comandi da testare e tramite `cat /dev/ttyUSB1` sono state valutate le risposte.

La seguente tabella riporta i comandi effettivamente funzionanti e le relative risposte:

COMANDO	PARAMETRI	AZIONE	RISPOSTA
a	1xxxyyyzzz xxx = numero di lampeggi da eseguire (da 001 a 999, in cui 999 corrisponde ad un numero di lampeggi infinito) yyy = tempo di accensione di un lampeggio in ms (da 001 a 999) zzz = tempo di spegnimento tra un lampeggio e l'altro in ms (da 001 a 999)	lampeggio LED	Aa A: indica l'inizio del lampeggio a: indica la fine del lampeggio
d		accensione LED	A
D		spegnimento LED	a
g		leggi posizione	posizione del motore a passo (da 0 a 199); restituisce -1 se non è stato eseguito il riposizionamento all'avvio
m	xxxyzzz	azionamento motore	SMm

	xxx = periodo di esecuzione di un passo espresso in ms (da 001 a 999) y = direzione di rotazione (f per senso orario, r per senso antiorario) zzz = numero di passi (da 001 a 999, in cui 999 corrisponde ad un numero infinito di passi)		S: indica l'accensione della scheda driver del motore M: indica l'inizio del movimento m: indica la fine del movimento
l		arresto del motore	m
n		accensione della scheda driver del motore	S
N		spegnimento della scheda driver del motore	s
p		riporta il motore in posizione 0	SMm S: indica l'accensione della scheda driver del motore M: indica l'inizio del movimento m: indica la fine del movimento N.B. se il motore è già in posizione 0 non viene fornita alcuna risposta
s		spegne sia la scheda driver del motore che i LED	abcm
v		attiva la modalità verbose	nessuna risposta
V		disattiva la modalità verbose	nessuna risposta

Tabella 3.1 – Comandi e risposte del microcontrollore

3.2. Test del software preesistente

All'inizio di questo lavoro è stato fornito anche del software che gestisce il marker del robot *Morgul*. Tale software utilizza la libreria ARIA per la comunicazione con il marker, che è connesso alla porta seriale ausiliaria del robot.

Poiché *Speedy* non possiede questa porta, era stata implementata anche una classe per la connessione al marker attraverso un adattatore USB-RS232, in modo da poter collegare il marker direttamente al calcolatore di *Speedy*. Sfortunatamente tale classe è andata persa e, nel tentativo di implementarla nuovamente, si è visto che sarebbe stato più conveniente creare un nuovo programma per la gestione del marker senza utilizzare la libreria ARIA.

3.3. Sviluppo del software

Si è scelto di sviluppare un'applicazione assimilabile ad un proxy, che si interpone tra un client remoto ed il microcontrollore del marker (che in questo caso ha funzione di server), inoltrando le richieste e le risposte dall'uno all'altro. Il proxy può essere avviato come demone sul calcolatore di *Speedy* restando in ascolto su una specifica porta TCP in attesa di eventuali comandi. Il client si collega al proxy tramite una connessione TCP e gli invia i comandi per il marker. Il proxy verifica la correttezza dei comandi ricevuti e, se validi, li inoltra al microcontrollore attraverso la porta seriale. Per ogni comando eseguito il marker fornisce una risposta che viene letta dal proxy sulla porta seriale ed inviata al client con la connessione TCP. Se il comando ricevuto non è valido, il proxy avvisa il client, ma resta comunque attivo fino a quando non riceve il comando di arresto dal clienti (*stop*).

In figura 3.1 è rappresentato lo schema di comunicazione appena descritto.

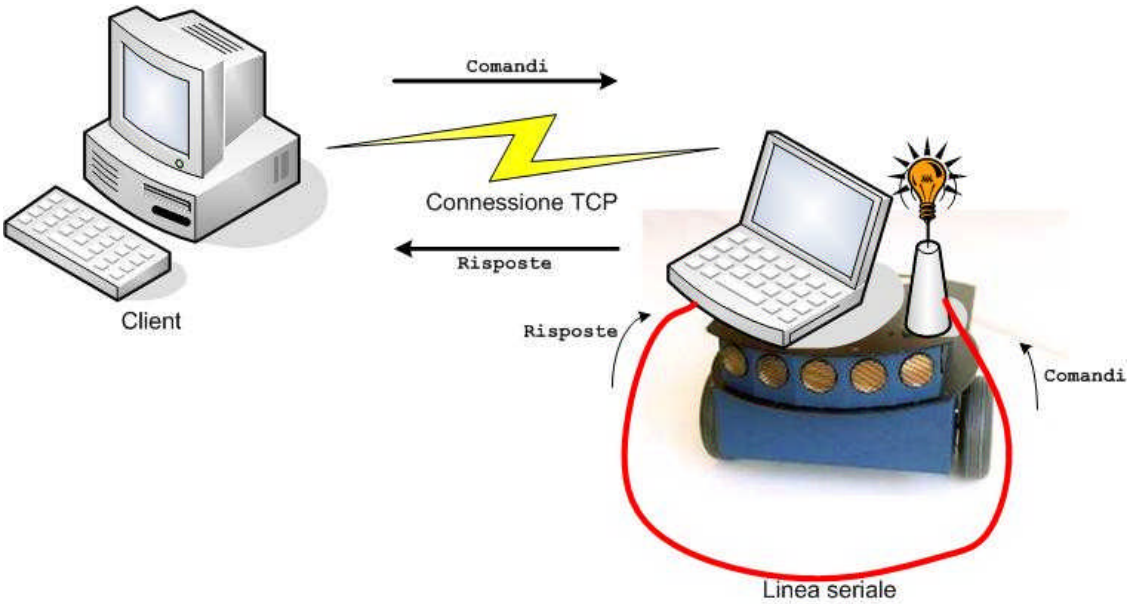

Figura 3.1 – Modello di comunicazione implementato

3.3.1. Descrizione del proxy

Il programma proxy è stato scritto in linguaggio *C*, compilato con la versione 4.0.3 di *gcc* ed è composto dai file sorgenti *proxyMarker.c* e *auxMarker.c*; il primo contiene il corpo principale del proxy, mentre il secondo raggruppa alcune funzioni ausiliarie ed è incluso nel primo tramite l'istruzione `#include "auxMarker.c"`.

All'avvio dell'applicazione, *proxyMarker* inizializza il marker riportandolo "a casa" (posizione 0), spegnendo i LED e disattivando la modalità "verbose". In seguito, crea ed inizializza un socket, assegnandogli una porta e mettendolo in ascolto su tutte le interfacce di rete disponibili, nell'attesa di connessioni TCP.

Quando un client remoto instaura una connessione ed invia un comando il proxy:

1. crea un socket temporaneo per gestire la connessione;
2. fa una lettura sulla porta seriale per controllare la presenza di eventuali risposte pendenti ai comandi eseguiti precedentemente;
3. legge il comando inviato dal client (fare riferimento alla tabella 3.2 per i comandi effettivamente implementati e le relative risposte) e ne verifica la validità;
4. invia al marker i comandi validi tramite la linea seriale;
5. legge la risposta restituita dal marker sulla porta seriale;

6. invia al client le eventuali risposte pendenti e, separatamente, la risposta all'ultimo comando eseguito. Se l'ultimo comando non è risultato valido la risposta è *nop*;
7. chiude il socket temporaneo e resta in attesa di nuove connessioni.

Se il comando ricevuto è quello di chiusura (*stop*), il proxy:

1. inoltra le eventuali risposte pendenti;
2. inoltra la risposta al comando *stop*;
3. chiude il socket temporaneo;
4. spegne i LED e riporta il marker "a casa";
5. chiude anche il socket principale e termina la propria esecuzione.

Nell'implementazione dei comandi resi disponibili dal proxy sono state apportate alcune modifiche rispetto ai comandi presentati in tabella 3.1

La tabella 3.2 mostra come vengono gestiti i comandi ricevuti dal proxy.

COMANDO	PARAMETRI	AZIONE	RISPOSTA
a	1xxxyyyzzz xxx = numero di lampeggi da eseguire (da 001 a 999, in cui 999 corrisponde ad un numero di lampeggi infinito) yyy = tempo di accensione di un lampeggio in ms (da 001 a 999) zzz = tempo di spegnimento tra un lampeggio e l'altro in ms (da 001 a 999)	lampeggio LED	Aa A: indica l'inizio del lampeggio a: indica la fine del lampeggio
d		accensione LED	A
D		spegnimento LED	a
g		leggi posizione	posizione del motore a passo (da 0 a 199); restituisce -1 se non è stato eseguito il riposizionamento all'avvio
m	xxxyzzz xxx = periodo di esecuzione di un passo espresso in ms (da 001 a 999) y = direzione di rotazione (f per senso orario, r per senso antiorario) zzz = numero di passi (da 001 a 999, in cui 999 corrisponde ad un numero infinito di passi)	azionamento motore	SMm S: indica l'accensione della scheda driver del motore M: indica l'inizio del movimento m: indica la fine del movimento
l		arresto del motore	m
n		accensione della scheda driver del	S

		motore	
N		spegnimento della scheda driver del motore	s
p		riporta il motore in posizione 0	SMm S: indica l'accensione della scheda driver del motore M: indica l'inizio del movimento m: indica la fine del movimento N.B. Se il motore è già in posizione 0 la risposta è 'H'.
s		spegne sia la scheda driver del motore che i LED	abcm
stop		termina l'esecuzione del proxy	Proxy chiuso
comando non valido		il comando non è valido quindi non viene inviato al marker	nop

Tabella 3.2 – Comandi e risposte resi disponibili dal proxy

Ø **L'attivazione e la disattivazione della modalità "verbose" (comandi *v* e *V*) non sono implementate dal proxy in quanto tale modalità non funziona correttamente sul microcontrollore per i comandi di lampeggio e rotazione. Dalle prove effettuate risulta che in modalità "verbose" il microcontrollore non è in grado di leggere comandi costituiti da più di tre caratteri.**

3.3.2. Descrizione del client

Al fine di testare il corretto funzionamento di *proxyMarker*, è stato realizzato un client in *java* che stabilisce una connessione con il proxy ed invia un comando per il marker.

Ad ogni esecuzione il client instaura una connessione con *proxyMarker* ed invia un solo comando; in seguito si aspetta due risposte: la prima è quella eventualmente pendente, la seconda è quella relativa al comando corrente. Una volta ricevute le risposte, il client chiude la connessione e si arresta.

La figura 3.2 mostra il protocollo di comunicazione tra client, *proxyMarker* e marker.

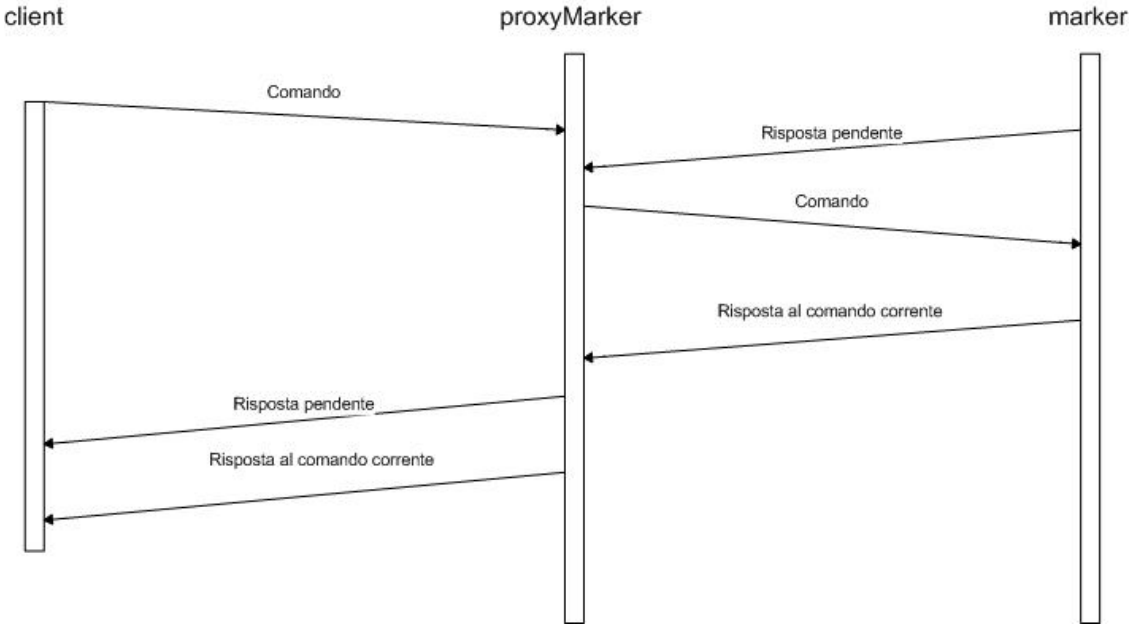

Figura 3.2 – Protocollo di comunicazione adottato

Il client è stato volutamente implementato senza interfaccia grafica ed in grado di inviare un solo comando in modo da essere facilmente integrato in altre applicazioni o script e per garantire la corretta interpretazione delle risposte inviate dal microcontrollore.

4. Modalità operative

In questo paragrafo viene mostrato come installare l'adattatore USB-RS232 sul calcolatore di Speedy, quali sono i parametri da settare per il suo corretto funzionamento e come utilizzare il software realizzato.

4.1. Componenti necessari

Per poter utilizzare *proxyMarker* sono necessari:

- calcolatore dotato di porta USB, interfaccia di rete wireless e sistema operativo Linux;
- marker attivo composto da un microcontrollore PIC 16F876 programmato come descritto in [1], un gruppo di LED ad alta intensità e un motore a passo con relativa scheda driver;
- adattatore USB-RS232;
- compilatore *gcc*.

Se s'intende utilizzare anche il client java serve anche la java virtual machine ed un compilatore *java* nel caso si possieda solo il sorgente.

4.2. Modalità di installazione

In questo paragrafo è mostrato come predisporre l'hardware necessario al funzionamento del marker e come compilare il codice sorgente delle applicazioni sviluppate.

4.2.1. Collegamento del marker al calcolatore di *Speedy*

La prima cosa da fare per poter utilizzare il marker è posizionare l'adattatore USB-RS232 (figura 4.1) poiché il calcolatore utilizzato su *Speedy* non possiede alcuna porta seriale.

Figura 4.1 – Adattatore USB-RS232

L'adattatore va inserito nella porta USB contrassegnata dalla sigla *USB1* visibile in figura 4.2, in modo tale da consentire il corretto assegnamento dei dispositivi USB all'avvio del calcolatore:

- *USB0* per il collegamento con il robot;
- *USB1* per il collegamento con il marker;
- *USB2* per il collegamento con l'eventuale telecamera.

Figura 4.2 – Ordine di assegnamento dei dispositivi USB all'avvio del calcolatore

A questo punto si deve collegare anche il cavo di alimentazione del marker ad una delle due prese di alimentazione per accessori su *Speedy* (figura 4.3) ed accertarsi che l'interruttore *AUX POWER* sia in posizione *ON* (figura 4.4).

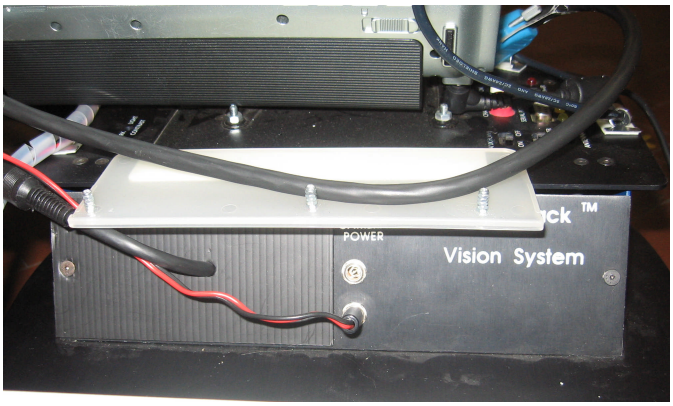

Figura 4.3 – Prese di alimentazione per gli accessori di *Speedy*

Figura 4.4 – Interruttore per l'alimentazione degli accessori di *Speedy*

4.2.2. Compilazione del codice sorgente

Per compilare il programma *proxyMarker*, dalla directory principale dell'elaborato (`markerSpeedy`) è sufficiente eseguire il comando `make`, in quanto è già predisposto all'interno di tale cartella un *Makefile* che permette di compilare il programma *proxyMarker*. I file sorgenti si trovano nella cartella `src`, mentre i file eseguibili vengono messi nella cartella `bin`.

La versione del compilatore *C* utilizzata nell'elaborato è *gcc* 4.0.3.

Per compilare il programma *client* bisogna controllare che sia installato il compilatore *java* (sul calcolatore di *Speedy*, al momento della stesura del presente elaborato, non c'era) e lanciare il comando `javac client.java` dalla cartella `client`.

La versione del compilatore *java* utilizzata nell'elaborato è *javac* 1.5.0.

4.3. Configurazione della porta seriale

Per far funzionare correttamente l'invio e la ricezione dei comandi e delle risposte è necessario impostare alcuni parametri relativi alla porta seriale collegata al calcolatore tramite l'adattatore USB-RS232. A tal fine si utilizza da shell il comando:

```
stty 9600 –icanon cstopb –echo –F nome_dispositivo_seriale
```

Ø **È necessario verificare tramite il comando `stty –aF nome_dispositivo_seriale` che la porta seriale sia configurata come segue:**

```
speed 9600 baud; rows 0; columns 0; line = 0; intr = ^C; quit = ^\; erase
= ^?; kill = ^U; eof = ^D; eol = <undef>; eol2 = <undef>; swtch = <undef>;
start = ^Q; stop = ^S; susp = ^Z; rprnt = ^R; werase = ^W; lnext = ^V;
flush = ^O; min = 1; time = 0; -parenb -parodd cs8 hupcl cstopb cread
clocal –crtscts -ignbrk brkint ignpar -parmrk -inpck istrip -inlcr -igncr
icrnl ixon -ixoff -iuclc -ixany -imaxbel -iutf8 opost -olcuc -ocrnl onlcr
-onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0 vt0 ff0 isig -icanon iexten
-echo echoe echok -echonl -noflsh -xcase -tostop -echoprt echoctl echoke
```

Ø **Se per avviare *proxyMarker* si utilizza lo script descritto nel paragrafo seguente la configurazione del dispositivo viene fatta automaticamente dallo script stesso.**

4.4. Utilizzo del software

Per avviare *proxyMarker*, dopo aver configurato correttamente la porta seriale del calcolatore, dalla cartella `bin` si deve eseguire il comando

```
./proxyMarker nome_dispositivo_seriale [numero_porta]
```

Esempio: `./proxyMarker /dev/ttyUSB1 9000`

Al fine di avviare *proxyMarker* in background, è stato creato lo script *startMarker.sh* da avviare dalla cartella `bin` tramite il comando

```
./startMarker.sh nome_dispositivo_seriale [numero_porta] &
```

Per avviare il *client*, dalla cartella `client` eseguire il comando

```
java client [indirizzo_IP numero_porta] comando
```

I parametri opzionali `indirizzo_IP` e `numero_porta` vanno omessi o specificati entrambi.

In tutti i casi se il numero della porta non viene specificato, i programmi utilizzano come porta predefinita la 9000.

Ø **Se si decide di non utilizzare la porta predefinita bisogna indicare lo stesso numero di porta sia a *proxyMarker* (o allo script *startMarker.sh*) che al *client*.**

5. Conclusioni e sviluppi futuri

In questo elaborato è stata implementata un'applicazione che, fungendo da proxy, rende disponibili a client remoti le funzionalità del marker attivo installato sul robot *Speedy*.

Il client realizzato in linguaggio *java* è servito per testare il corretto funzionamento del proxy, ma può essere anche impiegato come componente per altre applicazioni che intendono servirsi del marker attivo per effettuare la localizzazione del robot *Speedy*. Un possibile sviluppo in tal senso potrebbe essere l'integrazione del client nel sistema di telecomando via applet di *Speedy*.

Appendice

Codice sorgente di *proxyMarker.c*

```
/* Autori: Simone Borda
           Guido Pollini
           Sara Sabbadini
   Data ultima modifica: 29/06/2007
*/

/* Questo file contiene il corpo principale del programma per
   controllare il marker attivo sul robot Speedy.
   All'avvio l'applicazione inizializza il marker riportandolo "a
   casa" (posizione 0), spegnendo i LED e disattivando la modalità
   "verbose".
   In seguito crea ed inizializza un socket, assegnandogli una porta
   e mettendolo in ascolto su tutte le interfacce di rete
   disponibili, in attesa di ricevere comandi per il marker.
*/


#include "auxMarker.c" //include le funzioni ausiliare e tutte le
librerie utilizzate

#define MAXBUF    1024  // lunghezza massima del segmento TCP inviato
#define Max       128   // lunghezza massima del buffer per la porta
seriale
#define CAR_COM_GEN 2  // numero di caratteri di un comando generico


int main(int Count, char *Strings[])
{
    int myPort = 9000; // numero della porta di ascolto predefinita


    /*---Controllo degli argomenti passati dalla riga di comando---*/
    if (Count<2)
    {
      perror("Specificare la porta seriale da usare passandola come
parametro dalla riga di comando.\n");
      exit(1);
    }
    if (Count==3)
      {
            myPort = atoi(Strings[2]);
            if(myPort > 65535)
            {
                  printf("Numero di porta non accettabile, il numero
della porta deve essere minore di 65535\n");
                  exit(1);
            }
        }


    /*---VARIABILI PORTA SERIALE---*/
    char buff[Max];
    int fd = open_port(Strings[1]);

    /*---VARIABLI SOCKET---*/
```

```
    int sockfd;
    struct sockaddr_in self;
    char buffer[MAXBUF];

    char chiusura[] = "stop\n";  // comando di chiusura server
    int n = 0; // contatore


   /* All'inizio si spengono i LED e la scheda driver del motore, poi
      si accende la scheda driver e si posiziona il motore in
      corrispondenza della fotocellula (home)
   */
   printf("\n-- Avvio procedura di inizializzazione marker --\n");
   goHome(n, fd, buff);
   printf("-- Inizializzazione del marker completata --\n\n");


   /*---Avvio del socket---*/
   printf("-- Inizializzazione del socket --\n");

   // Creazione del socket
   if ( (sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0 )
      {
            perror("Socket error");
            exit(errno);
      }

   // Inzializzazione socket
   bzero(&self, sizeof(self));
   self.sin_family = AF_INET;
   self.sin_port = htons(myPort);
   self.sin_addr.s_addr = INADDR_ANY;

   // Assegnazione di una porta al socket
   if ( bind(sockfd, (struct sockaddr*)&self, sizeof(self)) != 0 )
      {
            perror("socket--bind");
            exit(errno);
      }

   // Il socket viene messo in ascolto
   if ( listen(sockfd, 20) != 0 )
      {
            perror("socket--listen");
            exit(errno);
      }


   /*---Esecuzione comandi---*/
   while (1)
      {
            printf ("\nIn attesa di una connessione...\n");

            char  *risposta;  // variabile per leggere la risposta
inviata dal PIC
            char  vecchia_risposta[Max];   // variabile per risposta
spegnimento motore e led
            vecchia_risposta[0] ='\0';
            char risposte_pendenti[Max];  // variabile per controllare
eventuali risposte non ancora lette dalla seriale
            risposte_pendenti[0] = '\0';
```

```
            int clientfd;
            struct sockaddr_in client_addr;
            int addrlen=(int)sizeof(client_addr);
            clientfd         =         accept(sockfd,         (struct
sockaddr*)&client_addr,(socklen_t   *)   &addrlen);   //   accetta   la
connessione del client
            printf("--     %s:%d     connessione     avvenuta     --\n",
inet_ntoa(client_addr.sin_addr), ntohs(client_addr.sin_port));


            /*---Ricezione dei comandi---*/
            // Controllo di eventuali risposte rimaste sulla seriale
            memset(&buff, '\0', sizeof(buff));  // pulizia buffer
            leggiSeriale(fd, buff);
            strcpy(risposte_pendenti, buff);

            // Ricezione dei dati dal client
            int i;  // lunghezza del comando ricevuto
            if ((i = read(clientfd, buff, Max)) <= 0)
            {
                printf("Errore nella chiamata READ\n");
                close(clientfd);
                continue;
            }
            buff[i] = '\0';
            printf("> Dati ricevuti : %s\n", buff);


            /*---Controllo comandi---*/
            // Verifica se il client ha inviato il comando di chiusura
            if (strcmp(buff, chiusura)==0)
            {
              printf ("Il processo verrÃ  terminato...\n");

              /*---Invio delle risposte---*/
              memset(&buffer, 0, sizeof(buffer));

              //   Vengono   concatenate   le   risposte   ai   comandi
precedentemente inviati al PIC
              strcpy(vecchia_risposta,       strcat(risposte_pendenti,
vecchia_risposta));

              // Invio delle eventuali risposte ai comandi precedenti
              strcpy(buffer, strcat(vecchia_risposta, "\r"));
              send(clientfd, buffer, strlen(buffer), 0);
              printf ("> Invio risposte ai comandi precedenti: %s\n",
buffer);

              // Invio della risposta del PIC al comando ricevuto
              memset(&buffer, 0, sizeof(buffer));
              strcpy(buffer, "Proxy chiuso");
              send(clientfd, buffer, strlen(buffer), 0);
              printf ("> Invio risposta al comando corrente: %s\n",
buffer);

              /*---Procedura di terminazione---*/
              // Chiusura della connessione temporanea
              close(clientfd);
              printf ("> Richiesta eseguita\n");

              // Riposizionamento del marker in posizione iniziale
```

```
  memset(&buff, 0, sizeof(buff));
  printf("%s\n", buff);
  goHome(n, fd, buff);

  // Chiusura dei file descriptor
  close(sockfd);
  close(fd);
  return EXIT_SUCCESS;
  return 0;
}

// Verifica della ricezione di altri comandi
if (isLampeggio(buff))
{
            printf ("> Comando di lampeggio\n");
            scriviSeriale(fd, buff);
            leggiSeriale(fd,buff);
      risposta = buff;
      }
else if (isRotazione(buff))
{
            printf ("> Comando di rotazione\n");
            scriviSeriale(fd, buff);
            leggiSeriale(fd,buff);
      risposta = buff;
      }
else if (i==CAR_COM_GEN)
  {
      // Contatori
      int i = 0;
      int cont = 0;
      int cont1 = 0;

      switch (buff[0])
      {
      case 'd': printf("> Comando accensione led\n");
              scriviSeriale(fd, buff);
                    leggiSeriale(fd,buff);
              risposta=buff;
              break;
        case 'D': printf("> Comando spegnimento led\n");
              scriviSeriale(fd, buff);
                    leggiSeriale(fd,buff);
              risposta=buff;
                    break;
      case 'g': printf("> Comando ritorna posizione\n");
                scriviSeriale(fd, buff);
                leggiSeriale(fd,buff);

              for (i=0; i<strlen(buff); i++)
              {
                if (isdigit(buff[i]))
                {
                  risposta[cont] = buff[i];
                  cont++;
                }
                else
                {
                  vecchia_risposta[cont1] = buff[i];
                  cont1++;
                }
```

```
                        }
                        risposta[cont] = '\0';
                        vecchia_risposta[cont1] = '\0';
                          break;
                case 'l': printf("> Comando arresta motore\n");
                        scriviSeriale(fd, buff);
                              leggiSeriale(fd, buff);
                        risposta = buff;
                                break;
                case  'n':  printf(">  Comando  accensione  scheda
driver\n");
                        scriviSeriale(fd, buff);
                          leggiSeriale(fd, buff);
                        risposta = buff;
                          break;
                case  'N':  printf(">  Comando  spegnimento  scheda
driver\n");
                        scriviSeriale(fd, buff);
                              leggiSeriale(fd,buff);
                        risposta = buff;
                              break;
                case 'p': printf("> Comando torna a casa\n");
                        scriviSeriale(fd, buff);
                        leggiSeriale(fd, buff);
                        if (buff[0] == 'p')
                            *risposta = 'H';
                        else
                              risposta = buff;
                              break;
                case 's': printf("> Comando spegni tutto\n");
                        scriviSeriale(fd, buff);
                        leggiSeriale(fd, buff);
                        risposta = buff;
                              break;
                default: printf ("> Comando errato!\n");
                       strcpy (risposta, "nop");
                            break;
                     }
            }
            else
            {
                printf("> Comando errato!\n");
                strcpy (risposta, "nop");
            }

        printf("\n");


        /*---Invio delle risposte---*/
        memset(&buffer, 0, sizeof(buffer));

      // Vengono concatenate le risposte ai comandi precedentemente
inviati al PIC
      strcpy(vecchia_risposta,             strcat(risposte_pendenti,
vecchia_risposta));

      // Invio delle eventuali risposte ai comandi precedenti
      strcpy(buffer, strcat(vecchia_risposta, "\r"));
        send(clientfd, buffer, strlen(buffer), 0);
        printf  ("> Invio  risposte  ai  comandi  precedenti:  %s\n",
buffer);
```

```
       // Invio della risposta del PIC al comando ricevuto
       memset(&buffer, 0, sizeof(buffer));
         strcpy(buffer, risposta);
         send(clientfd, buffer, strlen(buffer), 0);
         printf ("> Invio risposta al comando corrente: %s\n",
buffer);


         /*---Chiusura della connessione---*/
         close(clientfd);
         printf ("> Richiesta eseguita\n");
    }
}
```

Codice sorgente di *auxMarker.c*

```
/* Autori: Simone Borda
         Guido Pollini
         Sara Sabbadini
   Data ultima modifica: 29/06/2007
*/

/* Questo file contiene alcune funzioni ausiliarie per il programma
   proxyMarker e include le librerie utilizzate.
*/


#include <stdio.h>
#include <iostream.h>
#include <string.h>
#include <stdlib.h>
#include <sys/socket.h>
#include <resolv.h>
#include <arpa/inet.h>
#include <sys/types.h>
#include <unistd.h>     /* UNIX standard function definitions */
#include <fcntl.h>      /* File control definitions */
#include <errno.h>      /* Error number definitions */
#include <termios.h>    /* POSIX terminal control definitions */
#include <time.h>       /* Libreria per la gestione delle attese */
#include  <sys/ioctl.h>   /*  Libreria  per  la  gestione  della  porta
seriale */

#define  CAR_LAMPEGGIO  12   //  numero  di  caratteri  del  comando  di
lampeggio
#define  CAR_ROTAZIONE  9    //  numero  di  caratteri  del  comando  di
rotazione



/*---Funzione per aprire la porta seriale---*/
int open_port(char *seriale)
    {
      int fd;  // file descriptor per la porta

      fd = open(seriale, O_RDWR | O_NOCTTY | O_NDELAY);
      if (fd == -1)
      {
      // impossobile aprire la porta
```

```
      perror("Impossibile aprire la porta seriale.\nVerificare la
correttezza del parametro inserito.\n");
      exit(1);
      }
      else
      fcntl(fd, F_SETFL, 0);

      return (fd);
    }


/*---Funzione per inviare i dati sulla seriale---*/
void scriviSeriale(int fd, char *buff)
{
      if(write(fd, buff, strlen(buff))<0)
            fputs("Scrittura fallita.\n", stderr);
}


/*---Funzione che legge i caratteri presenti sulla seriale---*/
void leggiSeriale(int fd, char *buff)
{
      int avaible =0;

      while(1)
      {
            sleep(1);

            if (ioctl(fd, FIONREAD, &avaible)<0) // controlla quanti
caratteri da leggere ci sono sulla seriale
                  perror("La funzione ioctl() ha generato un
errore\n");
            if (avaible == 0)
            {
                  break; // non ci sono caratteri da leggere
            }
            if (read(fd, buff, avaible)>0)
            {
                  buff[avaible]='\0';
                  return;
            }
      }
}


/*---Funzione che controlla se il comando ricevuto è quello di
lampeggio---*/
int isLampeggio(char *buff)
{
      if (strlen(buff) != CAR_LAMPEGGIO)
            return 0;
      char testa[3];
      testa[2] = '\0';
      strncpy(testa, buff, 2);
      if (strcmp(testa, "a1") != 0)
            return 0;
      for(int i=2; i<11; i++)
      {
            if (!isdigit(buff[i]))
                  return 0;
      }
```

```
	return 1;
}


/*---Funzione che controlla se il comando ricevuto è quello di rotazione---*/
int isRotazione(char *buff)
{
	if (strlen(buff) != CAR_ROTAZIONE)
		return 0;
	if (buff[0] != 'm')
		return 0;
	for(int i=1; i<4; i++)
	{
		if (!isdigit(buff[i]))
			return 0;
	}
	if (buff[4] != 'r' && buff[4] != 'f')
		return 0;
	for(int i=5; i<8; i++)
	{
		if (!isdigit(buff[i]))
			return 0;
	}
	return 1;
}


/*---Funzione che riporta a casa il marker spegnendo i LED
     Viene chiamata all'inizio ed alla fine dell'esecuzione di proxyMarker---*/

void goHome(int n, int fd, char *buff)
{
	char off[] = "VN"; // comando per spegnere la scheda driver del motore

	scriviSeriale(fd, off);
	leggiSeriale(fd,buff);

	char *ack_off = buff; // variabile per la risposta al comando VN

	if (strcmp(ack_off,"s")==0 or strcmp(ack_off, "Vs")==0)

		printf("> Spegnimento scheda driver [OK]\n");
	else
	{
		perror("> Spegnimento scheda driver [FAILED]\n");
		exit(1);
	}

	char home[] = "ps"; // comando per posizionare il motore in 0 e spegnere i LED

	scriviSeriale(fd, home);
	leggiSeriale(fd,buff);

	char *ack_home = buff; // variabile per risposta al comando ps

	if	(strcmp(ack_home,"abcm")==0	||	strcmp(ack_home, "SMmabcm")==0)
```

```
            printf("> Spegnimento LED [OK]\n> Marker in posizione 0
[OK]\n");
      else if (strcmp(ack_home, "SM")==0)
      {
            printf("> Spegnimento LED [OK]\n> Marker in posizione 0
[OK]\n");
            leggiSeriale(fd,buff);
            char *ack_home2 = buff;
            if (strcmp(ack_home2, "mabcm")!=0)
            {
                  perror("> Spegnimento LED [FAILED]\n> Marker in
posizione 0 [FAILED]\n");
                  exit(1);
            }
      }
      else
      {
            perror("> Spegnimento LED [FAILED]\n> Marker in posizione
0 [FAILED]\n");
            exit(1);
      }
}
```

Codice sorgente di *client.java*

```
/* Autori: Simone Borda
         Guido Pollini
         Sara Sabbadini
   Data ultima modifica: 29/06/2007
*/

/* Questo programma stabilisce una connessione con proxyMarker
   ed invia un comando per il marker.
   Ad ogni esecuzione dell'applicazione, viene instaurata una
   connessione con proxyMarker ed inviato un solo comando; in
   seguito sono attese due risposte: la prima è quella eventualmente
   pendente, la seconda è quella relativa al comando corrente.
   Una volta ricevute le risposte, l'applicazione chiude la
    connessione e si arresta.
*/


import java.io.*;
import java.net.*;

public class client
{
  public static void main(String[] args) throws java.io.IOException
  {
    String ip_address="192.0.2.11";
    int port = 9000;
    String command = "";

    try
    {
      if (args.length == 1) {
        if (args[0].equals("--help")) {
          System.out.println("Usage:");
          System.out.println("client [ip-address] [port] [command to
send]");
          System.out.println(
```

```
              "client  [command  to  send]    ,(Default  IP=192.0.2.11,
port=9000)");
          System.exit(1);
        }
        else { // Configurazione di default
          command = args[0];
          initClient(ip_address, 9000, command);
        }
      }

      else if (args.length == 3) {
        ip_address = args[0];
        port = Integer.parseInt(args[1]);
        command = args[2];
        initClient(ip_address, port, command);
      }
    }
    catch(Exception e)
        {
          System.out.println("Wrong parameters.");
          System.exit(1);
        }
  }
  /*Inizializzazione della connessione TCP ed invio del comando*/
  private static void initClient(String ip, int porta, String comando)
{
    Socket socket = null; //creo il socket
    PrintWriter out = null; //creo lo scrittore
    BufferedReader in = null; //creo il lettore

    try {
      socket  =  new  Socket(ip,  porta);  //inizializzo  il  socket  che
comunica col server
      out    =    new    PrintWriter(socket.getOutputStream(),    true);
//inizializzo lo scrittore sul socket
      in              =              new             BufferedReader(new
InputStreamReader(socket.getInputStream())); //inizializzo il lettore
sul socket
      System.out.println("Client started.");
      String datiDaInviare = "";
      String datiRimasugli = "";
      String datiRicevuti = "";
      BufferedReader      stdIn      =      new      BufferedReader(new
InputStreamReader(System.in));

      datiDaInviare = comando;
      out.println(datiDaInviare);
      System.out.println("Client sends: " + datiDaInviare);
      datiRimasugli = in.readLine();
      System.out.println("Client    reads    (last    command):"    +
datiRimasugli);
      datiRicevuti = in.readLine();
      System.out.println("Client reads: " + datiRicevuti);
      out.close(); //chiudo lo scrittore
      in.close(); //chiudo il lettore
      stdIn.close(); //chiudo il buffered reader
      socket.close(); //chiudo il socket
      System.out.println("Socket closed");
    }
    catch (UnknownHostException e) {
      System.err.println("Unknown host exception");
```

```
    }
    catch (IOException e) {
      System.err.println(
          "Server unreachable.");
    }

  }
}
```

Bibliografia

[1] Manzini, M.: “Sviluppo di un dispositivo hardware e relativo software per la gestione di un marker e di un motore a passo”, 2005 (http://www.ing.unibs.it/~cassinis/docs/projects/Mor_04.pdf)

[2] Piccardi, S.: “Guida alla Programmazione in Linux”, 2007 (http://gapil.truelite.it)

[3] Sweet, M.: “Serial Programming Guide for POSIX Operating Systems”, 2005 (http://www.easysw.com/~mike/serial/serial.html#2_3)

[4] Kernighan, B.W., Ritchie, D.M.: *Il linguaggio C – Principi di programmazione e manuale di riferimento*, 2° edizione, Pearson Prentice Hall, 2004

[5] Comer, D.: *Internetworking con TCP/IP volume1. Principi, protocolli e architetture*, 4° edizione, Pearson Education Italia, 2002

Indice

SOMMARIO **1**

1. INTRODUZIONE **1**

2. IL PROBLEMA AFFRONTATO **2**

3. LA SOLUZIONE ADOTTATA **2**
3.1. Test dell'hardware preesistente **2**
3.2. Test del software preesistente **3**
3.3. Sviluppo del software **4**
3.3.1. Descrizione del proxy 4
3.3.2. Descrizione del client 6

4. MODALITÀ OPERATIVE **7**
4.1. Componenti necessari **7**
4.2. Modalità di installazione **7**
4.2.1. Collegamento del marker al calcolatore di *Speedy* 8
4.2.2. Compilazione del codice sorgente 9
4.3. Configurazione della porta seriale **9**
4.4. Utilizzo del software **10**

5. CONCLUSIONI E SVILUPPI FUTURI **10**

APPENDICE **11**
Codice sorgente di *proxyMarker.c* **11**
Codice sorgente di *auxMarker.c* **16**
Codice sorgente di *client.java* **19**

BIBLIOGRAFIA **22**

INDICE **23**