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)

Interfacciamento laser scanner Hokuyo UHG-08LX

Elaborato di esame di:

Francesca Facchetti, Samuele Fausti, Lorenzo Marini

Consegnato il: **15 giugno 2009**

Sommario

Il progetto sviluppato durante il corso di Robotica Mobile ha riguardato l'interfacciamento del laser scanner UHG-08LX della Hokuyo al notebook Acer Aspire One e la realizzazione di un'applicazione per la visualizzazione grafica delle misure effettuate dal sensore.

1. Introduzione

La presente relazione descrive tutte le attività svolte per interfacciare il laser scanner UHG-08LX al notebook Acer Aspire One.

2. Il problema affrontato

Lo svolgimento del progetto comporta la risoluzione di due problemi principali:

- l'interfacciamento del laser scanner UHG-08LX con il notebook Acer Aspire One;
- la realizzazione di un'applicazione che legga le misure effettuate dal sensore e le mostri all'utente in una semplice interfaccia grafica.

3. La soluzione adottata

La soluzione adottata per lo svolgimento del lavoro si compone di due fasi principali:

- la preparazione dell'ambiente;
- la realizzazione dell'applicativo.

3.1. Preparazione dell'ambiente

In questa fase si sono reperiti e installati sul notebook tutti i pacchetti, librerie e moduli necessari al corretto svolgimento del progetto.

In particolar modo sono stati installati:

- Driver USB/seriale (CDC ACM): per il corretto interfacciamento tra il notebook e il sensore;
- Librerie URG: necessarie per la lettura delle misure dal sensore;
- Strumenti di sviluppo (gcc e make): necessari per la realizzazione dell'applicativo

3.2. Realizzazione dell'applicativo

La seguente fase prevede la realizzazione del programma per la lettura e visualizzazione delle misure.

Per questa operazione si utilizza il protocollo SCIP 2.0 per la lettura dei dati, tramite le librerie URG fornite dal produttore del sensore, mentre per l'interfaccia grafica sono utilizzate le librerie grafiche OpenGL e SDL (Simple DirectMedia Layer)

4. Preparazione dell'ambiente

4.1. Componenti necessari

Laser Scanner UHG-08LX

Il laser scanner della Hokuyo della serie UHG-08 ha le seguenti caratteristiche:

- intervallo di rilevamento: 30÷11000mm;
- angolo di rilevamento: 270°;
- angolo di risoluzione: 0.36°;
- tempo di scansione: 67msec/scansione;
- protocollo: SCIP 2.0;
- interfaccia: USB 2.0;
- alimentazione: 12V DC tramite alimentatore.

Notebook Acer Aspire One

Il notebook utilizzato ha le seguenti caratteristiche:

- Sistema Operativo: Linpus, distribuzione Linux basata su Fedora;
- Intel Atom 1.6 Ghz;
- Ram 512 MB;
- HD 8 GB;
- Monitor TFT widescreen (1024 x 600);
- dimensioni: 249x29x170 mm. Peso: 1 kg.

Librerie

Per lo svolgimento del presente progetto sono necessarie le seguenti librerie:

- SDL
- OpenGL
- Boost
- Ncurses
- URG

4.2. Modalità di installazione

4.2.1. Installazione ambiente di sviluppo

Aprire un terminale e installare il compilatore *gcc* e la utility *make* digitando:

```
$sudo yum install gcc make
```

4.2.2. Installazione driver CDC ACM

Aprire un terminale e installare le librerie per il menù grafico utilizzato nella ricompilazione del kernel eseguendo il seguente comando:

```
$sudo yum install ncurses-deve
```

Successivamente scaricare i sorgenti del kernel e dei moduli relativi digitando:

```
$wget ftp://guest@csdftp.acer.com.tw/Aspire One Linpus Linux/Aspire One
Source/linux-2.6.23.9.lw.zip
$unzip linux-2.6.23.9.lw.zip
```

```
$cd linux-2.6.23.9
```

Copiare il file config_lawson_080516v1 che contiene la configurazione di fabbrica dell'Aspire One tramite il seguente comando:

```
$cp config_lawson_080516v1 .config
```

Avviare la riconfigurazione del kernel digitando:

```
$make menuconfig
```

A questo punto, nella configurazione, selezionare la voce:

```
Device Driver -> USB Support -> USB Modem (CDC ACM)
```

e impostarne la compilazione come modulo premendo <M>.

Successivamente compilare i moduli digitando i seguenti comandi:

```
$ rm -fr include/asm
$ ln -sv include/asm-i386 include/asm
$ make modules
```

Infine è necessario installare il modulo compilato all'interno del kernel eseguendo i seguenti comandi:

```
$ sudo cp drivers/USB/class/cdc-acm.ko
/lib/modules/2.6.23.9lw/kernel/drivers/USB/class
$ sudo depmod -ae
```

Per verificare la corretta esecuzione delle operazioni descritte in precedenza collegare il dispositivo alla porta USB e digitare il comando:

```
$ dmesg | grep acm
```

Il sistema visualizzerà le informazioni riportate in Figura 1, che informano l'utente del corretto caricamento del modulo CDC-ACM e della corretta creazione di un device (a caratteri) denominato /dev/ttyACM0.

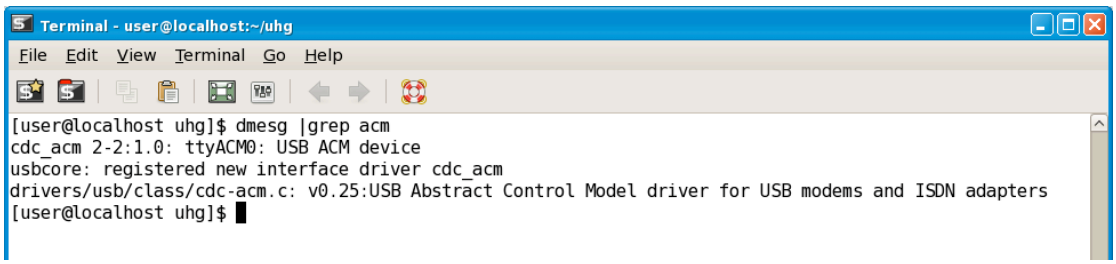

Fig. 1 - Messaggio di corretto riconoscimento del laser

4.2.3. Installazione librerie per lo sviluppo dell'applicativo

Il seguente comando installa tutte le librerie necessarie allo sviluppo dell'applicativo:

```
$ sudo yum install SDL-devel boost-devel boost
```

4.2.4. Installazione librerie URG

In questa fase vengono installate tutte le librerie necessarie alla corretta comunicazione tra l'applicativo e il laser scanner.

Per ottenere questo scaricare e decomprimere i file contenenti i sorgenti delle librerie URG digitando:

```
$ wget http://www.hokuyo-aut.jp/cgi-bin/urg_program_en/urg-0.0.2.zip
$ unzip urg-0.0.2.zip
$ cd urg-0.0.2
```

A questo punto è necessario compilare e installare le librerie scaricate al punto precedente tramite:

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

4.3. Realizzazione applicativo

L'applicazione realizzata è logicamente strutturata in tre fasi:

- *inizializzazione*: in cui viene inizializzato il contesto grafico e viene creata la connessione al sensore;
- *lettura dei dati*: nella quale si acquisiscono i dati dal sensore;
- *visualizzazione*: adibita alla visualizzazione grafica delle misure rilevate.

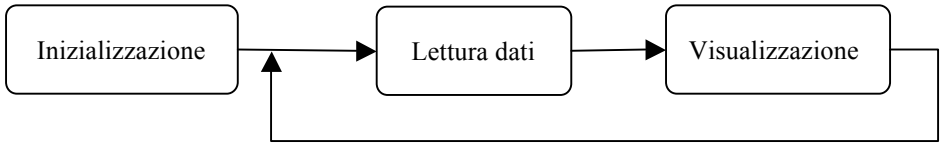

Fig. 2 - Schema a blocchi applicativo realizzato

Nella prima fase innanzitutto si inizializza, utilizzando le primitive fornite dalla libreria SDL, la modalità video.

```
// inizializzazione OpenGL
if (SDL_Init(SDL_INIT_VIDEO) < 0)
    {return 1;}

SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);

if (!(screen = SDL_SetVideoMode(WIN_SIZE, WIN_SIZE, 16, SDL_HWSURFACE | SDL_OPENGL)))
{
    fprintf(stderr, "Cannot SetVideoMode: %s\n", SDL_GetError());
    SDL_Quit();
    return 1;
}
```

Fig. 3 - Istruzioni inizializzazione SDL

Tramite la primitiva `SDL_Init()` si inizializzano le librerie SDL ed in particolare il sottosistema video (specificato dal flag SDL_INIT_VIDEO passato come parametro).

Successivamente utilizzando `SDL_GL_SetAttribute()`, viene abilitato il double buffering, ed infine con `SDL_SetVideoMode()`, viene inizializzata la modalità video specificando la larghezza e l'altezza della finestra, i bit per pixel e una serie di flags.

In questo caso i flag utilizzati sono:

- SDL_HWSURFACE: per creare una superficie video nella memoria video;
- SDL_OPENGL: per utilizzare OpenGL come rendering.

Per la creazione della connessione con il laser scanner si utilizzano le primitive fornite dalla libreria URG.

```
//connessione sensore

  ret = urg_connect(&urg, device, 115200);
  if (ret < 0) {
    printf("urg_connect: %s\n", urg_getError(&urg));
    SDL_Quit();
    return 1;
  }
  ret = urg_getParameters(&urg, &params);
if (ret < 0)
{
        urg_disconnect(&urg);
        SDL_Quit();
        return 1;
}

//calcola parametri: angoli e distanze min e max
min_area = params.area_min_;
max_area = params.area_max_;
min_length = urg_getDistanceMin(&urg);
max_length = urg_getDistanceMax(&urg);
min_angle = urg_index2deg(&urg, min_area) - 90;
max_angle = urg_index2deg(&urg, max_area) - 90;
```

Fig. 4 - Istruzioni connessione al laser e lettura parametri

La primitiva `urg_connect()` effettua la connessione al laser, a questa è necessario passare come parametri:

- il riferimento alla struttura dati che rappresenta il sensore URG;
- il nome del dispositivo (vale a dire la porta a cui è connesso il sensore, in questo caso ttyACM0);
- il baudrate (cioè la quantità di dati trasmessi al PC ogni secondo).

Successivamente tramite `urg_getParameters()` vengono memorizzati in opportune variabili i parametri del laser:

- min_length e max_length: distanza minima e massima rilevabili dal sensore;
- min_area e max_area: angolo minimo e massimo (espressi in rad) spazzati dal sensore, secondo il suo sistema di riferimento;
- min_angle e max_angle: angolo minimo e massimo spazzati dal sensore, secondo il sistema di riferimento di OpenGL. Tali angoli sono convertiti in gradi tramite `urg_index2deg()`.

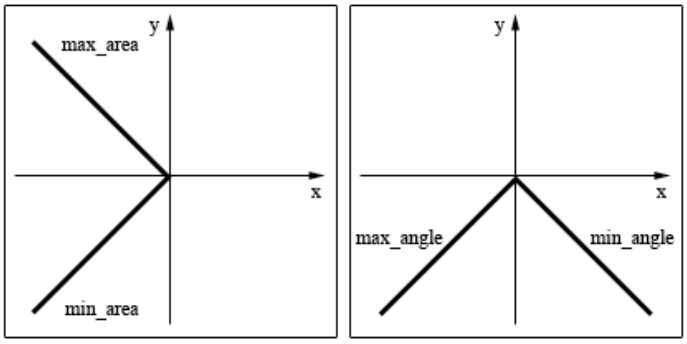

Fig. 5 - Angolo minimo e angolo massimo

```
//crea vettore dati
data_max = urg_getDataMax(&urg);
data = (long*)malloc(sizeof(long) * data_max);
if (data == NULL) {
  perror("data buffer");
  SDL_Quit();
  return 1;
}
```

Fig. 6 - Struttura dati per la memorizzazione delle misure

In questa fase viene inoltre allocato in memoria un vettore atto a contenere le misure effettuate dal sensore.

Infine sono presenti delle istruzioni che inizializzano il contesto di rendering.

```
//imposta sistema di riferimento per openGL
glClear( GL_COLOR_BUFFER_BIT );
glMatrixMode( GL_PROJECTION );
glLoadIdentity();
// NB -> glOrtho (bordo sx finestra, bordo dx, bordo inferiore, bordo superiore, -1.0, 1.0);
glOrtho(-max_length, max_length, max_length, -max_length, -1.0, 1.0);
glMatrixMode( GL_MODELVIEW );
glLoadIdentity();
```

Fig. 7 - Inizializzazione funzionalità di rendering

La primitiva `glClear()` inizializza il buffer passato come parametro, in questo caso il buffer del colore, mentre `glMatrixMode()` imposta la matrice di proiezione.

`glOrtho()` invece permette di fissare il sistema di riferimento per la visualizzazione, impostando il valore delle seguenti coordinate:

- ascissa del bordo sinistro;
- ascissa del bordo destro;
- ordinata del bordo inferiore;
- ordinata del bordo superiore;

Nel caso specifico l'origine del sistema di riferimento, il punto di coordinate (0,0), si troverà al centro della finestra.

La fase di Lettura dei dati si compone essenzialmente di due chiamate a primitive fornite da URG.

```
//Richiesta dati a sensore
ret = urg_requestData(&urg, URG_GD, URG_FIRST, URG_LAST);
if (ret < 0) {urg_exit(&urg, "urg_requestData()");}

//Ricezione dati da sensore
n = urg_receiveData(&urg, data, data_max);
if (n < 0) {urg_exit(&urg, "urg_receiveData()");}
```

Fig. 8 - Lettura dati dal laser

La primitiva `urg_requestData()` permette la richiesta dei dati al sensore, mentre `urg_receiveData()` permette la lettura vera e propria delle misure, le quali vengono memorizzate nel vettore data creato precedentemente.

La fase denominata Visualizzazione si occupa di mostrare graficamente le misure rilevate dal sensore.

L'utente ha la possibilità di modificare la visualizzazione delle misure premendo un carattere da tastiera. In particolare sono previste tre possibilità di modifica della modalità di visualizzazione:

- premendo C si passa alternativamente dalla visualizzazione a colori a quella in bianco e nero. I colori utilizzati sono quattro: rosso per le distanze al di sotto di un certo valore considerato "di

pericolo" (memorizzato nella costante D_EMERGENCY), arancione per le distanze comprese tra la distanza "di pericolo" e una considerata "di attenzione" (memorizzata nella costante D_CAUTION), verde per le altre distanze, azzurro per le misure fuori scala (cioè oltre la distanza massima rilevabile dal sensore);

- premendo V si modifica lo stile di visualizzazione, mostrando solo il contorno degli ostacoli rilevati oppure colorando l'area libera di fronte al sensore;
- premendo G si modifica la rappresentazione delle linee di livello, sempre distanziate di 1m, disegnando una griglia oppure dei cerchi concentrici.

A seconda del carattere premuto dall'utente, vengono eseguite le seguenti funzioni implementate:

- `renderAreaColor()` e `renderAreaBW()` per la visualizzazione tramite poligoni, a colori o in bianco e nero;
- `renderPointsColor()` e `renderPointsBW()` per la visualizzazione puntiforme dei contorni, a colori o in bianco e nero;
- `renderGrid()` e `renderCircles()` per la visualizzazione dei riferimenti con una griglia o con delle curve concentriche.

Infine la funzione `renderLayout()` permette di mostrare a video il contorno dell'area in cui vengono effettuate le misure.

4.4. Compilazione ed esecuzione

Per compilare l'applicazione è sufficiente aprire un terminale e digitare il comando:

```
$ make
```

Mentre per l'esecuzione, la quale è possibile anche senza i privilegi di superuser, basta digitare il comando

```
$ ./uhg
```

4.5. Modalità di visualizzazione

In questo paragrafo sono riportati degli screen shot che mostrano le varie modalità di visualizzazione delle misure implementate e descritte nel paragrafo 4.3**Errore. L'origine riferimento non è stata trovata.**.

4.5.1. Visualizzazione con aree a colori

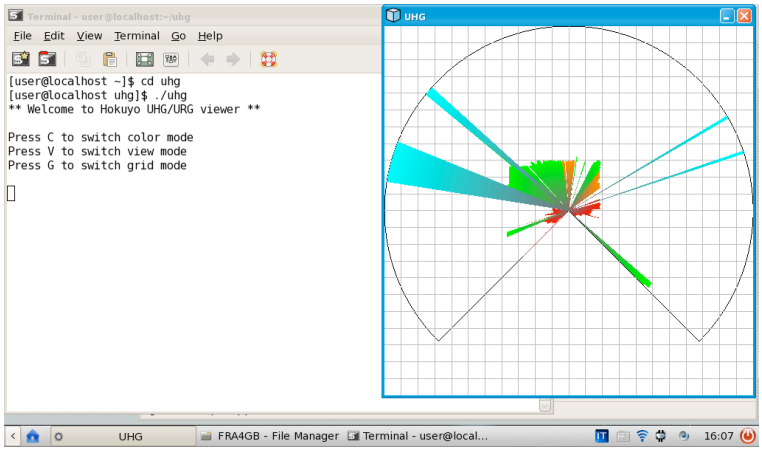

Fig. 9 - Visualizzazione misure: aree a colori

4.5.2. Visualizzazione con aree in bianco e nero

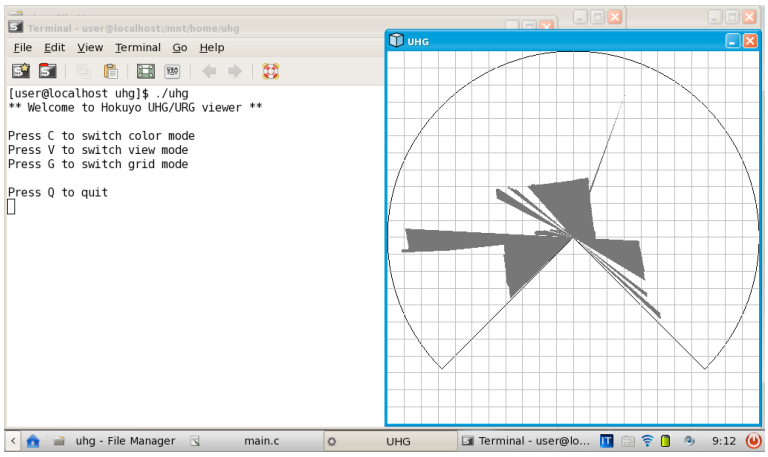

Fig. 10 - Visualizzazione misure: aree in bianco e nero

4.5.3. Visualizzazione con punti a colori

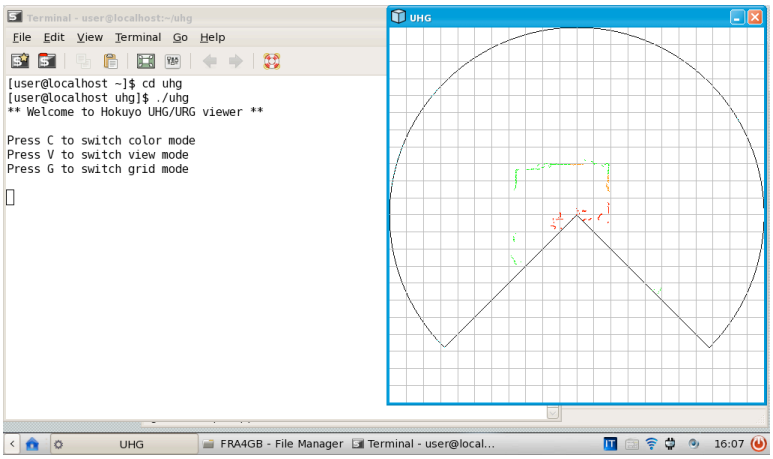

Fig. 11 - Visualizzazione misure: punti a colori

4.5.4. Visualizzazione con punti in bianco e nero

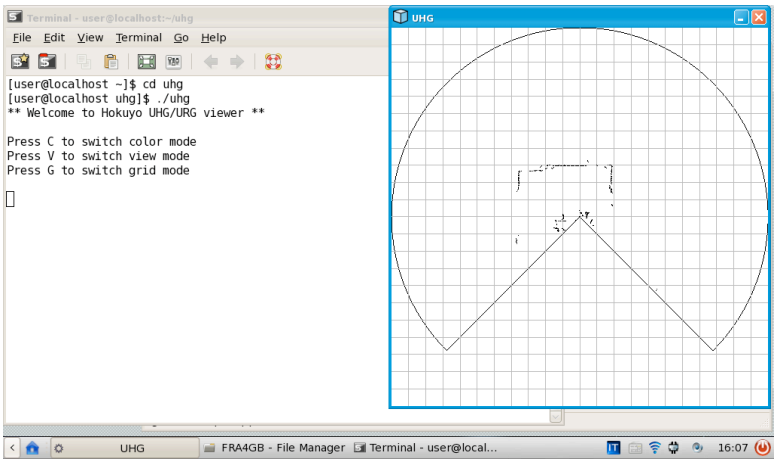

Fig. 12 - Visualizzazione misure: punti in bianco e nero

4.5.5. Visualizzazione del riferimento con curve concentriche

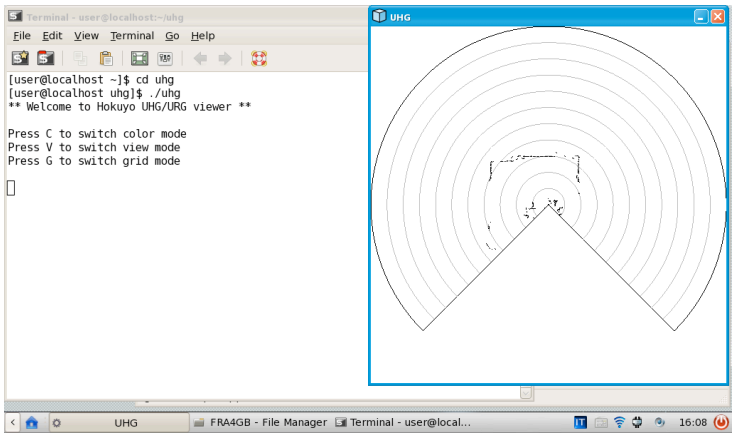

Fig. 13 - Visualizzazione misure: riferimenti a curve concentriche

5. Conclusioni e sviluppi futuri

Sono stati raggiunti tutti gli obiettivi prefissati, in particolare il sensore viene riconosciuto correttamente dal notebook e l'applicativo realizzato mostra correttamente le misure rilevate dal laser.

In aggiunta l'utente può modificare a suo piacimento la modalità di visualizzazione delle informazioni.

Possibili sviluppi futuri riguardano l'arricchimento dell'interfaccia grafica, per esempio tramite:

- l'aggiunta di bottoni per la modifica della modalità di visualizzazione;
- l'aggiunta di campi di testo per la modifica in tempo reale dei parametri di esecuzione (per esempio la distanza di sicurezza);
- aggiungere la possibilità di effettuare uno zoom sulla parte centrale della finestra, che rappresenta le distanze più vicine al sensore.

Bibliografia

[1] Kamimura, S.: "URG Programming Guide", Sito web Hokuyo (http://www.hokuyo-aut.jp/cgi-bin/urg_programs_en/index.html).

[2] Simple DirectMedia Layer, Sito web SDL (http://www.libsdl.org/).

[3] Ferri, Ghidini: "Interfacciamento sensore Hokuyo URG-04", Sito web (http://www.ing.unibs.it/~arl/docs/projects/Sen_05)

Appendice A: Codice sorgente applicativo realizzato

```
/*****************************************************************************
*   Interfacciamento sensore UHG-08LX
*
*   Copyright 2009 Francesca Facchetti, Samuele Fausti, Lorenzo Marini
*
*   This program is free software: you can redistribute it and/or modify
*   it under the terms of the GNU General Public License as published by
*   the Free Software Foundation, either version 3 of the License, or
*   (at your option) any later version.
*
*   This program is distributed in the hope that it will be useful,
*   but WITHOUT ANY WARRANTY; without even the implied warranty of
*   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
*   GNU General Public License for more details.
*
*   You should have received a copy of the GNU General Public License
*   along with this program.  If not, see <http://www.gnu.org/licenses/>.
*
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "urg_ctrl.h"
#include <SDL.h>
#include <GL/gl.h>

#define WIN_SIZE            500
#define PI                  3.141592653589
#define C_COLOR             1
#define C_BW                0
#define V_AREA              1
#define V_POINTS            0
#define G_GRID              1
#define G_CIRCLES           0
#define D_EMERGENCY         2000
#define D_CAUTION           3000


  long *data = NULL;
  urg_t urg;
  urg_parameter_t params;
  int min_length = 0;
  int max_length = 0;
  float min_angle = 0;
  float max_angle = 0;
  int min_area = 0;
  int max_area = 0;

//errore durante la comunicazione con il sensore
static int urg_exit(urg_t *urg, const char *message) {
  printf("%s: %s\n", message, urg_getError(urg));
  urg_disconnect(urg);
  SDL_Quit();
  return 1;
}

//visualizzazione modalita Area a colori
void renderAreaColor(SDL_Surface *screen){
    int i = 0;
    //punti distanze misurate
        glBegin(GL_POLYGON);
        glColor3f(0.4f, 0.4f, 0.4f);
         glVertex2f(0.0f, 0.0f);
        for (i = min_area; i <= max_area; i++)
          {
            long dist = data[i];
```

```
            //imposta colore punti
            if (dist == 0)
             {
               dist = max_length;
               //fuori scala: BLU
               glColor3f(0.0f, 1.0f, 1.0f);
             }
            else if (dist<D_EMERGENCY)
             {
               //molto vicino: ROSSO
               glColor3f(1.0f, 0.0f, 0.0f);
             }
            else if ((dist<D_CAUTION)&&(dist>=D_EMERGENCY))
             {
               //mediamente vicino: ARANCIONE
               glColor3f(1.0f, 0.5f, 0.0f);
             }
             else
             {
               //lontano: VERDE
               glColor3f(0.0f, 1.0f, 0.0f);
             }
            float angle = (float)(-(urg_index2deg(&urg, i) + 90) / 180.0 * PI);

            glVertex2f(dist * cos(angle), dist * sin(angle));

          }
         glColor3f(0.4f, 0.4f, 0.4f);
         glVertex2f(0.0f, 0.0f);
         glEnd();
   SDL_GL_SwapBuffers();
}

//visualizzazione modalità area in bianco e nero
void renderAreaBW(SDL_Surface *screen){
     int i = 0;
    //punti distanze misurate
        glBegin(GL_POLYGON);
        glColor3f(0.4f, 0.4f, 0.4f);
         glVertex2f(0.0f, 0.0f);
        for (i = min_area; i <= max_area; i++)
          {
            long dist = data[i];
            //imposta colore punti
            glColor3f(0.4f, 0.4f, 0.4f);
            if (dist == 0)
             {
               dist = max_length;
               //fuori scala: BIANCO
               glVertex2f(0.0f, 0.0f);
             }
             else
             {
               //valore valido: GRIGIO
               float angle = (float)(-(urg_index2deg(&urg, i) + 90) / 180.0 *
PI);

            glVertex2f(dist * cos(angle), dist * sin(angle));
             }

          }
         glColor3f(0.4f, 0.4f, 0.4f);
         glVertex2f(0.0f, 0.0f);
         glEnd();
   SDL_GL_SwapBuffers();
}

//visualizzazione modalità punti a colori
void renderPointsColor(SDL_Surface *screen){
    int i = 0;
    //punti distanze misurate
        glBegin(GL_POINTS);
        glColor3f(0.4f, 0.4f, 0.4f);
```

```
          glVertex2f(0.0f, 0.0f);
         for (i = min_area; i <= max_area; i++)
           {
             long dist = data[i];
             //imposta colore punti
             if (dist == 0)
              {
                dist = max_length;
                //fuori scala: BLU
                glColor3f(0.0f, 1.0f, 1.0f);
              }
             else if (dist<D_EMERGENCY)
              {
                //molto vicino: ROSSO
                glColor3f(1.0f, 0.0f, 0.0f);
              }
             else if ((dist<D_CAUTION)&&(dist>=D_EMERGENCY))
              {
                //mediamente vicino: ARANCIONE
                glColor3f(1.0f, 0.5f, 0.0f);
              }
              else
              {
                //lontano: VERDE
                glColor3f(0.0f, 1.0f, 0.0f);
              }
             float angle = (float)(-(urg_index2deg(&urg, i) + 90) / 180.0 * PI);

             glVertex2f(dist * cos(angle), dist * sin(angle));

           }
          glEnd();
   SDL_GL_SwapBuffers();
}

//visualizzazione modalità punti in bianco e nero
void renderPointsBW(SDL_Surface *screen){
int i = 0;
    //punti distanze misurate
        glBegin(GL_POINTS);
        glColor3f(0.4f, 0.4f, 0.4f);
         glVertex2f(0.0f, 0.0f);
        for (i = min_area; i <= max_area; i++)
          {
            long dist = data[i];
            //imposta colore punti: NERO
            glColor3f(0.0f, 0.0f, 0.0f);

            float angle = (float)(-(urg_index2deg(&urg, i) + 90) / 180.0 * PI);

            glVertex2f(dist * cos(angle), dist * sin(angle));

           }
          glEnd();
   SDL_GL_SwapBuffers();

}

//disegna la griglia
void renderGrid(SDL_Surface *screen){
         //colore grigio
         glColor3f(0.7f, 0.7f, 0.7f);    int c;
         glBegin(GL_LINES);
         for (c=0;c<max_length; c+=1000)
          {
              glVertex2f(c, max_length);
              glVertex2f(c, -max_length);
              glVertex2f(-c, max_length);
              glVertex2f(-c, -max_length);
              glVertex2f(max_length, c);
              glVertex2f(-max_length, c);
              glVertex2f(max_length, -c);
              glVertex2f(-max_length, -c);
```

```
          }

         glEnd();

}

//disegna i cerchi di livello
void renderCircles(SDL_Surface *screen){
        //colore grigio
        glColor3f(0.7f, 0.7f, 0.7f);
        int i;
        int c;
        for (c = 1000; c < max_length; c += 1000)
         {
           glBegin(GL_LINE_STRIP);
           for (i = min_angle; i <= max_angle; i++)
               glVertex2f(cos(i / 180.0 * PI) * c, sin(i / 180.0 * PI) * c);
           glEnd();
         }
}

//disegna il contorno dell'area in cui si effettuano misure
void renderLayout(SDL_Surface *screen){
        int i;
        //colore nero
        glColor3f(0.0f, 0.0f, 0.0f);
        //cerchio distanza massima
        glBegin(GL_LINE_STRIP);
           glVertex2f(0.0f, 0.0f);
           for (i = min_angle; i <= max_angle; i++)
             glVertex2f(cos(i / 180.0 * PI) * max_length, sin(i / 180.0 * PI) *
max_length);
           glVertex2f(0.0f, 0.0f);
        glEnd();
        //cerchio distanza minima
        glBegin(GL_LINE_STRIP);
           glVertex2f(0.0f, 0.0f);
           for (i = min_angle; i <= max_angle; i++)
             glVertex2f(cos(i / 180.0 * PI) * min_length, sin(i / 180.0 * PI) *
min_length);
           glVertex2f(0.0f, 0.0f);
        glEnd();
}

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

  const char device[] = "/dev/ttyACM0";
  int data_max;
  int ret;
  int n;
  int mod_color = C_BW;
  int mod_view = V_AREA;
  int mod_grid = G_GRID;
  SDL_Surface *screen;
  SDL_Event event;

  // inizializzazione OpenGL
  if (SDL_Init(SDL_INIT_VIDEO) < 0)
         {return 1;}

  SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);

  if (!(screen = SDL_SetVideoMode(WIN_SIZE, WIN_SIZE, 16, SDL_HWSURFACE |
SDL_OPENGL)))
  {
         fprintf(stderr, "Cannot SetVideoMode: %s\n", SDL_GetError());
         SDL_Quit();
         return 1;
  }

  SDL_WM_SetCaption("UHG", "UHG");
  glEnable( GL_TEXTURE_2D );
```

```
        //colore sfondo -> BIANCO
        glClearColor( 1.0f, 1.0f, 1.0f, 0.0f );

        //connessione sensore

     ret = urg_connect(&urg, device, 115200);
     if (ret < 0) {
       printf("urg_connect: %s\n", urg_getError(&urg));
       SDL_Quit();
       return 1;
     }
     ret = urg_getParameters(&urg, &params);
    if (ret < 0)
    {
                urg_disconnect(&urg);
                SDL_Quit();
                return 1;
    }

   //calcola parametri: angoli e distanze min e max
   min_area = params.area_min_;
   max_area = params.area_max_;
   min_length = urg_getDistanceMin(&urg);
   max_length = urg_getDistanceMax(&urg);
   min_angle = urg_index2deg(&urg, min_area) - 90;
   max_angle = urg_index2deg(&urg, max_area) - 90;

   //imposta sistema di riferimento per openGL
   glClear( GL_COLOR_BUFFER_BIT );
   glMatrixMode( GL_PROJECTION );
   glLoadIdentity();
   // NB -> glOrtho (bordo sx finestra, bordo dx, bordo inferiore, bordo
superiore, -1.0, 1.0);
   glOrtho(-max_length, max_length, max_length, -max_length, -1.0, 1.0);
   glMatrixMode( GL_MODELVIEW );
   glLoadIdentity();

   //crea vettore dati distanze
   data_max = urg_getDataMax(&urg);
   data = (long*)malloc(sizeof(long) * data_max);
   if (data == NULL) {
     perror("data buffer");
     SDL_Quit();
     return 1;
   }

   printf("** Welcome to Hokuyo UHG/URG viewer **\n\n");
   printf("Press C to switch color mode\n");
   printf("Press V to switch view mode\n");
   printf("Press G to switch grid mode\n\n");
   printf("Press Q to quit\n");

   int done = 0;
   while (!done)
   {
          while (SDL_PollEvent(&event))
          {
                 if (event.type == SDL_QUIT)
                        {done = 1;}
                 if (event.type == SDL_KEYDOWN )
                        {       //gestione evento tastiera
                                switch(event.key.keysym.sym)
                                {
                                    case SDLK_c: mod_color = !mod_color;
                                                 break;
                                    case SDLK_v: mod_view = !mod_view;
                                                 break;
                                    case SDLK_g: mod_grid = !mod_grid;
                                                 break;
                                    case SDLK_q: done = 1;
                                                 break;
                                    default:     break;
                                }
```

```
                        }
                {
                 if(event.key.keysym.sym == SDLK_ESCAPE )
                 done = 1;
                }
         }
        //Richiesta dati a sensore
        ret = urg_requestData(&urg, URG_GD, URG_FIRST, URG_LAST);
        if (ret < 0) {urg_exit(&urg, "urg_requestData()");}

        //Ricezione dati da sensore
        n = urg_receiveData(&urg, data, data_max);
        if (n < 0) {urg_exit(&urg, "urg_receiveData()");}

  //cancella lo schermo prima di ridisegnarlo
        glClear(GL_COLOR_BUFFER_BIT);

        renderLayout(screen);

        if (mod_grid == G_GRID){
            renderGrid(screen);
        } else {
            renderCircles(screen);
        }

         if (mod_view == V_AREA){
            if (mod_color == C_COLOR){
              renderAreaColor(screen);
              }
            else {
              renderAreaBW(screen);
            }
         }
         else {
            if (mod_color == C_COLOR){
              renderPointsColor(screen);
              }
            else {
              renderPointsBW(screen);
            }
         }


  }
  urg_disconnect(&urg);
  free(data);
  SDL_Quit();
  return 0;
}
```

Appendice B: Makefile

```
URG = /home/user/urg-0.0.2
CPP = gcc
COMPILE = ${CPP}
CFLAGS = -Wall -g -c `sdl-config --cflags` -I/usr/local/include/urg/ -
I${URG}/src/connection/c/
LIBS = `sdl-config --libs` -lGL -lc_connection -lc_urg
LINK = ${CPP}
LDFLAGS = -Wall -g ${LIBS} -Wl,--rpath -Wl,/usr/local/lib

all: uhg

uhg: main.o
  $(LINK) $(LDFLAGS) -o uhg main.o

main.o: main.c
  $(COMPILE) $(CFLAGS) main.c

clean:
  rm uhg *.o
```

Indice