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 sensore Hokuyo URG-04

Elaborato di esame di:

Diego Ferri, Andrea Ghidini

Consegnato il: **30 maggio 2009**

Sommario

Con questo lavoro si è interfacciato il laser scanner URG-04 di Hokuyo, previa installazione e configurazione di un adeguato ambiente, con il netbook Acer Aspire One. In seguito si è creata un'applicazione che visualizza in maniera grafica i dati provenienti dal sensore.

1. Introduzione

Questa relazione descrive le attività svolte per la riuscita del progetto assegnato durante il corso di Robotica Mobile e nasce dall'esigenza di utilizzo dei nuovi sensori laser scanner in dotazione all'ARL in accoppiata con un calcolatore.

2. Il problema affrontato

Il primo problema da affrontare riguarda l'interfacciamento del sensore laser scanner URG-04 prodotto da Hokuyo con un netbook Acer Aspire One. Infatti Hokuyo fornisce un'interfaccia USB non utilizzabile direttamente senza l'ausilio di un emulatore di interfaccia seriale su porta USB.

In seguito si vuole sviluppare un'applicazione in grado di leggere i dati del sensore e mostrarli graficamente (URGUI). Questo obiettivo si divide nel reperimento di librerie che incapsulino i comandi del protocollo utilizzato dal sensore e dell'individuazione di un ambiente grafico adatto a rappresentare i dati.

Infine l'obiettivo finale sarà la scrittura vera e propria dell'applicazione.

3. La soluzione adottata

3.1. Preparazione dell'ambiente

Prima di iniziare lo sviluppo dell'applicazione è necessario assicurarsi che siano presenti tutti i tool necessari. In particolare tutti i programmi che saranno poi richiesti dalla compilazione e dall'esecuzione di URGUI sono:

- strumenti di sviluppo (gcc, make);
- driver USB/seriale (CDC ACM);
- librerie URG, fornite da Hokuyo.

3.2. Sviluppo di URGUI

L'applicazione vuol essere in grado di leggere i dati forniti dal sensore e disegnarli in tempo reale. Per la lettura dei dati il sensore implementa due protocolli differenti, SCIP 1.1 e 2.0. Il protocollo 2.0 è stata la scelta più ovvia poichè fornisce una precisione di misurazione maggiore.

Per soddisfare il requisito di rappresentazione grafica in tempo reale si è utilizzata l'accelerazione fornita dalla scheda grafica in quanto il processore non è in grado di reggere il carico di lavoro necessario. Si sono quindi utilizzate le librerie grafiche OpenGL. Inoltre per rendere l'applicazione cross-platform, su

qualunque piattaforma supporti OpenGL, si è astratta la creazione del contesto grafico con l'utilizzo delle librerie SDL (Simple DirectMedia Layer).

I dati vengono rappresentati su un diagramma radiale vista dall'alto e centrato sul sensore stesso.

L'applicazione è stata sviluppata nel linguaggio C, per omogeneità con l'ambiente URG/OpenGL/SDL che espongono tutti interfacce native in C. Inoltre il compilatore gcc è standard *de facto* dell'ambiente Linux.

4. Modalità operative

4.1. Componenti necessari

4.1.1. URG-04

Il laser scanner ha le seguenti specifiche:

- intervallo di rilevamento: 20~5600mm;
- angolo di rilevamento: 240°;
- angolo di risoluzione: 0.36°;
- tempo di scansione: 100ms/scansione;
- protocolli: SCIP 1.1/2.0;
- interfaccia USB 2.0;
- alimentazione a 5V tramite interfaccia USB.

4.1.2. Acer Aspire One

Il calcolatore in dotazione, utilizzato per il progetto è un netbook Acer Aspire One, modello Z5G. Sono riportate di seguito le specifiche di interesse per il progetto:

- sistema operativo Linux Linpus (derivato di Fedora);
- processore Atom 1,6Ghz;
- chipset N270;
- 1Gb di ram;
- ssd da 8Gb;
- 3 connettori USB.

4.1.3. Librerie necessarie

- SDL e OpenGL;
- Boost;
- Ncurses;
- URG.

4.2. Modalità di installazione

4.2.1. Strumenti di sviluppo

Aprire un terminale.

```
$ sudo yum install gcc-c++ make
```

4.2.2. Driver CDC ACM

Aprire un terminale, installando le librerie ncurses-devel che servono per il menu grafico utilizzato per la ricompilazione del kernel.

```
$ sudo yum install ncurses-devel
```

Scaricare ora i sorgenti del kernel e dei moduli relativi al kernel già installato sull'Aspire One.

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

Avviare la riconfigurazione del kernel dopo aver copiato il file config_lawson_080516v1 contenente la configurazione di fabbrica dell'Aspire One.

```
$ cp config_lawson_080516v1 .config
$ make menuconfig
```

Nella configurazione, selezionare la voce e impostarne la compilazione come modulo, premendo <M>.

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

Compilare i moduli rimuovendo prima la cartella indicata.

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

> ➢ **Per motivi che non si è stati in grado di isolare, su alcuni calcolatori si è reso necessario l'utilizzo dei privilegi di superuser prima del comando precedente.**

Infine per "installare" il modulo compilato all'interno del kernel corrente.

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

Ora in seguito all'inserimento del connettore dati USB, il modulo cdc-acm verrà caricato automaticamente e verrà creato il device /dev/ttyACM0.

4.2.3. Librerie URG

Installare i pacchetti richiesti per la compilazione delle librerie.

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

Scaricare e scompattare il file contenente i sorgenti delle librerie URG.

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

Compilare ed installare le librerie.

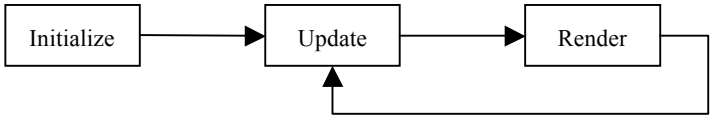
```
$ ./configure
$ make
$ sudo make install
```

4.2.4. URGUI

Il codice di URGUI scritto e il relativo *makefile* sono riportati in appendice. Di seguito si riportano le scelte progettuali più significative.

Per la compilazione di URGUI è stato necessario scrivere a mano il makefile in modo da includere tutte le librerie: SDL, OpenGL e URG.

L'applicazione esegue questo flusso finchè non viene interrotta dall'utente.



Si descrive di seguito brevemente l'utilità delle singole fasi:

- **Initialize**: inizializzazione del contesto grafico OpenGL tramite SDL e creazione della connessione con il sensore;
- **Update**: lettura dei dati dal sensore tramite API URG;
- **Render**: trasformazione dei dati in coordinate cartesiane e disegno delle stesse tramite primitive grafiche OpenGL.

4.2.4.1 Initialize

Si utilizza SDL per richiedere il contesto grafico, con la creazione della finestra impostando risoluzione, profondità e accelerazione grafica hardware OpenGL.

In seguito si inizializza la connessione con il sensore che viene interrogato a riguardo dei propri parametri di funzionamento (angolo, distanza massima, distanza minima, numero di punti).

OpenGL è una libreria grafica che permette il disegno accelerato di primitive grafiche 3D. OpenGL utilizza coordinate omogenee (composte dalle coordinate x, y, z e w) per rappresentare tutti i punti nello spazio, in modo da consentire così tutte le traslazioni e proiezioni possibili utilizzando semplicemente una matrice di proiezione. Siccome le primitive utilizzate da URGUI saranno semplicemente 2D, è necessario inibire le funzionalità 3D ed impostare un sistema di coordinate grafico 2D. Per fare ciò si crea una matrice di proiezione ortogonale, che permette di mappare le coordinate 2D utilizzate da URGUI (espresse in millimetri) nei pixel della superficie grafica. Inoltre si dimensiona il sistema di coordinate in relazione alla distanza massima rilevabile dal sensore, in modo da utilizzare al meglio lo spazio a disposizione.

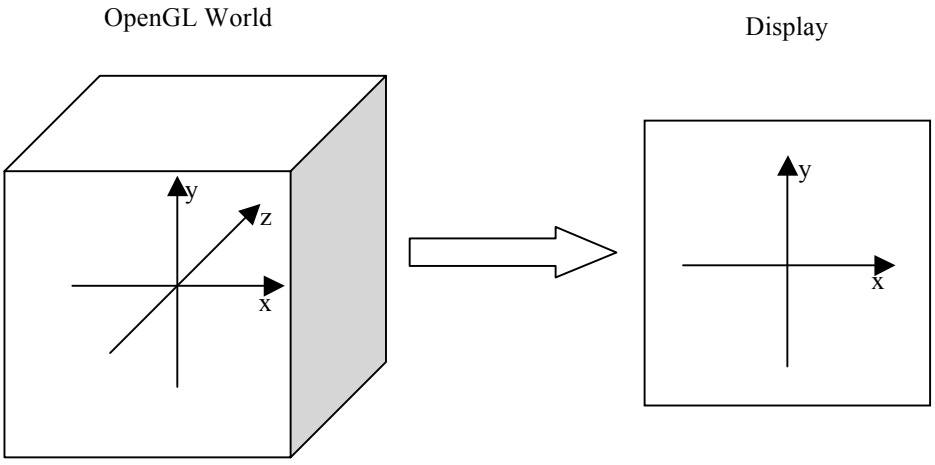


Fig. 1 - Trasformazione coordinate.

4.2.4.2 Update

Si richiedono e ricevono i dati al sensore attraverso le librerie URG. I dati vengono ricevuti all'interno di un array di tante misure quanti sono i punti comunicati precedentemente. Il codice C che permette di ottenere i dati dal sensore utilizza due funzioni fornite dalla libreria URG:

```
int res = urg_requestData(&urg, URG_GD, URG_FIRST, URG_LAST);
int n = urg_receiveData(&urg, data, data_max);
```

La prima chiamata effettua la richiesta al sensore dei dati rilevati da una lettura completa e ritorna un valore minore di zero in caso di errore. La seconda chiamata invece trasferisce i valori letti dal sensore all'interno dell'array `data` e ritorna il numero di valori effettivamente letti. L'array data è semplicemente

un array di `long` che contiene le distanze rilevate in mm. Per trasformare l'indice dell'array nell'angolo corrispondente si utilizza la funzione

```
urg_index2deg(&urg, index)
```

che trasforma `index` nel suo equivalente in gradi. Nel caso la misura sia fuori scala (oggetto troppo distante oppure oggetto riflettente il raggio) il valore contenuto nell'array sarà 0. Questo valore è distinguibile da qualunque altro valore valido perché sicuramente minore della distanza minima rilevabile.

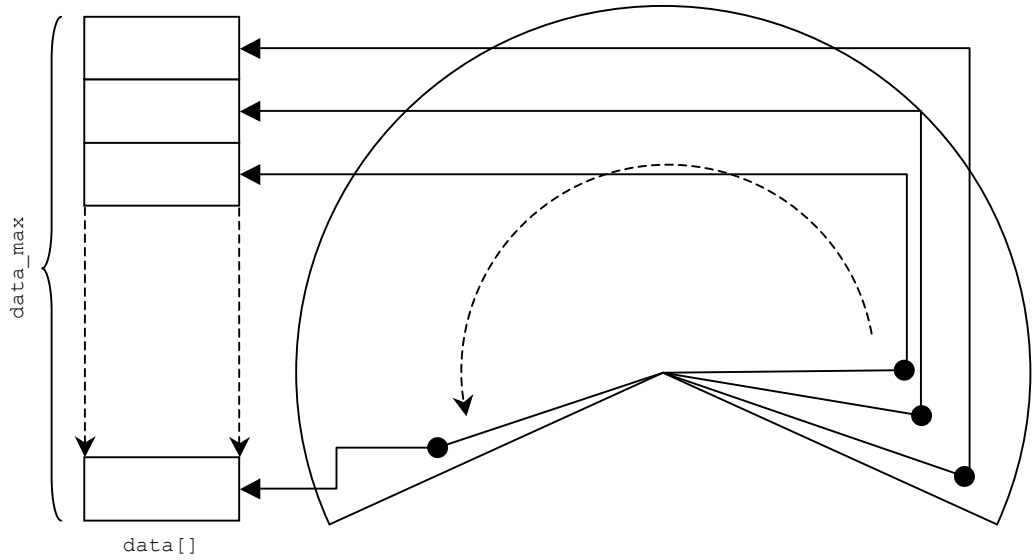


Fig. 2 - Mapping delle misure rilevate dal sensore nell'array `data`.

4.2.4.3 Render

Innanzittutto si disegna lo sfondo nero e la griglia di archi di circonferenze concentriche. La distanza tra due archi consecutivi rappresenta 1000 mm nel mondo reale. Su di essi ogni valore letto precedentemente viene disegnato con una linea bianca che parte dall'origine e le cui lunghezza e direzione rappresentano la misura effettuata. In caso di valore fuori scala, la linea visualizzata coprirà la distanza massima e sarà disegnata in rosso.

4.2.5. Compilazione ed esecuzione di URGUI

Per compilare URGUI è sufficiente lanciare da un terminale il comando

```
$ make
```

Per eseguire URGUI sono necessari i privilegi di superuser in quanto l'applicativo richiede accesso in lettura/scrittura al device a caratteri /dev/ttyACM0. Il comando da utilizzare sarà quindi

```
$ sudo ./urgui
```

5. Conclusioni e sviluppi futuri

Le attività svolte hanno raggiunto gli obiettivi del progetto: il sensore viene correttamente collegato al netbook e l'applicazione sviluppata soddisfa tutti i requisiti prefissati.

Dai test effettuati URGUI funziona correttamente e sfrutta fino in fondo le potenzialità offerte dal netbook e dal sensore in modo abbastanza ottimizzato. L'applicazione è compilabile su qualsiasi piattaforma che supporta SDL e URG (al momento Windows, Linux, Mac). URGUI dovrebbe anche risultare compatibile con qualunque laser range scanner che supporta il protocollo SCIP su USB. Infatti dai test compiuti in laboratorio si è dimostrato compatibile anche con il sensore Hokuyo UHG-08LX.

L'applicazione sviluppata, data la sua essenzialità, si candida come valido esempio di future applicazioni grafiche basate sul sensore.

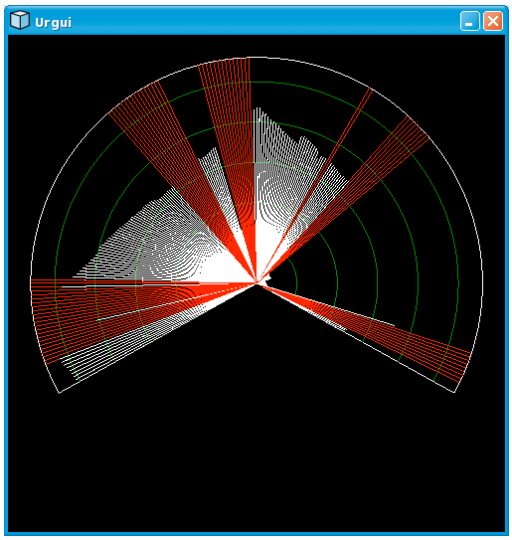


Fig. 1 - Scansione di una porzione del laboratorio di robotica.

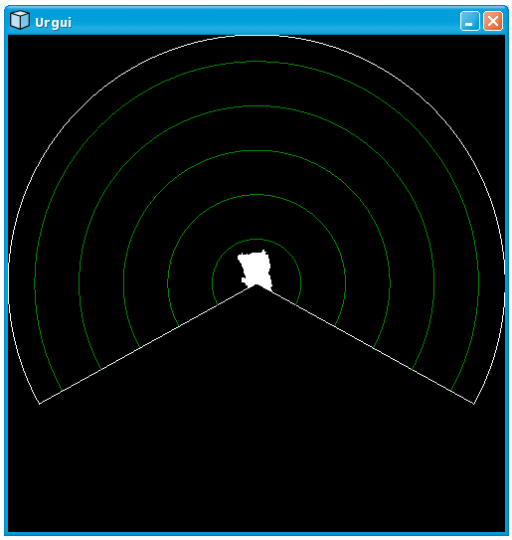


Fig. 2 - Scansione di un ambiente con ostacoli posti a meno di 1m.

Bibliografia

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

[2] AA VV: "OpenGL API Documentation", OpenGL Website (http://www.opengl.com/documentation/).

[3] AA VV: "SDL Documentation Wiki", SDL Website (http://sdl.beuc.net/sdl.wiki/).

Appendice A: Codice Sorgente (urgui.c)

L'applicazione viene rilasciata sotto licenza GNU GPL ed il testo della licenza è riportato nel codice sorgente.

```
/**
 * URGUI
 *
 * urgui.c
 * Written by Diego Ferri, Andrea Ghidini
 * May 2009
 *
 */

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

#define WIN_SIZE            500
#define PI                  3.141592653589

urg_t urg;
urg_parameter_t params;
int data_max;
float angleMin, angleMax;
long *data;

int initURG(const char *port)
{
  int res = urg_connect(&urg, port, 115200);
  if (res < 0)
    return 0;
  res = urg_getParameters(&urg, &params);
  if (res < 0)
  {
    urg_disconnect(&urg);
    return 0;
  }
  data_max = urg_getDataMax(&urg);
  data = (long*)malloc(sizeof(long) * data_max);
  if (!data)
  {
    perror("malloc");
    urg_disconnect(&urg);
    return 0;
  }

  angleMin = urg_index2deg(&urg, params.area_min_) - 90;
  angleMax = urg_index2deg(&urg, params.area_max_) - 90;

  return 1;
}

void initGL()
{
  double disp_range = params.distance_max_;
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(-disp_range, disp_range, disp_range, -disp_range, -1.0, 1.0);
  glMatrixMode(GL_MODELVIEW);
  glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
  glDisable(GL_DEPTH_TEST);
}

int update()
{
  int res = urg_requestData(&urg, URG_GD, URG_FIRST, URG_LAST);
  if (res < 0)
    return 0;
  int n = urg_receiveData(&urg, data, data_max);
  if (n < 0)
    return 0;
  return 1;
}

void render(SDL_Surface *screen)
{
  int i;
```

```
  glClear(GL_COLOR_BUFFER_BIT);
  glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
  glBegin(GL_LINE_STRIP);
    glVertex2f(0.0f, 0.0f);
    for (i = angleMin; i <= angleMax; i++)
      glVertex2f(cos(i/180.0*PI) * params.distance_max_, sin(i/180.0*PI) * params.distance_max_);
    glVertex2f(0.0f, 0.0f);
  glEnd();
  glBegin(GL_LINE_STRIP);
    glVertex2f(0.0f, 0.0f);
    for (i = angleMin; i <= angleMax; i++)
      glVertex2f(cos(i/180.0*PI) * params.distance_min_, sin(i/180.0*PI) * params.distance_min_);
    glVertex2f(0.0f, 0.0f);
  glEnd();

  glColor4f(0.0f, 0.5f, 0.0f, 1.0f);
  int c;
  for (c = 1000; c < params.distance_max_; c += 1000)
  {
    glBegin(GL_LINE_STRIP);
      for (i = angleMin; i <= angleMax; i++)
        glVertex2f(cos(i / 180.0 * PI) * c, sin(i / 180.0 * PI) * c);
    glEnd();
  }

  glBegin(GL_LINES);
    for (i = params.area_min_; i <= params.area_max_; i++)
    {
      glVertex2f(0.0f, 0.0f);
      long dist = data[i];
      if (dist == 0)
      {
        dist = params.distance_max_;
        glColor4f(1.0f, 0.0f, 0.0f, 1.0f);
      }
      else
        glColor4f(1.0f, 1.0f, 1.0f, 1.0f);
      float angle = (float)(-(urg_index2deg(&urg, i) + 90) / 180.0 * PI);
      glVertex2f(dist * cos(angle), dist * sin(angle));
    }
  glEnd();
}

int main(int argc, char *argv[])
{
  SDL_Surface *screen;
  SDL_Event event;

  const char *device = "/dev/ttyACM0";

  if (SDL_Init(SDL_INIT_VIDEO) < 0)
    return 1;
  SDL_GL_SetAttribute(SDL_GL_RED_SIZE, 5);
  SDL_GL_SetAttribute(SDL_GL_GREEN_SIZE, 5);
  SDL_GL_SetAttribute(SDL_GL_BLUE_SIZE, 5);
  SDL_GL_SetAttribute(SDL_GL_DEPTH_SIZE, 16);
  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("Urgui", "Urgui");

  if (!initURG(device))
  {
    fprintf(stderr, "Cannot initialize URG device on port %s.\n", device);
    SDL_Quit();
    return 1;
  }
  printf("URG device found on port %s.\n", device);

  initGL();

  int done = 0;
  while (!done)
  {
    while (SDL_PollEvent(&event))
    {
      if (event.type == SDL_QUIT)
        done = 1;
```

```
      if (event.type == SDL_KEYDOWN )
      {
        if(event.key.keysym.sym == SDLK_ESCAPE )
          done = 1;
      }
    }
    if (!update())
    {
      fprintf(stderr, "Cannot get data.\n");
      break;
    }
    render(screen);
    SDL_GL_SwapBuffers();
  }

  urg_disconnect(&urg);
  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: urgui

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

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

clean:
	rm urgui *.o
```

Indice