UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Ingegneria dell'Informazione

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica
(Prof. Riccardo Cassinis)

Laser Safety

Elaborato di esame di:

Christian Deldossi

Consegnato il: **20 dicembre 2013**

Sommario

Il mio lavoro è volto a creare un'applicazione che interfacci il sensore laser scanner URG-04 di Hokuyo con un calcolatore e mostri per via grafica le sue rilevazioni.

L'applicazione dovrà prevedere la possibilità di distinguere una zona di interesse particolare definita dall'utente, all'interno della quale verranno visualizzati in maniera differente eventuali nuovi ostacoli, stampando a video le loro coordinate.

1. Introduzione

Il mio progetto lavora per creare un'applicazione, basandosi sull'applicativo "mfbm", per sfruttarne alcuni metodi ed alcune caratteristiche.

Questo lavoro si occupa di interfacciare il sensore laser scanner URG-04 di Hokuyo con un calcolatore e di mostrarne su un piano cartesiano i dati da esso rilevati.

Fig. 1 - 1-1 Sensore laser scanner URG-04 HOKUYO.

2. Il problema affrontato

Lo scopo del mio lavoro è stato, in una prima fase, quello di rappresentare sul piano cartesiano gli ostacoli e i dati rilevati dal sensore, in maniera chiara, uniforme e precisa.

Durante questa fase sono stati sfruttati alcuni metodi dell'applicativo "mfbm".

Nella seconda e principale fase del mio lavoro, l'obiettivo è stato quello di permettere all'utente di definire una zona di interesse ben precisa e fare in modo che, al suo interno, vi fosse un'attenzione particolare ai dati rappresentati e rilevati: il programma deve mostrare eventuali nuovi ostacoli in maniera evidente e permettere la visualizzazione delle loro coordinate cartesiane.

Se l'utente non fosse interessato ad utilizzare questa soluzione, il programma deve continuare la sua normale esecuzione, mostrando sul calcolatore tutti i dati rilevati nella stessa modalità.

3. La soluzione adottata

3.1. Prima fase: visualizzazione ostacoli

Nella prima fase del mio lavoro ho provveduto a realizzare la visualizzazione di tutti gli ostacoli rilevati dal laser scanner su un piano cartesiano.

Per fare questo ho utilizzato diversi metodo dell'applicativo "mfbm".

Alcuni di questi per ovvie ragioni di diverse esigenze di utilizzo, è stato necessario modificarli.

La modifica più significativa è stata fatta al metodo "drawsegment", settando la visualizzazione dei dati come tanti punti della stessa dimensione nello spazio di coordinate cartesiane.

È stata fatta questa scelta geometrica perché ritenuta la più efficace ed efficiente, anche in considerazione dell'obiettivo finale del mio lavoro.

Di seguito è riportato il nuovo codice del metodo:

```
Void drawsegment(pelement segmenttodisplay, float red, float green, float blue, float alpha){

        glColor4f(red, green, blue, alpha);

        point p1,p2;
        p1.x=coords[segmenttodisplay->firstpoint].x; p1.y=coords[segmenttodisplay->firstpoint].y;
        p2.x=coords[segmenttodisplay->lastpoint].x; p2.y=coords[segmenttodisplay->lastpoint].y;

        glBegin(GL_QUADS);
        glVertex2f(p1.x-LARGHEZZA_QUADRATINO/2, p1.y-LARGHEZZA_QUADRATINO/2);
        glVertex2f(p1.x-LARGHEZZA_QUADRATINO/2, p1.y+LARGHEZZA_QUADRATINO/2);
        glVertex2f(p1.x+LARGHEZZA_QUADRATINO/2, p1.y+LARGHEZZA_QUADRATINO/2);
        glVertex2f(p1.x+LARGHEZZA_QUADRATINO/2, p1.y-LARGHEZZA_QUADRATINO/2);

        glVertex2f(p2.x-LARGHEZZA_QUADRATINO/2, p2.y-LARGHEZZA_QUADRATINO/2);
        glVertex2f(p2.x-LARGHEZZA_QUADRATINO/2, p2.y+LARGHEZZA_QUADRATINO/2);
        glVertex2f(p2.x+LARGHEZZA_QUADRATINO/2, p2.y+LARGHEZZA_QUADRATINO/2);
        glVertex2f(p2.x+LARGHEZZA_QUADRATINO/2, p2.y-LARGHEZZA_QUADRATINO/2);

        glEnd();
}
```

Nella figura seguente viene riportato un esempio di esecuzione del programma con lettura e visualizzazione dei dati.

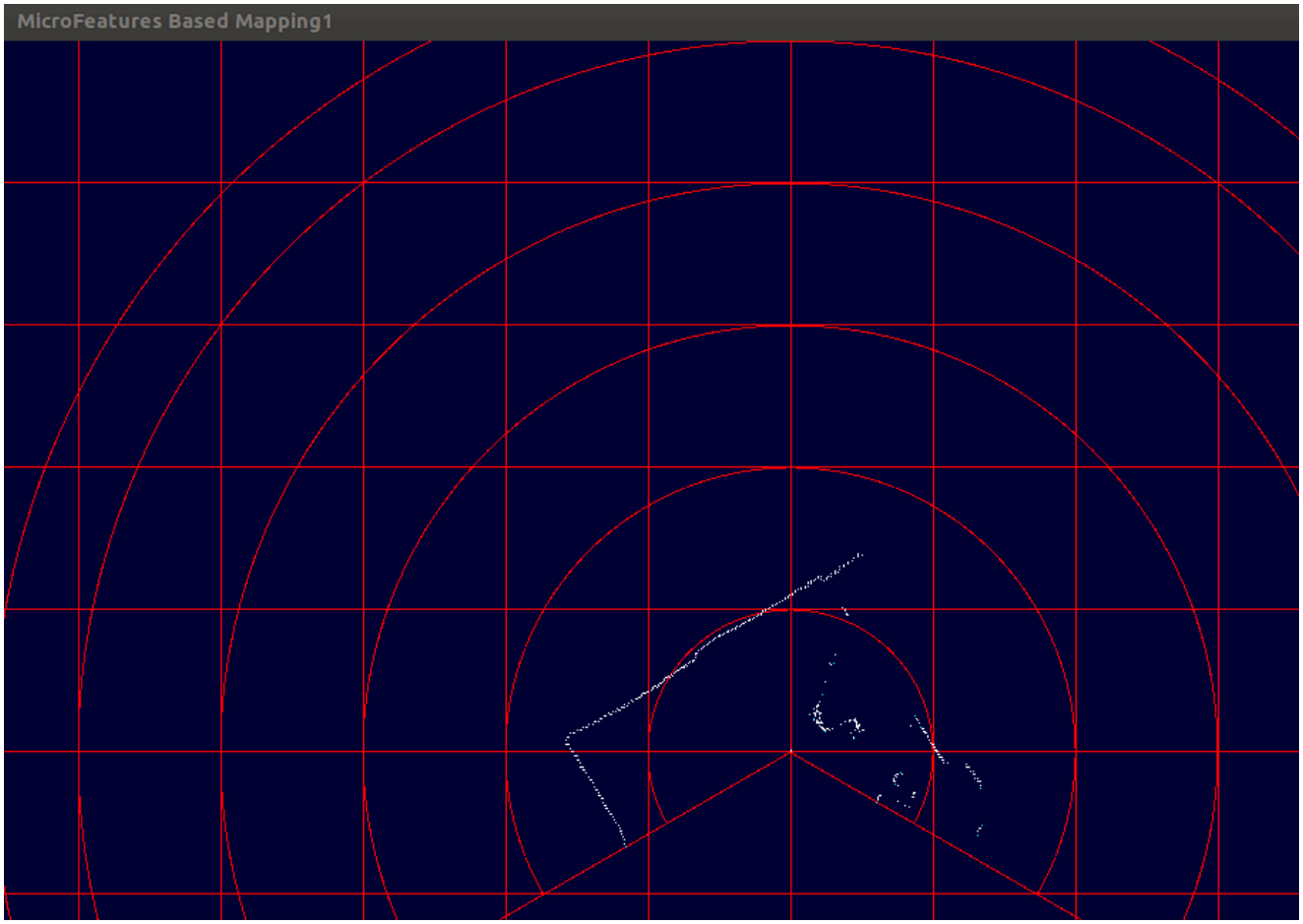

Fig. 2 - Figura 2-1: Esempio di normale esecuzione del programma LASERSAFETY

3.2. Seconda Fase: Laser Safety

3.2.1. Interazione tra utente e mouse

Per prima cosa si è reso necessario scegliere un metodo per permettere all'utente di definire la sua zona di interesse: la scelta è stata fatta rilevando i click del mouse che definissero una zona poligonale ben precisa.

Il programma rileva e salva le coordinate dei click del mouse e attiva la nuova modalità di visualizzazione degli ostacoli solo dopo il completamento dell'area poligonale (di un massimo di dieci lati); eventuali altri interventi dell'utente non vengono rilevati e salvati.

Questa rilevazione avviene durante il loop del main del programma: di seguito viene riportata la nuova versione.

```
SDL_Event event;
//                                                              loops=1;
        while (loops--) {                                               // MAIN LOOP
                while (SDL_PollEvent(&event)) {
                        if (event.type == SDL_QUIT)
                                loops = 0;
                        if (event.type == SDL_KEYDOWN ) {
                                if(event.key.keysym.sym == SDLK_ESCAPE )
                                        loops = 0;
                        }

      if( event.button.button == BUTTON_SX && event.type == SDL_MOUSEBUTTONDOWN )

                        {
                        contaClick++;
                                while((contaClick) < 11 && (contaClick != q )
                                        ){
                                        mouse_x[q] =traduci_coordX(event.button.y);
                                        mouse_y[q]= traduci_coordY(event.button.x);
                                        q++;

                                        }

                        }
```

3.2.2. Ridefinizione e adattamento coordinate

Le coordinate del click del mouse salvate non sono conformi e coerenti con gli assi e la metodologia di rappresentazione di tutti gli ostacoli rilevati.

Pertanto si è resa necessaria l'introduzione di due nuovi metodi molto semplici che permettano la conversione: di seguito è riportato il codice.

```
int traduci_coordX (int num)
{
return (-num + 500)*10;
}

int traduci_coordY (int num)
{
return (-num + 600)*10;
}
```

3.2.3. Definizione e rappresentazione a video della zona poligonale

Contestualmente alla rilevazione e al salvataggio dei click del mouse dell'utente, il programma permette la visualizzazione della zona poligonale desiderata.

Al primo click viene mostrato a video un punto giallo nella posizione esatta; ad ogni click seguente viene mostrato un punto giallo nella posizione esatta ed una linea congiungente questo punto con il precedente.

Questa rappresentazione prosegue fino a quando l'utente clicca sul primo punto selezionato, chiudendo e definendo la zona poligonale.

Per evitare problemi di precisione è stata introdotta una condizione: per chiudere il poligono è sufficiente cliccare ad una distanza ravvicinata al primo punto.

Una volta terminata la rappresentazione il programma salva nella variabile *nVert* il numero di vertici rappresentati e quindi dell'area poligonale.

Di seguito viene riportato il codice introdotto nel metodo *render* per eseguire queste azioni.

```
for(w=0; w < 10; w++ ){

        if(mouse_x[w] != 0){

        glBegin(GL_QUADS);
        glColor4f(1.0f, 1.0f, 0.0f, 1.0f);
        glVertex2f(mouse_x[0]-(LARGHEZZA_QUADRATINO),
                        mouse_y[0]-(LARGHEZZA_QUADRATINO));
        ……..
                        glEnd();
```

```
point p1,p2;
p1.x=mouse_x[0];
p1.y=mouse_y[0];
p2.x=mouse_x[w];
p2.y=mouse_y[w];


if(ppdistance(p1,p2) > 250){

glBegin(GL_QUADS);
glColor4f(1.0f, 1.0f, 0.0f, 1.0f);
glVertex2f(mouse_x[w]-(LARGHEZZA_QUADRATINO),
                    mouse_y[w]-(LARGHEZZA_QUADRATINO));
……..

                                glEnd();
nVert=w;


                    glBegin(GL_LINES);
                    glColor4f(1.0f, 1.0f, 0.0f, 1.0f);
                    glVertex2f(mouse_x[w],mouse_y[w]);
                    glVertex2f(mouse_x[(w-1)],mouse_y[(w-1)]);
                    glEnd();
}
else if( w != 0){

                    glBegin(GL_LINES);
                    glColor4f(1.0f, 1.0f, 0.0f, 1.0f);
                    glVertex2f(mouse_x[0],mouse_y[0]);
                    glVertex2f(mouse_x[(w-1)],mouse_y[(w-1)]);
                    glEnd();

nVert=(nVert + 1);
a++;
break;
}
}
```

}

3.2.4. Inizializzazione matrice

Definita la zona di interesse viene creata ed inizializzata una matrice con valori zero: questa matrice verrà utilizzata per il confronto con gli eventuali nuovi ostacoli.

È stato scelto il valore zero per indicare l'assenza di ostacoli e il valore uno per la presenza di un ostacolo.

Per evitare che ad ogni chiamata del metodo, che corrisponde ad ogni lettura dei dati, la matrice venga riinizializzata a zero, questa parte di codice viene inserita in un ciclo condizionale: la condizione posta è la seguente:

```
if( c == 0 ){
```

Questo significa che l'inizializzazione deve avvenire solo una volta, per una corretta ed efficiente esecuzione del programma.

Nel codice riportato di seguito, si nota infatti che alla fine la variabile *c* (inizialmente uguale a 0) viene incrementata dopo l'inizializzazione.

```
if( c == 0 ){
for( w=0; w < 8000; w++)
                    {
          for(  s=0;s < 8000; s++)  {
                              matrice[w][s] = 0;
                                        }
                    }
c++;
}
```

Dopo aver inizializzato *matrice*, alla prima iterazione il programma la setta e la modifica andando a segnalare(porre =1) tutti gli ostacoli già presenti.

3.2.5. Definizione algoritmo punti interni

Essendo stata data la possibilità all'utente di definire una zona poligonale, si è reso necessario introdurre un metodo per verificare quali fossero i punti interni a questa zona e che vadano quindi analizzati e quali quelli esterni.

Per fare questo è stato introdotto un metodo *pointInPolygon* che riceve in ingresso:

- Il numero di vertici del poligono: rappresentati dalla variabile *nVert*;
- I vettori contenenti le coordinate x e y dei vertici del poligono: rappresentati da *mouse_x* e *mouse_y*;
- Le coordinate x e y del punto da verificare: rappresentato da *coords[i].x* e *coords[i].y*;

Il metodo restituisce il valore 1 se il punto verificato risiede all'interno del poligono; restituisce il valore 0 in caso contrario.

Questo metodo deve essere eseguito per tutti gli ostacoli rilevati dal sensore, ma deve essere eseguito solo se e quando l'utente definisce tutta la zona poligonale di interesse.

Per questo la sua esecuzione viene inserita nella seguente istruzione condizionale.

if((pointInPolygon(nVert, mouse_x, mouse_y,coords[i].x, coords[i].y)) == 1 && (a != 0))

Di seguito viene riportato il codice dell'intero metodo, inserito nella classe *utils*.

```
int pointInPolygon(int nvert, float vertx[], float verty[], int testx, int testy)
{
  int i, j, c = 0;
  for (i = 0, j = nvert-1; i < nvert; j = i++)
 {
   if ( ( (verty[i]>testy) != (verty[j]>testy))  &&
          ( testx < (vertx[j]-vertx[i]) * (testy-verty[i]) / (verty[j]-verty[i]) + vertx[i]) )
     c = !c;
  }
  return c;
}
```

3.2.6. Visualizzazione nuovi ostacoli

Ad ogni rilevazione e lettura dei dati il programma esegue questo procedimento, di cui viene riportato il codice:

1. Viene prese in considerazione e scandita solo la zona interessata dall'utente, sfruttando le coordinate del mouse, opportunamente tradotte e adattate;
2. Viene scandito il vettore contenente tutti gli ostacoli rilevati in quella zona(*data[i]* tramite le relative coordinate);
3. Viene effettuato il confronto punto per punto in quella zona: se un ostacolo ad una data posizione era già presente(quindi *matrice[][]==1*), si passa all'ostacolo successivo; se ad una data posizione non c'era nessun ostacolo (quindi *matrice[][]==0*) ed ora è comparso, il programma disegna in quella posizione l'ostacolo in verde e di dimensioni maggiori(per una questione visiva) e indica la sua presenza(viene settata a 1 la posizione di *matrice[][]* corrispondente); vengono inoltre stampate a video le coordinate;
4. Il programma continua la ripetizione di queste operazioni fino all'interruzione dell'utente o fino alla fine della lettura dei dati.

In seguito a diverse esecuzioni di prova e ad un colloquio col Docente si è notato che, dato:

- il numero piuttosto elevato di ostacoli rilevati;
- l'altissimo numero di iterazioni eseguite dal programma;
- la presenza di un inevitabile rumore e disturbo nella lettura dei dati;

la visualizzazione dei nuovi ostacoli risultava essere troppo precisa e troppo veloce: per questo motivo non forniva una reale trasposizione degli effettivi nuovi ostacoli.

Si è deciso quindi di diminuire la precisione dell'algoritmo di confronto, per rendere migliore e più performante il programma, tenendo in considerazione l'ambito di utilizzo futuro.

Di conseguenza il punto 3 descritto in precedenza è stato modificato e quindi, anziché effettuare il confronto *punto per punto*, si è deciso di confrontare *ogni punto con i 4 punti successivi e i 4 punti precedenti:* [1] in questo modo vengono visualizzati solo cambiamenti significativi (di almeno 9 punti contigui).

Il resto dell'algoritmo è rimasto invariato.

Di seguito viene riportato il codice della versione finale del metodo.

```
for(w=0; w < 10; w++ ){
        if(mouse_x[w] != 0){
        glBegin(GL_QUADS);
        glColor4f(1.0f, 1.0f, 0.0f, 1.0f);
        glVertex2f(mouse_x[0]-(LARGHEZZA_QUADRATINO),
                        mouse_y[0]-(LARGHEZZA_QUADRATINO));
        ……
                        glEnd();
        point p1,p2;
        p1.x=mouse_x[0];
        p1.y=mouse_y[0];
        p2.x=mouse_x[w];
        p2.y=mouse_y[w];
        if(ppdistance(p1,p2) > 250){
        glBegin(GL_QUADS);
        glColor4f(1.0f, 1.0f, 0.0f, 1.0f);
        glVertex2f(mouse_x[w]-(LARGHEZZA_QUADRATINO),
                        mouse_y[w]-(LARGHEZZA_QUADRATINO));
        …….
                        glEnd();
        nVert=w;
                glBegin(GL_LINES);
                glColor4f(1.0f, 1.0f, 0.0f, 1.0f);
                glVertex2f(mouse_x[w],mouse_y[w]);
```

[1] In questo caso i termini *successivi* e *precedenti* indicano i punti con coordinate cartesiane immediatamente prima e immediatamente dopo il punto esaminato: di conseguenza vengono presi in considerazione, oltre al punto, gli 8 punti vicini in termini di spazio.

```
                        glVertex2f(mouse_x[(w-1)],mouse_y[(w-1)]);
                        glEnd();
                }
                else if( w != 0){
                        glBegin(GL_LINES);
                        glColor4f(1.0f, 1.0f, 0.0f, 1.0f);
                        glVertex2f(mouse_x[0],mouse_y[0]);
                        glVertex2f(mouse_x[(w-1)],mouse_y[(w-1)]);
                        glEnd();
                nVert=(nVert + 1);
                a++;
                break;
                }
                }
}
if( c == 0 ){
for( w=0; w < 8000; w++)
                {
        for(  s=0;s < 8000; s++)  {
                        matrice[w][s] = 0;
                                }
                }
c++;
}
if((pointInPolygon(nVert, mouse_x, mouse_y,coords[i].x, coords[i].y)) == 1 && (a != 0))
{
                if(b < 2000)        {
                matrice[coords[i].x][coords[i].y] = 1;
                        }
                int next=i+1;
                int prev=i-1;
                int next2=i+2;
                int prev2=i-2;
                int next3=i+3;
                int prev3=i-3;
                int next4=i+4;
                int prev4=i-4;
```

```
if(matrice[coords[i].x][coords[i].y] == 0 && matrice[coords[next].x][coords[next].y] == 0 && matrice[coords[next2].x][coords[next2].y] == 0 && matrice[coords[prev].x][coords[prev].y] == 0 && matrice[coords[prev2].x][coords[prev2].y] == 0 && matrice[coords[prev3].x][coords[prev3].y] == 0 && matrice[coords[next3].x][coords[next3].y] == 0 && matrice[coords[prev4].x][coords[prev4].y] == 0 && matrice[coords[next4].x][coords[next4].y] == 0 && b > 2000){
                glBegin(GL_QUADS);
                glColor4f(0.0f, 1.0f, 0.0f, 1.0f);
                glVertex2f(coords[i].x-(LARGHEZZA_QUADRATINO*2),
                                coords[i].y-(LARGHEZZA_QUADRATINO*2));
                ……….
                glEnd();
        printf("Le coordinate del nuovo ostacolo sono x=%d e y=%d \n " , coords[i].x ,coords[i].y );
                matrice[coords[i].x][coords[i].y] = 1;
                                                        }
b++;
}
```

4. Modalità operative

Il programma richiede una modalità di avvio molto semplice: dopo essersi posizionato nella cartella contenente tutti i sorgenti, va avviato normalmente da terminale.

Viene data la possibilità di fornire alcune opzioni all'avvio:

- "./LaserSafety [-i <inputfile>]": con questa opzione è possibile passare un file in input al programma;
- "./LaserSafety [-o <output file>]": con questa opzione è possibile salvare la lettura dei dati in un file;
- "./LaserSafety [-l <loops>]": con questa opzione è possibile specificare al programma il numero di cicli, quindi di letture consecutive, che il programma deve eseguire.

Tutte queste opzioni sono consultabili nel file "usage.c".

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;
- interfaccia USB 2.0;
- alimentazione a 5V tramite interfaccia USB.

4.1.2. Librerie necessarie

Per un corretto funzionamento del programma è necessaria l'installazione di alcune librerie sul calcolatore in uso:

- build-essential;
- SDL: Simple DirectMedia Layer è una libreria libera multimediale multipiattaforma, scritta in C, che crea un livello astratto al di sopra di varie piattaforme software grafiche e sonore e dunque può controllare video, audio digitale, CD-ROM, suoni, thread, caricamento condiviso di oggetti, timer e networking. Versione utilizzata: 1.2;
- OPENGL: Open Graphics Library è una specifica che definisce API per più linguaggi e per più piattaforme per scrivere applicazioni che producono computer grafica 2D e 3D. L'interfaccia consiste in 250 diverse chiamate di funzione che si possono usare per disegnare complesse scene tridimensionali a partire da semplici primitive. Versione utilizzata: 2.0
- URG: libreria necessaria per interfacciarsi ed interagire con il sensore laser scanner URG-04 di Hokuyo.

4.1.3. Linux

Questo lavoro è stato svolto su un calcolatore con installato Ubuntu 11.10(versione 32 bit) e con versione del kernel 3.0.0.

4.2. Modalità di installazione

Per la compilazione e l'esecuzione del programma il calcolatore deve avere installato le librerie descritte nel paragrafo precedente; in particolare è necessario installare i seguenti pacchetti:

- build-essential;
- libsdl1.2-dev;
- liburg0-dev;
- libsdl1.2-net1.2;
- urg-utils;
- automake 1.10.

Essendo presente un makefile, riportato in appendice, per la compilazione è sufficiente da terminale digitare il comando

```
$ make
```

4.3. Avvertenze

L'applicativo richiede accesso in lettura e scrittura al device a caratteri /dev/TTYACM0: per questo motivo è necessario inserire il proprio utente nel gruppo DIALOUT.

5. Conclusioni

L'attività svolta ha consentito il raggiungimento degli obiettivi prefissati: l'applicativo "LaserSafety" permette una corretta e coerente visualizzazione grafica degli ostacoli rilevati dal sensore laser scanner.

Viene data inoltre la possibilità all'utente di scegliere una zona preferenziale poligonale, tramite click del mouse sullo schermo: in questa zona l'applicativo mostra tutti gli ostacoli rilevati, mettendo in evidenza, colorati di verde, solo gli eventuali nuovi ostacoli e stampando a video le relative coordinate cartesiane.

Nel caso l'utente non fosse interessato ad una particolare zona, l'applicativo continua a mostrare tutti gli ostacoli nella stessa forma e con lo stesso colore.

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/).

[4] Prof. Riccardo Cassinis : Applicativo “mfbm”

Appendice A: Makefile

```
PROGRAM_NAME=LaserSafety
VERSION=1.0

# Attenzione! quanto segue serve per la creazione dei tarball! Non dimenticare nulla!
# ALL_FILES = *.c *.h Makefile TODO CHANGELOG COPYING README *.txt

# Definitions
CC=cc
CXXINC=`c_urg-config --cflags` `sdl-config --cflags` -Iinc/
CXXLINK= `c_urg-config --libs` `sdl-config --libs`  -lGL

#
BARECXXFLAGS=  -g -O0 -Wall -c
CFLAGS =$(BARECXXFLAGS)

COMPILE = $(CC) $(CFLAGS)
#

OBJFILES := $(patsubst src/%.c,obj/%.o,$(wildcard src/*.c))

all : $(PROGRAM_NAME)

$(PROGRAM_NAME): $(OBJFILES)

	$(CC)  $(OBJFILES) -o $(PROGRAM_NAME) $(CXXLINK)

obj/%.o: src/%.c inc/%.h inc/LaserSafety.h

	$(COMPILE) $(CXXINC) -o $@ $<

clean:
	rm -f obj/*.o
	rm -f */*~
	rm -f *~
	rm -f $(PROGRAM_NAME)
	rm -f */._*
	rm -f ._*
	rm -f .DS_Store
	rm -f */.DS_Store
```

```
install:    all
            cp $(PROGRAM_NAME) /usr/local/bin/$(PROGRAM_NAME)

dist:       clean
#           mkdir $(PROGRAM_NAME)
#           cp $(ALL_FILES) $(PROGRAM_NAME)
            tar czvf ../$(PROGRAM_NAME)-V$(VERSION).tgz ../LaserSafety
#           rm -rf $(PROGRAM_NAME)
            @echo All done!

rebuild:    clean       all

# For debug purpose only
print:
            @echo $(CXXINC)
            @echo ==================================
            @echo $(OBJFILES)
```

Indice