UNIVERSITA' degli STUDI di BRESCIA

Facoltà di Ingegneria

Corso di robotica prof. Riccardo Cassinis

Progetto

FOLLOW MY SOCKS

Gabriele Filippini 022749

Gianpaolo Gasparini 022270

Gli autori di *Follow my Socks*

Gabriele Filippini

Gianpaolo Gasparini

Obiettivo del progetto

Lo scopo del progetto è la realizzazione di un robot in grado di seguire un uomo mentre cammina. A tal fine, viene identificato un blob di colore di riferimento che il robot sarà in grado di ricercare e seguire.

In sede operativa, si è scelto di utilizzare una fascia colorata da applicare al polpaccio della persona da seguire.

Materiale impiegato

- Robot *Speedy* tipo Pioneer1 dotato di sistema di visione Cognachrome.
- Sistemi pc-linux
- Materiale del Laboratorio di Robotica Avanzata della facoltà.

Realizzazione Pratica

Il progetto è stato realizzato in maniera da semplificare al massimo la taratura e l'utilizzo del robot. La prima operazione da eseguire è la memorizzazione delle caratteristiche del target nella EEPROM del Cognachrome. A questo punto, il robot è già pronto per la caccia.
Una volta lanciato il programma, il robot è autonomo, anche se è stata prevista una semplice interfaccia a tastiera del PC di guida per i comandi di start-stop (per soste temporanee) e la disconnessione.

Manuale Utente

Vengono descritte dettagliatamente le operazioni da eseguire per il set-up e l'utilizzo del robot.

Configurazione del Cognachrome

Il Cognachrome è il sistema hardware di acquisizione ed elaborazione delle immagini provenienti dalla telecamera installata sul robot.
Per configurarlo, occorre lanciare il programma mini-arc.
Al prompt di Linux digitare:

```
arc -saphira
```

Compaiono altre due finestre: in quella titolata "Saphira Interaction" digitare:

```
q
```

Nella finestra "ARC stream" compare il prompt "Tpuvis[x]" di cui è mostrata la schermata in fig. 1

Figura 1 : Prompt di Tpuvis mode

Esistono tre canali di ingresso, contraddistinti da *a,b,c*. Per scegliere il canale desiderato digitare .

```
a
```

seguito dalla lettera del canale. A questo punto, è possibile posizionare l'oggetto colorato di fronte alla telecamera, e digitare

```
s (minuscolo)
```

per acquisire il blob. Per far sì che il sistema riconosca il blob in condizioni realistiche (zone d'ombra, diverse temperature di colore della luce…) è possibile effettuare una serie di acquisizioni premendo ogni volta il tasto

```
j
```

Terminate le varie acquisizioni nelle diverse condizioni di luce, occorre salvare queste informazioni nella EEPROM del robot, tramite il comando

```
S (maiuscolo)
```

Per uscire, digitare

```
Q
```

NOTA

Uscendo da miniARC, il terminale XTERM si blocca ed è necessario utilizzare il comando Kill.

Interfaccia utente

La navigazione del robot è autonoma, tuttavia sono stati previsti alcune funzioni disponibili a tastiera; in effetti, il sistema è pensato come robot con pc portatile annesso.
Una volta avviato il programma, sono disponibili i comandi:

- SPACE: Ferma il robot, per una sosta o in caso di emergenza
- G "Go", riavvia il robot
- Q Esce dalla navigazione

Logica di comando

In questo capitolo, si esaminano le strategie di navigazione e di movimentazione del robot e le strade seguite durante lo studio, per arrivare al risultato finale.

Un primo approccio al problema

Il robot viene controllato da un programma proprietario che si interfaccia con l'ambiente SAPHIRA, il quale provvede all'esecuzione di comandi I/O a basso livello.

In Saphira è possibile definire dei Behaviors e delle Activities per il controllo del Robot, scritti nel linguaggio Colbert.
Il primo studio è stato il tentativo di organizzare behaviors predefiniti, quali

- AvoidCollision
- TurnLeft
- AttendAtPos

Tale studio si è rivelato un insuccesso, poichè:

- Saphira è in generale un sistema poco affidabile ed i behaviors predefiniti presentano diversi malfunzionamenti.
- Non esiste sufficiente documentazione sul linguaggio Colbert e l'unico behavior presente per il controllo di direzione del robot è TurnLeft. Questo significa che, durante la ricerca del blob, il robot può girare solo a sinistra. Ciò è potenzialmente scorretto (cosa accade se il blob scompare alla destra del robot?) e sicuramente inefficente.

La soluzione seguita

Visto lo studio precedente, è stata scelta la via della movimentazione diretta.

Logica di funzionamento

Il problema è scomponibile in due parti:

- Ricerca del blob colorato
- Approccio al blob

Nella fase di ricerca, il robot gira su se stesso fino a trovare un blob di colore con dimensioni accettabili, cioè abbastanza grande da non poter essere confuso con un riflesso o un oggetto sullo sfondo.
In questa fase, viene processata in modo continuo l'immagine fornita dalla telecamera: se non è presente un blob, il robot ruota di un angolo prefissato.

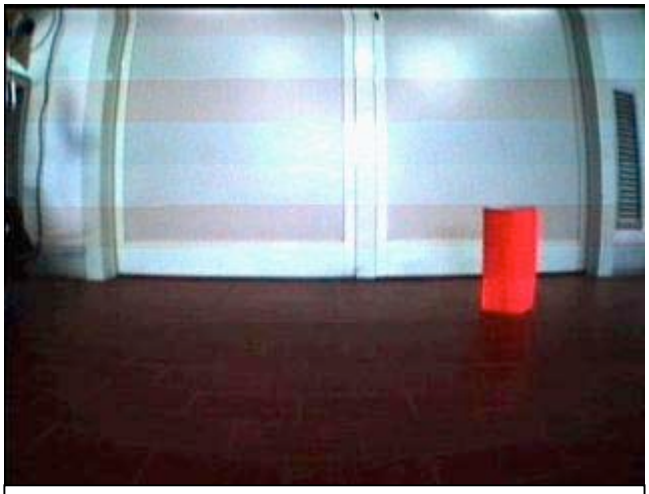


Figura 2 : Immagine dalla telecamera

Figura 3 : Immagine elaborata

Se invece l'immagine presenta un blob, e si trova di fronte al robot (entro una certa tolleranza) inizia la seconda fase.

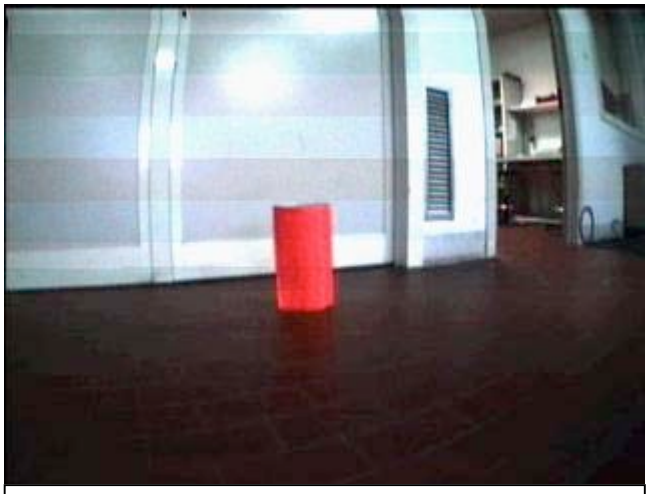
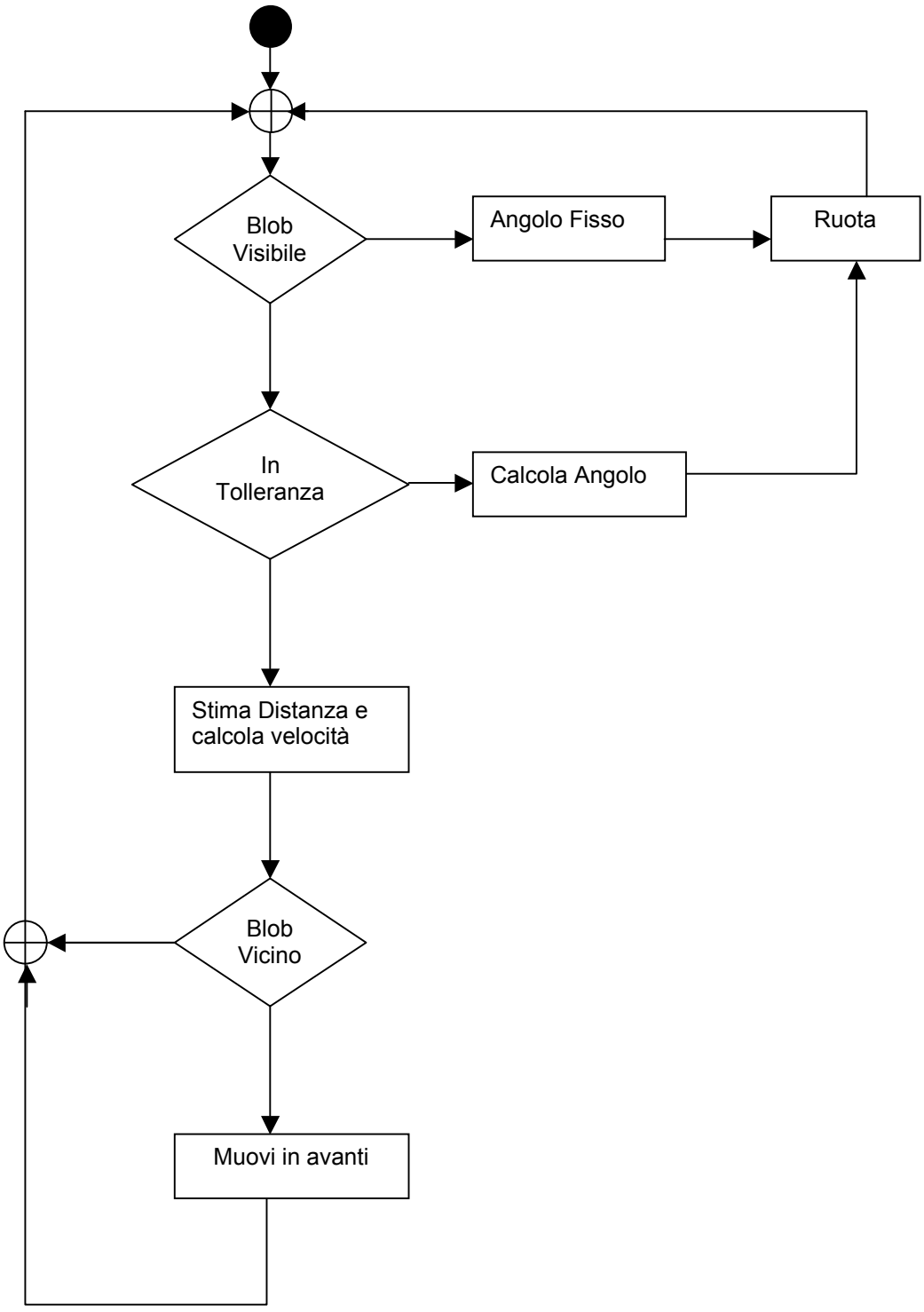


Figura 4 : Blob centrato

Figura 5 : Approccio terminato

L'approccio viene eseguito con velocità variabile in funzione della distanza, stimata rispetto alla dimensione del blob.
Anche in questa seconda fase viene processata l'immagine acquisita dalla telecamera: in questo modo il robot può istantaneamente seguire una nuova rotta, qualora il blob cambi posizione.
Se il blob scompare dalla vista del robot, si ritorna nella prima fase. Al contrario, se il blob ha dimensioni superiori ad un certo limite, è stata certamente raggiunta la posizione: si fermano i motori e si attende che il blob si muova nuovamente.

Il diagramma di flusso illustra ciò detto:



Conclusione

I risultati sperimentali conseguiti sono soddisfacenti, in quanto dimostrano la fattibilità di un sistema di navigazione autonomo per robot Pioneer 1.
Il moto è risultato essere sufficientemente fluido e il target viene seguito con buona precisione.
Il limite fondamentale è dato dalla bassa velocità con cui il robot raggiunge il goal, dovuto alla politica di monitoraggio continuo della posizione del blob.

Un secondo livello di sviluppo potrà affrontare alcune tematiche come:

- Migliore interazione tra Saphira e miniARC per eliminare la necessità di utilizzare direttamente quest'ultimo a riga di comando
- Ottimizzazione dei movimenti, aumentando il parallelismo tra ricerca e approccio al blob

Note finali

Problemi

La documentazione è ASSOLUTAMENTE INSUFFICIENTE, in particolare per ciò che riguarda i behavior e la gestione della visione in ambiente Saphira.

Saphira è poco stabile e mostra una serie di problemi quali:

- Crash improvvisi, senza messaggi di errore che possano indirizzare un debugging
- Casuale impossibilità connessione al Pioneer, lasciando immuato il codice (funzionante fino a poco prima!)
- Assenza di funzionalità fondamentali (behavior TurnRight!)

Il sistema miniARC sembra alquanto instabile e, stante il fatto che è impossibile uscire dall'ambiente senza provocare il crash del terminale, in alcune occasioni non scrive correttamente i dati nelle EEPROM del robot.

Il robot presenta problemi hardware probabilmente dovuti ad un cattivo cablaggio (non è stato possibile verificare esattamente quale componente ne sia responsabile, anche se sembra sia il Cognachrome).

Appendice

Taratura della telecamera

Per poter utilizzare i dati provenienti dal Cognachrome nella navigazione è stato necessario calcolare una funzione che, partendo dalla coordinata x del blob, restituisse l'angolo tra la direzione attuale del robot ed il blob stesso. Per evitare che la luce proveniente dall'alto interferisse con la lettura della telecamera, si è inclinata quest'ultima di circa 20 gradi verso il basso.
A tal fine si sono ricavati empiricamente i valori restituiti dalla funzione "found_blob" per alcune ben precise posizioni del blob rispetto al robot.
La telecamera ha un campo di circa 90 gradi orizzontalmente, si è scelto quindi di posizionare il blob lungo una retta distante 1000 mm dal robot in modo da formare un angolo di 10, 20, 30 e 40 gradi.

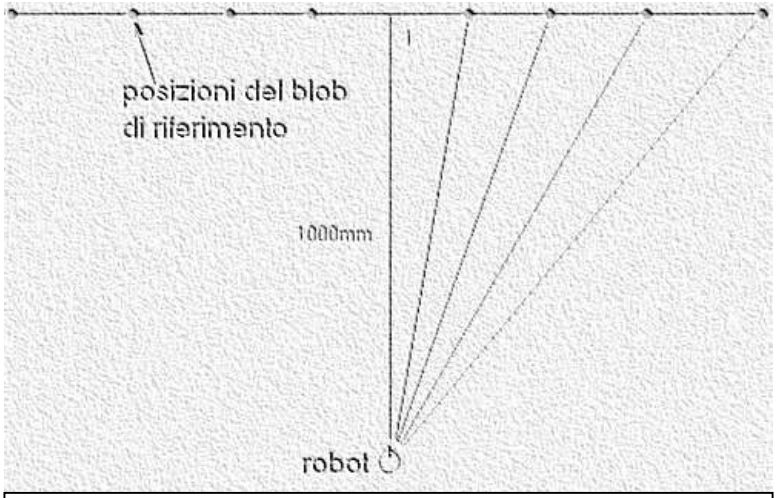


Figura 6 : Misure di regolazione della telecamera

Dopo aver calcolato la lunghezza l come

$$l=1000\tan(angle)$$

si sono acquisiti i valori restituiti dalla funzione sopra citata riassunti nella tabella 1.

Tabella 1 : Parametri telecamera

angolo	l	x_inclin	x_orizz
-40	-839	-89	-84
-30	-577	-67	-65
-20	-364	-44	-43
-10	-176	-22	-20
0	0	0	0
10	176	23	20
20	364	45	43
30	577	68	66
40	839	89	85

Si possono notare 2 serie di valori x_incl e x_orizz: il primo si riferisce al caso in cui la telecamera è inclinata verso il basso, nel secondo caso la telecamera è orizzontale.
In figura 7 sono rappresentati i valori in forma grafica, si nota come le due serie siano molto simili: ciò semplifica la sintesi della funzione.

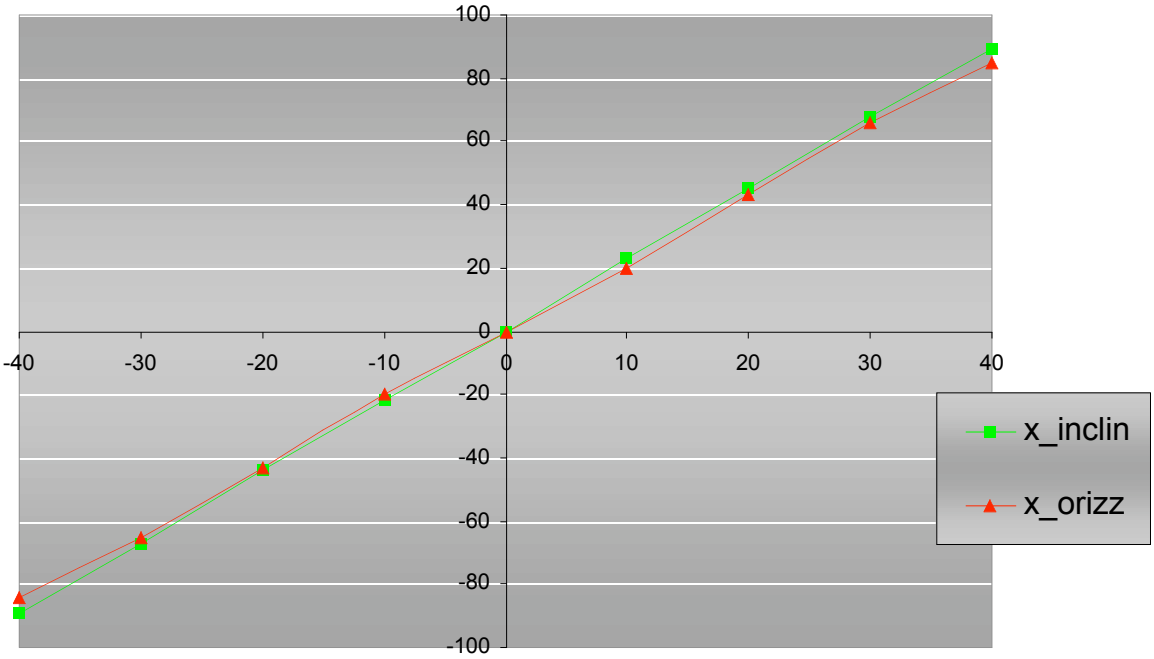


Figura 7 : Parametri telecamera

Si nota pure che la distribuzione è quasi rettilinea ed una semplice retta di regressione può approssimare bene i dati.
La retta cercata ha equazione

$$f(angle)=2.3*angle$$

Ora, partendo dalle coordinate x del blob restituite da "found_blob", è facile stimare l'angolo come

$$angle=x/2.3=x*0.44$$

Listati

Vengono allegati i sorgenti C del sistema di navigazione.
Il file principale è *fms.c* il quale richiama *pio.c* che gestisce l'I/O su robot .
Mychroma.c contiene la gestione di visione e movimentazione.

```
// ##############################################
//
// Fms.c
//
//   by G. Filippini and G. Gasparini
//
//   Last Revision Sept. 1999
//
// ##############################################

#include "pio.h"

void main()
{
  pioConnectRobot();
}


// ##############################################
//
// pio.c
//
//   by G. Filippini and G. Gasparini
//
//   Last Revision Sept. 1999
//
// ##############################################

#include "pio.h"
#include "mychroma.h"


int pioDisconnectRobot(void); // prototype


void beep(void)
{
  sfRobotComStrn(sfCOMSAY,"\25\003",2);
}


void boop(void)
{
  sfRobotComStrn(sfCOMSAY,"\25\060",2);
}


void pioVision(void)
{
  switch(process_state)
    {
    case sfINIT:
```

```
      setup_vision_system();
      sfStartBehavior(sfStop,"stop",0,4,0);                  // safety procedure
      sfInitIntention(draw_blobs,"draw_blobs",0,sfEND);      // update the
saphira window
                                                             // with what's
camera seeing
      sfInitIntention(search_and_go_blob,"s_a_g_b",0,sfEND); // find blob and
approach it
      setup_vision_system();                                 // set all vision
parameters
      process_state=20;
      break;
    }
}


void myStartupFn(void)
{
  sfSetDisplayState(sfDISPLAY,2); // refresh Saphira window every 200mS
}


void myConnectFn(void)
{
  sfInitControlProcs();                 // startup communication processes
  //sfInitRegistrationProcs();
  //sfInitAwareProcs();
  //sfInitInterpretationProcs();
  //sfInitSpecialProcs();
  sfInitProcess(pioVision,"pioVision"); // startup vision procedures
}


int myKeyFn(int ch) // managing of keyboard
{
  switch (ch)
    {
    case SPACEKEY:  // 'space' key stand-by the robot
      boop();
      boop();
      sfBehaviorControl=0;
      sfSetRVelocity(0);
      sfSetVelocity(0);
      sfRemoveTask("s_a_g_b");
      break;
    case 'g':       // 'g' or 'G' key start/resume navigation
    case 'G':
      beep();
      sfBehaviorControl=1;
      sfInitIntention(search_and_go_blob,"s_a_g_b",0,sfEND);
      break;
    case 'q':       // 'q' or 'Q' key halt and disconnect the robot
    case 'Q':
      pioDisconnectRobot();
      break;
    }
}


int pioConnectRobot(void)     // activate link between pc and Pioneer robot
{
  sfOnStartupFn(myStartupFn); // register a startup function
```

```
  sfKeyProcFn(myKeyFn);       // startup keyboard management
  sfOnConnectFn(myConnectFn); // register a connection function
  sfStartup(1);  // 1 start up the Saphira window, and then keep going
                 // 0 start up the Saphira window, and execute the microtask
  sfMessage("Connect to robot ..");
  sfConnectToRobot(sfTTYPORT, sfCOM2); // startup a connection
  while (!sfIsConnected) sfPause(100); // attend till connection is estabilished
  sfSMessage("Waiting for connection ...");
  sfMessage("***  READY  *** ");
  while (!sfIsExited) sfPause(1000);
  if (sfIsConnected)
    {
      sfDisconnectFromRobot();
      sfPause(1000);
    }
  return 0;
}


int pioDisconnectRobot(void)  // disconnection
{
  sfMessage("Disconnecting from Robot ...");
  sfDisconnectFromRobot();
  sfMessage("***  DISCONNECTED  *** ");
  return 0;
}


int pioStopRobot(void) // halt the robot
{
  sfSetVelocity(0);
  sfMessage("STOP");
  sfPause(5000);
  sfDisconnectFromRobot();
  return 0;
}

#ifndef PIO_H
#define PIO_H

#undef ABS
#undef VERSION
#undef min
#undef max
#include"saphira.h"

#undef max
#define max(a,b) ((a)>(b) ? (a) : (b))

#undef min
#define min(a,b) ((a)<(b) ? (a) : (b))

#ifdef __cplusplus
extern "C" {
#endif                          /* __cplusplus */


int pioConnectRobot(void);  // attivazione collegamento con il pioneer

int pioDisconnectRobot(void);  // disattivazione collegamento con il pioneer

int pioStopRobot(void);
```

```
void pioMoveRobot(void);


#ifdef __cplusplus
}                               /* extern "C" */
#endif                          /* __cplusplus */
#endif
```

```
//###############################################

// mychroma.c
//
//   by G. Filippini and G. Gasparini
//
//   Last Revision Sept. 1999
//

//###############################################


#include "saphira.h"
#include "mychroma.h"

#define DELTA 10      // tollerance: center of camera view from center of blob
#define DEFAULT_ROTATION 20 // if the blob is invisible, the robot turn this
quantity
#define NEAR_AREA 1800      // area of blob when robot is near goal
#define DRAW_BLOB_TH 20     // min blob area

extern float sfBlobConst;

int blob_area=0;              // look at the name ;-)
float sfBlobConst = 40000.0;
int found_blob_area = 50;     // min area of recognized blob
int isTollOK=0;               // boolean (1=blob centred)
int xdist=0;                  // x-coordinate of blob (projected onto ground
plane)
int last_pos=0;               // last position of blob in the camera view field

SFPROCESS StopApproach = NULL;


void setup_vision_system(void)
{
  sfRobotComStr(VISION_COM,"pioneer_a_mode=2"); // blob bb mode
  sfRobotComStr(VISION_COM,"pioneer_b_mode=2"); //      "
  sfRobotComStr(VISION_COM,"pioneer_c_mode=2"); //      "
}


void stop_vision_system(void)
{
  sfRobotComStr(VISION_COM,"pioneer_visrun=0");
}


int foundb(struct vinfo *v, int delta) // low-level function (finding blobs)
{
  isTollOK=0;
  if (v->area >= found_blob_area) // area is OK
    {
      last_pos=ABS(v->x);         // last blob position
      blob_area=v->area;
      if (ABS(v->x) <= delta)     // blob in front of robot
      isTollOK=1;
      return v->x;
    }
  else
    {
      isTollOK=0;
```

```
      return 10000;               // no blob found
    }
}


int found_blob(int channel, int delta) // high-level function (finding blobs)
                                       // is useful if you want to get
                                       // all three channels from Cognachrome
{
  switch(channel)
    {
    case 0:
      return foundb(&sfVaInfo, delta);
    case 1:
      return foundb(&sfVbInfo, delta);
    case 2:
      return foundb(&sfVcInfo, delta);
    }
}


void draw_one_blob(struct vinfo *v, int color) // low-level function
                                               // (drawing one blob onto screen)
{
  int x, y, h, w;
  if (v->area >= DRAW_BLOB_TH) // blob is large enough
    {
      x = BLOB_X(v);   // x-coordinate of center of mass
      y = BLOB_Y(v,x); // y-            "
      h = v->h;        // height of blob
      w = v->w;        // width
      sfSetLineColor(color);
      draw_rw_cbox(x,y,w,h);
    }
}


void draw_blobs(void) // high-level function (drawing blobs)
{
  switch(process_state)
    {
    case sfINIT:
    case sfRESUME:
      draw_one_blob(&sfVaInfo,sfColorRed);
      draw_one_blob(&sfVbInfo,sfColorDarkOliveGreen);
      draw_one_blob(&sfVcInfo,sfColorYellow);
      break;
    }
}


void find_blob(void)
     // This function's goal is to rotate till one blob enter into camera view
field,
     // then the blob is centered
{
  int angle; // angle between front of robot and blob

  switch(process_state)
    {
    case sfINIT:
```

```
    case 22:
      xdist=found_blob (0 , DELTA);
      if (isTollOK==1)                    // blob in front of robot
      {
        process_state=sfSUCCESS;
      }
      if ((xdist!=10000) && (isTollOK==0)) // blob visible but not in front of
robot
      {
        angle=(int) ceil(xdist*0.44);     // rotation is proportional to the
                                          // x-coord of blob (see manual for
details)
        sfBehaviorControl=0;              // disabling behaviors before sending
                                          // direct motion commands to robot
        sfSetRVelocity(30);
        sfSetDHeading(angle);
        sfBehaviorControl=1;              // abiliting behaviors
      }
      if (xdist==10000)                     // invisible blob
      {
        angle=DEFAULT_ROTATION*last_pos/ABS(last_pos); // rotation=default angle

        sfBehaviorControl=0;
        sfSetRVelocity(60);
        sfSetDHeading(angle);
        sfBehaviorControl=1;
      }
      break;
    }
}


void search_and_go_blob(void)
     // This is the "core" of movimentation.
     // It consists of two steps:
     // 1) it launch "find_blob" so the target is centred
     // 2) sends direct motion commands to robot that approachs blob.
     // Speed and step length are proportional to the distance of blob.
{
  static sfprocess *ptrIntFindBlob = NULL; // pointer to task "find_blob"
  static sfprocess *ptrBehAvColl = NULL;   // pointer to Behavior
"avoid_collision"

  switch(process_state)
    {
    case sfINIT:
      ptrBehAvColl= sfStartBehavior(sfAvoidCollision,"stop collision", 0, 0, 0,
                            0.5,               // front sensitivity
                            0.5,                  // side sensitivity
                            5,                    //turning gain
                            flakey_radius + 200.0, // standoff
                            sfLEFTTURN            // preferred turn direction
                            );
      sfStartBehavior(sfStop, "stop", 0, 4, 0);            // stop the robot (is
a safety procedure)
      process_state=20;

    case 20:
      ptrIntFindBlob = sfInitIntention(find_blob, "find a blob", 0,
                               sfINT, 0,
                               sfEND);
      process_state = 30;
```

```
      break;

    case 30:
      if (ptrIntFindBlob->state == sfSUCCESS) // wait 'till "find_blob" return a
success
      {
        sfKillIntention(ptrIntFindBlob);    // safety procedure
        ptrIntFindBlob=NULL;
        sfBehaviorControl=0;
        if (blob_area<=NEAR_AREA)           // robot is not close to the blob
          {
            if (blob_area<(int) ceil(NEAR_AREA*2/3)) // robot is far from blob
            {
              sfSetMaxVelocity(2000);
              sfSetPosition(300);
            }
            else                                    // robot is near the blob
            {
              sfSetMaxVelocity(200);
              sfSetPosition(100);
            }
          }
        sfBehaviorControl=1;
        process_state = 20;
      }
      break;

    case sfFAILURE:
    case sfTIMEOUT:
      break;
    }
}

#ifndef MYCHROMA_H
#define MYCHROMA_H

// #undef ABS
#undef BLOB_Y
#undef VERSION
#undef min
#undef max
#include"saphira.h"

#undef max
#define max(a,b) ((a)>(b) ? (a) : (b))

#undef min
#define min(a,b) ((a)<(b) ? (a) : (b))

#ifdef __cplusplus
extern "C" {
#endif                          // __cplusplus


  // Pixels and angles for small ARC camera

#define DEG_TO_PIXELS 3.0     // approximately 3 pixels per degree
#define CAM_CENTER 140        // image center

#define BLOB_CONST sfBlobConst      // disparity x distance = BLOB_CONST
(100000.0)
#define LINE_CONST sfLineConst  // disparity x distance = LINE_CONST   (1000.0)
```

```
  // X,Y distance from camera image plane to object

#define DELTA_GRAD 5    // difference from camera and robot axe

#define BLOB_X(v) (BLOB_CONST/(float)((v)->h))
#define BLOB_Y(v,xx) (xx * tan((float)(v)->x * DEG_TO_RAD / DEG_TO_PIXELS) - xx
* DELTA_GRAD * DEG_TO_RAD / DEG_TO_PIXELS)

#define LINE_X(v) (LINE_CONST/(float)((v)->h))
#define LINE_Y(v) sin((float)((v)->x) * DEG_TO_RAD / DEG_TO_PIXELS)

  //void speak_start(void);
  //void speak_wait(void);
  //void speak_bye(void);
void find_blob(void);
void approach_blob(void);
void search_and_go_blob(void);
void test_control_proc(void);
void turn_off_everything(void);

void setup_vision_system(void);
void stop_vision_system(void);
int foundb(struct vinfo *v, int delta);
void draw_blobs(void);

#ifdef __cplusplus
}                               // extern "C"
#endif                          // __cplusplus
#endif
```

```
#-----------------------------------------------------------------
#
#  Follow My Socks
#
#
#   by G. Filippini and G. Gasparini
#
#   Last Revision Sept. 1999
#
#
#-----------------------------------------------------------------

CC = gcc
OFLAGS = -O -g -DUNIX -DLINUX

X11D = /usr/X11
BIND = $(HOME)/fms/
SRCD = $(HOME)/fms/
OBJD = $(HOME)/obj/
INCD = -I$(SAPHIRA)/handler/include/ -I$(X11D)include/ -Iinclude/
LIBD = -L$(SAPHIRA)/handler/obj/ -L$(X11D)lib/ -L/lib/ -Llib/
COLBERT = $(SAPHIRA)/colbert/
OAALIB = $(SAPHIRA)/oaa/agents/lib/
LIBS = -ldl  -lm -lc -lXm -lXt -lX11 -lXext -lXpm  -lsf

include $(SAPHIRA)/handler/include/os.h

CFLAGS = -g -D$(CONFIG) $(OFLAGS) $(INCD)
INCLUDE = -I$(INCD) -I$(X11D)include

# executable file
all: $(BIND)fms
      touch all

# sources
$(OBJD)fms.o : $(SRCD)fms.c
      $(CC) -c $(CFLAGS) $(SRCD)fms.c  $(INCLUDE) -o $(OBJD)fms.o
$(OBJD)pio.o : $(SRCD)pio.c
      $(CC) -c $(CFLAGS) $(SRCD)pio.c  $(INCLUDE) -o $(OBJD)pio.o
$(OBJD)mychroma.o : $(SRCD)mychroma.c
      $(CC) -c $(CFLAGS) $(SRCD)mychroma.c  $(INCLUDE) -o $(OBJD)mychroma.o

# if  more .c files must be compiled, copy the two rows above and change the
file name,
# then update the obj files raw below.

# object files
OBJS = $(OBJD)fms.o $(OBJD)pio.o $(OBJD)mychroma.o

# don't touch this!!!
$(BIND)fms : $(OBJS)
      $(CC) $(OBJS) -o $(BIND)fms  \
-L$(MOTIFD)lib $(LIBD) -lsf $(LIBS) -lc -lm

clean:
      -rm -f $(OBJD)* *~ $(SRCD)*~
      -rm -f $(BIND)fms  $(BIND)*.pgm $(BIND)*.ppm
```

Ringraziamenti

Gli autori desiderano ringraziare tutti i tesisti del Laboratorio di Robotica Avanzata (LDRA) ed in particolare Stefano Inelli per i preziosi consigli e il tempo loro dedicato.