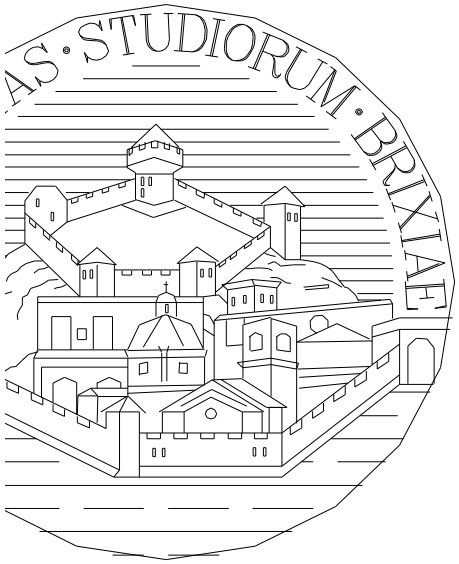
UNIVERSITÀ degli STUDI DI BRESCIA

Facoltà di Ingegneria



Progetto

PostMaster - Fase II

Corso di Robotica *Prof. Riccardo Cassinis*

REALIZZATO DA:

ANDREINI LUIGI

CHIARINI NICOLA

Anno Accademico 1998/'99

I realizzatori del progetto "***PostMaster – Fase II***":

Luigi Andreini

Nicola Chiarini

SOMMARIO:

OBIETTIVI DEL PROGETTO: 5

HARDWARE E SOFTWARE A DISPOSIZIONE: 6

Hardware: 6
Software: 6

COMPOSIZIONE DEL PROGRAMMA: 7

Dipendenze tra i vari files del programma: 7

LOGICA DI FUNZIONAMENTO DEL PROGRAMMA: 7

Premessa (PostMaster – Fase I): 7
PostMaster – Fase II: 8
Osservazioni: 8

FILE "PM2.C": 9

Costanti utilizzate: 10
Tasks utilizzati: 12
Funzioni: 14

FILE "COGNA.C" 16

Tasks utilizzati: 16

APPENDICI:

LISTATI: 17

File "pm2.c": 17
File "cogna.c": 40
File "cogna.h": 48
File "my-chroma.c": 49
Routines utili di uso comune: 54

STRUTTURE DI SAPHIRA NON DOCUMENTATE: 54

PROBLEMI RISCONTRATI E SOLUZIONI PROPOSTE: 56

Telecamera: 56
Cognachrome Vision System: 57
Osservazioni: 58

COGNACHROME VISION SYSTEM: 60

Variabili persistenti Telecamera NTSC / Sistema di Acquisizione: 60
Variabili persistenti Sistema di Apprendimento: 60
Nota: 61

BIBLIOGRAFIA E RIFERIMENTI: 61

NOTE: 61

OBIETTIVI DEL PROGETTO:

Il progetto "Postmaster - Fase II" si occupa della navigazione autonoma del robot Pioneer I assegnatoci (denominato "Speedy") in un ufficio qualsiasi del DEA (Dipartimento di Elettronica per l'Automazione) al fine di consegnare la posta dipartimentale (vedi progetto "Postmaster - Fase I"). Più precisamente Speedy, trovandosi inizialmente nel corridoio in prossimità della porta dell'ufficio obiettivo deve essere in grado di riconoscerla, entrarvi, effettuare una breve navigazione nell'ufficio per poi, consegnata la posta, "togliere il disturbo" e riportarsi nel corridoio.

Ai progettisti non è consentito modificare l'ambiente in alcun modo.

Per raggiungere l' obiettivo vengono utilizzate due strategie:

1. lettura dei sonar per rilevare la porta aperta;

2. interpretazione dei frames provenienti dal sistema visivo di Speedy (Cognachrome) per rilevare la porta chiusa.

Una volta identificata la porta, Speedy capisce se questa è aperta o chiusa: nel primo caso entra nell'ufficio e procede alla consegna della posta, nel secondo si posiziona davanti ad essa e ne richiede l'apertura.

HARDWARE E SOFTWARE A DISPOSIZIONE:

Hardware:

- Robot modello *Pioneer I* della *ActivMedia* denominato "**Speedy**" dotato di telecamera analogica NTSC e del sistema di visione Cognachrome della Newton Labs.

- PC portatile denominato "**Frost**" collegato a Speedy, usato per il controllo del robot.

- PC server denominato "**Golem**", usato per l'attività di sviluppo e debugging del software e per l'accesso ai servizi di rete della facoltà.

- Altro materiale in dotazione a LDRA (manuali, radio modem, connettori vari, ecc.).

Software:

- Ambiente *Saphira* della *SRI International*, usato per lo sviluppo software di questo progetto.

- Tool *mini-ARC* della *Newton Labs*, usato per la gestione del sistema di visione Cognachrome.

- Tool *xwpe*, usato per la scrittura del codice sorgente relativo al progetto.

- Utility *vplay*, usata per la riproduzione dei files audio in formato wav.

- Utility *xmixer*, usata per la calibrazione del sistema audio di Frost.

Nota: il sofware sopra citato è utilizzabile con il sistema operativo *Linux*.

COMPOSIZIONE DEL PROGRAMMA:

Il programma consta dei seguenti files:

"pm2.c": file principale in cui sono contenute le funzioni per la connessione al robot, le funzioni di inizializzazione, il main del programma e tutti i task che riguardano il funzionamento tramite sonars e la movimentazione del robot.

"cogna.c": file che governa le rutines della visione usate per la ricerca della porta.

"cogna.h": file che definisce le costanti usate dalle rutines di visione e le macro per la stima delle distanze dei blob riconosciuti dal sistema visivo.

"my-croma.c": file per l'inizializzazione del sistema di visione cognachrome e la definizione delle costanti e funzioni fondamentali di interfaccia con saphira;

Dipendenze tra i vari files del programma:

pm2.c: all
cogna.c: cogna.h my-chroma.c

LOGICA DI FUNZIONAMENTO DEL PROGRAMMA:

Premessa (PostMaster – Fase I):

Nella prima fase di questo progetto (PostMaster – Fase I) il robot viene portato in prossimità della porta dell'ufficio da raggiungere (a circa due metri da essa) e conosce da che parte essa si trova (destra oppure sinistra).

PostMaster – Fase II:

La prima operazione che viene effettuata dal robot è la ricerca di una porta aperta tramite l'utilizzo dei sonars laterali (TaskSonar), contemporaneamente viene anche eseguito TaskMisura che calcola la distanza percorsa dal robot.
Si possono a questo punto presentare due possibilità:

1. Viene individuata una porta aperta prima che la distanza percorsa sia pari al valore della costante LUNGHEZZA (definita in "pm2.c"), quindi viene interrotto TaskMisura e il robot si posiziona davanti ad essa;

2. Nessuna porta aperta è stata localizzata e la distanza percorsa ha raggiunto il valore DISTANZA: Il robot ricerca la porta che a questo punto per forza deve essere chiusa col sistema di visione, per questo torna indietro a circa metà della strada coperta, interrompe TaskSonar e TaskMisura, quindi chiama TaskVisione. Quest'ultimo task fa ruotare il robot finché non viene riconosciuto il blob associato al colore della porta, a questo punto il robot effettua un calcolo in base alle dimensioni del blob riconosciuto in modo da poter stimare la posizione della porta affinché si possa portare davanti ad essa. Se nessuna porta (blob) viene riconosciuta il programma finisce.

Osservazioni:

Durante tutta la movimentazione del robot rimane sempre attivo il behavior Avoid Collision garantendo che il robot eviti qualunque tipo di contatto con altri oggetti. Tuttavia durante l'esecuzione del suo compito il robot utilizza l'Avoid Collision con parametri differenti. Infatti durante la sua navigazione nel corridoio è molto importane che venga avvertita tempestivamente la presenza di qualunque tipo di ostacolo davanti ad esso mentre non deve essere data molta importanza a quelli

posti ai lati (potrebbero essere semplicemente persone che camminano), al contrario nell'attraversare la porta e nell'ufficio devono avere un maggior peso le rivelazioni dei sonars laterali rispetto a quelli frontali.

Il robot è messo in grado di navigare in qualunque tipo di ambiente e non solo nel dipartimento di elettronica poiché non carica alcuna mappa dall'esterno ma ne crea lui stesso una runtime con l'ausilio dei sonars; infatti il riconoscimento del corridoio e della porta avviene mediante interpretazione della pointlist e continuo riaggiornamento di quest'ultima durante l'avanzamento.

L'emissione di segnali sonori avviene mediante il sistema audio presente nel computer portatile portato in giro dal robot Speedy e consta della semplice esecuzione mediante il comando vplay di file di tipo wav appositamente creati e posti nella directory windows/system/wav.
In tale directory ci sono i files:

open.wav: richiede l'apertura della porta;
posta.wav: avviso dell'arrivo di posta;
ciao.wav: saluto prima di uscire da un ufficio.

Oltre a questi segnali sonori il robot emette tramite lo speaker montato su di esso dei veri e propri segnali acustici ogni qual volta completa con successo un' azione.
Il tipo di file supportato dal sistema è U-LAW 22050 Hz 8 bit Mono.
Qualora si vogliano eseguire altri files di tipo wav assicurarsi che siano di tale tipo. Se si vuole posizionare i files da eseguire in un altro percorso è necessario cambiare il path indicato nel listato del programma come argomento dell' esecuzione dell' istruzione VPLAY.

FILE "PM2.C":

In tale file si trovano tutte le costanti, tutte le funzioni e tutti i tasks utilizzati nella parte inerente la navigazione tramite sonars, inoltre si trovano le procedure

per la connessione di Saphira al robot e la procedura con la quale l'utente indica al robot da quale parte si trova la porta da ricercare.

Costanti utilizzate:

SPOSTVEL: Indica la velocità con la quale il robot si sposta nel corridoio durante la fase di ricerca della porta.

AVVVEL: Indica la velocità con la quale il robot attraversa la porta ormai individuata e con la quale si muove all'interno dell'ufficio da visitare; ovviamente questa è minore di SPOSTVEL in quanto in questa fase il robot si muove in uno spazio molto ristretto e pieno di ostacoli.

BUBBLEMAX: E' indicativamente il raggio dello scudo che il robot crea attorno a se e col quale viene attivato Avoid Collision quando il robot si trova nel corridoio (Tutti gli ostacoli che si trovano all'interno del cerchio di tale raggio centrato nel baricentro del robot vengono tempestivamente segnalati e il robot devia dalla propria traiettoria onde evitare un eventuale urto.

BUBBLEMIN: Ha la stessa funzione di BUBBLEMAX solo che viene utilizzata per attivare Avoid Collision all'interno dell'ufficio e durante l'attraversamento della porta; un valore maggiore di tale costante comporta un'enorme difficoltà al robot di attraversare la porta e spostarsi all'interno dell'ufficio, infatti verrebbero segnalati ostacoli ovunque con conseguente arresto del robot.

HFRSENS-LFRSENS-HSDSENS-LSDSENS:

Sono tutti parametri con i quali viene attivato il behavior Avoid Collision nei diversi casi secondo le diverse esigenze e

raprresentano rispettivamente massima e minima sensibilità dei sonars frontali e laterali.

MAXREAD: Indica la massima distanza entro cui la lettura dei sonars è ritenuta significativa.

OSTACLE: Indica la massima distanza entro cui la lettura dei sonars è ritenuta significativa quando il robot si trova ormai posizionato davanti alla porta in base a tale lettura viene deciso lo stato della porta (Aperto/Chiuso).

DIST: Indica la distanza alla quale il robot si posiziona davanti alla porta, tale valore è strettamente in dipendenza con quello indicato dalla costante Ostacle, infatti un valore di Ostacle minore di DIST potrebbe causare il mal funzionamento del programma poiché i sonars in tal caso potrebbero non rilevare la presenza della porta essendo questa situata al di fuori dal loro range di validità.

FERMA: Indica il tempo che il robot attende in ufficio prima di uscirne.

ATTESA: Indica il tempo che il robot attende davanti alla porta prima di richiederne una nuova apertura oppure di rinunciare ad entrare e passare così alla sconnessione.

NMRICHIESTE: Indica il numero di volte che il robot richiede l' apertura della porta mediante segnale acustico.

LUNGHEZZA: Indica la lunghezza del tratto che il robot deve percorrere prima di decidere che la porta non si riesce ad individuare con i sonars e dunque attivare il sistema visivo.

Tasks utilizzati:

MYCONNECTFN: Indica i processi base (già predefiniti in saphira) che si devono attivare alla connessione al robot, questi sono:
ControlProcs: controllo dei behaviors;
RegistrationProcs: Creazione ed riempimento della pointlist.
InterpretationProcs: Interpretazione della pointlist con consecutiva individuazione di porte, giunzioni e corridoi.

TASKSONAR: E' il task che parte subito dopo la connessione contemporaneamente al TaskMisura. Il suo compito è quello di scorrere ad intervalli regolari di tempo la pointlist creata dal robot durante il suo movimento trovarein essa eventuali corridoi e porte aperte, nel caso venga individuato quest'ultimo oggetto viene immediatamente rimosso il TaskMisura ed il robot viene riportato indietro in modo da posizionarsi davanti alla porta segnalata pronto per entrare. A questo punto viene iniziato il TaskEsplora ed il processo si interrompe.

TASKMISURA: Inizia parallelamente al TaskSonar e calcola in continuazione la distanza percorsa dal robot dal suo punto di partenza confrontandola immediatamente dopo col valore indicato dalla costante LUNGHEZZA, non il cammino risulta essere maggiore di tale valore viene rimosso il TaskMisura (il robot rinuncia alla ricerca di una porta aperta tramite sonars) ed inizia la ricerca per mezzo della visione, se tale ricerca ha dato esito negativo allora il robot si sconnette altrimenti viene posizionato davanti alla porta chiusa.

Va osservato che il Taskmisura ed il tasksonar partendo in parallelo e controllandosi reciprocamente sono pensati in modo tale che per forza uno dei due deve avere buon esito, il primo dei due che raggiunge tale stato (sfSUCCESS)

“killa“ il suo parallelo, prende il controllo del robot per portarlo davanti alla porta. Alla fine di questa prima fase dunque il robot o si trova non connesso in quanto non è stata individuata alcuna porta aperta o chiusa oppure si trova già davanti ad essa orientato pronto per entrare ed è dunque a questo punto che inizia la seconda fase che porta il robot a tentare di entrare nell’ufficio.

TASKESPLORA: Parte dalla certezza che il robot si trovi di fronte alla porta e che debba essere verificato lo stato di quest’ultima.
Come prima cosa viene attivato il behavior Avoid Collision con una sensibilità minore in modo da poter permettere al robot di attraversare la porta e muoversi all’interno dell’ufficio senza difficoltà qualora questa sia o venga aperta.
Il passo immediatamente successivo è quello di accertarsi dello stato della porta e dunque viene chiamata la funzione provaindoor in base al valore da questa tornato capisce se la porta è aperta.
In caso affermativo la attraversa, entra nella stanza contigua emette un segnale acustico per avvisare del suo arrivo, attende alcuni secondi, saluta emettendo un altro suono e ritorna nel corridoio. Il ritorno viene garantito dal TaskPollicino che memorizza periodicamente alcuni punti toccati dal robot durante la sua “ escursione “ nella nuova stanza.
In caso negativo attiva il TaskParla con il quale ne richiede l’apertura.

Alla fine di questa seconda fase dunque il robot ha terminato il suo compito e si trova dunque nuovamente nel corridoio pronto per una nuova “missione”.
All’interno del TaskEsplora si sono chiamati i seguenti tasks:

TASKPOLLICINO: crea in memoria una lista di punti che il robot tocca da quando inizia ad entrare nell’ufficio a quando inizia ad uscire. Questi vengono messi in una lista semplice di tipo LIFO (Last In First Out).

In tal modo si evita di capovolgerla quando il robot la utilizza come pista per ritornare nel corridoio.
La lista viene riempita di un nuovo elemento ad intervalli di tempo regolari.

TASKPARLA: Presiede all' esecuzione di file wav per la richiesta di apertura della porta. Infatti partendo dal presupposto che essa sia chiusa (dato di fatto visto il punto ed il modo in cui viene iniziato) esegue il file open.wav appositamente creato e sistemato nel percorso windows/system/wav mediante la chiamata all' istruzione vplay. Va osservato che la chiamata a questa non avviene in maniera diretta ma tramite un controllo if sull' istruzione fork. Quest' ultima altro non fa che duplicare il processo in esecuzione con una dipendenza di tipo padre_figlio, in tal modo si evita di mandare in crash la temporizzazione del programma.
Inoltre sia che il processo così creato abbia buon esito (viene trovato ed eseguito correttamente il file audio) sia che restituisca un errore il processo figlio alla fine si auto-elimina e
Il programma chiamante (processo padre) può normalmente continuare.
La richiesta di apertura viene ripetuta più volte(il numero è specificato con la costante NMRICHIESTE) qualora la porta non venga aperta e tra una richiesta e la successiva viene atteso un tempo pari al valore espresso in nS dalla costante ATTESA. In ogni caso sia che la porta venga aperta o meno il task si rimuove all'atto della sua interruzione e il controllo viene ridato al task chiamante.

Funzioni:

MYKEYFN: E' in assoluto la prima funzione che viene chiamata all' atto della connessione e serve per settare la vaiabile globale turn al valore di ± 90° in base alla direzione della porta da ricercare.
Questa variabile a sua volta viene utilizzata per far ruotare il robot quando si deve iniziare la ricerca tramite visione.
Viene inoltre settata un' altra variabile globale (dir) alla è possibile fare riferimento in qualunque punto del programma per conoscere la direzione della ricerca.

FINDARTIFACTBYTYPE: Ricerca nella pointlist l' artifatto il cui tipo è specificato dal parametro type e restituisce il puntatore ad esso.

ACFINDDOOR: Utilizzando la funzione appena sopra indicata ricerca la porta che sia anche situata nella posizione giusta rispetto al robot (destra oppure sinistra) e permette di ignorare quelle situate in posizione errata. Una volta individuata crea un punto esattamente davanti al suo centro ed attiva il behavior sfgotopos in modo da portare il robot davanti ad essa.

PROVAINDOOR: Mediante la lettura dei tre sonars frontali (#2, #3, #4)
Rileva se la porta è aperta oppure chiusa; nel primo caso restituisce il valore 1 altrimenti restituisce il valore 0.
C' è da osservare che si utilizza un if con al suo interno tre condizioni or sulla lettura dei tre sonars frontali. L'utilizzo della lettura di tre sonars anziché del solo centrale è stato determinato dal fatto che esso quando il robot è situato davanti alla porta si trova esattamente in corrispondenza della griglia di areazione delle porte degli uffici perciò non sempre può fornire valori corrispondenti alla reale situazione.

GOIN: Tale funzione può sembrare abbastanza singolare infatti riceve come parametro un puntatore a processo e restituisce quale valore di ritorno lo stesso puntatore al processo.

Scopo di questa infatti è quello di abbassare la sensibilità del behavior Avoid Collision in modo da permettere al robot di attraversare la porta e di navigare nell'ufficio.
L'ingresso del robot nell'ufficio viene eseguito con l'attivazione del comando sfSetPositon in modo tale da poter controllare ed eventualmente modificare di quanto il robot si deve "inoltrare" nell'ufficio semplicemente modificando il valore di AVANZATA costante globale e parametro del comando citato prima.

CREALISTA: Crea semplicemente una lista di punti in coordinate assolute toccati dal robot. La lista da essa realizzata è di tipo LIFO,
e come detto prima (vedi TaskPollicino) serve al robot come indicazione della strada da percorrere per poter tornare dall'ufficio al corridoio.

BEEP: Funzione che ogni volta che viene chiamata fa emettere un segnale acustico al robot.

FILE "COGNA.C"

In tale file si trova la logica di funzionamento del sistema di visione per la ricerca della porta chiusa.

Tasks utilizzati:

find_blob: questo task fa partire il behavior *sfTurnLeft* in modo da far ruotare il robot (verso sinistra) al fine di ricercare la porta. Si presentano due possibilità: se questa non viene trovata entro un angolo di 120° il processo termina nello stato sfFAILURE; nel caso invece di blob riconosciuto viene stimata la distanza della porta e viene creato un artefatto di tipo *point* (*blob_pt*) che memorizza il punto

che si trova di fronte ad essa alla distanza *FRONT_DIST*, il processo termina quindi nello stato *sfSUCCESS*.

approach_blob: questo task assume che la porta sia stata trovata (*blob_pt != NULL*) e fa partire il behavior *sfAttendAtPos* per raggiungere il punto in questione.

search_and_go_blob: questo task invoca *find_blob* per poi verificarne lo stato di uscita: se questo termina in *sfFAILURE* allora anch'esso termina in questo stato, se invece termina in *sfSUCCESS* viene invocato *approach_blob* dopo di che il task termina nello stesso stato di *approach_blob*.

APPENDICE

LISTATI:

File "pm2.c":

```
/*  PROGETTO POSTMASTER - FASE II  */

/*   REALIZZAZIONE EFFETTUATA DA:  */

/*         Andreini Luigi  024431  */
/*         Chiarini Nicola 024991  */


#include "saphira.h"
#include "cogna.c"
#include <stdlib.h>
#include <math.h>


/* COSTANTI */
```

```
#define sfNoTimeOut 0
#define sfNoSuspension 0
#define SPOSTVEL 250.0        // velocità di spostamento nel corridoio
#define AVVVEL 150            // velocità di spostamento nell' entrare nell'ufficio
#define MAXREAD 4000.0        // range di lettura valida dei sonar
#define BUBBLEMAX 400.0       // bubble massima di sicurezza dell' avoid collison utilizzato vnel moto nel corridoio
#define BUBBLEMIN 90.0        // bubble massima di sicurezza dell' avoid collison utilizzato vnel moto nel corridoio
#define OSTACLE 700.0         // validità lettura sonar dabvanti alla porta (decide se la porta è aperta o chiusa)
#define DIST 800.0            // distanza a cui il robot si mette di fronte alla porta prima di entrarvi
#define ATTESA 150            // tempo di attesa davanti alla porta dopo averne richiesto apertura
#define FERMA 100             // tempo di attesa in ufficio prima di uscire (numero va moltiplicato per 100 mS)
#define NMRICHIESTE 3         // numero richieste di apertura della porta prima di riununciare ad entrare
#define LUNGHEZZA 3000        // lunghezza che deve percorrere prima di attivare la ricerca tramite visione
#define HFRSENS 1.4           // alta sensibilità sonar frontale
#define LFRSENS 0.9           // bassa sensibilità sonar frontale
#define HSDSENS 1.4           // alta  sensibilità sonar laterale
#define LSDSENS 0.9           // bassa sensibilità sonar laterale
#define POSIZIONATI 500.0     // spostamento da effettuare per radrizzare la rotella pilota


/* STRUTTURE DATI */


struct listpt
      {
       float px;
       float py;
       float pth;
       struct listpt *prev;
      };
```

```
struct listpt *ListEntry=NULL;

typedef enum {acLEFT, acRIGHT, acANY} acSide;



/* VARIABILI GLOBALI */


int  turn; // può assumere +90 oppure -90 gradi e decide la direzione in cui
girarsi per attivare la visione
char tasto; // valore tornato dalla funzione mykeyfunction
door *d;



/* PROTOTIPI DI FUNZIONI */


void myConnectFn(void);
void myStartupFn(void);
int  myKeyFn(int ch);
static point *findArtifactByType(int type);
static door *acFindDoor( acSide dir);
void TaskSonar(void);
static sfprocess * vaidavanti(door *porta);
int provaindoor(void);
void TaskParla(void);
static sfprocess *goin(sfprocess *behavior);
void crealista(struct listpt **base, struct listpt *elemento);
void TaskPollicino(void);
void TaskMisura(void);
void TaskEsplora(void);
void beep(void);



/* VERIFICA DEL SISTEMA OPERATIVO SOTTO CUI SI LAVORA */


#ifdef IS_UNIX
void main(int argc, char **argv)
```

```
#endif
#ifdef MS_WINDOWS
int PASCAL WinMain (HANDLE hInst, HANDLE hPrevInstance, LPSTR lpszCmdLine, int
nCmdShow)
#endif
{

  sfKeyProcFn(myKeyFn);
  sfOnStartupFn(myStartupFn);
  sfOnConnectFn(myConnectFn);


/* INIZIO DEI MICRO-TASK DI SAPHIRA  */

  printf("Starting...\n");

#ifdef IS_UNIX
  sfStartup(1);                    // modalità asincrona
#endif
#ifdef MS_WINDOWS
  sfStartup(hInst, nCmdShow, 1);  // modalità asincrona
#endif

  while (!sfIsExited) sfPause(1000);

  if (sfIsConnected)
    {
    sfDisconnectFromRobot();  // mi assicuro di essere sconnesso dal robot prima
di uscire...
    sfPause(1000);
    }

#ifdef MS_WINDOWS
  return 0;
#endif
}


void
myStartupFn(void)
{
  sfSetDisplayState(sfDISPLAY, 2);  /* set it to 5 Hz */
```

```
}

void
myConnectFn(void)        /* start those processes */
{

  sfInitControlProcs();          // for behavior control
  sfInitRegistrationProcs();     // register the robot using sensed artifacts
  sfInitAwareProcs();            // figure out where we are
  sfInitInterpretationProcs();    // find walls and doors (NB: Saphira 6.1 only
have walls)
  sfInitSpecialProcs();


  sfInitProcess(test_control_proc,"test it");   // from cogna.c


  sfSMessage("INTRODURRE DA CHE PARTE SI TROVA LA PORTA RISPETTO AL ROBOT");
  sfSMessage("d=destra s=sinistra");
}



/* DEFINIZIONE DEI TASK UTILIZZATI */


void TaskSonar(void)
/* CERCA IL CORRIDOIO E LA PORTA MEDIANTE INTERPRETAZIONE DELLA LETTURA DEI
SONAR SE LA PORTA VIENE TROVATA
    INTERROMPE IL TASKMISURA, PORTA IL ROBOT DAVANTI AD ESSA ED ATTIVA IL
TASKESPLORA */

{

  static point *goal;
  static point* MyGoal=NULL;
  static door*  MyDoor=NULL;
  static corridor* MyCorridor=NULL;
  static sfprocess *processo = NULL;
  static sfprocess *prova = NULL;
  static sfprocess *collisione = NULL;
```

```
  static float angle;

 switch(process_state)
  {
  case sfINIT:
    if(sfIsConnected)
     {
      sfResetRobotVars();
      collisione=  sfStartBehavior(sfAvoidCollision,  "AvoidCollision",
sfNoTimeOut,0, sfNoSuspension,
                                HFRSENS,             // sensibilità frontale(0.5
- 3.0)
                                                          LSDSENS,
laterale(0.5 - 3.0)
                                                    //     velocità     di
rotazione(slow: 4.0 - fast: 10.0)
                                           );            BUBBLEMAX  //MAXandoff  (mm):
security bubble
      sfFrontMaxRange = MAXREAD;
      process_state=10;
     }

  break;

  case 10:
    beep();
    sfSMessage(" SETTATO VEL A %f mm/s " ,SPOSTVEL);
    processo        =        sfStartBehavior(sfConstantVelocity,
"CostantVelocity",0,0,0,SPOSTVEL);
    process_state = 15;
  break;

  case 15:
    MyCorridor = (corridor *)findArtifactByType(sfCORRIDOR);
    if (MyCorridor!=NULL)
      {
       sfKillIntention(processo);
       beep();
       sfSMessage("  TROVATO  CORRIDOIO  X:  %f;  Y:  %f;TH:  %f  ",MyCorridor-
>x,MyCorridor->y,MyCorridor->th);
       process_state=20;
      }
```

```
  break;

  case 20:
    if (tasto =='d' )
      MyDoor = acFindDoor(acRIGHT);
    else
      MyDoor = acFindDoor(acLEFT);
    if (MyDoor!=NULL)
      {
       sfKillIntention(processo);
       sfSetVelocity(0);
       beep();
       sfSMessage("    TROVATO    PORTA    XT:H:  %f%f  Y:",MyDoor->x,MyDoor-
>y,MyDoor->th);
       angle=-(MyDoor->th);
       sfRemoveTask("TaskMisura");
       process_state=30;
      }
  break;

  case 30:
    sfMessage(" VADO DAVANTI ALLA PORTA ");
    beep();
    processo=vaidavanti(MyDoor);
    process_state=31;
  break;

  case 31:
    if(processo->state!=sfSUCCESS)
     {
      process_state=31;
     }
    else
     {
      beep();
      sfMessage("PUNTO RAGGIUNTO");
      sfKillIntention(processo);
      process_state=32;
     }
  break;

  case 32:
```

```
    beep();
    sfSMessage("MI POSIZIONO A %f GRADI",angle);
    sfSetHeading(angle);
    process_state=33;
  break;

  case 33:
    if(sfDoneHeading(5)==0)
     {
      process_state=33;
     }
    else
     {
      beep();
      sfMessage("HO COMPLETATO IL GIRO ");
      process_state=40;
     }
  break;

  case 40:
    sfKillIntention(collisione);
    process_state=sfINTERRUPT;
    sfInitProcess(TaskEsplora,"TaskEsplora");
  break;

  case sfINTERRUPT:
    sfMessage("FINE TASKSONAR");
    sfRemoveTask("TaskSonar");
  break;

  case sfREMOVE:
    sfKillIntention(collisione);
  break;

  }
}


void TaskParla(void)
  /* RICHIEDE L'APERTURA DELLA PORTA TRAMITE L' ESECUZIONE DEI FILE .WAV QUANDO
ESSA E' CHIUSA
```

```
     OGNI VOLTA CHE VIENE EFFETTUATA TALE RICHIESTA IL ROBOT ATTENDE UN TEMPO
PARI ALLA COSTANTE ATTESA E POI
     MEDIANTE LA FUNZIOME PROVAINDOOR VERIFICA CHE LA PORTA SIA STATA APERTA
     LA RICHIESTA QUALORA NON ABBIA SUCCESSO VIENE ESEGUITA UN NUMERO DI VOLTE
INDICATO DALLA COSTANTE NMRICHIESTE */

{
 static float nmrichieste=0;

 switch(process_state)
  {

  case sfINIT:
    process_state=10;
  break;

  case 10:
    sfMessage(" MI PUO' APRIRE LA PORTA PER FAVORE ? ");
    if(fork()==0)
      system("vplay -s 22050 /dos/windows/system/wav/open.ok");
    nmrichieste=nmrichieste+1;
    process_state=20;
  break;

  case 20:
    sfSuspendSelf(ATTESA);
  break;

  case sfRESUME:
  case 21:
    if(provaindoor()==0)
     {
      if(nmrichieste < NMRICHIESTE )
       {
        process_state=10;
       }
      else
       {
        sfMessage("NESSUNO MI APRE E IO MI FERMO QUI ");
        beep();
        beep();
        sfDisconnectFromRobot();
```

```
        process_state=sfINTERRUPT;
        sfRemoveTask("TaskParla");
       }
     }
    else
     {
      sfRemoveTask("TaskParla");
      process_state=sfINTERRUPT;
     }
  break;

  case sfINTERRUPT:
    sfRemoveTask("TaskParla");
  break;

  }
}


void TaskPollicino(void)
   /* MEMORIZZA I PUNTI  TOCCATI  DAL  ROBOT  DA  QUANDO  INIZIA  IL  TASK  A  QUANDO
FINISCE CREANDO UNA LISTA SEMPLICE
       VIENE  UTILIZZATA  PER  RICOSTRUIRE  I  PUNTI  CHE  IL  ROBOT  DEVE  TOCCARE  PER
POTER USCIRE DALL' UFFICIO  */

{
 static int conta;
 static struct listpt *tp;


 switch(process_state)
  {

  case sfINIT:
    tp=malloc(sizeof(struct listpt));
    tp->px=sfRobot.ax;
    tp->py=sfRobot.ay;
    tp->pth=(sfRobot.ath+180);
    crealista(&ListEntry,tp);
    conta=0;
    process_state=20;
  break;
```

```
  case 20:
    if(conta<50)
     {
      conta=conta+1;
      process_state=20;
     }
    else
      process_state=21;
  break;

  case 21:
    tp=malloc(sizeof(struct listpt));
    tp=malloc(sizeof(struct listpt));
    tp->px=sfRobot.ax;
    tp->py=sfRobot.ay;
    tp->pth=(sfRobot.ath+180);
    crealista(&ListEntry,tp);
    conta=0;
   process_state=20;
 break;

  case sfINTERRUPT:
    sfRemoveTask("TaskPollicino");
  break;
  }
}


void TaskMisura(void)
   /* MISURA LA DISTANZA PERCORSA DAL ROBOT DAL PUNTO DI PARTENZA ALLA SUA
POSIZIONE ATTUALE E LA CONFRONTA CON
     LA COSTANTE DISTANZA QUANDO IL VALORE E' SUPERIORE ALLA COSTANTE INTERROMPE
IL TASKSONAR ED INIZIA IL TASK VISIONE */

{

 static point start;
 static point posizione;
 static float percorrenza,radice;
 static sfprocess *ctrlurto = NULL;
```

```
 switch(process_state)
 {

  case sfINIT:
    start.x=sfRobot.ax;
    start.y=sfRobot.ay;
    start.th=sfRobot.ath;
    process_state=20;
  break;

  case 20:
    posizione.x=sfRobot.ax;
    posizione.y=sfRobot.ay;
    process_state=25;
  break;

  case 25:
    radice = ( (posizione.x - start.x) * (posizione.x - start.x) ) + (
(posizione.y - start.y) * (posizione.y - start.y));
    process_state=30;
  break;

  case 30:
    percorrenza = sqrt(radice);
    process_state=40;
  break;

  case 40:
    if(percorrenza<LUNGHEZZA)
     {
      process_state=20;
     }
    else
     {
      beep();
      sfMessage(" ESAMINATO TUTTO E NON HO TROVATO ALCUNA PORTA APERTA");
      sfRemoveTask("TaskSonar");
      sfSetVelocity(0);
      ctrlurto=sfStartBehavior(sfAvoidCollision, "A v o i d      Collision",
sfNoTimeOut,0, sfNoSuspension,
                           HFRSENS,          // sensibilità frontale (0.5 -
3.0)
```

```
                                                    // sensibilità laterale (0.5 -
3.0)
                                6.0,                // velocità di rotazione(slow: 4.0 -
fast: 10.0)
                                                    );          // security bubble(mm): security
bubble
      beep();
      sfMessage(" TORNO INDIETRO A CERCARE UNA PORTA CHIUSA ");
      process_state=50;
     }
  break;

  case 50:
    sfSetDHeading(180);
    process_state=55;
  break;

  case 55:
    if(sfDoneHeading(10)==1)
     {
      sfSetPosition(LUNGHEZZA/2);
      process_state=60;
     }
    else
     {
      process_state=55;
     }
  break;

  case 60:
    if(sfDonePosition(100)==1)
      {
      sfSetDHeading(turn);
      process_state=65;
      }
    else
      process_state=60;
  break;

  case 65:
    if(sfDoneHeading(10)==1)
      {
```

```
      sfKillIntention(ctrlurto);
      sfMessage(" INIZIO VISIONE ");
      beep();
      SearchTheDoor();
      process_state=70;
     }
   else
     {
      process_state=65;
     }
 break;

 case 70:

   if(  sfTaskFinished("search door")  )  //  ||
(sfGetProcessState(sfFindProcess("search door")) == sfINTERRUPT) ||
(sfGetProcessState(sfFindProcess("search door")) == sfF) )
     {
     sfMessage("FINITO TASK VISIONE ");
     beep();
     process_state=75;
     }
   else
     process_state=70;
 break;

 case 75:
   if(sfGetProcessState(sfFindProcess("search door")) != sfSUCCESS)
    {
     sfMessage("TUTTI I TENTATIVI SONO FALLITI NON ESISTE PORTA");
     sfDisconnectFromRobot();
     sfRemoveTask("search door");
     sfRemoveTask("TaskMisura");
    }
   else
    {
     process_state=sfINTERRUPT;
     turn_off_everything();
     sfRemoveTask("search door");
     sfInitProcess(TaskEsplora, "TaskEsplora");
    }
 break;
```

```
  case sfINTERRUPT:
    sfKillIntention(ctrlurto);
    sfRemoveTask("TaskMisura");
  break;
 }
}


void TaskEsplora(void)
/* INIZIA L' ESPLORAZIONE ALL' INTERNO DELL' UFFICIO
   COME PRIMA COSA VIENE VERIFICATO CHE NON CI SIANO OSTACOLI DAVANTI AL ROBOT
IN TAL CASO IL ROBOT VIENE PORTATO
   ALL' INTERNO DELL' UFFICIO VIENE FATTO ATTENDERE IN ESSO UN TEMPO PARI ALLA
COSTANTE FERMA E POI VIENE FATTO
   RIUSCIRE NEL CORRIDOIO PRONTO PER UNA NUOVA MISSIONE */

{
  static point *goal;
  static sfprocess *prova = NULL;
  static sfprocess *urto = NULL;


switch(process_state)
  {
  case sfINIT:
    urto=sfStartBehavior(sfAvoidCollision, "Stop Collision", sfNoTimeOut,0,
sfNoSuspension,
                         LFRSENS,   // sensibilità frontale (0.5 - 3.0)
                         LSDSENS,   // sensibilità laterale (0.5 - 3.0)
                                 6.0,              // velocità di rotazione (slow: 4.0 -
fast: 10.0)
                         BUBBLEMIN);    //  standoff (mm): security bubble
    process_state=20;
  break;

/* LE ISTRUZIONI DAL CASE 20 AL CASE 40 SERVONO SOLO PER RADDRIZZARE LA ROTELLA
PILOTA
   E PERMETTERE AL ROBOT DI ENTRARE NON OBLIQUO */

  case 20:
    sfSetPosition(-POSIZIONATI);
```

```
  process_state=25;
break;

case 25:
  if(sfDonePosition(50)==1)
    {
    sfSetPosition(POSIZIONATI);
    process_state=30;
    }
  else
    process_state=25;
break;

case 30:
  if(sfDonePosition(50)==1)
    {
    process_state=40;
    }
  else
    process_state=30;
break;

case 40:
  if(provaindoor()==1)
   {
    beep();
    sfSMessage(" O.K. NESSUN OSTACOLO: ENTRO!");
    sfMessage(" INIZIO TASK POLLICINO");
    sfInitProcess(TaskPollicino,"TaskPollicino");
    urto  =  goin(urto);
    process_state=42;
   }
  else
   {
    sfSMessage("ATTENZIONE PORTA CHIUSA IMPOSSIBILE ENTRARE ");
    process_state=41;
    sfInitProcess(TaskParla,"TaskParla");
   }
break;

case 41:
  if (sfTaskFinished("TaskParla"))
```

```
    {
     if(provaindoor()==1)
       {
        sfInitProcess(TaskPollicino,"TaskPollicino");
        urto  =  goin(urto);
        process_state=42;
       }
     else
       {
        sfMessage("NESSUNO MI APRE E IO MI FERMO QUI ");
        beep();
        process_state=sfINTERRUPT;
       }
     }
    else
     {
      process_state=41;
     }
  break;

  case 42:
    if(sfDonePosition(50)==0)
     {
      process_state=42;
     }
    else
     {
      sfMessage("SONO ENTRATO! ");
      beep();
      sfKillIntention(urto);
      urto= sfStartBehavior(sfStopCollision, "Stop Collision", sfNoTimeOut,0,
sfNoSuspension,
                                          ,     // sensibilità frontale(0.5 -
3.0)
                                                // sensibilità laterale(0.5 -
3.0)
                                             BUBBLEMIN // standoff (mm): security
bubble
       // EVENTUALE VISITA DELLA STANZA ATTENZIONE AVOID COLLISION
      sfRemoveTask("TaskPollicino");
      process_state=45;
     }
```

```
  break;

  case 45:
    sfSetVelocity(0);
    sfMessage(" POSTA! POSTA!");
    if(fork()==0)
      system("vplay -s 22050 /dos/windows/system/wav/posta.ok"); // avviso posta
    process_state=48;
  break;

  case 48:
    sfSuspendSelf(FERMA);
  break;

  case sfRESUME:
    sfMessage("CIAO IO  ESCO ");
    if(fork()==0)
      system("vplay -s 22050 /dos/windows/system/wav/ciao.ok"); //   saluto per
uscire
    process_state = 50;
  break;

  case 50:
    if(ListEntry!=NULL)
     {
      goal=sfCreateGlobalPoint(ListEntry->px,ListEntry->py,ListEntry->pth);
      sfAddPointCheck(goal);
      ListEntry=ListEntry->prev;
      sfSMessage("posizionato punto");
      beep();
      process_state=52;
     }
    else
     {
      sfSMessage("ARRIVATO BYE BYE ");
      sfDisconnectFromRobot();
      beep();
      beep();
      sfKillIntention(urto);
      process_state=sfINTERRUPT;
     }
  break;
```

```
  case 52:
    prova=sfStartBehavior(sfGoToPos, "GoToPos",0,0,0,
           SPOSTVEL,
           goal,
           150.0);
    sfSMessage("STO TORNANDO ALLA POSIZIONE: X %f, Y %f, Th %f",goal->x,goal-
>y,goal->th );
    beep();
    process_state=53;
  break;

  case 53:
    if(prova->state!=sfSUCCESS)
     {
      process_state=53;
     }
    else
     {
      sfMessage("GOAL RAGGIUNTO");
      beep();
      sfKillIntention(prova);
      sfRemPoint(goal);
      process_state=50;
     }
  break;

  case sfINTERRUPT:
    sfRemoveTask("TaskEsplora");
  break;
  }
}



/* FUNZIONI UTILIZZATE ALL' INTERNO DEI TASK  */


int  myKeyFn(int ch)
/* RICHIEDE ALL' UTENTE DI INDICARE DA CHE PARTE SI TROVA LA PORTA RISPETTO AL
ROBOT */
```

```
{

 switch(ch)
  {
   case 'd':
   case 'D':
     sfMessage("RICERCO LA PORTA ALLA MIA DESTRA");
     sfMessage(" INIZIO TASK ");
     sfInitProcess(TaskSonar,"TaskSonar");
     sfInitProcess(TaskMisura,"TaskMisura");
     turn=50;
     tasto='d';
   return 1;

   case 's':
   case 'S':
     sfMessage("RICERCO LA PORTA ALLA MIA SINISTRA");
     sfMessage(" INIZIO TASK ");
     sfInitProcess(TaskSonar,"TaskSonar");
     sfInitProcess(TaskMisura,"TaskMisura");
     turn=-130;
     tasto='s';
   return 1;

   default:
     sfMessage("ATTENZIONE s=SINISTRA d=DESTRA");
   return 1;
  }
 return 1;
}


static point *findArtifactByType(int type)
/* SCORRE LA POINTLIST E TROVA GLI ARTEFATTI DI TIPO SPECIFICATO DA type */

{
    point **p = (point **)sfPointList->p;
    int i, check = TRUE;

    while(check)
    {
      check = FALSE;
```

```
      for (i=0; i<sfPointList->n; i++)
          if(p[i]->cat == ARTIFACT && p[i]->matched > 500 &&
            p[i]->matched < 100000 && !check)
          {
            switch(p[i]->type)
            {
                case CORRIDOR:
                case DOOR:
                  sfRemPoint(p[i]);
                  check = TRUE;
                  break;

                default:
                  break;
            }
          }
    }

    for (i=0; i<sfPointList->n; i++)
      if(p[i]->cat == ARTIFACT && p[i]->type == type && p[i]->viewable)
          return(p[i]);
    return(NULL);
}


static door *acFindDoor(acSide dir)
/*  TROVA  UNA  PORTA  NELLA  POINTLIST  SITUATA  RISPETTO  AL  ROBOT  SULLA  PARTE
INDICATA DA acSide */

{
  d = (door *)findArtifactByType(sfDOOR);

  if(d == NULL )
    return( NULL);
  else
    if ( dir == acANY )
     return(d);
    else
      if ((dir == acLEFT)&&(d->th>=180))
       return(d);
      else
```

```
       if ((dir == acRIGHT)&&(d->th<=180))
        return(d);
       else
         return( NULL);
}


static sfprocess * vaidavanti(door *porta)
/* UNA VOLTA INDIVIDUATA UNA PORTA APERTA TRAMITE I SONAR QUESTA PROCEDURA PORTA
IL ROBOT A POSIZIONARSI DAVANTI AD ESSA*/

{

  static point *p1,*p2;
  static sfprocess *processo=NULL;

  p2=sfCreateLocalPoint(0.0,0.0,0.0);
  p1 = sfCreateLocalPoint(porta->x, porta->y,0.0);
  if(porta->th<180)
   sfPointMove(p1,0,DIST,p2);//PORTA SULLA DX DEL ROBOT
  else
   sfPointMove(p1,0,-DIST,p2);//PORTA SULLA SX DEL ROBOT
  p2->th=porta->th;
  sfAddPoint(p2);
  beep();
  sfSMessage(" INIZIO GOTOPOS");
  processo=sfStartBehavior(sfGoToPos, "GoToPos",0,0,0,
           SPOSTVEL,
           p2,
           100.0);
  sfSMessage("STO RAGGIUNGENDO LA POSIZIONE: X %f, Y %f, Th %f",p2->x,p2->y,p2-
>th );
  beep();
  return processo;
}


int provaindoor(void)
/*TESTA LA PRESENZA DI O =STACOLI PRIMA DI ENTRARE: via libera=1 ostacolo=0*/

{
```

```
if((sfSonarRange(3)<OSTACLE)||(sfSonarRange(2)<OSTACLE)||(sfSonarRange(4)<OSTACL
E))
  {
   sfMessage("PROVAINDOOR HA TROVATO UN OSTACOLO E RITORNA 0");
   return 0;
  }
 else
  {
   sfMessage("PROVAINDOOR HA DATO VIA LIBERA E RITORNA 1");
   return 1;
  }
}


static sfprocess *goin(sfprocess *behavior)
/* ABBASSA LA BUBBLE DI AVOID COLLISION ENTRA RIALZA LA BUBBLE DI AVOID COLL */

{
 static float  avanzata;

 sfKillIntention(behavior);
 avanzata =  DIST+DIST;
 behavior= sfStartBehavior(sfAvoidCollision, "StopCollision", sfNoTimeOut,0,
sfNoSuspension,
                                          //FRONT SENS,  sensibilità frontale (0.5 -
3.0)
                                      ,     // sensibilità laterale LEFT SENS (0.5 -
3.0)
                                6.0,          // velocità di rotazione(slow: 4.0 -
fast: 10.0)
                                             //  standoff  (mm):  security     B
bubble
 beep();
 sfSMessage("ATTENZIONE STO ENTRANDO PISTA!!!");
 sfSetMaxVelocity(AVVVEL);
 sfSetPosition(avanzata);
 beep();
 return behavior;
}
```

```
void crealista(struct listpt **base, struct listpt *elemento)
/* CREA UNA LISTA DI PUNTI IN COORDINATE ASSOLUTE TOCCATI DAL ROBOT  */

{

 if(*base==NULL)
   {
    elemento->prev = NULL;
    *base=elemento;
    sfSMessage("MESSO PUNTO BASE NELLA LISTA" );
    beep();
   }
 else
   {
    elemento->prev=*base;
    *base=elemento;
    sfSMessage("MESSO NUOVO PUNTO NELLA LISTA ");
    beep();
   }
}


void beep(void)
/* EMETTE UN SUONO TRAMITE LO SPEAKER MONTATO SUL ROBOT */
{
  sfRobotComStrn(sfCOMSAY,"\25\003",2);
}
```

File “cogna.c”:

```
/* ################################################
 * "cogna.c" based on <btech.c>
 * ################################################ */



#include "saphira.h"
#include "cogna.h"
```

```
#define DELTA 25  /* camera window size */

#define DIST_OUT 600



point *blob_pt;
extern float sfBlobConst; /* from chroma.c */

extern char tasto;



/*
 * set up initial behaviors and intentions
 */



/* Turn until facing a blob, channel A */
/* Starting code for activity find_blob  */
void
find_blob(void)
{
  static sfprocess *_sf_p;
  static sfprocess *p = NULL;
  int x;
  float xx, yy;

  float door_th = 0;

  switch(process_state)
    {

    case sfINIT:
    case 20:
      p = sfStartBehavior (sfTurnLeft, "sfTurnLeft", 80, 2 , 0,
                0.5 );   /* Turn velocity: from 0 (low) to 1 (high) */
      process_state = 21;
```

```
      break;

    case 21:
      if (sfFinished (p ) )
        {
        process_state = 22; break;  /* exit point case 21 */
        }

      {
      x = found_blob (_process_params [ 0 ] . i , DELTA ) ;
      if ( x > - 500  )
        {
        sfSMessage ("Found a blob!!! %d" , x ) ;
        if (_process_params[0].i == 0)
          {
          xx = BLOB_X (&sfVaInfo );
          yy = BLOB_Y (&sfVaInfo, xx );
          }
        else
          {
          xx = BLOB_X (&sfVbInfo );
          yy = BLOB_Y (&sfVbInfo, xx );
          }

        if (tasto == 'd') door_th = 90;
        else if (tasto == 's') door_th = -90;

        blob_pt = sfCreateLocalPoint (xx - (DIST_OUT*abs(cos(sfRobot.ath))), yy
- (DIST_OUT*abs(sin(sfRobot.ath))), door_th);
        sfAddPoint (blob_pt ) ;
        sfKillIntention (p ) ;
        p = NULL ;
        process_state = sfSUCCESS; break; /* exit point case 21 */
      }      /* end if ( x > - 500  ) */
      }

      process_state = 21 ;
      break;

    case 22:
      sfKillIntention (p ) ;
      p = NULL ;
```

```
      sfMessage ("No blob found!!!" ) ;
      process_state = sfFAILURE; break;
      break;

    case sfINTERRUPT:
      if (p ) sfKillIntention (p ) ;
      sfSuspendSelf(0); break;
      break;

    case sfRESUME:
      process_state = 20; break;
      process_state = sfSUCCESS;

    }
  /* end switch */

}
/* Ending code for activity find_blob  */



SFPROCESS StopApproach = NULL;

void                    /* assumes we've found one... */
approach_blob(void)
{
  static sfprocess *p = NULL;
  int x;
  float xx, yy;

  switch(process_state)
    {
    case sfINIT:
    case sfRESUME:
      process_state = 20;
      break;

    case 20:
      p = sfStartBehavior(sfAttendAtPos,"approach blob", 300, 2, 0,
               200.0,   /* velocity (mm/s) */
               blob_pt, /* goal position */
               100.0);  /* success radius (mm) */
```

```
      process_state = 10;
      break;

    case 10:
      /*
      if (found_blob(_process_params[0].i, DELTA) > -500)
          {
        xx = BLOB_X(&sfVaInfo);
        yy = BLOB_Y(&sfVaInfo,xx);
        blob_pt->x = xx + FRONT_DIST;
        blob_pt->y = yy;
        sfSetGlobalCoords(blob_pt);

        // ------------------------------------------------------- //
        blob_pt = sfCreateLocalPoint (xx + FRONT_DIST, yy , 0.0 ) ;
        sfAddPoint (blob_pt ) ;
        // ------------------------------------------------------- //
          }
      */

         if (((SFPROCESS *)_process_params[1].p)->bcl->act_v  >  .3  /*  when
we have stop collision */
      if (sfFinished(p))      /* we're stopping here */
        {
        process_state = sfSUCCESS;
        blob_pt->viewable = 0;
        break;
      }
    }
}


void                    /* find it, then go to it */
search_and_go_blob(void)
{
  static sfprocess *p = NULL;
  switch(process_state)
    {

    case sfINIT:
```

```
       process_params[1].p = sfStartBehavior(sfStopCollision,"stop collision",
0, 0, 0,
                0.4,     // front sensitivity
                0.4,     // side sensitivity
                flakey_radius + 50.0 // standoff
      );

      sfStartBehavior(sfStop, "stop", 0, 4, 0);
      sfSuspendSelf(0);
      break;

    case sfRESUME:
      process_state = 20;
      break;

    case 20:
      p = sfInitIntention(find_blob, "find a blob", 0,
               sfINT, _process_params[0].i,
               sfEND);
      process_state = 30;
      break;

    case 30:

      if (p->state == sfSUCCESS)    /* did it, now go get it */
      {
        sfKillIntention(p);
        p = NULL;
        process_state = 40;
        break;
      }      // else

      if (sfFinished(p)) /* sfFAILURE or sfTIMEOUT */
      {
        process_state = p->state;
        sfKillIntention(p);
        p = NULL;

        // sfSMessage("Failure or TimeOut, find blob sate: %i", process_state);

        break;
      }
```

```
      break;

    case 40:
      p = sfInitIntention(approach_blob, "go get it", 0,
               sfINT, _process_params[0].i,
               sfPTR, _process_params[1].p,
               sfEND);
      process_state = 50;
      break;

    case 50:
      if (sfFinished(p))
      {
        sfKillIntention(p);
        p = NULL;
        sfSuspendSelf(0);

        process_state = sfSUCCESS;  /* search door OK */
        sfMessage("Search door OK!!!");
        break;
      }
      break;

    case sfINTERRUPT:
      if (p) sfKillIntention(p);
      p = NULL;
      sfSuspendSelf(0);

      sfMessage("Search door interrupted!!!");
      sfSMessage("Search door sate: %i", process_state);

      break;
    }
}



void setup_vision_system(void);     /* from chroma.c */
void draw_blobs(void);        /* from chroma.c */
```

```
void
test_control_proc(void)
{
  switch(process_state)
    {
    case sfINIT:
      setup_vision_system();  /* set all vision parameters */

      sfStartBehavior(sfStop, "stop", 0, 4, 0);

      sfInitIntention(draw_blobs, "draw blobs",0,sfEND);

      sfInitIntention(search_and_go_blob, "search door", 0, sfINT, 0, sfEND);
      // sfInitIntention(search_and_go_blob, "search blue", 0, sfINT, 1, sfEND);

      setup_vision_system();  /* set all vision parameters, just in case */
      process_state = 20;
      break;

    }
}




/*
 * Startup sequence
 */

/*
void
myConnectFn(void)
{
  sfInitControlProcs();
  sfInitProcess(test_control_proc,"test it");
}
*/
```

```
void
turn_off_everything(void)
{
  sfSetProcessState(sfFindProcess("search door"), sfINTERRUPT);
}



void
SearchTheDoor(void)
{
  turn_off_everything();
  sfSetProcessState(sfFindProcess("search door"), sfRESUME);
}
```

File “cogna.h”:

```
/*
 * Translation from camera to RW coordinates
 */



/* 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)
```

File “my-chroma.c”:

```
/*
##############################################################################
"my-chroma"  ---  functions  for  interpreting  chroma  vision  results
##############################################################################
*/



#include "saphira.h"



/*
 * Set up the vision parameters
 */

void
setup_vision_system(void)
{
  sfRobotComStr(VISION_COM,"pioneer_a_mode=2");        /*  2  =  blob  bb  (bounding
box) mode */
  sfRobotComStr(VISION_COM,"pioneer_b_mode=2");      /* 1 = line mode */
  sfRobotComStr(VISION_COM,"pioneer_c_mode=2");      /* 0 = blob mode */
```

```
  sfRobotComStr(VISION_COM,"line_bottom_row=10");    // 180 //
  sfRobotComStr(VISION_COM,"line_num_slices=18");    //   6 //
  sfRobotComStr(VISION_COM,"line_slice_size=10");     //   10 //   /* number of
rows in each slice */
  sfRobotComStr(VISION_COM,"line_min_mass=10");       //   10 //   /* how wide
slices have to be */

   // sfRobotComStr(VISION_COM,"boxsize=10");           //    20 // affects 'j'
incremental color training command
  // sfRobotComStr(VISION_COM,"colorgrow=10");        //    5 // affects '<' '>'
commands
   //sfRobotComStr(VISION_COM,"diffthresh=5000");     // 5000 // affects 's'
command (colors bounding box sensitivity)

  sfRobotComStr(VISION_COM,"edge_thresh=100");         // 100 // 1 to 255 =
sensitivity of visual sonars (1 = MAX)

  sfRobotComStr(VISION_COM,"orient_subsamp=0");      // 0 //

  sfRobotComStr(VISION_COM,"pio_max_blobs_a=1");     // 0 //
  sfRobotComStr(VISION_COM,"pio_max_blobs_b=1");     // 0 //
  sfRobotComStr(VISION_COM,"pio_max_blobs_c=1");     // 0 //

  // sfRobotComStr(VISION_COM,"pline_row_min=10");   //  50 //
  // sfRobotComStr(VISION_COM,"pline_row_max=190");  // 170 //

  sfRobotComStr(VISION_COM,"sony_apl=5");             // 127 // Auto gain now is
low = image is more lightened
  sfRobotComStr(VISION_COM,"sony_hue=250");             // 127 // Affects only
colors convertion for the video rapresentation
  sfRobotComStr(VISION_COM,"sony_saturation=250");    // 127 // Brown now is a
saturated color
  sfRobotComStr(VISION_COM,"sony_sharpness=250");     // 127 // Image appears
more crisp (less fuzzy)
}


void                    /* don't do this, or you'll have to run ARC */
stop_vision_system(void)
{
  sfRobotComStr(VISION_COM,"pioneer_visrun=0");
```

```
}



/*
 * Finding blobs and lines
 */

float sfBlobConst = 100000.0;       /* // 40000.0 // from chroma.c */
float sfLineConst = 1000.0;     // non serve!!! //

int found_blob_area = 250;    // 30 //
int found_line_num = 2;       //  2 //




#define MAX_W 60

#define MIN_W 120


int
foundb(struct vinfo *v, int delta)
{
  if (v->type == BLOB_MODE)
    {
      if ( (v->area >= found_blob_area) && (v->w >= MIN_W) && (v->w <= MAX_W) &&
(ABS(v->x) <= delta) )
      return v->x;
      else
      return -1000;
    }
}

int
found_blob(int channel, int delta)
{
  switch(channel)
    {
```

```
    case 0:
      return foundb(&sfVaInfo, delta);
    case 1:
      return foundb(&sfVbInfo, delta);
    case 2:
      return foundb(&sfVcInfo, delta);
    }
}


int
foundl(struct vinfo *v)
{
  if (v->type == LINE_MODE)
    {
      if (v->num >= found_line_num)
      return v->x;
      else
      return -1000;
    }
}

int
found_line(int channel)
{
  switch(channel)
    {
    case 0:
      return foundl(&sfVaInfo);
    case 1:
      return foundl(&sfVbInfo);
    case 2:
      return foundl(&sfVcInfo);
    }
}



/* draw blobs and lines, extracting RW coordinates from image size */

int draw_blob_thresh = 25;    /* // 20 // must be this big to draw */
// int draw_line_thresh = 2;  /* must be this big to draw */
```

```
void
draw_one_blob(struct vinfo *v, int color)
{
  int x, y, w, h;
  /* if (v->area >= draw_blob_thresh) */
  if (v->area >= found_blob_area && ABS(v->x) <= draw_blob_thresh)
    {
      x = BLOB_X(v);
      y = BLOB_Y(v,x);
      h = v->h;
      w = v->w;
      sfSetLineColor(color);
      draw_rw_cbox(x, y, w, h);

      // sfSMessage("x:%i, y:%i, h:%i, w:%i", x, y, h, w);
    }
}

void
draw_one_line(struct vinfo *v, int color)
{
  float y;
  /* if (v->num >= draw_line_thresh) */
  if (v->num >= found_line_num)
    {
      y = LINE_Y(v);
      sfSetLineColor(color);
      draw_rw_cbox(1000.0, 1000.0*y, 4, 12);
    }
}


void
draw_blobs(void)        /* process for drawing raw blobs */
{
  switch(process_state)
    {
    case sfINIT:
    case sfRESUME:
      draw_one_blob(&sfVaInfo,sfColorLightRed);
      // draw_one_blob(&sfVbInfo,sfColorLightYellow);
```

```
      // draw_one_line(&sfVcInfo,sfColorSteelBlue);
      break;
    }
}
```

Routines utili di uso comune:

Vedi *acFindArtifactByType*, *acFindDoor* e *acFindAndFollowCorridor* in "***pm2.c***".

APPENDICE

STRUTTURE DI SAPHIRA NON DOCUMENTATE:

Durante lo svolgimento di questo elaborato sono state superate svariate difficoltà inerenti alla comprensione dell'ambiente di sviluppo Saphira: molte cose "difficili" sono diventate "facili" dopo essere state affrontate in modo diverso; purtroppo però per risolvere i nostri problemi nel modo "migliore" la sola lettura del manuale non è risultata sufficiente a causa delle scarse informazioni che esso fornisce.
Per quanto riguarda la nostra esperienza, è stato per noi essenziale analizzare i files di header di Saphira per arrivare a comprendere quelle strutture (non documentate) che potevano essere utili al fine del nostro elaborato. Ciò ha richiesto purtroppo molto tempo ed è per questo che riteniamo proficuo fare un elenco delle strutture nascoste all'interno di Saphira:

- ***process_state:***
  int process_state; <***process.pro***>

  Variabile persistente istanziata da ogni micro-task (sincrono) di Saphira; tiene traccia dello stato del processo, permettendo così di strutturarlo secondo il

paradigma degli Automi a Stati Finiti (FSA) proprio della struttura sincrona di Saphira. Segue un esempio esplicativo in pseudo-codice:

```
void micro-task (void) {
   …
   switch (process_state) {
   case FSA_State_0:
         …
   case FSA_State_N:
         …
   }
}
```

- ***_process_params:***

  typedef param beh_params;* <***struct.h***>

  beh_params _process_params; <**constr.h**>

  Array di variabili di tipo sfINT, sfFLOAT o sfPTR usato per la comunicazione fra due micro-task Saphira. Correlato alla funzione *sfInitIntention*:

  - ***sfInitIntention:***

    sfprocess sfInitIntention(void (*fn)(void), char* name, int timeout, ...);* <***saphira.pro***> /* Nota: la lista di parametri va chiusa con *sfEND* */

    Funzione per l'inizializzazione di un micro-task Saphira, ma che si differenzia da *sfInitProcess* per la possibilità di settare un timeout per l'esecuzione del processo e di passare a quest'ultimo una lista di parametri (vedi *_process_params*).

Un esempio applicativo che usa queste due ultime strutture può essere visto nel file "**cogna.c**" allegato a questa relazione (vedi l'appendice Listati).

- ***sfPointList::***

  point point_buf[MAXPOINTS];*

  *list _plist_ = {0, MAXPOINTS, (void**)point_buf};*

  list point_list = &_plist_;*

```
/* La struttura list viene definita in <constr.h> */
```

Lista in cui vengono memorizzati gli artefatti gestiti da Saphira. Fondamentale per chiunque voglia gestire in modo diretto la conoscenza del robot sul mondo reale (come risulta necessario in questo progetto).

Un esempio applicativo che usa questa struttura può essere visto nel file "**pm2.c**" allegato a questa relazione (vedi l'appendice Listati).

APPENDICE

PROBLEMI RISCONTRATI E SOLUZIONI PROPOSTE:

Telecamera:

Messa a fuoco:

Per il buon funzionamento del sistema di visione di Speedy è molto importante accertarsi che la telecamera montata su di esso sia a fuoco: con la telecamera montata attualmente sul robot capita molto spesso, infatti, che la messa a fuoco venga persa a causa della rotazione accidentale dell'obiettivo in quanto esso non può essere bloccato.
Il modo più facile e veloce per mettere a fuoco la telecamera consiste nei seguenti passi:

1. staccare il cavo che entra nella presa "Camera In" di Speedy ed inserirlo nella presa "Video In" del *video-transmitter*;

2. collegare il *video-receiver* a Golem ed accertarsi che sia impostato sullo stesso canale del *video-transmitter*;

3. infine, utilizzare l'applicativo Linux *XTVScreen* presente su Golem sotto ambiente X-Windows per visualizzare l'output proveniente dalla telecamera in modo da poter osservare l'andamento della messa a fuoco durante l'operazione di taratura.

Note:

Per l'alimentazione del video-transmitter si può far uso del secondo connettore "Camera Power" fornito da Speedy o, nel caso questo sia già occupato, mediante una batteria da 12V DC.
L'applicativo XTVScreen su Golem è già stato configurato in modo da funzionare correttamente con il video-modem in dotazione al laboratorio (modello *VTX2400 Eagle* della *Tango Systems*). Se si dovessero presentare problemi molto probabilmente è a causa della perdita di sincronismo fra ricevitore e trasmettitore del video-modem: la prima cosa da fare è quindi provare a resettare il video-receiver con l'apposito pulsante.

Cognachrome Vision System:

Vantaggi:

- Il sistema è molto flessibile ed effettua automaticamente (in tempo reale) l'elaborazione dei frames video provenienti dalla telecamera richieste dall'applicativo client.

- Il sistema svincola il programmatore dal lavoro di elaborazione dell'immagine visiva, consentendo di avere direttamente a disposizione parecchie informazioni relative ai blobs di interesse.

Svantaggi:

- Difficoltà nell'apprendimento dei colori, legate alla scarsa usabilità del tool mini-ARC.

- Impossibilità (con il tool mini-ARC) di memorizzare/leggere su/da file i settings del sistema, limitando così la possibilità di integrazione del lavoro di più gruppi di sviluppo.

- I colori devono essere scelti in modo oculato: possibilmente non devono appartenere a superfici lucide e riflettenti ma essere opachi e contrastare con lo sfondo nel quale sono inseriti.
  Per la scelta dei colori occorre fare delle prove e nulla si può dire in generale: tutto dipende dall'applicazione specifica e dalle condizioni di illuminazione dell'ambiente in cui essa verrà fatta funzionare.

- Non completa integrazione con Saphira, in quanto in questo ambiente si possono avere solo informazioni sulle dimensioni del blob riconosciuto. A chi fosse interessato ad avere informazioni sull'orientazione dei blob o sui contorni presenti nel frame visivo (funzione di "edge detection") deve dunque gestire il "Protocol String" descritto nel manuale Cognachrome.

Osservazioni:

Nell'analizzare i frames video provenienti dalla telecamera montata su Speedy emerge un problema distorsivo, dovuto principalmente alle dimensioni ridotte della lente utilizzata dall'obiettivo, che porta ad una errata percezione geometrica e cromatica di ciò che viene visto dal robot. Si nota inoltre che questi problemi (in particolar modo la distorsione cromatica) diventano consistenti avvicinandosi ai lati del frame visivo.
Tenuto conto di quanto appena detto e dei fini del nostro progetto, la soluzione più facilmente integrabile con l'ambiente Saphira consiste nel "buttar via" ciò che viene visto ai margini del campo visivo ignorando i blob riconosciuti al di fuori dell'area visiva centrale considerata "affidabile" (vedi variabile *delta* definita nella funzione *foundb* nel file "***my-chroma.c***").

Un altro problema tecnico osservato consiste nel non perfetto allineamento fra l'asse della telecamera e quello del robot. Questo difetto costitutivo porta ad un'errata stima della distanza laterale del blob riconosciuto (vedi macro *BLOB_Y* nel file "***cogna.h***") che può diventare considerevole nel caso di blob di piccole dimensioni e situati ai margini del campo visivo. Date le dimensioni del blob che deve essere riconosciuto (una porta situata frontalmente al robot, quindi molto grande) e l'operazione di finestramento centrale che viene operata (il blob viene sempre visto sotto un piccolo angolo) non si renderebbero necessari particolari aggiustamenti; tuttavia è stata operata la scelta di correggere questo inconveniente introducendo il fattore correttivo *DELTA_GRAD* (vedi definizione nel file "***cogna.h***" e macro *BLOB_Y* precedentemente citata).

Infine, esistono anche errori di stima che discendono dalla natura e dal colore del blob da riconoscere: la porta infatti è lucida (quindi soggetta a riflettere luce e ad apparire di colori diversi a seconda dell'angolazione sotto cui la si osserva) e possiede un colore insaturo (marrone, quindi ai margini della scala RGB e facilmente confondibile dal sistema di visione con altri colori insaturi che si trovano ad esso vicini nello spazio RGB – nella fattispecie tutti i colori scuri). Tutto questo porta, in particolari circostanze, ad un riconoscimento solo parziale del blob con la conseguente sovrastima della distanza (vedi macro *BLOB_X* nel file "***cogna.h***").
Questo difetto può essere minimizzato agendo sul valore delle variabili persistenti del sistema Cognachrome (vedi sezione seguente) che regolano la digitalizzazione dell'output analogico della telecamera (formato video NTSC) nei valori digitali dello spazio RGB e con considerazioni sulla forma geometrica della porta (altezza e larghezza della stessa legate alla distanza a cui avviene il riconoscimento del blob).

APPENDICE

COGNACHROME VISION SYSTEM:

Il sistema di visione Cognachrome viene configurato attraverso il setting delle variabili persistenti che esso usa per impostare i parametri del suo funzionamento. Passiamo in rassegna le più importanti, divise in due categorie:

Variabili persistenti Telecamera NTSC / Sistema di Acquisizione:

sony_apl — controlla l'auto guadagno del modulo che digitalizza il segnale NTSC proveniente dalla telecamera. Il suo valore va tenuto basso se non si vuole che il sistema compensi i cambiamenti di luminosità.

sony_saturation — il suo valore va tenuto basso se si hanno problemi di saturazione, alto se si vuole saturare colori normalmente insaturi come il marrone o il grigio.

sony_sharpness — influenza la definizione dell'immagine digitalizzata: un valore basso ne sfuma i contorni.

pioneer_x_mode — (x=a, b, c) per impostare la modalità di funzionamento BLOB_MODE (0), LINE_MODE (1), BLOB_BB_MODE (2).

Variabili persistenti Sistema di Apprendimento:

diffthresh — influenza il sistema di apprendimento del colore (comando *S* in mini-ARC Saphira Mode): se alto si associa al colore da riconoscere un volume elevato nello spazio RGB.

box_size — dimensioni (in pixel) della finestra di colori che si vanno ad aggiungere dopo un apprendimento incrementale (comando *J* in

mini-ARC Saphira Mode) allo spazio RGB del colore riconosciuto.

Nota:

Nell'ambiente *TPU_VIS MODE* di mini-ARC con il comando *P* vengono visualizzate tutte le variabili persistenti attualmente definite.

APPENDICE

BIBLIOGRAFIA E RIFERIMENTI:

pm2@golem.ing.unibs.it per avere informazioni sul progetto

nicoc@freenet.hut.fi per qualsiasi domanda su Saphira o Cognachrome

NOTE:

I file indicati tra "" sono quelli propri di questo progetto.

I file indicati tra <> sono gli headers di Saphira.