UNIVERSITA' degli STUDI di BRESCIA

Facoltà di Ingegneria

Corso di robotica prof. Riccardo CASSINIS

Progetto

<< PARKING >>

REALIZZATO DA:
DAVIDE BOTTURI
FRANCESCO CAGGIANO
DAVIDE TOZZO

Anno Accademico 1997/98

I realizzatori del progetto PARKING ed il loro Pioneer 1®:

Da sinistra:

DAVIDE TOZZO, DAVIDE BOTTURI, FRANCESCO CAGGIANO

Obiettivo:

L'obiettivo assegnatoci è stato quello di far parcheggiare il robot Pioneer denominato "Speedy" all'interno di un armadio nel Laboratorio di Robotica Avanzata utilizzando il sistema di visione Cognachrome Vision System®.
Più precisamente, il robot, posizionato in un qualsiasi punto della stanza, deve essere in grado di localizzare l'armadio, capire se l'anta è aperta, entrare e girarsi in modo che l'armadio si possa chiudere.

Naturalmente ai progettisti è data la facoltà di condizionare l'ambiente mediante blob di colore scelti opportunamente. È inoltre opportuno che il robot, durante il suo movimento, eviti gli ostacoli.

Materiali a disposizione (Tools):

- Robot "Speedy", tipo Pioneer1® equipaggiato con telecamera e sistema di visione Cognachrome®.
- Computer portatile denominato "Frost" da collegare al robot durante il funzionamento.
- Servizi di rete di facoltà.
- Materiale del **L**aboratorio di **R**obotica **A**vanzata della facoltà.

Realizzazione:

- Il progetto è stato ultimato in modo che il robot possa raggiungere il goal da qualsiasi parte della stanza venga fatto partire.
- Non è necessario che il robot veda il primo riferimento già all'avvio del programma.
- Non è necessario fornire al robot alcun dato sulla sua posizione iniziale: esso è in grado di muoversi e di localizzarsi in maniera autonoma (all'interno della stanza).
- Non è necessario che l'ambiente sia privo di ostacoli.
- La ricerca del punto finale non ha alcun limite di tempo ed il programma, se non interrotto volontariamente, si ferma solo a parcheggio avvenuto.

- E' richiesto che il fondo del parcheggio sia marcato con 2 blob colorati e che di fronte all'ingresso ne sia posizionato un terzo; non è necessario rispettare alcuna distanza relativa o assoluta per il posizionamento dei tre landmark.

INDICE:

Bibliografia

LOGICA DI FUNZIONAMENTO:

indice

Il parcheggio è riconosciuto attraverso una combinazione di tre colori: un blob verde che ha la proprietà di essere visto da lontano e dalla combinazione di due blob (uno rosso ed uno giallo) che individuano la posizione precisa del goal.
La prima operazione che viene effettuata dal robot è quindi la ricerca_del_blob_verde
L'operazione consiste nella ricerca, da parte del robot, di un blob di colore verde posto casualmente sul pavimento, di fronte all'armadio. Il robot, dal punto dove è stato fatto partire, esegue una rotazione esaminando l'ambiente circostante. Se non trova il blob si muove in avanti ed effettua una nuova rotazione. Se localizza il blob verde, esso si muove e va a posizionarsi in prossimità del riferimento colorato.

L'approccio al blob verde viene effettuato controllando le rilevazioni dei sonar. Se viene rilevato un ostacolo, il robot si ferma immediatamente, interrompe la sua attività di "approccio", ruota di 90°, avanza nella nuova direzione di 50cm ed effettua nuovamente una rotazione per cercare l'obiettivo che ora vedrà da un'altra angolazione. Se lo spazio da percorrere sarà libero da ostacoli si realizzerà "l'approccio", altrimenti verrà effettuata una nuova manovra di aggiramento dell'ostacolo. Questo tipo di "avoid collision" risulta particolarmente utile anche quando il robot vede erroneamente un blob colorato in un riflesso su una parete o su un mobile: il robot realizzerà l'approccio fino a circa 50cm dal falso blob e poi cercherà il nuovo blob ma non vedrà più il riflesso precedente perché ora esso ha cambiato l'angolo di vista.
Il posizionamento in prossimità del blob verde risulta necessario per permettere al robot di avere una localizzazione in un punto a lui visibile da qualsiasi parte della stanza venga fatto partire e per permettere al robot un successivo ingresso nell'armadio secondo una direzione più o meno perpendicolare all'armadio stesso. Ultimato il posizionamento sul blob verde il robot effettua una ***ricerca del blob rosso***.

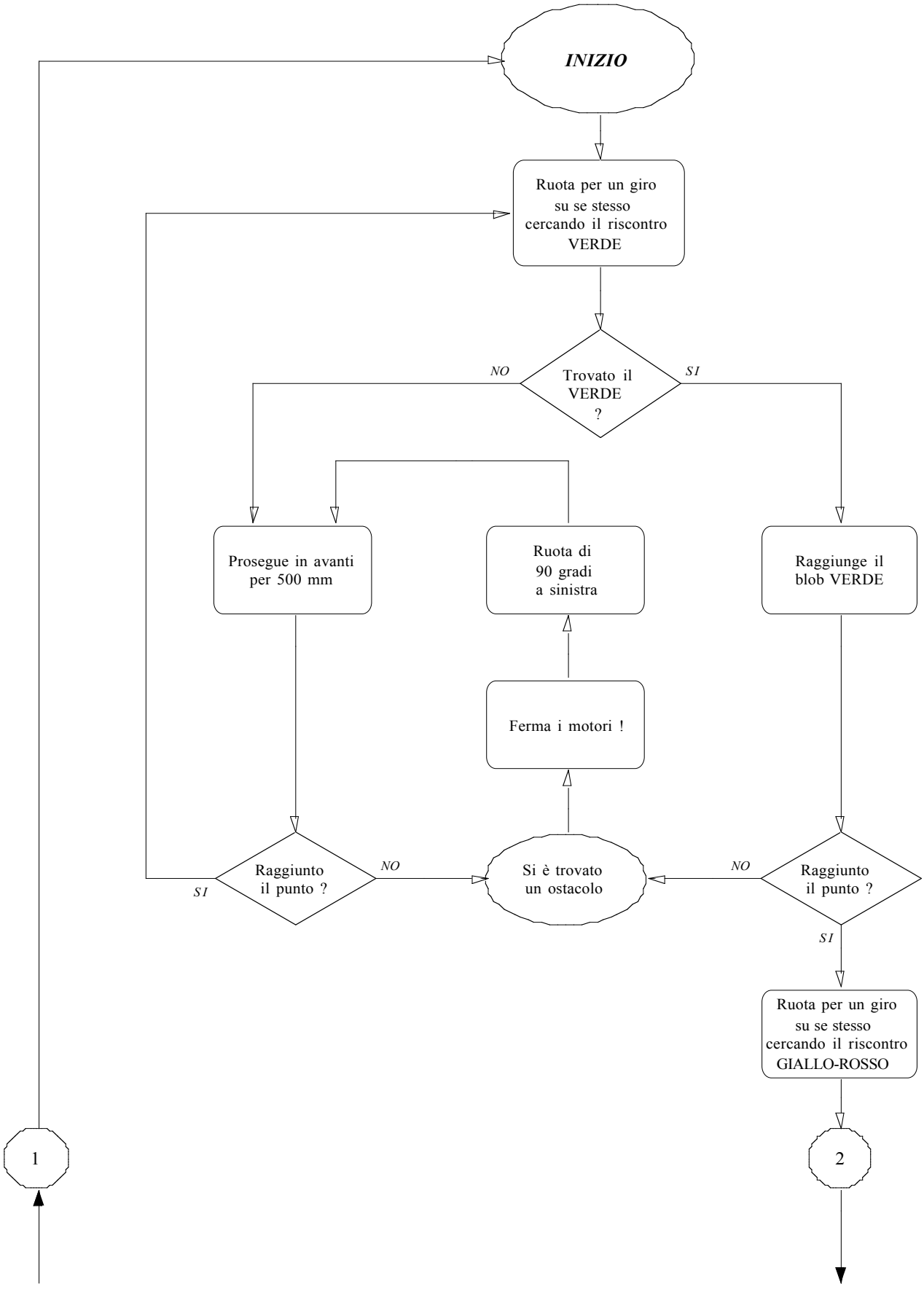
La ricerca viene fatta compiendo una rotazione. Se il robot non trova il blob rosso significa che le condizioni di luce non sono favorevoli oppure che la porta dell'armadio è chiusa. In questa fase, il robot, ogni volta che riconosce il landmark rosso verifica che ci sia, nel suo raggio di vista, anche il riscontro giallo.

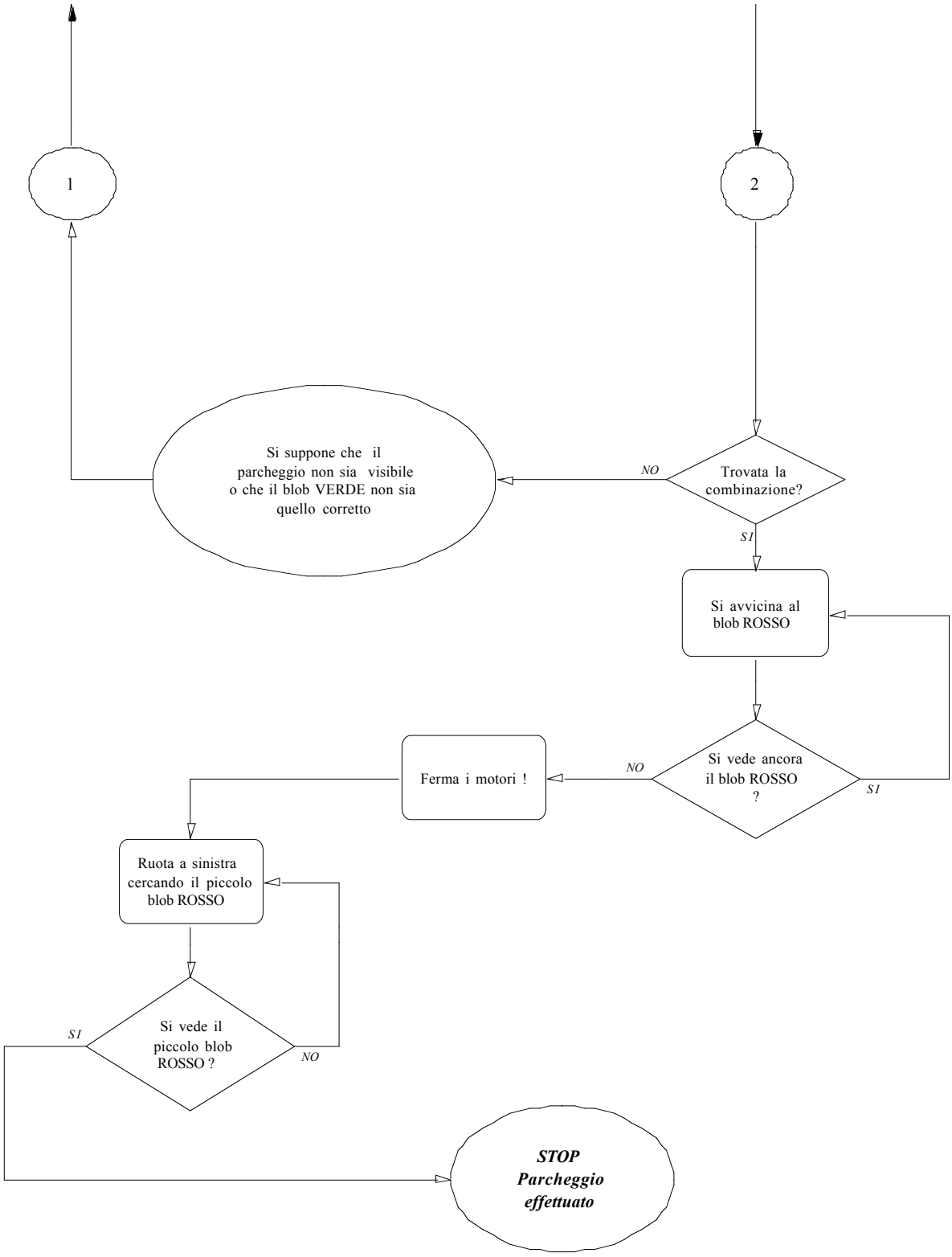
Il controllo sul blob giallo è necessario perché la rilevazione di quest'ultimo è indice per il robot di una porta dell'armadio aperta quanto basta per permettere il parcheggio.

Una volta verificate le condizioni il robot inizia l'avvicinamento all'armadio guidato esclusivamente dal sistema di visione (poiché i sonar non risultano più attendibili) il quale elabora i dati sulla posizione relativa al blob rosso.

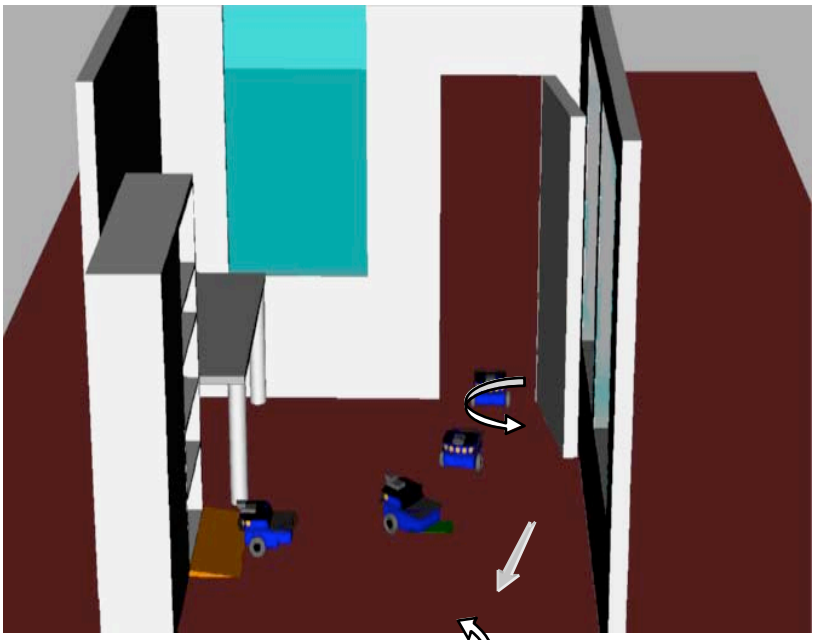
Raggiunta la posizione contro la parete posteriore dell'armadio il robot effettua una manovra di rotazione prendendo come riferimento un altro piccolo blob rosso posto sul montante centrale dell'armadio.

Di seguito è riportato un flow chart che sintetizza la logica con cui il Pioneer interagisce con il mondo.


INIZIO
Ruota per un giro su se stesso cercando il riscontro VERDE
Trovato il VERDE ?
NO
SI
Prosegue in avanti per 500 mm
Ruota di 90 gradi a sinistra
Raggiunge il blob VERDE
Ferma i motori !
Raggiunto il punto ?
NO
SI
Si è trovato un ostacolo
Raggiunto il punto ?
NO
SI
Ruota per un giro su se stesso cercando il riscontro GIALLO-ROSSO
1
2

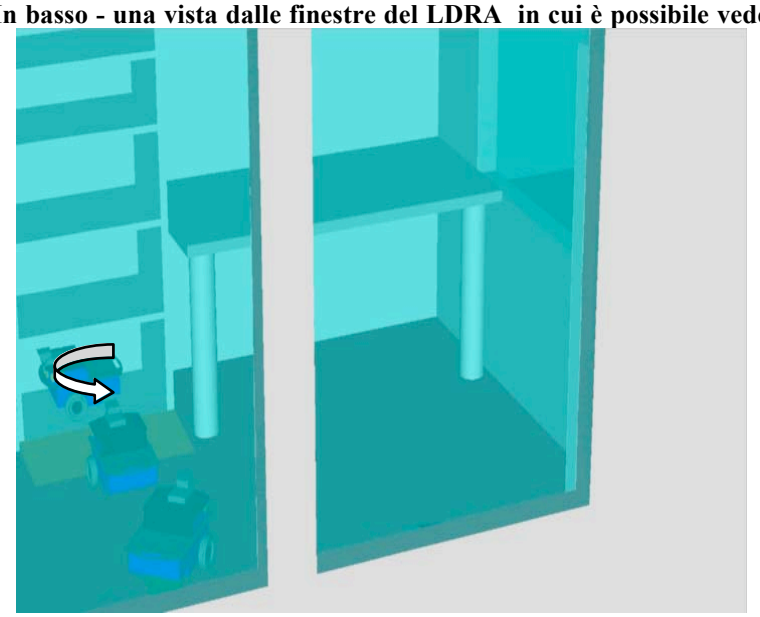

1
2
Si suppone che il parcheggio non sia visibile o che il blob VERDE non sia quello corretto
NO
Trovata la combinazione?
SI
Si avvicina al blob ROSSO
Si vede ancora il blob ROSSO ?
NO
SI
Ferma i motori !
Ruota a sinistra cercando il piccolo blob ROSSO
Si vede il piccolo blob ROSSO ?
SI
NO
STOP
Parcheggio effettuato

Riprendiamo ora la descrizione del progetto descrivendo con maggior dettaglio le fasi cardine del processo decisionale e per rendere più chiaro il percorso che si intende far seguire al robot si è disegnato un ufficio LDRA virtuale in cui abbiamo posizionato il Pioneer.



A destra - una visione d'insieme che evidenzia la presenza del blob verde nel mezzo della stanza e la ricerca della combinazione rosso/gialla.

In basso - una vista dalle finestre del LDRA in cui è possibile vedere il Pioneer nella sua posizione finale.



RICERCA ED APPROCCIO AL BLOB VERDE: indice

La ricerca del blob verde è la prima operazione che compie il robot dopo essere stato attivato.
Esso esegue una rotazione di 360° che viene interrotta non appena viene visto il blob. Se la rotazione si è conclusa senza alcun ritrovamento, il robot segnala l'accaduto emettendo un beep dal tono grave e si muove in avanti di 500mm. Dopo aver cambiato la sua posizione esegue una nuova rotazione e così via fino a quando la ricerca non dà esito positivo. A questo punto inizia la fase di approccio al blob verde.

Il calcolo della traiettoria viene effettuato mediante una stima iterativa della distanza del blob verde (Vd. Paragrafo successivo).
In questo approccio non richiediamo al robot una particolare precisione in modo da poter avere una maggiore velocità di avanzamento.
In questa fase il robot deve continuamente vedere attraverso i sonar il mondo circostante che è assolutamente casuale e ricco di insidie ed ostacoli, fissi e mobili. Quando i sonar rilevano un ostacolo (almeno a 550mm, distanza utile per la manovra di rotazione) il robot si ferma immediatamente, dà avviso mediante un beep sordo del fallimento del blob approaching, ruota di 90° ed avanza di 500mm (tenta cioè di aggirare l'ostacolo).

Dalla nuova posizione effettua una ulteriore rotazione. Se l'ostacolo è aggirato, il robot riuscirà a vedere il blob verde, in caso contrario, al termine della rotazione, si muoverà ancora di 500mm ed effettuerà una nuova rotazione continuando così finché non tornerà a vedere il blob.

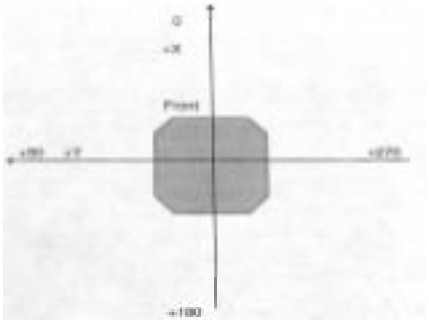
Il robot potrebbe vedere dei riflessi di colore verde nei muri oppure nei particolari cromati di alcuni elementi d'arredo (come è spesso accaduto durante i test). Per limitare questo fenomeno abbiamo inserito dei "filtri" che verranno descritti più avanti (vedi il_sistema_decisionale)

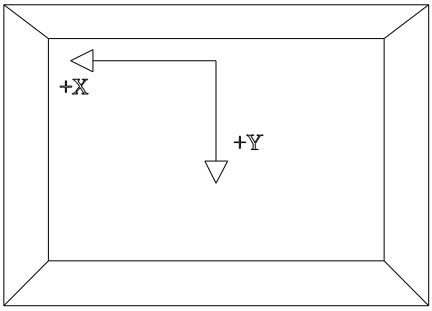
STIMA DELLA DISTANZA DEL BLOB VERDE:

Nell'utilizzo della telecamera è emerso il problema della distorsione associata ad essa. Questa distorsione è causata principalmente dalle deformazioni introdotte dalla lente di dimensioni ridotte utilizzata dalla telecamera e dalla non coassialità tra la telecamera ed il robot.
Il problema è stato risolto cercando una corrispondenza tra ciò che vede la telecamera e le dimensioni del mondo reale.

Abbiamo eseguito una serie di misure da posizioni note e le abbiamo confrontate con le misure fornite dal sistema di visione. Il confronto ha evidenziato un errore sulla misura della profondità nella direzione frontale rispetto al robot. Questo errore cresce aumentando l'angolo di incidenza tra il robot e l'obiettivo da raggiungere.

Attraverso una regressione lineare abbiamo cercato la funzione (lineare) che meglio correggesse questi errori.





Per la stima della distanza del blob verde, che si trova adagiato sul pavimento, la funzione di correzione dell'errore ha dovuto tener conto del fatto che la telecamera vede il blob con un angolo più basso rispetto alla linea dell'orizzonte (almeno nella parte finale del percorso di avvicinamento, quella parte nella quale si ha bisogno di un maggiore precisione).

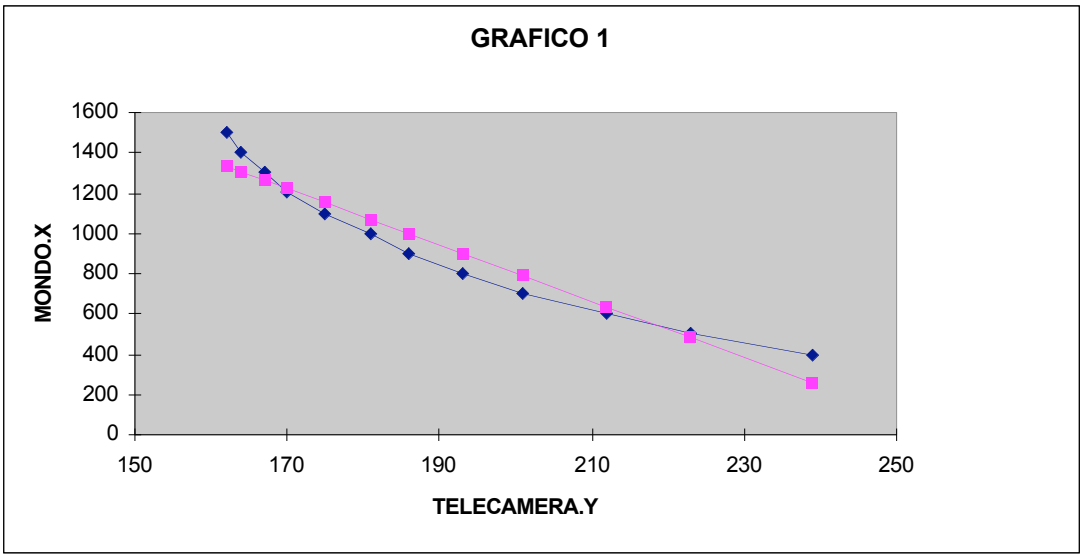
A sinistra è mostrato il sistema di riferimento per la telecamera mentre a destra il sistema di riferimento per il Pioneer1

Nella tabella successiva evidenziamo le misure effettuate e le diverse rette ottenute tramite regressione lineare (la funzione che utilizza il metodo dei minimi quadrati per calcolare una retta che meglio rappresenti i dati).

Xtelec.	Ytelec.	Xmondo	Ymondo	Retta di

		(mm)	(mm)	Approssimazione
9	239	400	0	
11	223	500	0	
10	212	600	0	Pendenza=-13.99
11	201	700	0	
12	193	800	0	Intecetta=3601.755
12	186	900	0	
11	181	1000	0	
12	175	1100	0	
13	170	1200	0	
12	167	1300	0	
13	164	1400	0	
11	162	1500	0	

Segue il grafico relativo a questa tabella



Xtelec.	Ytelec.	Xmondo (mm)	Ymondo (mondo)	Retta di approssimazione
46	220	500	150	
44	209	600	150	
40	197	700	150	Pendenza=-15.723
37	190	800	150	
34	183	900	150	Intercetta=3862.697
32	178	1000	150	
31	173	1100	150	
30	170	1200	150	

28.5	167	1300	150	
27	165	1400	150	
-22	219	500	-150	Pendenza=-16.014
-17	207	600	-150	
-14	198	700	-150	Intercetta=3911.032
-12	190	800	-150	
-10	184	900	-150	
-8	178	1000	-150	
-5	174	1100	-150	
-4	169	1200	-150	
-3	166	1300	-150	
-1	164	1400	-150	
64	194	700	300	Pendenza=-22.648
59	188	800	300	
55	183	900	300	Intercetta=5058.787
51	178	1000	300	
48	173	1100	300	
46	170	1200	300	
43	167	1300	300	
41	163	1400	300	
-39	194	700	-300	Pendenza=-23.765
-35	189	800	-300	
-30	182	900	-300	Intercetta=5261.673
-27	177	1000	-300	
-23	173	1100	-300	
-20	169	1200	-300	
-18	167	1300	-300	
-16	164	1400	-300	
-14	161	1500	-300	
68	175	1000	450	Pendenza=-34.198
65	172	1100	450	
60	169	1200	450	Intercetta=6986.321
57	167	1300	450	
55	163	1400	450	
-43	175	1000	-450	Pendenza=-36.911
-39	171	1100	-450	
-35	169	1200	-450	Intercetta=7438.743
-33	166	1300	-450	
-29	164	1400	-450	
-27	161	1500	-450	

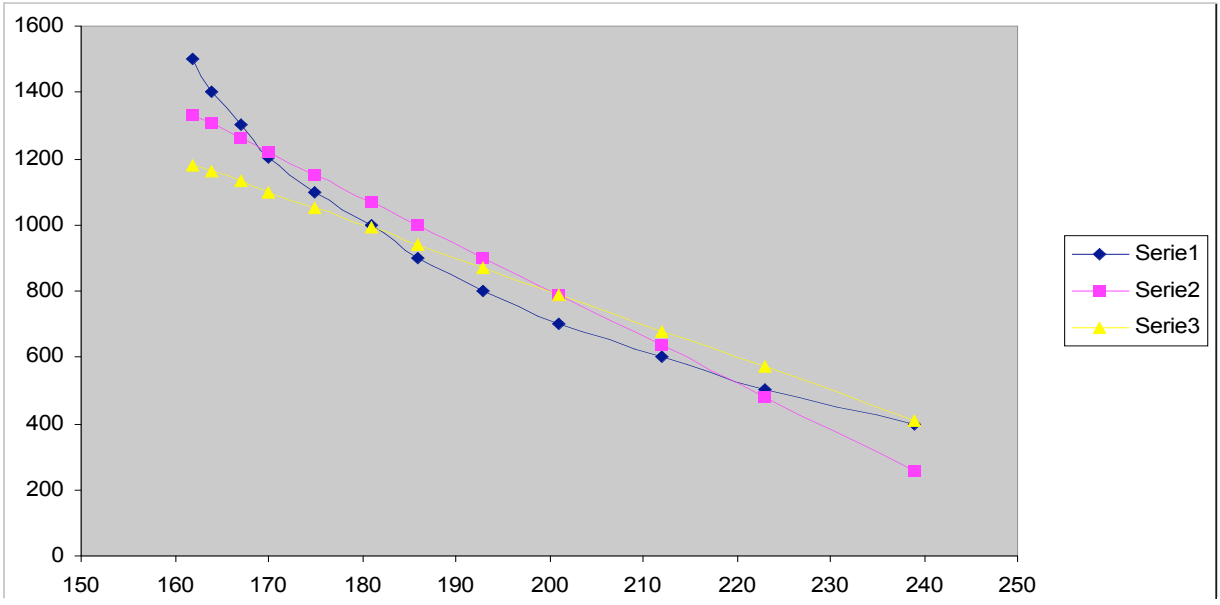
Si può chiaramente notare che variando la posizione di osservazione rispetto alla coordinata Ymondo la retta di approssimazione cambia notevolmente. Abbiamo scelto di tenere maggiormente in considerazione i dati ottenuti dalle stime sulla distanza di punti posti di fronte al robot in quanto nel tragitto di avvicinamento il robot si gira e viene a trovarsi di fronte al target proprio nel momento in cui occorre una maggiore precisione.
Dai dati precedenti si evince quindi che la retta di approssimazione dovrebbe essere:

$$Xmondo = 3601.75 - 13.99 \cdot Ytelecamera$$

La retta di regressione lineare non soddisffa le nostre aspettative, poiché produce un errore considerevole intorno ai 40 cm (distanza di ultima lettura) mentre è più preciso per distanze superiori. Noi abbiamo però l'esigenza opposta, cioè da lontano stimi in modo approssimativo, mentre durante l'avvicinamento migliori la stima. Quindi abbiamo utilizzato la serie 3 ottenuta ruotando la retta "serie 2" intorno al punto P al fine di farla coincidere con la curva reale nel punto 240,400 cioè alla coordinata corrispondente alla distanza di ultima visione.

Si è quindi scelta la seguente retta di approssimazione:

$$Xmondo = 2800 - 10 \cdot Ytelecamera$$



Nel grafico precedente con l'etichetta "Serie 1" sono mostrati i valori reali da noi misurati, con l'etichetta "Serie 2" sono mostrati i valori emersi dai calcoli mentre con l'etichetta "Serie 3" è mostrata la stima dei valori utilizzati da noi.

RICERCA ED APPROCCIO DEI BLOB ROSSO E GIALLO:

indice

In questa fase il robot ruota di 360° cercando il blob rosso. La rotazione viene interrotta non appena viene visto il colore rosso. Il robot si pone due domande: <<Vedo anche il blob giallo?>>, <<L'area del blob rosso è maggiore del valore impostato?>>. Solo se queste due domande hanno una risposta affermativa viene effettuato il blob approaching in quanto se non viene visto il blob giallo potremmo essere di fronte ad un riflesso oppure la porta dell'armadio potrebbe non essere aperta in modo sufficiente o ancora, se l'area del blob rosso non è abbastanza grande, potremmo essere davanti ad un riflesso.

Una volta localizzata in maniera corretta la combinazione Y&R inizia l'avvicinamento al blob rosso; sono state seguite due diverse strade per questo tipo di approccio: una via è stata quella di stimare la distanza del blob attraverso le misure sull'altezza del blob stesso che vengono restituite dal Cognachrome, mentre la seconda strada è stata quella di avvicinarsi al blob con velocità costante fino a quando il colore roso sparisce dalla vista del robot.

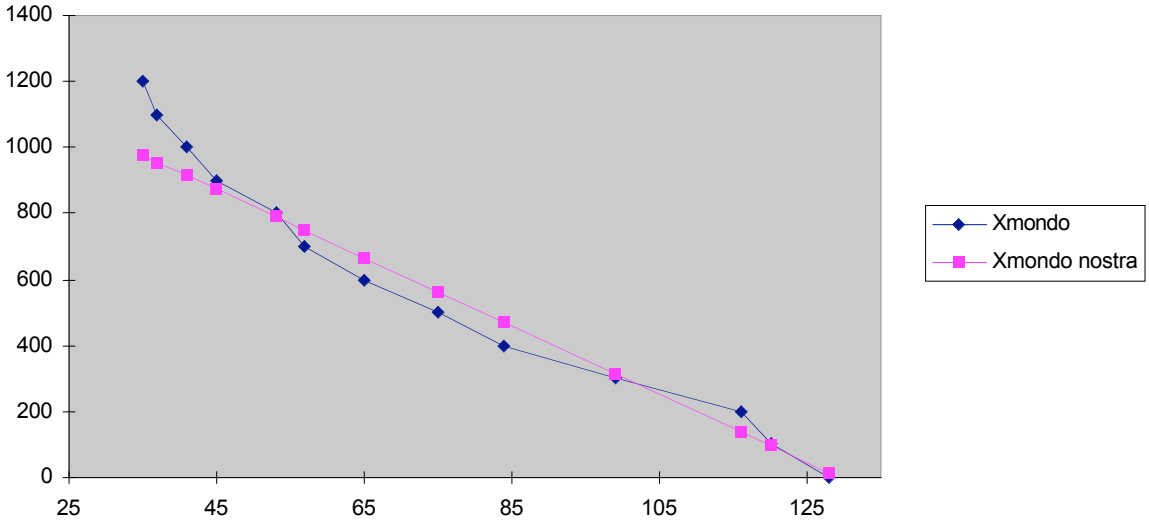
a) Stima della distanza del blob rosso.

indice

Per la stima della distanza del blob rosso, il discorso è leggermente diverso rispetto a quella del blob verde, in quanto il blob si trova di fronte alla telecamera. Abbiamo effettuato una serie di misure da posizioni note osservando i dati che il sistema di visione ci restituiva. Il dato più stabile è quello relativo all'altezza del blob. La stabilità del dato è stata un elemento discriminante in quanto, in questa fase, è molto importante avere una notevole precisione. Se il sistema di visione sottostima la distanza si ha un errato posizionamento del robot all'interno dell'armadio (parte del robot resta fuori) mentre se il sistema di visione sovrastima la distanza si ha un urto contro la parete posteriore dell'armadio.
Nella tabella seguente riportiamo i dati relativi alle misure effettuate per tarare il sistema:

Xtelec.	Ytelec.	H	v	Area	X mondo	Y mondo
11	134	35	20	700	1200	0
12	134	37	21	777	1100	0
10	136	41	24	984	1000	0
12	136	45	26	1170	900	0
12	137	53	29	1537	800	0
12	139	57	32	1824	700	0
10	139	65	36	2340	600	0

12	141	75	41	3075	500	0
11	151	84	45	3780	400	0
10	159	99	54	5346	300	0
10	172	116	65	7540	200	0
11	196	110	86	9460	100	0
11	184	128	121	15488	0	0



Come si può vedere dal grafico precedente la stima che abbiamo scelto del mondo reale, avvicinandosi all'obiettivo, è una leggera sottostima che ci pone in condizioni di sicurezza senza trascurare la precisione.

È importante inoltre sottolineare che, in caso di alterazione della situazione luminosa dell'ambiente, le aberrazioni cromatiche derivanti da queste possono alterare ulteriormente le misure effettuate dal sistema di visione ma con piccole correzioni alle rette di stima si possono ottenere nuovamente misure precise.

b) Movimento tipo ON-OFF: indice

Il robot si muove nella direzione del centroide del blob con modesta velocità. Quando esso giunge a pochi cm dal blob, non vede più il colore rosso in quanto la telecamera si viene a trovare più in alto del blob e quindi il cono di vista non comprende più il colore. Quando il sistema non rileva più il colore rosso, il robot ferma la sua corsa.
Questo sistema è particolarmente funzionale in quanto elimina i problemi che emergono nel caso a) cioè:

- mancanza di precisione dovuta ad alterazioni di forma del blob causate da variazioni delle condizioni luminose
- difficoltosa taratura del sistema.

Per tarare il sistema è infatti sufficiente modificare l'altezza del blob di riferimento in modo che il blob stesso sparisca dalla vista del robot nell'istante in cui si desidera che quest'ultimo si fermi.

Quest'ultimo algoritmo di movimentazione si è rivelato di maggiore efficacia rispetto al precedente anche se molto più semplice. Il vantaggio è imputabile soprattutto alla minore influenzabilità del sistema di visione rispetto alle condizioni di illuminazione.

RICERCA DEL PICCOLO BLOB ROSSO: indice

Una volta che il robot giunge a pochi millimetri dal fondo dell'armadio, esso non vede più il colore rosso ed il suo obiettivo diventa quello di posizionarsi in modo che l'armadio possa essere chiuso.
Per realizzare questo obiettivo, il robot ruota finché non viene a trovarsi di fronte ad un piccolo blob rosso posto sul montante centrale dell'armadio.
Il robot esegue due controlli su questo blob:

a) controlla che il blob rosso sia almeno ad una certa altezza
b) verifica le dimensioni dell'area del blob.

Il controllo a) serve per non tenere in conto eventuali parti di colore del precedente blob che durante la rotazione potrebbero entrare nuovamente nel cono di vista del robot.
Il controllo b) serve da filtro per eventuali riflessi che potrebbero essere visti sulle pareti lucide dell'armadio.

IL SISTEMA DECISIONALE:

indice

Le risposte dei "blocchi decisionali" disegnati nel flow_chart sono il risultato di un filtraggio introdotto per eliminare i disturbi causati dal mondo reale. Di seguito si elencano i filtri utilizzati:

1. Durante la ricerca del verde vengono verificate le seguenti condizioni:

   - L'area individuata deve essere compresa tra un valore minimo ed uno massimo. Questo permette di evitare riconoscimenti errati dovuti a riflessi (blob piccoli) o ad oggetti che essendo più grandi del blob ricercato devierebbero l'attenzione del sistema di visione.
   - La distanza stimata deve essere minore di una costante **X_MAX_VERDE**; questo consente di eliminare tutti i blob verdi che vengono visti al di sopra di una determinata coordinata Y_telecamera e che dovrebbero essere molto più ampi del blob ricercato.
   - Le coordinate X_camera devono essere all'interno di un determinato range e questo consente di valutare gli oggetti solo quando si trovano di fronte alla telecamera (quindi quando sono meno soggetti a deformazioni).

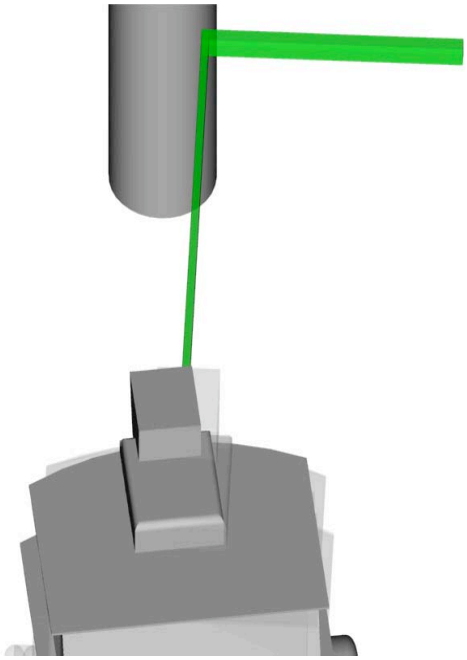
2. Durante il riconoscimento del parcheggio:
   si è effettuato un controllo sulla presenza del colore giallo e del colore rosso nel modo già descritto ed in più si è aggiunta la richiesta che il giallo fosse a sinistra del rosso. La presenza quindi della combinazione verde, giallo e rosso determina la presenza del parcheggio; è chiaro che non è consentito utilizzare queste stesse combinazioni nell'ambiente in cui opera il Pioneer durante la ricerca del proprio posteggio!.
   Mentre per il blob verde è stata stimata una distanza e successivamente comandato un movimento, per il riconoscimento della posizione degli altri landmark si è dovuto procedere in modo diverso a causa della consistente variazione di luce che influenzava i blob, soprattutto in relazione alla ridottissima possibilità di errore concessa al sistema di navigazione. Per poter avere un controllo efficiente si è quindi sostituito il controllo dinamico di *stima* con un controllo pur sempre dinamico ma di *presenza*. Durante l'approccio al blob rosso il Pioneer "scodinzola" e quindi il blob rosso esce dal cono di vista. Questo causerebbe l'arresto in quanto verrebbe interpretato come condizione di stop. Per risolvere questo si è aggiunto anche un controllo sulla effettiva crescita dell'area del blob rosso.

Durante il riconoscimento del piccolo blob rosso:
si è giocato sulle dimensioni delle aree e sul fatto che quello piccolo può essere osservato dal sistema di visione solamente quando non è visibile il blob più grande; inoltre le posizioni dei centroidi dei due blob rossi sono state studiate in modo tale

da non poter generare alcuna confusione e, più precisamente, il centroide del piccolo blob è in una posizione più alta di qualunque possibile riflesso del grande blob.

In generale:

Nello svolgimento dei test si è notato che per alcuni attimi durante la ricerca sul posto, la telecamera riusciva a intravedere il blob verde riflesso nei tubi lucidi degli sgabelli e ciò che più disturbava il sistema era il fatto che questi riflessi riuscivano spesso ad attraversare indenni i filtri predisposti. Si è reso necessario quindi introdurre un filtro dinamico che interpellasse il sistema di visione in due istanti di tempo diversi in modo da consentire al Pioneer di cambiare la propria posizione ed evitare i riflessi di questo particolare tipo (il robot in un istante vede il riflesso ma non nell'istante successivo).



LISTATO COMMENTATO DEL PROGRAMMA

indice

```
/*#####################################################################
 *   parking Ver 00.01.10.
 *#####################################################################
 */


#include "saphira.h"


#define X_MAX_VERDE  2000  /* massima distanza di riconoscimento blob verde
*/
#define X_MAX_ROSSO  3000  /* massima distanza di riconoscimento blob
rosso e giallo */
#define Time_out1    370        /* time out di rotazione con velocità
VEL_R_1 */
#define Time_out2    200        /* time out di rotazione con velocità VEL_R_2
*/

#define VEL_R_1            0.6     /* velocità di rotazione lenta */
#define VEL_R_2            0.8     /* velocità di rotazione rapida */
#define VEL1                 100.0 /* velocità di spostamento lenta */
```

```
#define VEL2                    200.0 /* velocità di spostamento rapida */

/* PROTOTYPE ************************************** */
void myConnectFn(void);
int myKeyFn (int ch);
int myButtonFn(int x, int y, int b);
void SearchGreen(void);
void SearchYellow(void);
void SearchRed (void);
void Park (void);
void beep(void);
void beeeep(void);
void boop(void);
void setup_vision_system(void);     /* from chroma.c */
void draw_blobs(void);        /* from chroma.c */


/* VARIABILI GLOBALI ********************************************** */
point *blob_pt;
int can, found_blob_M_area, found_blob_m_area;



/* ********************************************************************
*/
void
main(int argc, char **argv)
{

  sfButtonProcFn(myButtonFn);
  sfKeyProcFn(myKeyFn);
  sfOnConnectFn(myConnectFn);
  sfStartup(0);
}

/* ********************************************************************
*/
int
myButtonFn(int x, int y, int b)
{
  return 1;
}


/* myKeyFn: prende un carattere da tastiera ed esegue la funzione ad esso
associato */
int
myKeyFn(int ch)
{
  BEHCLOSURE b;
  static sfprocess *p = NULL;
  switch(ch)
      {
      case SPACEKEY:  /* ALT immediato */
                                  if (sfSingleStepMode) return 0;
                if (NULL != (b = sfFindBehavior("Constant Vel")))
                {
                  if (b->running) sfSMessage("Constant Vel OFF");
                      sfSetBehaviorState(b, sfOFF);
                }
                sfSetVelocity(0);
```

```
            break;

        case 'r': /* ricerca del blob rosso */
                  SearchRed();
                  sfMessage ("Going to Red blocK");
                  return 1;

            case 'g': /* ricerca del blob giallo */
                SearchYellow();
                sfMessage("Going to Yellow block");
                return 1;

     case 'v': /* ricerca del blob verde */
                        SearchGreen();
                 sfMessage("Going to Green block");
                 return 1;
        case 'p': /* funzione parcheggia */
              Park();
        sfMessage ("Parking in action");
              return 1;

        }
  return 0;
}


/* find_little :ricerca blob piccolo nell'armadio */
void
find_little(void)
{
  static sfprocess *_sf_p;
  static sfprocess *p = NULL;

  float X_MAX;
  int area, Time;
  float xx, yy;
  switch(process_state)
    {
    case sfINIT:
    case 20:  // inizia rotazione
           p = sfStartBehavior(sfTurnLeft,"sfTurnLeft", Time_out2, 2,
0, VEL_R_2);

    case 21:
      if ( !( ! sfFinished (p ) ) )  /* se è scaduto  il tempo */
          { process_state = 22;
            break;
          }

      {
         area = found_area_b (&sfVcInfo, 30, 100, 2000, 120 ); /*
controlla se esiste blob rosso */

          if ( area > - 500  )  /* trovato blob piccolo */
                {
        sfMessage ("Found a little blob!!!") ;
        sfKillIntention (p ) ;
        p = NULL ;
        process_state = sfSUCCESS; break;
                }
       }
```

```
        process_state = 21 ;
        break;

    case 22: / * tempo scaduto, ammazza il processo e fermati */
      sfKillIntention (p ) ;
      p = NULL ;
      sfMessage ("No blob found!!!" ) ;
      process_state = sfFAILURE;
      break;

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

    case sfRESUME:
      process_state = 20; break;

    }
}


/* find blob: ruota finché non viene trovato un blob di un certo colore
*/
void
find_blob(void)
{
  static sfprocess *_sf_p;
  static sfprocess *p = NULL;

  float X_MAX;
  int x;
  float xx, yy;

switch(process_state)
    {
     case sfINIT: /* inizia rotazione */
     case 20:
                 p = sfStartBehavior(sfTurnLeft,"sfTurnLeft", Time_out2,
2, 0, VEL_R_2);

     case 21:
            if ( !( ! sfFinished (p ) ) )  /* se  time out */
               { process_state = 22;
                  break;
               }




            {
      x = found_blob (_process_params [ 0 ] . i , 30 ) ; /* controlla se
è
                                                          stato trovato un
                                                  blob*/

               if ( x > - 500  )     /* trovato*/
                  {
                    switch (_process_params[0].i) /* controlo sul tipo di
            colore trovato */
                  {
```

```
            case 0: //VERDE (stima distanza)
          sfMessage ("Found a Green blob ");
          xx = 2800 - (10 * sfVaInfo.y) ;
          sfVaInfo.x = sfVaInfo.x - 21;
          yy = BLOB_Y(&sfVaInfo, xx);
          X_MAX=X_MAX_VERDE;
                        break;

            case 1:  // GIALLO (stima distanza)
                           sfMessage ("Found a Yellow blob");
          xx = 2800 - (10 * sfVbInfo.y )
          yy = BLOB_Y(&sfVbInfo, xx);
          X_MAX=X_MAX_ROSSO;
          break;

            case 2: //ROSSO (stima distanza)
          sfMessage ("Found a Red blob");
          xx = 1340 -(10.38 * sfVcInfo.h );
          sfVcInfo.x = sfVcInfo.x - 11;
          yy = BLOB_Y(&sfVcInfo, xx);
          X_MAX=X_MAX_ROSSO;
          break;
                      }

                        if (fabs(xx)<X_MAX) / se non _ troppo lontano crea
punto e ammazza il processo
                {
                  blob_pt = sfCreateLocalPoint (xx , yy , 0.0 ) ;
                   //sfSMessage ("blob_pt->x= %f  blob_pt->y= %f",blob_pt->x,
blob_pt->y);
                   sfAddPoint (blob_pt ) ;
                   sfKillIntention (p ) ;
                   p = NULL ;
                   process_state = sfSUCCESS;
                            break;
                 }
          }
          }
          process_state = 21 ;
          break;

         case 22: /* blob non trovato, ammazza il processo */
          sfKillIntention (p ) ;
          p = NULL ;
          sfMessage ("No blob found!!!" ) ;
          boop();
          process_state = sfFAILURE; break;
          break;

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

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

```
/* find_yellow_and_red: ricerca blob giallo e rosso */
void
find_yellow_and_red(void)
{
  static sfprocess *_sf_p;
  static sfprocess *p = NULL;
  float X_MAX;
  int x_red, x_yellow;
  float xx, yy;

switch(process_state)
    {
    case sfINIT:
    case 20:  // inizia rotazione
        p = sfStartBehavior(sfTurnLeft,"sfTurnLeft", Time_out1, 2, 0,
VEL_R_1);

    case 21:
      if ( !( ! sfFinished (p ) ) ) { process_state = 22; break; }
      {
      /* controlla presenza dei blob rosso  e giallo
              x_yellow= found_blob (1, 60);
      x_red = found_blob (2, 60 );



      if (( x_red > -500  ) && (x_yellow > -500))
      { // se trovati
                  beep();
         sfMessage ("Found both blob!!!");

                  //crea un punto nella direzione del centro blob rosso,
avanti 290mm
                  xx = 1340 -(10.38 * sfVcInfo.h );
         if (sfVcInfo.x < 0) // se blob a destra
                          { yy = BLOB_Y(&sfVcInfo, xx)*(1.2);}
            else // blob a sinistra
                          { yy = BLOB_Y(&sfVcInfo, xx)*(0.8);}
          xx = 290;

          blob_pt = sfCreateLocalPoint (xx , yy , 0.0 ) ;
           sfAddPoint (blob_pt ) ;
           sfKillIntention (p ) ;
           p = NULL ;
           process_state = sfSUCCESS;
                    break;
      }
      }
      process_state = 21 ;
      break;

     case 22: // blob non trovati
      sfKillIntention (p ) ;
      p = NULL ;
      sfMessage ("No blobs 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;
    }
}



/* approach_blob : avvicinamento al blob */

void
approach_blob(void)
{
  static sfprocess *p = NULL;
  int x;
  static  int area;
  float xx, yy, speed;
  switch(process_state)
    {
    case sfINIT:
    case sfRESUME:
      area = 0;
      process_state = 20;
      break;

    case 20:

       /* setta la velocità in base al bolob da raggiungere
      if (_process_params[0].i == 0) // VERDE
         {speed = VEL2;}
         else speed = VEL1;

      // avvicinamento
      p = sfStartBehavior(sfAttendAtPos,"approach blob",0,3,0,
                   speed, blob_pt, 110.0);

      process_state = 25;
      break;


    case 25:
      if (found_blob(_process_params[0].i, 60) > -500)
       // se vede il blob aggiorna il punto
      {
        switch (_process_params[0].i)
        {
      case 0: //VERDE

            xx = 2800 - (10 * sfVaInfo.y);
            sfVaInfo.x = sfVaInfo.x - 21;
            yy = BLOB_Y(&sfVaInfo, xx);
//                   sfSMessage("stimo la distanza del Verde  a x= %f  y=
%f", xx, yy);

                  // filtro sull' aggiornamento
                   // aggiorna solo se quello che vedi non _ pi— lontao
di 300mm dall'ultimo visto
                        if (  ((fabs(xx - blob_pt->x)) < 300 ) &&
                             ((fabs(yy - blob_pt->y)) < 300 ))
```

```
                        {
//                      sfMessage ("aggiornamento punto verde");
                  blob_pt->x = xx;
                  blob_pt->y = yy;
                              sfSetGlobalCoords(blob_pt);
                  }
            break;

         case 1: // GIALLO

            xx = 1340 - (10.38  * sfVbInfo.h ) ;
            yy = BLOB_Y(&sfVbInfo, xx);
            blob_pt->x = xx;
            blob_pt->y = yy;
                        sfSetLocalCoords(blob_pt);
             break;

       case 2: // ROSSO
            if (sfVcInfo.area > 5000) area = 1; // setta area=1 se sei
molto vicino
            // crea un punto nella direzione del centro del blob davanti
al robot  290mm
                              xx = 1340 -(10.38 * sfVcInfo.h );
            if (sfVcInfo.x < 0){ yy = BLOB_Y(&sfVcInfo, xx)*(1.2);}
                     else { yy = BLOB_Y(&sfVcInfo, xx)*(0.8);}
            blob_pt->x = 290;
            blob_pt->y = yy;
                        sfSetGlobalCoords(blob_pt);
                        break;
        }
      }
      else {//non vede il blob
               if ((_process_params[0].i == 2) && (area == 1))
              { // se st… approcciando al rosso e area =1 (molto vicino)
ferma il robot
               sfSetVelocity(0);
               sfKillIntention(p);
               p = NULL;
               sfMessage ("non vedo piu il rosso");
               process_state = sfSUCCESS;
      blob_pt->viewable = 0;
      break;
              }

           }
      if ( p->state==3) // arrivato
      {
                 sfSetVelocity(0);
        process_state = sfSUCCESS;
        blob_pt->viewable = 0;
        break;
      }
       }
}


/* search_and_go_blob: ricerca un blob e approccia */
void
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,
                    1.4, /* front sensitivity */
                    1.0, /* 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;
      }
      if (sfFinished(p))
      {
        sfKillIntention(p);
        p = NULL;
        sfSuspendSelf(0);
        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)))
      {

        sfStartBehavior (sfStop,"Stop", 0, 0, 0);
        sfKillIntention(p);
        p = NULL;
        sfSuspendSelf(0);
        break;
      }
      break;
    case sfINTERRUPT:
      if (p) sfKillIntention(p);
      p = NULL;
      sfSuspendSelf(0);
      break;
```

```
    }
}


/* parking : funzione di parcheggio
void
parking (void)
{
   static sfprocess *p = NULL;
   switch(process_state)
    {
    case sfINIT: // attiiva la stop collision
      _process_params[1].p = sfStartBehavior(sfStopCollision,"stop
collision", 0, 1, 0,
                    1.4, /* front sensitivity */
                    1.0, /* side sensitivity */
                    flakey_radius + 50.0 /* standoff */
                      );

      sfSuspendSelf(0);
      break;

    case sfRESUME:
      process_state = 20;
      break;

     case 20:

      /* ricerca del verde */
      beeeep();
      sfMessage("Searching GREEN blob.");
      p = sfInitIntention(find_blob, "find a blob", 0,
                    sfINT, 0, sfEND);
      process_state = 30;
      break;

    case 30:
      if (p->state == sfSUCCESS)
      {/* trovato  */
        printf("Kill intention\n");
        sfKillIntention(p);
        p = NULL;
        process_state = 40;
        break;
      }
      if (sfFinished(p)) /* se time out e blob non trovato*/
      {

        sfMessage ("GREEN blob not found. ");
                  boop();
        sfKillIntention(p);
        p = NULL;
        process_state=35;
        break;
       }
      break;

      case 35: /* avanza di 500 mm*/

          blob_pt = sfCreateLocalPoint (500, 0.0 , 0.0 ) ;
          sfAddPoint (blob_pt ) ;
```

```
          p = sfStartBehavior(sfGoToPos,"avanzamento",0,4,0,
                   VEL2, blob_pt, 110.0);

          process_state=36;
          break;
    case 36: /* durante l'avanzamento controlla presenza ostacoli a 550
mm dal robot */

         if ((_process_params[0].i==0)&&
              ((sfSonarRange(1)<550) || (sfSonarRange(2)<550) ||
(sfSonarRange(3)<550) ||                  (sfSonarRange(5)<550) ||
(sfSonarRange(4)<550)))
           { /* trovato un ostacolo : ferma il robot */
             sfMessage("vedo un ostacolo !!!");
             boop();
             sfSetVelocity(0);
             sfKillIntention(p);
             p = NULL;
             process_state = 51;
             break;
            }

          if (p->state == sfSUCCESS)
          {/* terminato l'avanzamento dei 500mm
           sfKillIntention(p);
           p = NULL;
           process_state = 20;
           break;
          }
         break;

    case 40: /* inizia l'approccio al verde se lo vedo ancora bene come
prima */

        if (!(!(!(found_blob(0, 60) > -500)))) /* filtro contro eventuali
riflessi verdi */
         { /* non vedo pi— il blob
           process_state = 20;
           break;
         }

      sfMessage ("GREEN blob found. I'm going there.");
      _process_params[0].i=0;
      p = sfInitIntention(approach_blob, "go get green", 0,
                   sfINT, _process_params[0].i,
                   sfPTR, _process_params[1].p, sfEND);

      process_state = 50;
      break;

    case 50: /* avvicinamento al blob verde */

       /* controllo presenza di ostacoli a meno di 550mm*/
      if ((_process_params[0].i == 0)&&
           ((sfSonarRange(1)<550) || (sfSonarRange(2)<550) ||
(sfSonarRange(3)<550) ||                  (sfSonarRange(4)<550) ||
(sfSonarRange(5)<550))
          )
      {/* trovato un ostacolo: ferma il robot */
        sfMessage("vedo un ostacolo !!!");
        boop();
```

```
        sfSetVelocity(0);
        sfKillIntention(p);
        p = NULL;
        process_state = 51;
        break;
       }

       if (sfFinished(p))
        { /* verde raggiunto */
      sfMessage ("Raggiunto il verde");
      sfKillIntention(p);
      p = NULL;
      process_state = 60;
      break;
        }

      break;

case 51: /* inizia rotazione di 90 gradi

       /*inizia rotazione */
       p = sfStartBehavior(sfTurnLeft,"sfTurnLeft", 50, 2, 0, VEL_R_2);
       process_state = 52;
       break;

    case 52:
       if (sfFinished(p))
        { /* fine rotazione 90 gradi */
        sfKillIntention(p);
        p = NULL;
        process_state = 35;
        }
       break;


    case 60:
      /* cerca il giallo e il rosso*/
       sfMessage ("Searching YELLOW and RED blobs. ");
       _process_params[0].i=2;
      p = sfInitIntention(find_yellow_and_red, "find red and yellow", 0,
                   sfINT,  _process_params[0].i, sfEND);
      process_state = 65;
      break;

    case 65 :
     if (p->state == sfSUCCESS)
      {/* blob trovati*/
                 sfMessage ("Found my park. ");
        sfKillIntention(p);
        p = NULL;
        process_state = 70;
        break;
      }
      if (sfFinished(p))
      {/* bolb ROSSO e GIALLO non trovati */
        sfMessage ("Park not found. I'm searching another. ");
        sfKillIntention(p);
        p = NULL;
        process_state=20;
        break;
      }
```

```
      break;

      case 70: /* avanza verso armadio*/

      sfMessage ("Parking in action. ");
     _process_params[0].i=2;
      p = sfInitIntention(approach_blob, "go get red", 0,
                  sfINT,  _process_params[0].i,
                  sfPTR, _process_params[1].p, sfEND);

      process_state = 90;
      break;

      case 90: /* attesa entrata nell'armadio */
      if (sfFinished(p))
      {/* entrato nell'armadio*/
        sfKillIntention(p);
        p = NULL;
        process_state = 95;
        break;
      }
      break;

 case 95: /*ricerca blob ROSSO piccolo*/

      p = sfInitIntention(find_little, "find a little blob", 0,
                  sfINT, 2, sfEND);

      process_state = 100;
      break;


      case 100:

      if (sfFinished(p))
      {
        beep();
       sfMessage("Mission accomplished.");
       sfKillIntention(p);
        p = NULL;
       sfDisconnectFromRobot();
        sfSuspendSelf(0);
        break;
               }
      break;


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

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 green", 0, sfINT, 0,
sfEND);
      sfInitIntention(search_and_go_blob, "search Yellow", 0, sfINT, 1,
sfEND);
      sfInitIntention(search_and_go_blob, "search red", 0, sfINT,2,
sfEND);
      sfInitIntention(parking, "effettua parcheggio", 0, sfINT, 0,
sfEND);

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

    }
}

/*
 * myConnectFn: sequenza d'avvio
 */

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


void
turn_off_everything(void)
{
  sfSetProcessState(sfFindProcess("search green"), sfINTERRUPT);
  sfSetProcessState(sfFindProcess("search Yellow"), sfINTERRUPT);
  sfSetProcessState(sfFindProcess("search red"), sfINTERRUPT);
  sfSetProcessState(sfFindProcess("effettua parcheggio"), sfINTERRUPT);
}


void
SearchGreen(void)
{
  turn_off_everything();
  sfSetProcessState(sfFindProcess("search green"), sfRESUME);
}

void
SearchYellow(void)
{
  turn_off_everything();
  sfSetProcessState(sfFindProcess("search Yellow"), sfRESUME);
}

void
SearchRed(void)
{
  turn_off_everything();
```

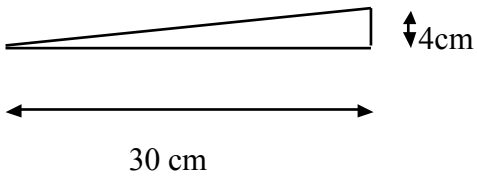
```
  sfSetProcessState(sfFindProcess("search red"), sfRESUME);
}

void
Park(void)
{
  turn_off_everything();
  sfSetProcessState(sfFindProcess("effettua parcheggio"), sfRESUME);
}

#include "saphira.h"

/*
 * setup_vision_system: esegue il settaggio del sistema di visione
 */

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");
  sfRobotComStr(VISION_COM,"line_bottom_row=180"); /* y-th row of image
*/
  sfRobotComStr(VISION_COM,"line_num_slices=6");  /* number of line
slices */
  sfRobotComStr(VISION_COM,"line_slice_size=10"); /* number of rows in
each slice */
  sfRobotComStr(VISION_COM,"line_min_mass=10");   /* how wide slices have
to be */
}


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


int
foundb(struct vinfo *v, int delta, int can)
{
  if (v->type == BLOB_MODE)
    {

     switch(can)
     {
      case 1:
      case 2:
      if (v->area >= found_blob_m_area && ABS(v->x) <= delta &&
      v->area <=found_blob_M_area)
      return v->x;
      else
      return -1000;
      break;

      case 0:
      if (v->area >= found_blob_m_area && ABS(v->x) <= delta &&
      v->area <= found_blob_M_area && v->y >110)
      return v->x;
```

```
      else
      return -1000;
      break;

     }
    }
}

int
found_area_b(struct vinfo *v, int delta, int area_m, int area_M, int
y_max)
{
  if (v->type == BLOB_MODE)
    {

      if (v->area >= area_m && ABS(v->x) <= delta && v->area <= area_M &&
      (v->y)< y_max)
      return v->area;
      else
      return -1000;
    }
}


/* found_blob: restituisce le coordinate x del blob riconosciuto sul
canale channel  se esso soddisfa le condizioni di dimensione minima,
proporzioni e i limiti imposti*/

int
found_blob(int channel, int delta)
{
  switch(channel)
    {
    case 0:
      found_blob_m_area = 10;
      found_blob_M_area = 500;
      can = 0;
      return foundb(&sfVaInfo, delta, can);
    case 1:
      found_blob_m_area = 100;
      found_blob_M_area = 10000;
      can = 1;
      return foundb(&sfVbInfo, delta, can);
    case 2:
      found_blob_m_area = 200;
      found_blob_M_area = 3000000;
      can = 2;
      return foundb(&sfVcInfo, delta, can);
    }
}



int draw_blob_thresh = 10;
int draw_line_thresh = 2;

void
draw_one_blob(struct vinfo *v, int color)
{
  int x, y, w, h;
  if (v->area >= 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("Coordinate camera x= %d  y= %d area %d h= %d w=%d", v-
>x, v->y, v->area, v->h, v->w);
     }
}


/*draw_blobs: disegna un rettangolo a monitor per  ogni canale che
rappresenta il blob riconosciuto sul canale corrispondente. Le dimensioni
del rettangolo sono proporzionali alle dimensioni del blob*/

void
draw_blobs(void)
{
  float xx, yy;
  switch(process_state)
    {
    case sfINIT:
    case sfRESUME:
      draw_one_blob(&sfVaInfo,sfColorDarkOliveGreen);
      draw_one_blob(&sfVbInfo,sfColorYellow);
      draw_one_blob(&sfVcInfo,sfColorRed);
      break;
    }
}

/* beep, beeeep, boop : emette segnali acustici di diverse tonalità e
durata*/

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

void
beeeep(void)
{
  sfRobotComStrn(sfCOMSAY,"\100\003",2);
}
void
boop(void)
{
  sfRobotComStrn(sfCOMSAY,"\25\060",2);
}
```

PROBLEMI E CONSIGLI:

indice



Per permettere l'ingresso nell'armadio abbiamo dovuto predisporre una pedana per poter superare il gradino di 4cm della base del mobile. La pedana è alta 4cm e lunga 30cm ed è stata realizzata in questo modo perché l'inclinazione risultante è la massima che il robot caricato con il computer portatile riesce a superare con batterie mediamente cariche.

La scelta dei colori deve essere fatta in modo attento tenendo conto dell'ambiente dove si svolgono le operazioni. I blob non devono essere lucidi (per evitare i riflessi dei neon che alterano la lunghezza d'onda riflessa e quindi anche la forma dei blob) e devono contrastare bene con lo sfondo nel quale sono inseriti. Il blob verde è stato realizzato mediante un pezzo di moquettes verde con piccoli pois neri. Questo tipo di tessuto è risultato particolarmente adatto in quanto risponde molto bene ai due requisiti precedenti: la sua superficie frastagliata rompe i raggi riflessi ed il suo colore ben contrasta con il pavimento rosso sul quale è adagiato. Il blob roso risulta valido in quanto il rosso è il colore che viene visto meglio dal sistema di visione e ben contrasta con il colore grigio-azzurro dell'armadio sul quale è applicato. Il giallo è un colore che favorisce le riflessioni ma è stato usato in quanto è servito solo come riferimento ed è stato l'unico colore, oltre al rosso, che ben contrastava con l'armadio.

Cognachrome Vision System:

Vantaggi incontrati :

- il sistema consente di avere a disposizione parecchie informazioni relative al blob di interesse in tempo molto rapido.

Svantaggi riscontrati :

- limitato controllo sulle operazioni effettuate dal sistema (dovuto anche alla esiguità del materiale informativo che si trova nel manuale).
- difficoltà nell'apprendimento dei colori quanto non si può intervenire direttamente sulla lunghezza d'onda. In questo modo le alterazioni dovute alla luce ed ai riflessi influenzano notevolmente l'apprendimento. Ci sono problemi anche a causa del contrasto eccessivo che il sistema crea nei suoi frames. Questo contrasto altera i colori aumentandone la luminescenza al punto che i colori più chiari diventano tutti bianchi ed i blob non vengono più riconosciuti.
  Il problema è stato risolto applicando un "cappellino" alla telecamera, una sorta di parasole di colore nero il quale fa in modo che in ogni frame sia presente una certa

quantità di colore nero, che mantiene fisso il livello del contrasto realizzato dal sistema di elaborazione dell'immagine, indipendentemente dalle variazioni delle condizioni di illuminazione.

- dovendo poi memorizzare su EEPROM i colori di interesse, se non si lasciano canali liberi per altri gruppi di lavoro, solo un gruppo alla volta può usufruire del sistema (si avverte l'esigenza di un apprendimento da file).
- I colori devono essere scelti in modo oculato: non devono essere superfici lucide e riflettenti ma opache (come per es: la moquettes) e devono essere colori che contrastano 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 specifiche condizioni di illuminazione.

Make_file e compilazione: indice

La programmazione in "C" con questo tipo di compilatore ci ha dato parecchi problemi anche perché in alcuni casi il compilatore interpretava in modo errato alcuni comandi standard del linguaggio "C" come per esempio le "//" , in quanto la riga seguente non viene completamente considerata come un commento.

```
#-----------------------------------------------------------------
#
#  Makefile per il file parking                    ottobre 1998
#
#-----------------------------------------------------------------


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

X11D = /usr/X11
BIND = ./
#Directory file bin

SRCD = src/
#Dir. dove ci deve essere il sorgente

OBJD = obj/
#Dir. dove mettera gli obj

INCD = -include/ -I$(SAPHIRA)/handler/include/ -I$(X11D)include/
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

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

```
CFLAGS = -g -D$(CONFIG) $(OFLAGS) $(INCD)

INCLUDE = -I$(INCD) -I$(X11D)include

#
# rules
#

all: $(BIND)parking  #nome eseguibile totale
      touch all

$(OBJD)parking.o: $(SRCD)parking.c
      $(CC) -c $(CFLAGS) $(SRCD)parking.c $(INCLUDE) -o $(OBJD)parking.o

 $(OBJD)fastrack1.o: $(SRCD)fastrack1.c
      $(CC) -c $(CFLAGS) $(SRCD)fastrack1.c $(INCLUDE) -o
$(OBJD)fastrack1.o


# Se ci sono piu' file .c copiare le 2 righe sopra cambiando il nomefile
# e aggiungere il .o sotto (es. $(OBJD)albero.o $(OBJD)albero1.o)

/*OBJS = $(OBJD)parking.o $(OBJD)fastrack1.o $(OBJD)btech.o
$(OBJD)chroma.o*/

OBJS = $(OBJD)parking.o

$(BIND)parking: $(OBJS)
      $(CC) $(OBJS) -o $(BIND)parking \
-L$(MOTIFD)lib $(LIBD) -lsf $(LIBS) -lc -lm


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

fine del make

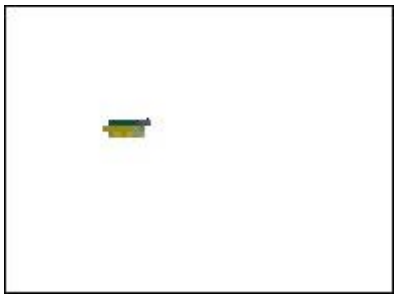
In Saphira:

Abbiamo scoperto, analizzando alcuni files dimostrativi provenienti da ricerche in Internet, l'esistenza di funzioni non comprese nel manuale di Saphira (e di altre poco commentate) e pertanto consigliamo, a chi intendesse svolgere dei lavori con questo tipo di linguaggio, di studiare i file che attualmente si trovano registrati nel computer denominato Golem del LDRA dell'Università di Brescia.

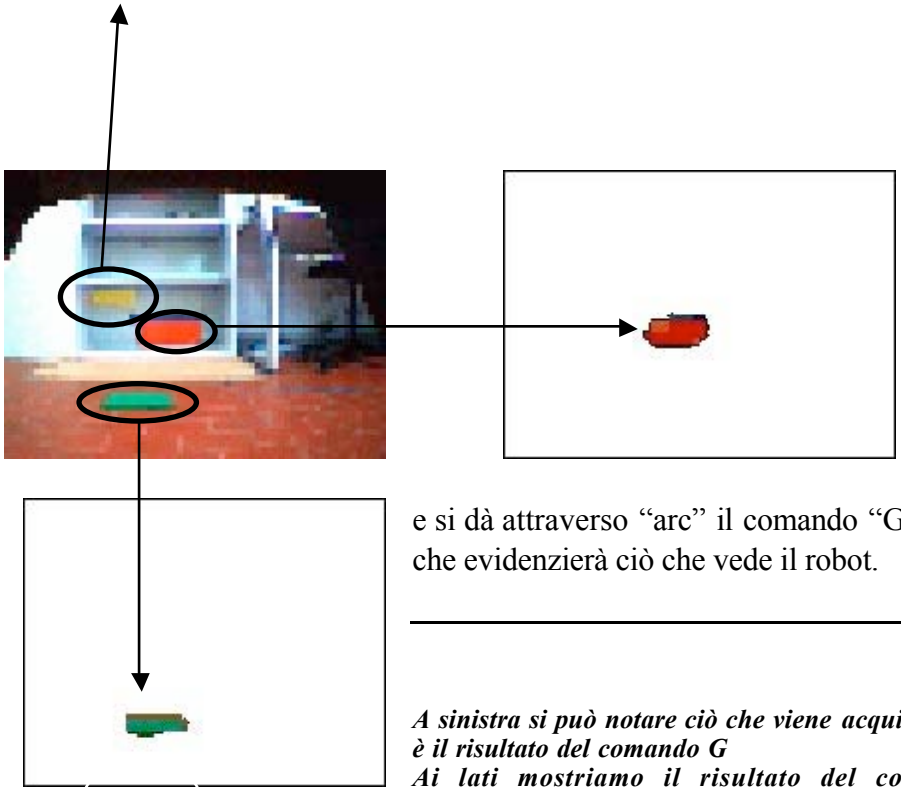
Come si usa il programma <<Parking>>:

indice

Per poter utilizzare il nostro programma anche in un altro contesto operativo occorre prima di tutto scegliere i colori (come spiegato in precedenza) in base al luogo ed all'illuminazione



dell'ambiente.
L'apprendimento dei colori deve essere effettuato chiamando il programma "arc" con la seguente riga di comando: << arc -saphira>>. In questo modo si è in comunicazione diretta con il sistema di visione. Si posiziona il robot di fronte al colore che si vuole far apprendere



e si dà attraverso "arc" il comando "G". apparirà un frame che evidenzierà ciò che vede il robot.

A sinistra si può notare ciò che viene acquisito dalla telecamera ed è il risultato del comando G
Ai lati mostriamo il risultato del comando Ta,Tb,Tc che corrispondono al
Verde (in basso), al rosso (destra) ed al giallo(in alto).

Occorre fare in modo che il colore che si vuole far apprendere stia al centro dell'immagine.
A questo punto si dà il comando "sX" dove X rappresenta il canale a, b o c che si vuole usare.

E' possibile verificare ciò che il sistema di visione ha acquisito utilizzando il comando TX; questo ordine mostra a video uno schermo in cui appare colorato solo il blob riconosciuto. Di seguito è riportato ciò che la telecamera vede da una posizione generica e i blob che ha riconosciuto rispettivamente sul canale a,b,c.
Per assicurarsi il correto apprendimento è necesario fare molte prove da posizioni diverse e con diverse condizioni di luce prima di memorizare i dati di acquiszione.
Per memorizzare i colori acquisiti occorre dare il comando "S" che salverà i dati sulla eeprom. Prima di memorizzare i colori occorrerà predisporre un cappellino nero, una aletta parasole attorno alla telecamera per risolvere il problema del contrasto. La telecamera deve essere sistemata in modo perfettamente verticale per non dover modificare le stime dei blob. Il blob verde (o del nuovo colore) dovrà essere posizionato di fronte all'armadio, sul pavimento, in un punto dal quale il robot potrà entrare nell'armadio senza urtare i montanti laterali e nessun altro ostacolo. Il blob giallo andrà sistemato in un punto dell'armadio che dovrà essere visibile solo quando l'anta è aperta. Il blob rosso dovrebbe essere sistemato sul fondo dell'armadio con una striscia di carta di colore simile all'armadio che ne copre l'estremità. In questo modo, coprendo il blob, si decide la posizione di stop.

I risultati del progetto: indice

Riportiamo di seguito una serie di immagini ottenute durante una delle esecuzioni del programma

Landmark VERDE

La manovra vista da un'altra angolazione:

Bibliografia: indice

E' possibile ottenne informazioni aggiuntive consultando gli indirizzi:

http://www.ai.sri.com per avere informazioni ed aiuti riguardanti il linguaggio Saphira

http://www.newtonlabs.com per avere informazioni e consigli sull'uso di Cognachrome Vision System

franzzip@yahoo.com per avere informazioni sul progetto o per contattarci.