ROBOTICA

Prof. R.Cassinis

Elaborato Sperimentale

A.A.1999/2000

Bandera Andrea (30059)
Bonisoli Alquati Livio (30063)
Donini Maurizio (30061)

Movimentazione in corridoio

L'obiettivo da raggiungere è quello di percorrere un qualsiasi corridoio sfruttando l'illuminazione derivante dalle lampade sul soffitto. Il robot, partendo da un generico punto all'interno del corridoio, cerca la presenza di sorgenti luminose e si avvicina ogni volta a quella più vicina; l'effetto che se ne ottiene è quello di muoversi sotto la fila di luci e quindi di percorrere nella maniera corretta il corridoio.
La posizione da raggiungere viene ricavata attraverso una sequenza di operazioni che vengono effettuate sull'immagine inviata dal robot al computer attraverso il sistema telecamera-ponte radio. L'immagine che viene acquisita è in formato PPM, ha dimensioni 320*240 ed ogni pixel è costituito da 3 byte, uno per ogni componente RGB che quindi può assumere valori compresi tra 0 e 255. La telecamera è stata inclinata di circa 70° rispetto alla posizione classica (quella frontale) così da poter vedere sul soffitto almeno 2 o 3 lampade consecutive, in modo che se anche una di queste non funzioni il robot possa comunque proseguire il movimento avendo a disposizione quella successiva.

Figura 1: Esempio di immagine acquisita con parametri standard.

Elaborazione dell'immagine.

L'elaborazione dell'immagine si rende necessaria per poter eliminare tutte quelle informazioni supplementari dovute ai riflessi percepiti sopratutto sulle pareti laterali che confondono il robot portandolo nella direzione sbagliata.
Per prima cosa occorre settare bene i parametri di acquisizione della telecamera, sopratutto quelli di luminosità e contrasto, che dipendono fortemente dall'ambiente in cui si sta lavorando o in cui si pensa che il robot debba agire; sperimentalmente abbiamo ottenuti buoni risultati acquisendo immagini a bassa luminosità e basso contrasto, ottenendo un effetto occhiale da sole.

Figura 2: Immagine acquisita migliorando i parametri.

Successivamente l'immagine viene sfuocata mediante un filtro passa-basso sulle 3 componenti RGB applicando una maschera di convoluzione sul vicinato N8, ossia rimpiazzando le varie componenti del pixel in esame tenendo conto anche di quelli che lo circondano. L'immagine risulta uniformata dato che vengono eliminate le alte frequenze spaziali che corrispondono a brusche variazioni nelle componenti di pixel adiacenti.
A questo punto sull'immagine viene applicata una soglia sul valore di luminosità dei pixel che viene calcolato mediante la formula[1]:

$$Y=0.299*R+0.587*G+0.114*B$$

Il valore della soglia è un altro parametro molto importante che deve essere settato a seconda dell'ambiente di lavoro e per questo fortemente variabile. Dato che stiamo isolando sorgenti luminose si tratta di un valore piuttosto alto, compreso nel range [0-255] dei possibili valori di Y. Nelle prove effettuate abbiamo utilizzato una soglia compresa tra 180 e 200.

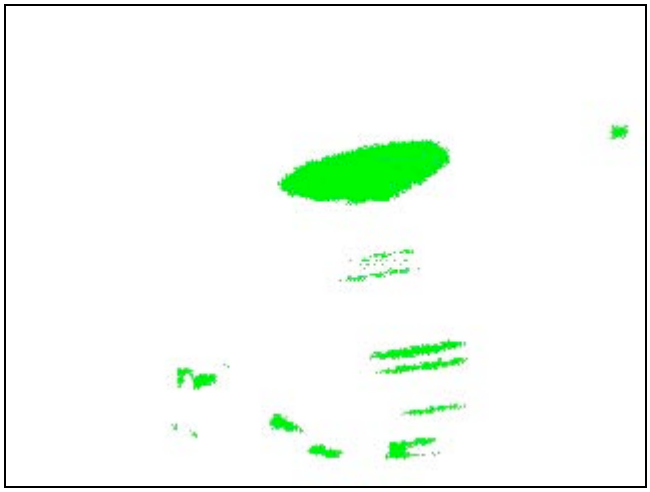
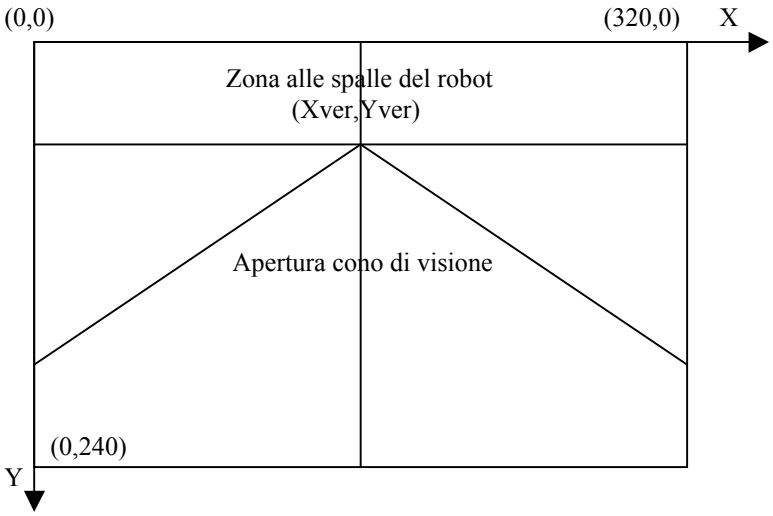


Figura 3: Immagine sogliata.[2]

[1] Formula valida per lo standard NTSC per il calcolo della luminosità.
[2] Le immagini elaborate originali hanno i pixel sotto soglia di colore nero.

A questo punto l'immagine viene segmentata, ossia vengono catalogati i blocchi di pixel che sono risultati sopra soglia, memorizzandone le caratteristiche principali quali posizione del baricentro, area, perimetro ed estremi in un opportuno array.
L'ultimo passo è l'eliminazione di ulteriori blocchi attraverso l'analisi delle loro caratteristiche calcolate precedentemente; sono stati imposti vincoli da rispettare sul valore dell'area e sul rapporto perimetro-area in modo da eliminare superfici troppo piccole o troppo frastagliate.
Dalle prove sperimentali abbiamo ottenuto ottimi risultati utilizzando il solo vincolo sull'area minima, lasciando invece aperti gli altri ed implementando una sorta di paraocchi sull'immagine. Questo sistema consiste nell'eliminare quei blocchi il cui baricentro non si trova all'interno dell'area caratterizzata da y>yver, dove (xver,yver) sono le coordinate del punto corrispondente alla verticale del robot che vanno modificate a seconda di come si sceglie di inclinare la telecamera, e da un parametro k che determina l'apertura del cono di visione utile (vedere disegno seguente).
Se consideriamo che la telecamera è puntata verso l'alto l'effetto che ne consegue è di guardare diritti verso la fila di luci senza tener conto degli spigoli tra le pareti laterali ed il soffitto. Ovviamente quei baricentri caratterizzati da y<yver vengono scartati perché corrispondono ad oggetti luminosi che sono alle spalle del robot.



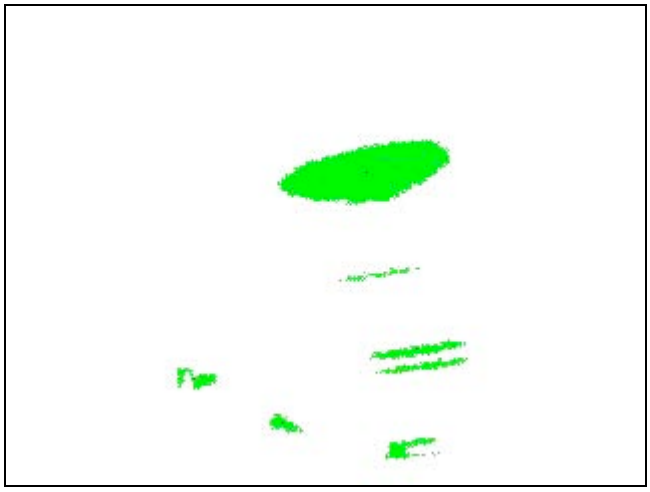


Figura 4: Immagine elaborata.

Di seguito sono riportate due immagini relative ad un'altra prova:

Figura 5: Immagine acquisita.

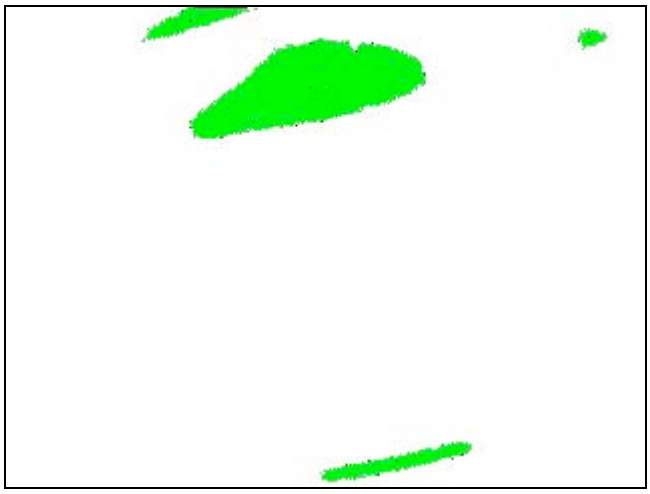


Figura 6: Immagine elaborata.

Estrazione delle informazioni dall'immagine.

A questo punto tra tutti i baricentri ottenuti dopo l'elaborazione si sceglie quello più vicino alla verticale e si calcola l'angolo di cui far ruotare il robot mediante l'equazione:

$$angolo=arcotan(x,y)$$

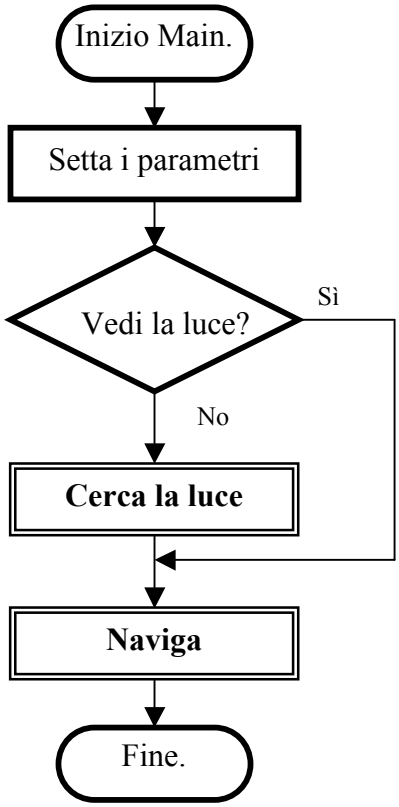
dove x ed y sono le distanze del baricentro selezionato dalla verticale lungo i due assi ed arcotan è una versione scalata di un fattore 2 dell'arcotangente:

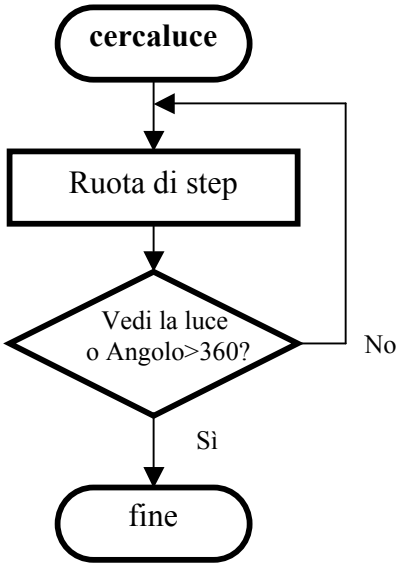
$$arcotan(x,y)=atan(x/2y)$$

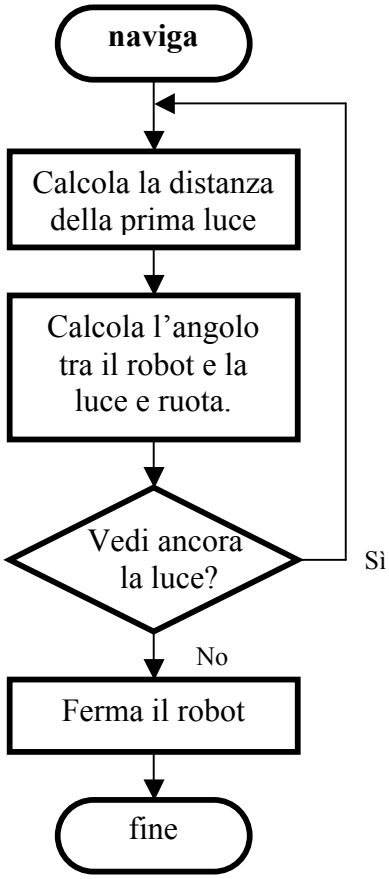
Questo fattore di scala è stato inserito dopo aver notato che quando x ed y erano quasi uguali, ossia il loro rapporto circa 1, una piccola variazione di uno di essi causava una variazione significativa del rapporto e di conseguenza dell'angolo calcolato causando vistose oscillazioni nel movimento del robot.

Algoritmo di movimento.

Di seguito è riportato lo schema a blocchi di massima dell'algoritmo implementato per la movimentazione del robot. Abbiamo disegnato separatamente le due activities chiamate dalla procedura principale. La stessa rappresentazione verrà utilizzata per descrivere gli altri due algoritmi.







Listato.

```
/*
Movimento in corridoio.
(C) 2000 Bandera Andrea, Bonisoli Alquati Livio, Donini Maurizio
*/

load proglib.so;        /* Carica lo sharedable object della libreria*/

/* activity per il movimento del robot sotto le luci */
act naviga()
{
 int dx;
 int dy;
 int stp;
 int vel;
 float angolo;
 int a;
 stp=0;
 vel=150;
 /* Ciclo di navigazione */
 while(stp<3)
 {
      a=procuno(&dx,&dy);           /*Calcola la distanza dalla verticale*/
      sfSMessage("Punto da raggiungere: X:%d Y:%d",dx,dy);
      if((dy<6)&&((dx<5)||(dx>-5))) stp=stp+1; else   stp=0;
      /*per fermarlo sotto la luce*/
      angolo=arcotan(dx,dy);
      sfSMessage("Angolo di rotazione: %f",angolo);
      sfSetVelocity(0);
      turn(-angolo);
      sfSetVelocity(vel);
 }

 /*Fermiamo il robot*/
 sfSetVelocity(0);
 sfSetRVelocity(0);
 halt;
}
/* Activity per la ricerca, in partenza, della prima luce*/
act cercaluce()
{
 int step;                          /* passo di rotazione*/
 int dx;
 int dy;
 int i;
 int a;
  step=30;
  dx=0;
  dy=0;
  i=0;
  sfSMessage("Mi giro x cercare qualche luce...o riflesso");
  /* Ruota finchè non vede la luce (max.360°)*/
  while (((dx==0)&&(dy==0))&&(i<360))
  {
      i=i+step;
      turn(step);
      a=procuno(&dx,&dy);
  }
  if (i>=360)     sfSMessage("Non vedo nessuna luce!");
            else  sfSMessage("Ho visto la luce");
}

act main
```

```
{
 int dx;
 int dy;
 int area;
 /* Settaggio dei parametri */
 luminosita=70;
 contrasto=100;
 xver=160;
 yver=40;
 areamin=100;
 areamax=200000;
 ratiomin=0.0;
 ratiomax=30000;
 livello=192;
 apertura=1.0;

 sfSMessage("PARAMETRI");
 sfSMessage("Soglia: %d", livello);
 sfSMessage("Luminosità: %d  Contrasto: %d",luminosita, contrasto);
 sfSMessage("Coordinate della verticale: X: %d Y: %d",xver, yver);
 sfSMessage("Area dei blocchi: Minima: %d  Massima: %d",areamin, areamax);
 sfSMessage("Rapporto per/area  Minimo: %f Massimo: %f",ratiomin, ratiomax);
 sfSMessage("Inizio navigazione...");

 area=procuno(&dx,&dy);
 /* Se non vede niente cerca la luce in partenza */
 if ((dx==0)&&(dy==0)) start cercaluce();
 start naviga() priority 2;
 sfSMessage("Fine navigazione.");
}

sfSetDisplayState(sfGLOBAL, 1);
start main;
```

Uscita da una stanza buia.

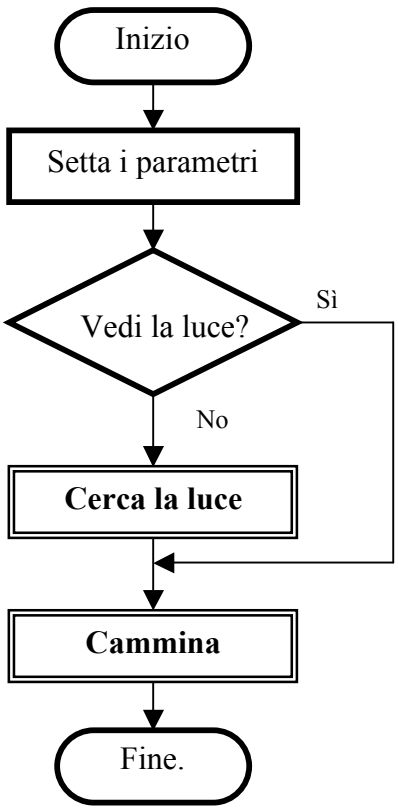
Elaborazione dell'immagine.

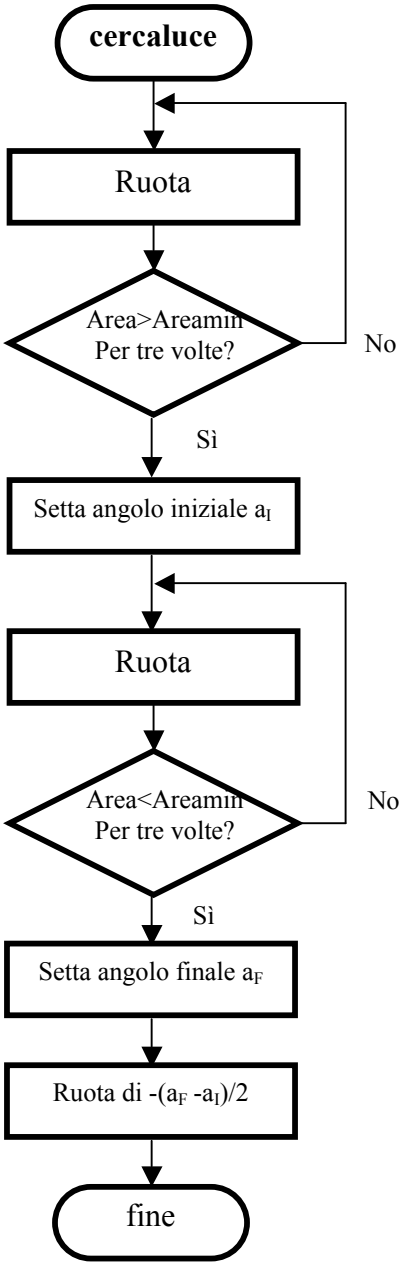
Siccome l'informazione necessaria per il controllo del robot era solamente l'area della zona luminosa che corrisponde all'uscita è sufficiente una elaborazione meno pesante a livello computazionale. Viene eseguita una sfocatura ed una soglia con un livello più basso rispetto al caso precedente (circa 150) per aumentare la sensibilità alla luce in modo da rilevare la luminosità maggiore all'esterno della stanza.

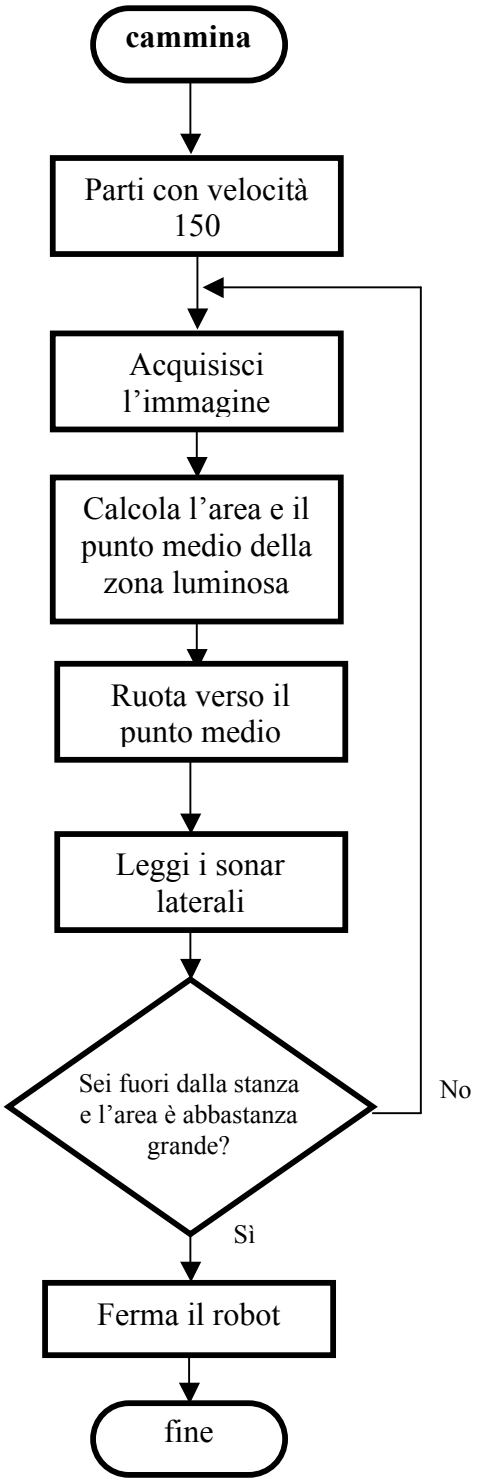
Estrazione delle informazioni dall'immagine.

Successivamente vengono calcolati gli estremi sinistro e destro della zona luminosa e l'area della stessa. Dagli estremi si ricava il punto medio e quindi la direzione da seguire mentre dall'area si ottiene un'informazione sulla distanza della porta dal robot, in quanto con l'avvicinarsi questa cresce.

Algoritmo.


Inizio
Setta i parametri
Vedi la luce?
Sì
No
Cerca la luce
Cammina
Fine.


cercaluce
Ruota
Area>Areamin
Per tre volte?
No
Sì
Setta angolo iniziale a_I
Ruota
Area<Areamin
Per tre volte?
No
Sì
Setta angolo finale a_F
Ruota di $-(a_F - a_I)/2$
fine


cammina
Parti con velocità 150
Acquisisci l'immagine
Calcola l'area e il punto medio della zona luminosa
Ruota verso il punto medio
Leggi i sonar laterali
Sei fuori dalla stanza e l'area è abbastanza grande?
No
Sì
Ferma il robot
fine

Listato.

```
/*
Uscita da una stanza buia
(C) 2000 Bandera Andrea, Bonisoli Alquati Livio, Donini Maurizio
*/

load proglib.so;  /* Carica lo lo shared object della libreria*/

/*Individua la porta e orienta il robot verso di essa*/
act cercaluce()
{
 int xi;
 int xf;
 int i;
 int area;
 int th;
 int st;
 st=0;
 area=0;
 i=0;
 sfSMessage("Continuo a girare fino a che trovo una luce");
 sfSetRVelocity(50);
 while ((area<areamin)&&(st<3))
 {
      /*turn(step) noblock;*/
      area=procdue(&xi,&xf);
      sfSMessage("Area: %d",area);
      if (area>areamin)  st=st+1;
      if (st==1)   sfRobot.ath=0;
 }
 st=0;
 while(st<3)
 {
      area=procdue(&xi,&xf);
      sfSMessage("B Area: %d Ang: %d",area,th);
      if (area<areamin)  st=st+1;
 }
 sfSetRVelocity(0);
 th=sfRobot.ath;
 turn(-th/2);
 sfSMessage ("Ho individuato la porta...");
}

/*Activity per far uscire il robot dalla stanza*/
act cammina()
{
 int D;
 int fermati;
 int area;
 int xi;
 int xf;
 int xm;
 int angolo;
 int sx;
 int dx;
```

```
   fermati=0;
   sfSetVelocity(150);
   while(fermati<3)
   {
    area=procdue(&xi,&xf);
    xm=(xi+xf)/2-xver;
    angolo=xm/6;
    turn(-angolo);
    dx=sfSonarRange(0);
    sx=sfSonarRange(6);
    if ((dx>600)&&(sx>600)&&(area>10000)) fermati=fermati+1;
    sfSMessage("Sonar %d %d",dx,sx);
  }
  sfSetVelocity(0);
  sfSetRVelocity(0);
}



act main
{
int xi;
int xf;
int a;
/*parametri*/
luminosita=70;
contrasto=100;
xver=160;
yver=0;
areamin=7000;
areamax=230000;
ratiomin=0.0;
ratiomax=30000;
livello=150;
sfSMessage("PARAMETRI");  sfSMessage("Soglia: %d",livello);
sfSMessage("Luminosità: %d  Contrasto: %d",luminosita,contrasto);
sfSMessage("Coordinate della verticale: X: %d Y:%d",xver, yver);
sfSMessage("Area dei blocchi: Minima: %d  Massima:%d",areamin, areamax);
sfSMessage("Rapporto per/area  Minimo: %f Massimo: %f",ratiomin,ratiomax);
sfSMessage("Inizio navigazione...");
a=procdue(&xi,&xf);
if (a<areamin) start cercaluce();
start cammina();
halt;
sfSMessage("Fine navigazione.");
}

sfSetDisplayState(sfGLOBAL, 1);
start main;
```

Ricerca della luce e avvicinamento.

Elaborazione dell'immagine.

In questo caso l'immagine acquisita, dopo essere stata sfuocata e dopo avervi applicato una soglia, deve essere elaborata per isolare le forme pressoché circolari corrispondenti alla sorgente luminosa (lampadina) da raggiungere.

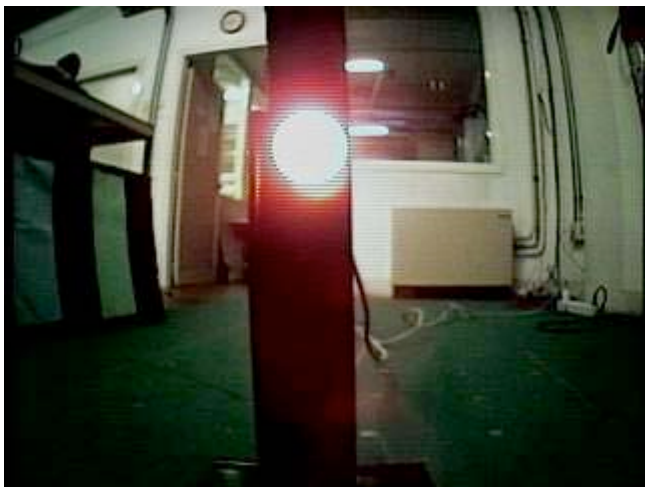


Figura 7: Esempio di immagine acquisita.

Di tutti i blocchi sopra soglia vengono calcolati i parametri e memorizzati in un opportuno array dinamico avente la seguente struttura:

- Id del blocco.
- Coordinate del baricentro.
- Area.
- Perimetro.
- Estremi lungo la coordinata X.
- Estremi lungo la coordinata Y.

Vengono eseguiti una serie di controlli sulle caratteristiche dei blocchi e di questi vengono eliminati quelli caratterizzati da:

- Dimensione orizzontale e verticale (calcolate come differenza degli estremi) troppo diverse fra loro.
- Raggio troppo piccolo. Il diametro viene calcolato come il valore medio delle dimensioni lungo X e Y.
- Area troppo piccola oppure troppo diversa dalla quantità $\pi(\text{raggio stimato})^2$.
- Percentuale pr troppo piccola (tipicamente sotto il 60%). Tale percentuale rappresenta il numero di pixel di perimetro che distano circa il raggio dal baricentro.
- Percentuale ar troppo piccola (tipicamente sotto il 60%). Tale percentuale rappresenta il numero di pixel del blocco appartenenti al cerchio avente il raggio stimato e centrato sul baricentro.

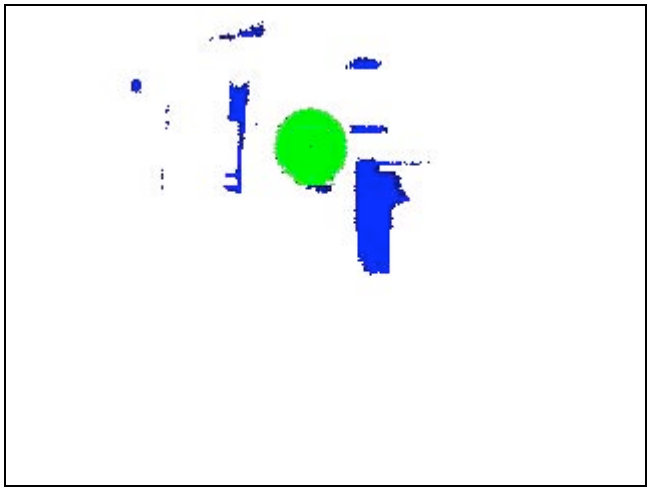


Figura 8: Immagine elaborata.

Di seguito sono riportate due immagini relative ad un'altra prova:

Figura 9: Immagine acquisita.

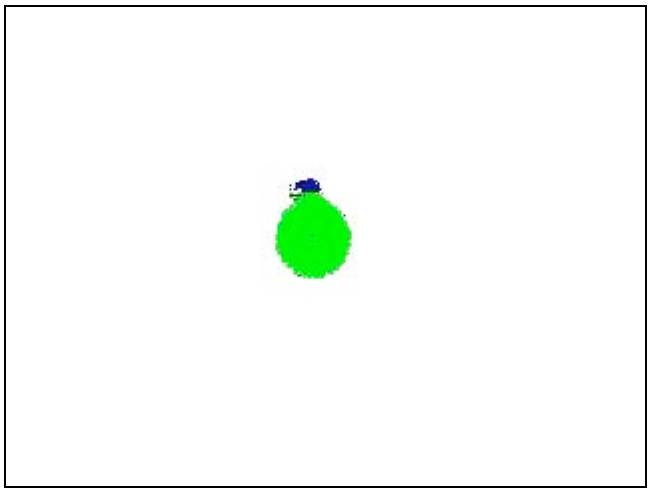
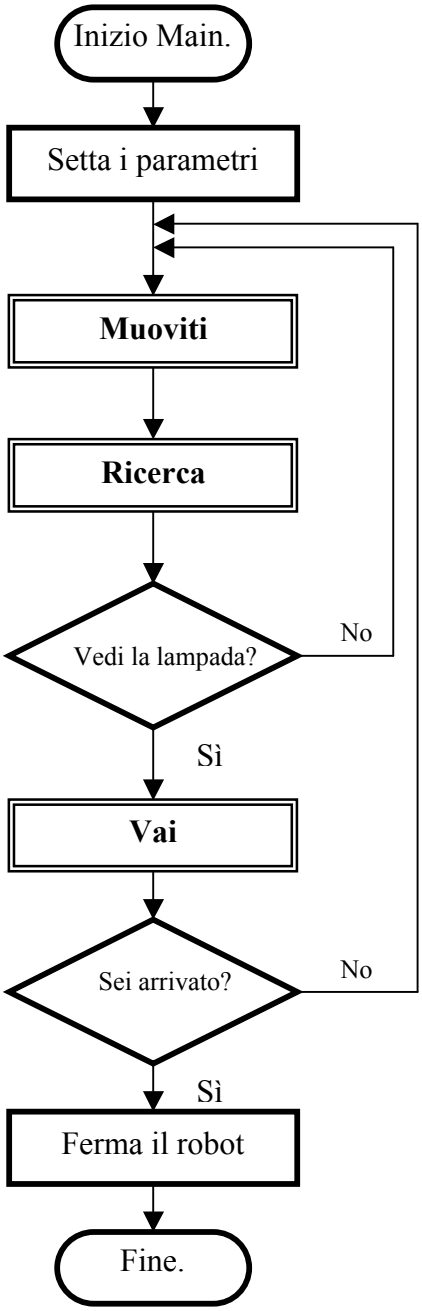


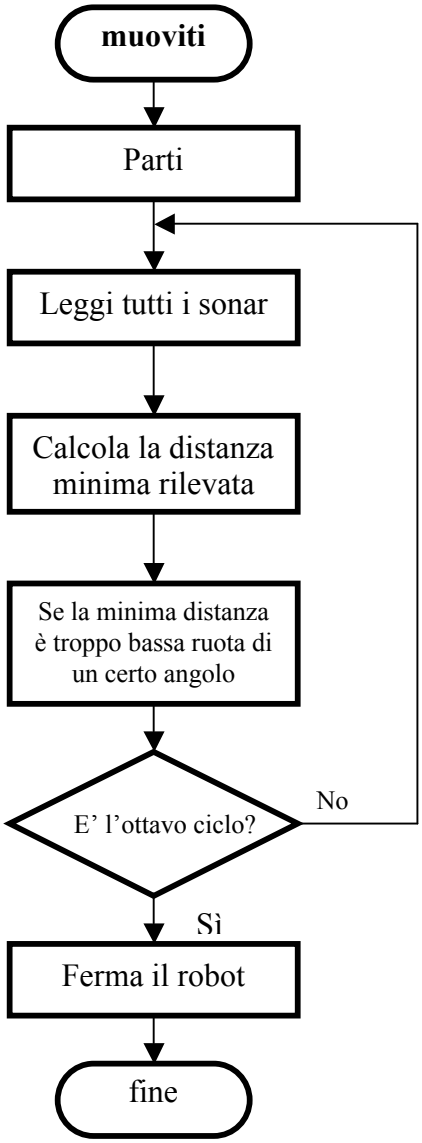
Figura 10: Immagine elaborata.

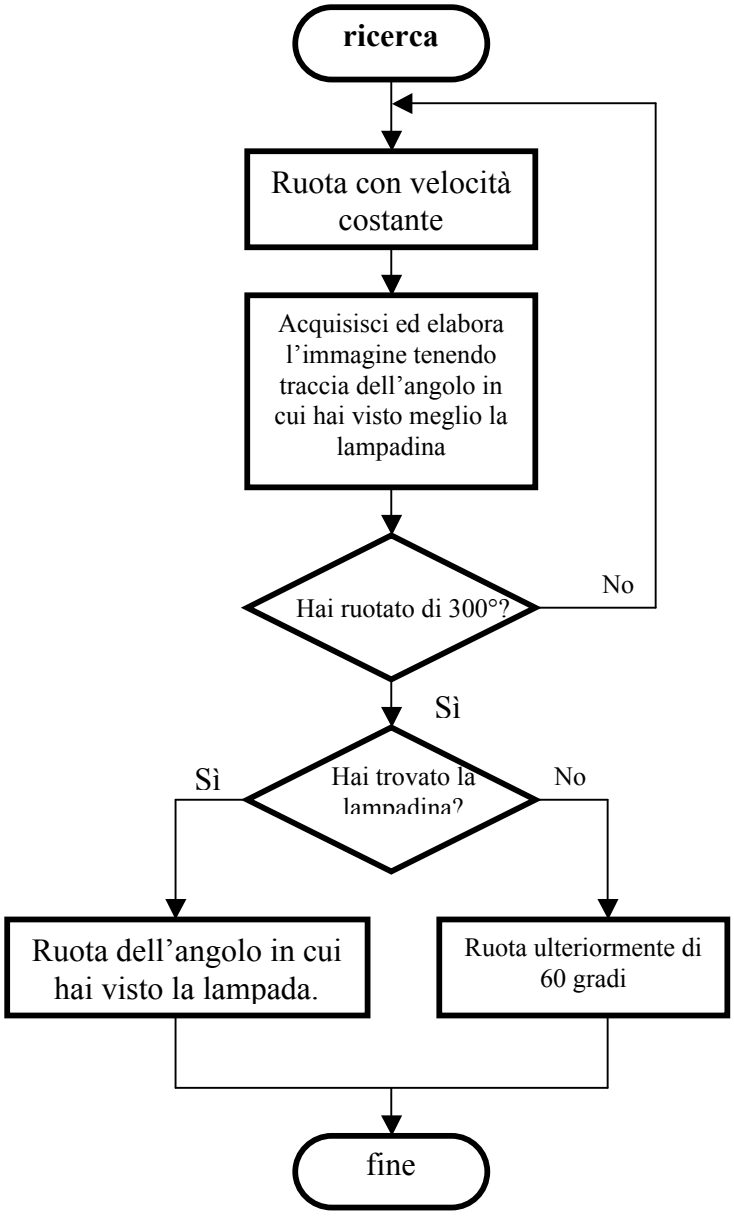
Estrazione delle informazioni dall'immagine.

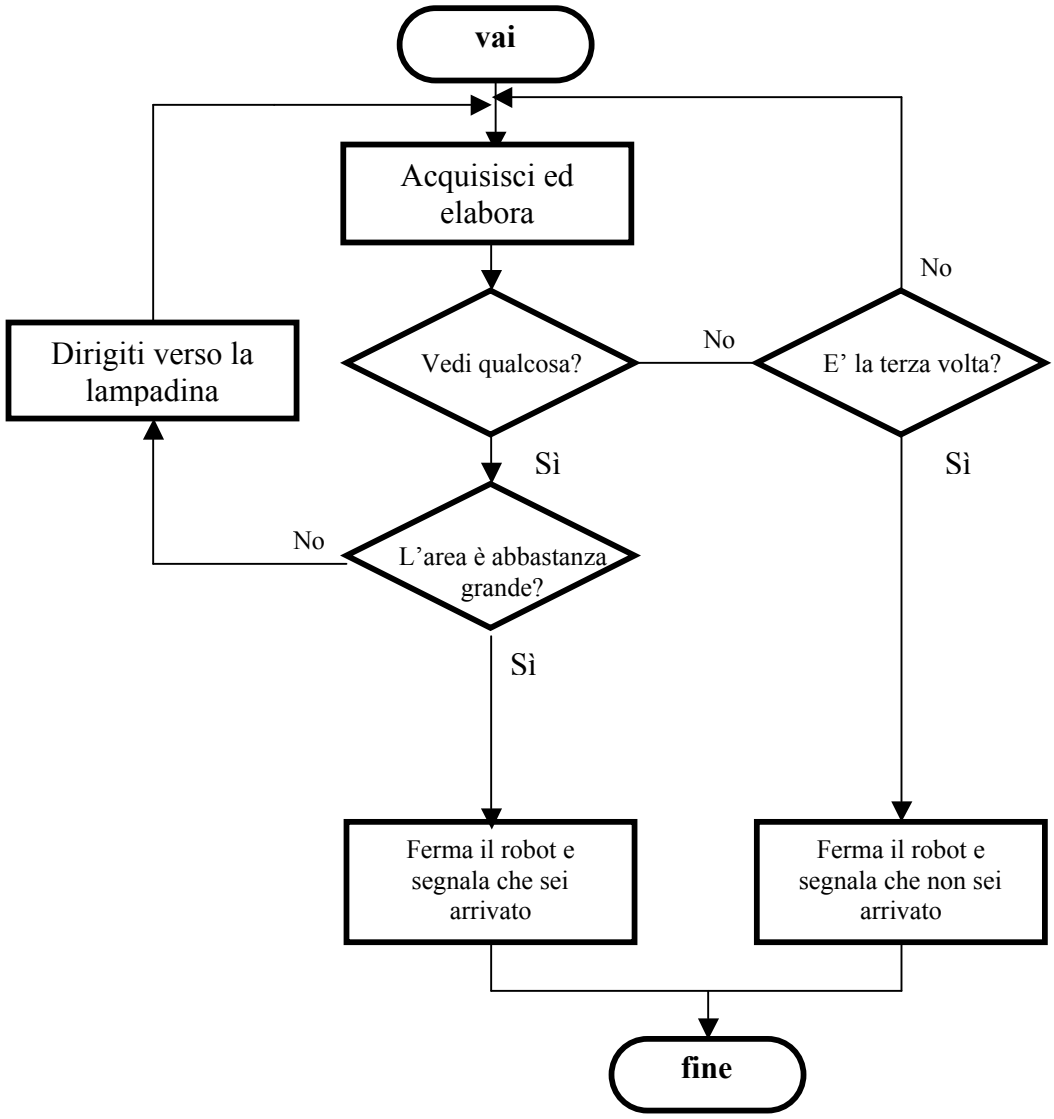
Tra tutti i blocchi rimanenti viene scelto quello che in base ai parametri risulta essere il più simile ad un cerchio e viene restituita l'area e la posizione del baricentro di tale blocco. L'area viene utilizzata per ricavare l'informazione sulla distanza e per far formare il robot quando questa supera un certo valore, mentre le coordinate del baricentro vengono usate per far muovere il robot nella direzione della lampada.

Algoritmo.






ricerca
Ruota con velocità costante
Acquisisci ed elabora l'immagine tenendo traccia dell'angolo in cui hai visto meglio la lampadina
Hai ruotato di 300°?
No
Sì
Hai trovato la lampadina?
Sì
No
Ruota dell'angolo in cui hai visto la lampada.
Ruota ulteriormente di 60 gradi
fine



Listato.

```
/* Ricerca della luce e avvicinamento
 (C) 2000 Bandera Andrea, Bonisoli Alquati Livio, Donini Maurizio
*/
load proglib.so;   /*Carica lo shared object della libreria*/

int blind;
int fermati;

/* Activity per il movimento casuale*/
act muoviti()
{
   int sonar;
   int D;
   int distanza;
   int nsonar;
   int alert;
   int ciclo;
   int angolo;
   int front;
```

```
   sfSMessage("Vago...");
   ciclo=0;
   sfSetVelocity(100); /*Parte piano*/
   while (ciclo<8)
   {
    nsonar=0;
    front=0;
    sonar=1;
    D=5000;
    alert=0;
    while (sonar<6)
    {
       distanza=sfSonarRange(sonar);
       if (distanza<D)
       {
          D=distanza;
          nsonar=sonar;
       }
       if (distanza<1000) front=front+1;
       sonar=sonar+1;
    }

    sfSMessage("Sonar max: %d",D);
    /*Controllo della direzione*/
    if (D<1500)
    {
       sfSetVelocity(100);
       if (nsonar<3) angolo=-45;
       if (nsonar>=3) angolo=45;
       /*Si gira solo se almeno 3 sonar vedono una distanza inferiore al metro*/
       if (front>2) angolo=180;
       turn(angolo);
    }
   else   sfSetVelocity(150);
   ciclo=ciclo+1;
   }/*while ciclo*/
   sfSetVelocity(0);
   sfSetRVelocity(0);
}


/*Activity per individuare la lampadina*/
act ricerca()
{
 int dx;
 int dy;
 int angolo;
 float best;
 float rapporto;
 int area;
 int mindx;
 rapporto=1;
 blind=0;
 angolo=0;
 best=1;
 mindx=160;
 sfRobot.ath=0; /* Setta a zero l'orientamento del robot*/
 sfSMessage("Cerco la luce...");
 sfSetRVelocity(40);
 while (sfRobot.ath<300)
 {
  /*Restituisce il grado di somiglianza del cerchio*/
  proctre(&dx,&dy,&area,&rapporto);
```

```
  sfSMessage("Rapporto: %f  Area:%d DX:%d",rapporto,area,dx);
  if ((rapporto<best)&&(rapporto<0.4))
    {
        best=rapporto;
      angolo=sfRobot.ath;
        mindx=dx;
    }
 }

 sfSetRVelocity(0);
 sfSMessage("Angolo con luce: %d",angolo);
 sfSMessage("Minimo dx : %d",mindx);
 blind= (best<0.99);
 if  (best>0.99)
 {
     sfSMessage("Non ho visto niente, continuo a vagare....");
     turn(60);
 }
 else
 {
     /*Si gira verso l'immagine migliore*/
     sfSMessage("HO TROVATO LA LAMPADA: %f",best);
     turn(-300+angolo-mindx/4);
  }
  sfSMessage("Punto da raggiungere: X:%d Y:%d Area: %d
Rapp:%f",dx,dy,area,rapporto);
}

/*Activity per dirigere il robot verso la lampadina*/
act vai()
{
 int dx;
 int dy;
 int stp;
 int ciclo;
 float angolo;
 int lost;
 int vel;
 float rapporto;
 int area;
 stp=0;
 angolo=0;
 lost=0;
 vel=150;
 ciclo=0;
 sfSMessage("Mi dirigo verso la luce...");
 /*Si ferma se sotto la lampada o se la perde*/
 while ((stp==0)&&(lost<3))
 {
   proctre(&dx,&dy,&area,&rapporto);
   sfSMessage("Punto da raggiungere: X:%d Y:%d Area: %d
Rapp:%f",dx,dy,area,rapporto);
   /*per fermarlo sotto la luce*/
   if (area>700)  {stp=1; vel=50;}          else stp=0;
   if ((area!=0)&&(rapporto<0.7)&&((dx>-150)&&(dx<150)||(ciclo==0)))
   {
       angolo=(dx/4);
       lost=0;
       sfSetVelocity(0);
       turn(-angolo);
       vel=150;
   }
   else
```

```
   {
       lost=lost+1;
       turn(-angolo/2.);
       vel=50;
   }
   sfSetVelocity(vel);
   ciclo=ciclo+1;
 }
 fermati=(lost!=2) ;
 sfSetVelocity(0);
 sfSetRVelocity(0);
 if (lost==2) sfSMessage("L'ho persa...");
}

act main
{
 /*settaggio parametri*/
 luminosita=60;
 contrasto=110;
 xver=160;
 yver=0;
 areamin=60;
 areamax=200000;
 ratiomin=0.0;
 ratiomax=30000;
 livello=202;
 sfSMessage("PARAMETRI");
 sfSMessage("Soglia: %d", livello);
 sfSMessage("Luminosità: %d  Contrasto: %d",luminosita, contrasto);
 sfSMessage("Coordinate della verticale: X: %d Y: %d",xver, yver);
 sfSMessage("Area dei blocchi: Minima: %d  Massima: %d",areamin, areamax);
 sfSMessage("Rapporto per/area  Minimo: %f Massimo: %f",ratiomin, ratiomax);
 sfSMessage("Inizio navigazione...");
 fermati=0;
 while(fermati==0)
 {
     start muoviti();
     start ricerca();
     fermati=1;
     if (blind==1)   start vai();
 }
  sfSetVelocity(0);
  sfSetRVelocity(0);
  halt;
 sfSMessage("Fine navigazione.");
}

sfSetDisplayState(sfGLOBAL, 1);     /* put display into global coords */
start main;             /* start up the toplevel activity */
```

Libreria Proglib.c

`unsigned char *leggimmagine(char filename[ ])`
Funzione che legge da disco l'immagine in formato ppm, specificata dal parametro *filename* , restituendo il puntatore all'area di memoria in cui viene copiata. La memoria necessaria deve venir allocata esternamente.

`int salvaimmagine(unsigned char *mem, char filename[ ])`
Funzione che scrive sul file specificato dal parametro *filename* il contenuto dell'area di memoria puntata da *mem.* Restituisce 0 se l'operazione si conclude correttamente.

`int grab(unsigned char *cop,int lum,int contr)`
Questa funzione acquisisce un' immagine in formato ppm dal frame grabber e la memorizza nell'area di memoria specificata da *cop*. E' possibile specificare i parametri di luminosità e contrasto della telecamera.
Il suo funzionamento si basa sulle funzioni contenute nella libreria "grab.c"[3] e in particolare abbiamo utilizzato la funzione *grab_open* per aprire il frame grabber, *grab_set_attr* per impostare i parametri di acquisizione, la funzione *grab_capture* che effettua il grabbing vero e proprio e *grab_close* che chiude il canale.

`int sfuoca(unsigned char *p)`
E' la funzione che si occupa di applicare un filtro di smoothing sul vicinato N_8 per ridurre il rumore puntuale, ossia tagliare le alte frequenze spaziali; il risultato è una sfocatura dell'immagine. Il parametro *p* è l'indirizzo dell'area di memoria in cui è memorizzata l'immagine.

`long int soglia(unsigned char *p, int S)`
Modifica l'immagine puntata da *p* annullando le componenti RGB dei pixel la cui luminosità risulta essere inferiore al parametro di soglia *S*. La luminosità viene calcolata con la formula, valida per lo standard NTSC:

$$Y= 0.299*R + 0.587*G +0.114*B$$

I pixel che possiedono luminosità sopra la soglia vengono impostati con la sola componente verde. Restituisce il numero di pixel sopra soglia.

`void scan(unsigned char *p,int x, int y, int id)`
Funzione ricorsiva che si occupa di distinguere i blocchi di pixel contenuti nell'immagine, attribuendo ad ogni blocco una diversa tonalità di rosso. Il parametro *p* punta all'immagine, *x* e *y* sono le coordinate del pixel da analizzare; il parametro *id* serve solo per distinguere i vari blocchi.

`int segmenta(unsigned char *p)`

[3] Libreria realizzata precedentemente dagli studenti Stefano Inelli, Alessandro Bonometti e Danilo Duina.

Scandisce l'immagine puntata da *p* mediante le funzione *scan* applicata ad ogni pixel. Restituisce il numero di blocchi trovati.

`void parametri(unsigned char *p,int totb)`
Calcola, per ogni blocco, i parametri (baricentro, area, perimetro ed estremi) e li memorizza nella *lista* dichiarata globalmente.

`void clean(unsigned char *p, int bl)`
Colora di blu il blocco *bl* nell'immagine *p.* Questa funzione serve solo per avere un ritorno grafico dei blocchi eliminati.

`int eliminablocchi(unsigned char *p,int totb)`
Funzione, utilizzata nel primo progetto, che elimina dalla lista i blocchi che non rispettano i vincoli di area minima e massima e rapporto perimetro/area.

`long int nearest(int vx,int vy,int totb,float k, int *px, int *py)`
Funzione, utilizzata per il primo progetto, che calcola la distanza lungo x e y (assegnandola ai parametri *px* e *py*) del baricentro all'interno del cono di visione (la cui apertura è *k*) più vicino alla verticale (specificata da *vx* e *vy*). Restituisce l'area del blocco selezionato. Il parametro *totb* indica il numero totale di blocchi.

`void similar(int vx,int vy,int totb,int *px, int*py, long int *area, float *min)`
Questa funzione, utilizzata per il terzo progetto, seleziona tra tutti i blocchi che sembrano dei cerchi quello più simile sulla base del rapporto fra l'area effettiva e quella stimata; inoltre calcola la distanza lungo x e y (assegnandola ai parametri *px* e *py*) del baricentro del blocco scelto. Al parametro *area* viene assegnato il valore dell'area mentre a *min* il coefficiente di somiglianza ad un cerchio calcolato come (area reale/area stimata)-1. Tale coefficiente vale 1 se nessun blocco è simile ad un cerchio.

`float cerchio(unsigned char *p,int blocco,int raggio)`
Restituisce la percentuale di pixel del *blocco* che si trovano all'interno di un cerchio centrato sul baricentro e caratterizzato dal parametro *raggio*.

`int riconosci(unsigned char *p,int totb)`
Analizza ognuno dei *totb* blocchi e elimina dalla lista quelli caratterizzati da parametri e caratteristiche che non sono proprie di un cerchio.

`void estremi(unsigned char *p, int *iniz, int *fin)`
Scandisce l'immagine *p* e assegna a *iniz* e *fin* le coordinate x degli estremi dell'immagine passata sopra soglia.

`float arcotan(int x, int y)`[4]
Questa funzione calcola l'angolo la cui tangente è il rapporto x/y. Si tratta di una arcotangente scalata di un fattore 2 in modo che le variazioni del rapporto x/y quando questo è circa 1 non comportino ampie variazioni dell'angolo.
`long int procuno(int *vaiax, int *vaiay)`

[4] Si è reso necessario la riscrittura della funzione atan propria del C in modo da poterla esportare nell' ambiente Saphira.

E' la funzione principale per il primo progetto. Viene esportata in ambiente Saphira e si occupa di chiamare nella sequenza opportuna le funzioni necessarie all'elaborazione ed all'estrazione delle informazioni utili. Assegna ai parametri *vaiax* e *vaiay* le coordinate rispetto alla verticale della lampada. Restituisce l'area della lampada. Si occupa di allocare e di liberare opportunamente la memoria per l'immagine acquisita e per la lista dei parametri.

`long int procdue(int *xi, int *xf)`
E' la funzione principale per il secondo progetto. Viene esportata in ambiente Saphira. Assegna ai parametri *xi* e *xf* gli estremi dell'immagine passata sopra soglia. Restituisce l'area della zona luminosa. Si occupa di allocare e di liberare opportunamente la memoria per l'immagine acquisita.
`int proctre(int *vaiax, int *vaiay, long int *area, float *rapp)`
E' la funzione principale per il terzo progetto. Viene esportata in ambiente Saphira. Assegna ai parametri *vaiax* e *vaiay* le coordinate del baricentro del cerchio. Assegna ai rispettivi parametri l'area del cerchio e il suo grado di somiglianza (*rapp*). Si occupa di allocare e di liberare opportunamente la memoria per l'immagine acquisita e i parametri.

Listato

```
//============================================================
// Progetto di Robotica
// (C) 2000 Bandera, Bonisoli Alquati, Donini
//
// Libreria di funzioni
//============================================================

#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include "tgmath.h"
#include "grab.h"
#include "grab.c"
#include "saphira.h"


//Variabili globali:

//Dimensione dell'immagine acquisita.
int width=320;
int height=240;
long int dim=230400;

//Parametri di acquisizione  ed elaborazione
int luminosita,contrasto,xver,yver,areamin,areamax,livello;
float apertura, ratiomin, ratiomax;

long int *lista;              //Lista dei blocchi individuati

#define C(x,y,k)    (*(copia+((y)*width+(x))*3+(k)))
#define P(x,y,k)    (*(p+((y)*width+(x))*3+(k)))
#define L(n,t)      (*(lista+(n)*9+(t)))


//Legge l'immagine in formato ppm da disco
int leggimmagine(unsigned char *mem,char filename[])
```

```
{
  FILE *ppm;
  int maxval;
  ppm=fopen(filename,"rb");
  if (ppm==NULL) {printf("Error: cannot open file %s\n",filename); return(1);}
  fscanf(ppm,"P6\n");
  fscanf(ppm," %d %d\n %d\n",&width,&height,&maxval);
  fseek(ppm,17,SEEK_SET);
  fread(mem,dim,1,ppm);
  fclose(ppm);
  return(0);
}

//Salva l'immagine in formato ppm su disco.
int salvaimmagine(unsigned char *mem, char filename[])
{
  FILE *ppm;
  int maxval=255;
  ppm=fopen(filename, "wb");
  if (ppm==NULL) {printf("\nError: cannot open file \n"); return 1;}
  fprintf(ppm,"P6\n");
  fprintf(ppm," %d %d\n %d\n",width,height,maxval);
  fwrite(mem,dim,1,ppm);
  fclose(ppm);
  printf("Salvata!\n");
  return(0);
}

//Acquisisce l'immagine tramite il frame grabber
int grab(unsigned char *cop,int lum, int contr)
{
  int device = 0;
  int channel = 1;
  int format = GRAB_MODE_NTSC;
  int bpp = 24;
  unsigned char *p;
  int i,j;
  // Apertura del frame grabber
  grab_open(width,height,device,channel,format,bpp);
  //Setta i parametri di acquisizione della telecamera.
  grab_set_attr(lum, 128, contr, 128);
  p = grab_capture();                        //Acquisizione
  if( p == NULL )
    {
      printf("\nInvalid image buffer\n");
      grab_close();
      return (1);
    }
  for(i=0;i<height;i++)                      //memorizzazione dell'immagine
     for(j=0;j<width;j++)
       {
       *(cop + (i * width + j)*3+2)=*(p + (i * width + j)*3);
       *(cop + (i * width + j)*3+1)=*(p + (i * width + j)*3+1);
       *(cop + (i * width + j)*3  )=*(p + (i * width + j)*3+2  );
       }
   grab_close();                            // Chiusura del frame grabber
   return(0);
}



//Sfuoca l'immagine
int sfuoca(unsigned char *p)
```

```
{
  int X=0,Y=0;
  float k=.111111;
  unsigned char *copia;
  copia=(unsigned char*) malloc(dim);
  if(copia==NULL)
  {
     printf("Memoria insufficiente\n");
     return(1);
  }
  for(X=0;X<dim;X++)        *(copia+X)=*(p+X);
  for(Y=1;Y<height-1;Y++)
  for(X=1;X<width-1 ;X++)
  {
    //R
    P(X,Y,0)=k*(C(X,Y-1,0)+C(X,Y,0)+C(X,Y+1,0)+
                        C(X-1,Y-1,0)+C(X-1,Y,0)+C(X-1,Y+1,0)+
                        C(X+1,Y-1,0)+C(X+1,Y,0)+C(X+1,Y+1,0));
    //G
    P(X,Y,1)=k*(C(X,Y-1,1)+C(X,Y,1)+C(X,Y+1,1)+
                        C(X-1,Y-1,1)+C(X-1,Y,1)+C(X-1,Y+1,1)+
                        C(X+1,Y-1,1)+C(X+1,Y,1)+C(X+1,Y+1,1));
    //B
    P(X,Y,2)=k*(C(X,Y-1,2)+C(X,Y,2)+C(X,Y+1,2)+
                        C(X-1,Y-1,2)+C(X-1,Y,2)+C(X-1,Y+1,2)+
                        C(X+1,Y-1,2)+C(X+1,Y,2)+C(X+1,Y+1,2));
  }
  free(copia);
  return(0);
}

//Elimina i pixel con luminosità inferiore ad S.
//Ritorna il numero di pixel sopra soglia
long int soglia(unsigned char *p,int S)
{
  int i,j;
  float Y;
  long int pix;
  pix=0;
  for(j=0;j<height;j++)
  for(i=0;i<width;i++)
  {
   Y=0.299*P(i,j,0)+0.587*P(i,j,1)+0.114*P(i,j,2);
   P(i,j,0)=P(i,j,1)=P(i,j,2)=0;
   if(Y>S)
    {
    P(i,j,1)=255;       //colora di verde le zone con luminosità superiore a S
    pix++;
   }
  }
  return(pix);
}

//Dato il pixel (x,y) marchia i pixel adiacenti, assegnando una diversa tonalità
di rosso al blocco a cui appartengono
void scan(unsigned char *p,int x, int y, int id)
{
  P(x,y,0)=id;    //Assegna alla componente R il numero corrispondente al blocco
  if( (P(x+1,y,0)==0)   && (P(x+1,y,1)==255)   && (x+1<width) )
scan(p,x+1,y,id);
  if( (P(x+1,y+1,0)==0) && (P(x+1,y+1,1)==255) && (x+1<width) && (y+1<height) )
scan(p,x+1,y+1,id);
```

```
  if( (P(x,y+1,0)==0)   && (P(x,y+1,1)==255)   && (y+1<height) )
scan(p,x,y+1,id);
  if( (P(x-1,y+1,0)==0) && (P(x-1,y+1,1)==255) && (x-1>0)     && (y+1<height) )
scan(p,x-1,y+1,id);
  if( (P(x-1,y,0)==0)   && (P(x-1,y,1)==255)   && (x-1>0) )
scan(p,x-1,y,id);
  if( (P(x-1,y-1,0)==0) && (P(x-1,y-1,1)==255) && (x-1>0)     && (y-1>0) )
scan(p,x-1,y-1,id);
  if( (P(x,y-1,0)==0)   && (P(x,y-1,1)==255)   && (y-1>0) )
scan(p,x,y-1,id);
  if( (P(x+1,y-1,0)==0) && (P(x+1,y-1,1)==255) && (x+1<width) && (y-1>0) )
scan(p,x+1,y-1,id);
}

//Scandisce l'immagine e, tramite scan, isola i blocchi nell'immagine
//Restituisce il numero di blocchi trovati
int segmenta(unsigned char *p)
{
 int q,w,i,trovati=0;
 for(q=0;q<height;q++)         //righe
 for(w=0;w<width;w++)        //colonne
 {
   if ((P(w,q,1)==255)&&(P(w,q,0)==0))
   {
    trovati++;
    scan(p,w,q,trovati);
   }
 }
 return(trovati);
}


//Calcola i parametri dei vari blocchi:
//Id - Xb - Yb - Area - Perimetro - Xmin - Xmax - Ymin - Ymax
void parametri(unsigned char *p,int totb)
{
 int q,w,bx,by;
 long int x,y;
 int selez=0;
 for (w=0;w<totb;w++)
 {
   L(w,0)=w+1;          //id del blocco
   L(w,5)=width;
   L(w,7)=height;
 }
 for (y=0;y<height;y++)
 for (x=0;x<width;x++)
 {
      //perchè la lista parte da 0 mentre la numerazione del blocco parte da 1
      selez=P(x,y,0)-1;
      if (((P(x+1,y,0)==0) ||
           (P(x+1,y+1,0)==0) ||
           (P(x,y+1,0)==0) ||
           (P(x-1,y+1,0)==0) ||
           (P(x-1,y,0)==0) ||
           (P(x-1,y-1,0)==0) ||
           (P(x,y-1,0)==0) ||
           (P(x+1,y-1,0)==0) ) && (P(x,y,1)!=0))  L(selez,4)++;  //perimetro



       if (P(x,y,1)!=0)
       {
```

```
          L(selez,3)++;          //area
          if(x<L(selez,5)) L(selez,5)=x; //xmin
          if(y<L(selez,7)) L(selez,7)=y; //ymin
          if(x>L(selez,6)) L(selez,6)=x; //xmax
          if(y>L(selez,8)) L(selez,8)=y; //ymax
          L(selez,1)+=x;    //integra la x
          L(selez,2)+=y;    //integra la y
       }
  }
  for(q=0;q<totb;q++)
  {
    L(q,1)=L(q,1)/L(q,3);  //Calcola i baricentri   somma(x)/area
    L(q,2)=L(q,2)/L(q,3);
    //Colora i baricentri:
    P(L(q,1),L(q,2),1)=0;     //verde=0
    P(L(q,1),L(q,2),2)=255;  //blu=255 per distinguerli
  }
}

//Funzione, non è indispensabile, che semplicemente elimina le informazioni
//sul colore solo nell'immagine (utile per vedere quello che accade ai blocchi)
//Il blocco bl da eliminare viene colorato di blu.
void clean(unsigned char *p, int bl)
{
 int y,x;
 for (y=0;y<height;y++)
 for (x=0;x<width;x++)
 {//bl parte da 1...nblocchi
  if(P(x,y,0)==bl) {P(x,y,0)=0;P(x,y,1)=0;P(x,y,2)=255;}
 }
}

//Elimina i blocchi in base all'area ed al rapporto area perimetro.
int eliminablocchi(unsigned char *p,int totb)
{
   int i=0,j=0;
   float ratio;
   for(i=0;i<totb;i++)
   {
      ratio=(float)(L(i,4))/(float)(L(i,3));
      if (((L(i,3)<areamin)||(L(i,3)>areamax))||
          ((ratio>ratiomax)||(ratio<ratiomin)))
      {
        clean(p,i+1);                          //Elimina dall'immagine;
        for(j=0;j<9;j++) L(i,j)=0;
        //Cancella la linea nel vettore blocco, elimina anche l'id
      }
   }
}

//Assegna ad x e y le coordinate del baricentro più vicino alla verticale
//Restituisce l'area del blocco individuato
long int nearest(int vx,int vy,int totb,float k, int *px, int *py)
{
  //X>0 Y>0 avanti a destra
  //X<0 Y>0 avanti a sinistra
  //X>0 Y<0 dietro a destra
  //X<0 Y<0 dietro a sinistra
 long int dist,min;
 long int area;
 int i,x,y;
 int prima;
 *px=0;
```

```
 *py=0;
 area=0;
 prima=0;
 for(i=0;i<totb;i++)
 {
  if (L(i,0)!=0)          //controlla se l'id !=0
  {
   x=(L(i,1)-vx);
   y=(L(i,2)-vy);
   dist=x*x+y*y;
   // apertura del "paraocchi" virtuale: k=1 sta per 45 gradi, <1 cono + largo
   if ((y+k*(float)x>0)&&(y-k*(float)x>0))
   {
     if ((dist<min) ||(prima==0))
         {
          min=dist;
          *px=x;
          *py=y;
          area=L(i,3);
          prima=prima+1;
         }
     }
   }
 }
 return(area);
}

//Assegna ad x e y le coordinate del baricentro del blocco più simile ad un
cerchio
//Assegna ad area e min rispettivamente l'area del blocco ed il relativo
rapporto area/area stimata
void similar(int vx,int vy,int totb,int *px, int*py, long int *area, float *min)
{
 int i,x,y;
 float raggio,dist;
 int prima;
 *px=0;
 *py=0;
 *area=0;
 *min=1;
 prima=0;
 for(i=0;i<totb;i++)
 {
 //controlla se l'id !=0  e la fascia centrale dell'immagine
  if ((L(i,0)!=0)&&(L(i,2)>50)&&(L(i,2)<180))
    {
   x=(L(i,1)-vx);
   y=(L(i,2)-vy);
   raggio=(L(i,6)-L(i,5)+L(i,8)-L(i,7))/4;
   dist=(L(i,3)/(3.14*raggio*raggio))-1;
   if (dist<0) dist=-dist;
   if ((dist<*min)||(prima==0))
         {
          *min=dist;
          *px=x;
          *py=y;
          *area=L(i,3);
          prima++;
         }
     }
  }}

//restituisce il num di pixel all'interno di un cerchio centrato sul baricentro
```

```
float cerchio(unsigned char *p,int blocco,int raggio)
{
 //float rapp;
 int i,j;
 int pixel=0;
 float dist;

 for(i=-raggio;i<raggio;i++)
 for(j=-raggio;j<raggio;j++)
 {
   if ((i*i+j*j)<=raggio*raggio)
   {
     //Blocco parte da uno
     if (P(L(blocco,1)+i,L(blocco,2)+j,0)==blocco+1) pixel++;
   }
 }
 return((float)pixel/L(blocco,3));
}



int riconosci(unsigned char *p,int totb)
{
 int x,y,i,j;
 int delta,deltax,deltay;
 float raggio,distanza,pr,ar;
 int pixel;
 for(i=0;i<totb;i++)
 {
   pixel=0;
   deltax=L(i,6)-L(i,5);
   deltay=L(i,8)-L(i,7);
   delta=deltax-deltay;
   if (delta<0) delta=-delta;
   //entra se la differenza tra le dimensioni orizzontale e verticale non sono
troppo diverse
   if ((delta>-4)&&(delta<+4))
   {
    raggio=(float)(deltax+deltay)/4;
     //Calcolo dei punti di perimetro che distano dal baricentro raggio +/- 3
pixel
     for (y=L(i,7);y<L(i,8);y++)
     for (x=L(i,5);x<L(i,6);x++)
     {
      if (((P(x+1,y,0)==0) ||
           (P(x+1,y+1,0)==0) ||
           (P(x,y+1,0)==0) ||
           (P(x-1,y+1,0)==0) ||
           (P(x-1,y,0)==0) ||
           (P(x-1,y-1,0)==0) ||
           (P(x,y-1,0)==0) ||
           (P(x+1,y-1,0)==0)) && (P(x,y,1)!=0))
           {
             distanza=pow(x-L(i,1),2)+pow(y-L(i,2),2);
             if ((distanza<pow(raggio+3,2))&&(distanza>pow(raggio-3,2)))
pixel++;
           }
       }//Fine ciclo perimetro

       pr=(float)pixel/L(i,4);  //percentuale di pixel di perimetro che approx
un cerchio
       ar=cerchio(p,i,(int)raggio);  // % di pixel del blocco all'interno di un
cerchio centrato sul baricentro
```

```
       printf("\nRapporti del blocco %d --> Perimetro:%f    Area:%f",i+1,pr,ar);

        //Condizioni per cancellare i blocchi
      if ( (raggio<=3) || (L(i,3)<2*pow(raggio,2)) || (L(i,3)>5*pow(raggio,2))
|| (pr<0.61) || (ar<0.61) || (L(i,3)<areamin) )
             {
               clean(p,i+1);
               for(j=0;j<9;j++) L(i,j)=0;
             }
     } //IF delta<-5
   else
   {
     clean(p,i+1);
     for(j=0;j<9;j++) L(i,j)=0;
   }
  }//Ciclo di scansione dei blocchi
}//Fine riconosci

//Calcola la x dei punti estremi dell'immagine
void estremi(unsigned char *p, int *iniz, int *fin)
{
  int y,x;
  *iniz=319;
  *fin=0;
  for (y=0;y<height-50;y++)            //non considero il pavimento...
  for (x=0;x<width;x++)
  {
     if((P(x,y,1)!=0)&&(x<*iniz))  *iniz=x;
     if((P(x,y,1)!=0)&&(x>*fin))  *fin=x;
  }
}

//Calcola l'arcotangente, scalata di un fattore 2
float arcotan(int x, int y)
{
 float a;
 if(y!=0) a=atan((float)x/(float)y/2)*(180/3.14);
 else a=0;
 return(a);
}

//Funzione per il primo progetto:
//Assegna ad vaiax e vaiay le coordinate della luce sul soffitto più vicina
//Restituisce l'area della luce
long int procuno(int *vaiax, int *vaiay)
{
 int blocchitot,q,w;
 long int rip;                              //area in uscita della luce
 //Alloca la memoria per l'immagine
 unsigned char *originale;
 originale = (unsigned char*) malloc(dim);
  if (originale==NULL)
  {
     printf("Memoria insufficiente.\n");
     return(1);
  }

 grab(originale,luminosita,contrasto);
         //leggimmagine(originale,"prova.ppm");         //PC
       salvaimmagine(originale,"tmp.ppm");
 sfuoca(originale);                                     //sfuoca l'immagine
       //salvaimmagine(originale,"tmpsfucata.ppm");
 soglia(originale,livello);               //applica la soglia all'immagine
```

```
       //salvaimmagine(originale,"tmpsogliata.ppm");
 blocchitot=segmenta(originale);               //ora segmento l'immagine
if(blocchitot>0)
 {
  //Alloca la  memoria per i parametri
  lista = (long int *) calloc(blocchitot*9,sizeof(long int));
  if(lista==NULL)
  {
    printf("Memoria insufficiente.\n");
    return(1);
  }
  parametri(originale,blocchitot);
  eliminablocchi(originale,blocchitot); //1 beav
  salvaimmagine(originale,"tmpeliminati.ppm");
  for(q=0;q<blocchitot;q++) { for(w=0;w<9;w++) printf("%d ",L(q,w));
printf("\n"); }

   rip=nearest(xver,yver,blocchitot,apertura,vaiax,vaiay);
   free(lista);
 }
 else
 {
  *vaiax=0;
  *vaiay=0;
   rip=0;
 }
 free(originale);
 return(rip);
}

//Funzione per il secondo progetto:
//Assegna ad xi e xf la x degli estremi dell'immagine
//Restituisce l'area della luce
long int procdue(int *xi, int *xf)
{
 int blocchitot,q,w;
 long int rip;
 unsigned char *originale;
 originale = (unsigned char*) malloc(dim);
   if (originale==NULL)
  {
     printf("Memoria insufficiente.\n");
     return(1);
  }
 grab(originale,luminosita,contrasto);
       salvaimmagine(originale,"tmp.ppm");
 sfuoca(originale);
 rip=soglia(originale,livello);
      salvaimmagine(originale,"tmpeliminati.ppm");
 estremi(originale,xi,xf);
 free(originale);
 return(rip);
}


//Funzione per il terzo progetto:
//Assegna ad vaiax e vaiay le coordinate del baricentro del blocco più simile ad
un cerchio
//Assegna ad area e rapp rispettivamente l'area di tale blocco ed il rapporto
area/area stimata
int proctre(int *vaiax, int *vaiay, long int *area, float *rapp)
{
 int blocchitot,q,w;
```

```
 unsigned char *originale;
 originale = (unsigned char*) malloc(dim);
   if (originale==NULL)
  {
     printf("Memoria insufficiente.\n");
     return(1);
  }
 grab(originale,luminosita,contrasto);
 //leggimmagine(originale,"prova.ppm");
       salvaimmagine(originale,"tmp.ppm");
 sfuoca(originale);
       //salvaimmagine(originale,"tmpsfucata.ppm");
 soglia(originale,livello);
       //salvaimmagine(originale,"tmpsogliata.ppm");
 blocchitot=segmenta(originale);
 if(blocchitot>0)
 {
  lista = (long int *) calloc(blocchitot*9,sizeof(long int));
  if(lista==NULL)
  {
    printf("Memoria insufficiente\n");
    return(1);
  }
  parametri(originale,blocchitot);
  riconosci(originale,blocchitot);
       salvaimmagine(originale,"tmpeliminati.ppm");
  similar(xver,yver,blocchitot,vaiax,vaiay,area,rapp);
       for(q=0;q<blocchitot;q++) { for(w=0;w<9;w++) printf("%d ",L(q,w));
printf("\n"); }
  free(lista);
 }
 else
 {
   *rapp=1;
   *area=0;
 }
 free(originale);
 return(0);
}

//Esporta variabili e funzioni
EXPORT void sfLoadInit (void)
{
  sfSMessage("Let's grab the  world.......");
  sfAddEvalFn("arcotan",arcotan,sfFLOAT,2,sfINT,sfINT);
  sfAddEvalFn("procuno",procuno,sfINT,2, sfPTR, sfPTR);
  sfAddEvalFn("procdue",procdue,sfINT,2, sfPTR, sfPTR);
  sfAddEvalFn("proctre",proctre,sfINT,4,sfPTR, sfPTR, sfPTR,sfPTR);
  sfAddEvalVar("luminosita", sfINT,(fvalue *)&luminosita);
  sfAddEvalVar("contrasto", sfINT,(fvalue *)&contrasto);
  sfAddEvalVar("xver", sfINT,(fvalue *)&xver);
  sfAddEvalVar("yver", sfINT,(fvalue *)&yver);
  sfAddEvalVar("areamin", sfINT,(fvalue *)&areamin);
  sfAddEvalVar("areamax", sfINT,(fvalue *)&areamax);
  sfAddEvalVar("ratiomin", sfFLOAT,(fvalue *)&ratiomin);
  sfAddEvalVar("ratiomax", sfFLOAT,(fvalue *)&ratiomax);
  sfAddEvalVar("livello", sfINT,(fvalue *)&livello);
  sfAddEvalVar("apertura",sfFLOAT,(fvalue *)&apertura);
}
```

Makefile.

```
# Progetto di Robotica
#
#
# (C) 2000 Bandera, Bonisoli Alquati, Donini
#
# Makefile per compilare proglib.c come file shared object
# caricabile dal saphira

SHELL = /bin/sh

############################################################

SRCD = ./
OBJD = ./
INCD = $(SAPHIRA)/handler/include/
LIBD = $(SAPHIRA)/handler/obj/
BIND = ./ver62/bin/
COLBERT = $(SAPHIRA)/colbert/
OAALIB = $(SAPHIRA)/oaa/agents/lib/

# find out which OS we have
include $(SAPHIRA)/handler/include/os.h

CFLAGS =  -g -D$(CONFIG) $(PICFLAG)
CC = gcc
INCLUDE = -I$(INCD) -I$(X11D)include


############################################################

all: proglib.so
      touch all


 clean:
      rm *.o *~

# Questo crea il file proglib.so contenente tutte le funzioni in C per la
gestione
# dell'acquisizione e elaborazione delle immagini.
# Per caricarlo nel saphira digitare:    load proglib.so

$(OBJD)proglib.o: $(SRCD)proglib.c $(INCD)saphira.h
      $(CC) $(CFLAGS) -c $(SRCD)proglib.c $(INCLUDE) -o $(OBJD)proglib.o

proglib.so: $(OBJD)proglib.o
      $(LD) $(SHARED) $(OBJD)proglib.o -o proglib.so
```

Note sull'utilizzo dei programmi.

Per poter utilizzare i programmi occorre copiare nella stessa directory tutto il contenuto della cartella `/codice` e compilare (tramite il comando make) la libreria *proglib.c* utilizzando il makefile in essa contenuto rispettando i percorsi specificati nelle righe iniziali (oppure modificarli opportunamente).
Aprire il Saphira[5], e dopo aver connesso il robot caricare il programma desiderato tramite il comando:

load <percorso> / <progettox.act>

Affinché i vari programmi funzionino correttamente occorre regolare l'orientamento della telecamera di Tobor che deve essere di circa 70° rispetto al pavimento per la movimentazione in corridoio e parallela al pavimento per gli altri due programmi.
E' comunque possibile regolare con precisione la mappatura del punto di verticale del robot nell'immagine mediante i parametri globali Xver e Yver.
I parametri impostati nei tre programmi sono stati scelti per un corretto funzionamento durante i test nel dipartimento di elettronica.

Appendice – Utilizzo della Quickcam.

Durante lo sviluppo del progetto abbiamo trovato il modo di utlizzare la quickcam per acquisire le immagini del robot[6]; siccome il tempo di refresh delle immagini era troppo elevato (circa un'immagine ogni secondo) abbiamo ritenuto opportuno abbandonare l'idea.
Di seguito riportiamo il file Documentazione gqcam.txt che abbiamo incluso nella cartella /codice/quickcam e che spiega come installare e utilizzare la quickcam disponibile in laboratorio.

```
#29/11/2000
#Bandera, Bonisoli Alquati, Donini, Bonometti


Per installare i driver della Quickcam occorre:
================================================

1) Verificare la presenza di Video4Linux; dovrebbero essere presenti i seguenti
file:

/lib/modules/2.2.x./misc/videodev.o
/lib/modules/2.2.x./misc/bw-qcam.o
/lib/modules/2.2.x./misc/c-qcam.o

(x dipende dalla versione del kernel  installata)

2) E' necessario fare login come root ed avere accesso ai codici sorgente del
kernel.

3) Il modulo c-qcam.c necessita di una modifica per poter essere utilizzato
correttamente. In particolare occorre modificare
```

[5] I programmi sono stati sviluppati e testati per la versione 6.2
[6] La procedura sotto descritta è stata eseguita sul PC Herbie del LDRA con Linux Red Hat 6.2 in previsione di un futuro utilizzo su Frost.

la funzione qc_detect, facendogli ritornare in ogni caso 1, come segue:

```
static int qc_detect(struct qcam_device *qcam)
{
      unsigned int stat, ostat, i, count = 0;

      parport_write_control(qcam->pport, 0xc);

      /* look for a heartbeat */
      ostat = stat = parport_read_status(qcam->pport);
      for (i=0; i<250; i++)
      {
            mdelay(1);
            stat = parport_read_status(qcam->pport);
            if (ostat != stat)
            {
                  if (++count >= 3) return 1;
                  ostat = stat;
            }
      }

      /* no (or flatline) camera, give up */
      return 1;
}
```

4) Bisogna ricompilare i moduli, digitando "make modules".

5) Ora è possibile caricare nel sistema i moduli per la quickcam a colori usando modprobe:

```
   modprobe -a videodev
   modprobe -a c-qcam
```

6) Per controllare se sono stati correttamente caricati digitare lsmod e dovrebbe apparire:

```
   c-qcam                 6964   0  (unused)
   videodev               2368   1  [c-qcam]
```

7) Per utilizzare gqcam con l'interfaccia grafica occorre:

   a) Installarlo scompattando il file gqcam-0.8.tar
   b) Digitare il comando "make"
   c) Lanciare il programma gqcam.

8) Per compilare ed eseguire il file di esempio 'prova.c' occorre utilizzare il Makefile presente nella directory corrente.