UNIVERSITA' DEGLI STUDI DI ROMA
"TOR VERGATA"

FACOLTA' DI INGEGNERIA

Tesi di Laurea in Ingegneria Informatica

"Sviluppo di sistemi integrati di visione per il controllo di robot mobili tramite retroazione basata su telecamere"

Relatore
Ing. F. Martinelli

Laureando
Fabrizio Romanelli

Anno Accademico 2002/2003

Ai miei genitori

Indice

CAPITOLO 1

Introduzione

1 Introduzione

1.1 Generalità

Lo scopo del presente lavoro è stato quello di realizzare un sistema integrato di navigazione costituito da una piattaforma mobile (il Robot *Nomad 150*) e da una Webcam (Philips *ToUcam Pro 740*) programmabili in *C++* in ambiente *Linux*.

Come applicazione dimostrativa del sistema integrato Robot-Webcam è stato sviluppato del codice tramite il quale il Robot è in grado di inseguire un *Landmark* in movimento in un ambiente non noto ingombro di ostacoli. Il *Landmark* può spostarsi a una velocità paragonabile a quella massima di movimento del Robot e la sua posizione viene rilevata mediante la Webcam. Il Robot utilizza invece i sensori sonar per rilevare gli ostacoli presenti nel percorso.

Il Robot *Nomad 150* non è dotato di un proprio sistema ottico di scansione dello spazio, ma il suo sistema di navigazione è regolato soltanto da 16 sensori ad ultrasuoni per l'individuazione degli ostacoli, e da 20 *switch* chiamati *bumper*, utilizzati come sensori di pressione per individuare la presenza di eventuali ostacoli a contatto con il Robot. Sono inoltre presenti sensori odometrici (*encoder* posizionati sulle tre ruote del Robot), che permettono di conoscere la posizione del Robot in ogni istante.

La scelta di adottare un sistema di rilevamento ottico per guidare la navigazione costituito da un dispositivo prettamente commerciale, quale la Webcam Philips *ToUcam Pro 740*, ne limita tuttavia le capacità e le potenzialità di acquisizione ed elaborazione per il riconoscimento delle immagini soprattutto in termini di *real-time*; la bassa risoluzione fornita dalla Webcam ed utilizzata nell'elaborazione ha anch'essa contribuito a limitare le potenzialità dell'elaborazione così come il limitato campo visivo della *ToUcam Pro 740*. Il sistema permette tuttavia un'applicazione che consente di ottenere risultati apprezzabili sotto l'aspetto dimostrativo.

Il software dimostrativo consiste di due programmi separati, uno per la navigazione del Robot, e l'altro per il riconoscimento del *Landmark* tramite Webcam. Il programma per il riconoscimento del *Landmark* effettua operazioni di elaborazione dell'immagine e passa i dati relativi alla posizione del *Landmark* individuato al programma di navigazione, che si occupa di far raggiungere al Robot il *Landmark* senza urtare contro gli ostacoli rilevati dai sensori sonar.

L'accoppiamento tra i dati letti dai sonar ed elaborati dal programma di gestione della Webcam e quelli utilizzati nell'elaborazione del software per la navigazione e per l'*obstacle avoidance*, ha messo in evidenza le limitazioni sostanziali imposte dall'hardware utilizzato[1].

[1]La principale limitazione è dovuta al fatto che la torretta, e quindi i sonar su essa,

Il software di gestione del sistema è stato elaborato in linguaggio C++ e compilato con *gcc* su sistema operativo *Linux* e con l'impiego di un PC portatile *ASUS* dotato di processore *Pentium 4* a 2 GHz e 256 MB di RAM. Nella fase esecutiva, l'elaborazione sequenziale del programma di riconoscimento del *Landmark* e quello della navigazione del Robot eseguita dal PC ha in parte contribuito ad incrementare i tempi di risposta del sistema; si è cercato tuttavia di contenere tali tempi a scapito della precisione nel riconoscimento del *Landmark*.

Si ritiene che il sistema *Robot-PC-Webcam* possa essere migliorato utilizzando una telecamera più adatta alle applicazioni di riconoscimento, allo scopo di eliminare alcune fasi di preelaborazione contenute nel software e ridurre i tempi di elaborazione delle immagini; ancora, ulteriore e fondamentale si ritiene sia l'utilizzo di una torretta sulla quale poter posizionare la Webcam in modo da rendere indipendenti la rotazione della torretta del Robot, e quindi dei sonar, e la rotazione della Webcam.

Alcuni problemi incontrati nel riconoscimento del *Landmark* in particolari condizioni di luminosità dell'ambiente in cui è stata eseguita la prova, potevano essere evitati via software con un'opportuna scelta dei parametri di luminosità e contrasto. Soluzioni di questo tipo sono tutta-

sono solidali con la Webcam; ne consegue quindi che una rotazione della stessa, operata dal programma che gestisce il riconoscimento del *Landmark*, provoca una rotazione dei sonar, indesiderata per la navigazione verso il *Landmark*, perché cambia il riferimento per il riconoscimento degli ostacoli.

via state scartate a priori per l'applicazione eseguita, per non incrementare ulteriormente i tempi di elaborazione e interpretazione delle immagini.

La *Computer Vision* è un campo in continuo sviluppo [1], e, se associata a sistemi robotici [3] [7] [8] [9] può fornire prestazioni e qualità adattabili ad una vasta gamma di applicazioni. A titolo di esempio si vuole segnalare l'applicazione studiata e realizzata da Coleman e altri [8], nella quale è stato realizzato un sistema di navigazione tramite telecamera e sistema *GPS*. In tale sistema un Robot[2] viene fatto alzare in volo e, utilizzando la telecamera (con un sistema integrato di *Computer Vision*) e il *GPS*, raggiunge una piattaforma di atterraggio, evitando gli ostacoli che trova nel suo percorso.

[2]Nel caso specifico un modellino di elicottero sul quale sono presenti oltre al *GPS*, anche un PC ed una telecamera.

CAPITOLO 2

Descrizione Hardware

2 Descrizione Hardware

2.1 Generalità

In questo capitolo viene descritto l'hardware utilizzato nel presente lavoro; esso è composto da:

- Robot *Nomad 150*
- Webcam Philips *ToUcam Pro 740*
- PC portatile *ASUS*

2.2 Robot *Nomad 150*

Il *Nomad 150* è un Robot dotato di un sistema meccanico di spostamento su di un piano costituito da tre ruote, che traslano tramite il controllo di un primo motore e possono contemporaneamente ruotare sotto l'azione di un secondo motore; un ulteriore motore permette la rotazione della torretta.

Il Robot può raggiungere una velocità traslazionale di 20 inch[3]/sec mentre la velocità di rotazione delle ruote e della torretta non può superare i 45 gradi/sec. Il sistema sensoriale del *Nomad 150* è composto da due sistemi di sensori: il sistema "tattile" ed il sistema "ad ultrasuoni". Il primo è costituito da 20 sensori di pressione indipendenti (*switch*), disposti su due corone, ciascuna con 10 sensori, in modo da coprire 360 gradi con una risoluzione

[3] 1 inch = 2.54 cm.

di 18 gradi.
Il sistema "ad ultrasuoni" è invece formato da 16 sonar che possono fornire la distanza con un *range* [17-255] inch. La risoluzione ottenibile con questo sistema sensoriale è dunque di 22.5 gradi.

Inoltre il *Nomad 150* è dotato di encoder sulle tre ruote che permettono di conoscere la posizione relativa del Robot, definita mediante le coordinate (x,y), relative al sistema di coordinate avente centro nella posizione in cui si trova inizialmente il Robot.

Figura 1: Robot *Nomad 150*

2.3 Webcam Philips *ToUcam Pro 740*

La Webcam *ToUcam Pro 740* è dotata di un sensore CCD (*Charged Coupled Device*) che permette di raggiungere risoluzioni di 640x480 pixel (VGA) con una profondità di colore di 24 bit. La lente integrata è una 6mm f2.0 ed il massimo *framerate* raggiungibile è 30 fps, ad una risoluzione di 320x240.

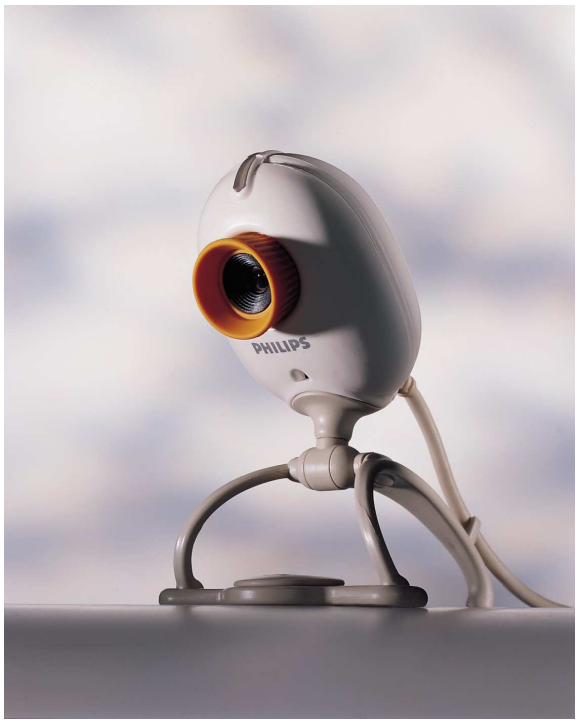

Figura 2: Webcam Philips *ToUcam Pro 740*

La *ToUcam Pro 740* ha la capacità di adattarsi a molti tipi di luminosità, aggiustando automaticamente i valori di luminosità e contrasto a seconda delle necessità. Nonostante la sua versatilità sotto questo punto di vista,

la Webcam risponde al cambiamento di luminosità con un ritardo che in alcuni, seppur rari, casi non risulta accettabile per l'applicazione nella quale è stata utilizzata, avendo determinato talvolta problemi di definizione dell'immagine ripresa in particolari condizioni di luminosità dell'ambiente.

2.4 PC portatile *ASUS*

Per l'elaborazione del sistema e l'applicazione, è stato impiegato un PC portatile *ASUS*. La velocità del processore *Pentium 4* 2 GHz, e i 256 MB di RAM di cui è dotato il portatile della *ASUS* hanno contribuito in maniera determinante al conseguimento di risultati soddisfacenti in termini di tempi di elaborazione, essendo fondamentale nella presente applicazione la parte relativa alla elaborazione delle immagini. Inoltre il computer, di ridotte dimensioni ed alimentato da una batteria propria, è stato sistemato sullo stesso Robot, con conseguenti vantaggi dal punto di vista della compattezza del sistema hardware.

2.5 *Landmark*

Come *Landmark* si è scelta una sfera di colore nero e la si è posta su di un supporto di colore bianco, in modo da garantire un livello di contrasto sufficiente per il corretto riconoscimento dell'oggetto.

La scelta della sfera come *Landmark* è stata dettata dal fatto che, scegliendo un oggetto differente, come ad esempio un quadrato o un poligono, sarebbe sorto il problema della variazione di prospettiva, e quindi della forma dell'oggetto stesso, nel caso in cui la Webcam non fosse centrata esattamente con il *Landmark*:

la sfera ha infatti la caratteristica che da qualunque direzione la si guardi, non cambia la sua forma, ed in particolare in una immagine a 2 dimensioni la sfera mantiene sempre la forma di un cerchio.

Figura 3: *Landmark* utilizzato per il riconoscimento

Inoltre può essere calcolata in modo relativamente semplice la relazione che intercorre tra il raggio della circonferenza individuata nell'immagine e la distanza alla quale si trova il *Landmark*; conoscendo quindi il raggio è pos-

sibile conoscere la distanza alla quale si trova l'oggetto dal Robot. Tutto ció allo scopo di non appesantire eccessivamente la mole di elaborazione delle immagini e dei dati che avrebbe contribuito a tempi di esecuzione non compatibili con l'applicazione scelta.

2.6 Posizionamento dell'hardware sul Robot

Il PC portatile è stato posizionato, come si può notare dalla figura 4, sulla parte superiore del Robot collegando il cavo dalla porta seriale RS232C del PC alla porta *HOST* del Robot.

Figura 4: Sistema *Robot-PC-Webcam*

Alla porta USB del PC è stata collegata la Webcam e quest'ultima è stata posta sulla parte anteriore della torretta del Robot in corrispondenza del Sonar 0[4].

In questo modo il sistema *Robot-PC-Webcam* risulta essere un sistema indipendente.

[4]Da notare che la Webcam e la torretta del Robot ruotano contemporaneamente essendo solidali.

CAPITOLO 3

Software di gestione

3 Software di gestione

3.1 Generalità

Lo scopo del software è quello di permettere al *Nomad 150* di navigare verso un obiettivo specificato (*Landmark*) che cambia nel tempo la sua posizione rispetto al Robot. Il software sviluppato segue le seguenti fasi della *computer vision*:

- percezione
- preelaborazione
- segmentazione
- descrizione
- riconoscimento
- interpretazione

Nella fase di *percezione* viene fornita una immagine dell'ambiente di lavoro. Nella fase di *preelaborazione* avviene il filtraggio dell'immagine, necessario per ridurre il rumore e per migliorare i particolari dell'immagine. Nella *segmentazione* l'immagine viene divisa in oggetti di interesse. Nella fase di *descrizione* avviene il calcolo delle caratteristiche utilizzabili per differenziare un oggetto da un altro. Nella fase di *riconoscimento* viene identificato l'oggetto (in questo caso il *Landmark*). Infine nell'*interpretazione* viene interpretata l'informazione per conferire un significato all'oggetto riconosciuto.

Per il software descritto in questo capitolo si sono utilizzate le librerie preesistenti in *Linux*[5] per i dispositivi di acquisizione immagini (schede TV o altro) adattandole alla Webcam, e le librerie fornite con il *Nomad 150* per la programmazione del Robot, tutte librerie in linguaggio *C/C++*, e adatte esclusivamente a sistemi operativi *Linux* (nel caso in esame la versione di *Linux* utilizzata è la distribuzione *Mandrake* versione *9.0*).

L'applicazione dimostrativa funziona mandando in esecuzione contemporaneamente due programmi:

1. Programma per la gestione della Webcam e per la movimentazione della torretta del Robot;
2. Programma per la gestione della navigazione del Robot.

In questo capitolo viene descritto il software necessario alla elaborazione delle immagini (che ne rappresenta la parte più imponente), e quello necessario alla navigazione del Robot.

3.2 Acquisizione dell'immagine

Il processo di acquisizione dell'immagine avviene nel modo seguente:

la Webcam, inizializzata in modo da acquisire immagini di dimensione 320x240 pixel a 24 bit di profondità

[5]Le librerie utilizzate sono le *Video 4 Linux* (*V4L*).

e a 30 fps, cattura l'immagine corrente che si presenta nell'ambiente di lavoro e la inserisce in un'area dedicata di memoria nella quale risiede in formato *YUV 420P*[6].

La prima fase ha quindi lo scopo di catturare l'immagine corrente per mezzo della Webcam e di trasferirla in memoria. Si riportano di seguito le linee di codice che, essendo stata precedentemente inizializzata la Webcam, permettono l'acquisizione dell'immagine:

ioctl(webcam, VIDIOCMCAPTURE, grab_buf);
ioctl(webcam, VIDIOCSYNC, grab_buf);

Nella prima linea di codice si effettua l'acquisizione dell'immagine corrente (parametro *VIDIOCMCAPTURE* passato alla funzione *ioctl*) da parte del dispositivo richiamato come primo parametro da *ioctl* (*webcam*). L'immagine viene trasferita in memoria all'interno di un array di elementi di tipo *unsigned char*. La seconda linea permette la sincronizzazione tra la Webcam e la memoria del PC per il processo di acquisizione dati.

L'immagine residente in memoria è rappresentata con la *Palette YUV420P*: questo è il formato con cui in origine vengono acquisite le immagini dalla Webcam[7] che, in genere, è quello di molti dispositivi dedicati per la trasmissione video o di dispositivi in cui è necessaria una certa compressione delle immagini (a scapito della qualità).

[6]Il formato *YUV 420P.* è quello di base con il quale la Webcam acquisisce le immagini.

[7]Il driver Philips per la Webcam permette **esclusivamente** l'acquisizione delle immagini nel formato *YUV420P.*

Il formato *YUV420P* ha un canale per la luminosità (Y) e due canali per i colori (U e V), con i quali possono essere rappresentati tutti i colori possibili; la sigla *420P* in linguaggio tecnico si legge 4:2:0 planare, ed indica che nella codifica digitale dell'immagine sono scritti per primi tutti i valori Y, esattamente 320x240 byte (nel caso in esame), seguiti poi dai valori delle U, in numero pari ad 1/4 dei pixel Y, cioè 320x240/4, ed infine dalla stessa quantità (320x240/4) di valori di V[8]. Pertanto, per ogni 4 valori di Y ce n'è uno per la U ed uno per la V, ovvero ogni valore di U e V si applica ad un blocco di 2x2 valori di Y.

Figura 5: Immagine acquisita dalla Webcam

[8]Le dimensioni dell'immagine in formato *YUV420P* contenuta in memoria sono 320x240x1.5 byte, essendo 320x240 (canale Y) + 320x240/4 (canale U) + 320x240/4 (canale V).

La modalità appena descritta, con la quale la Webcam acquisisce le immagini e con la quale esse risiedono in memoria, non si presta bene al filtraggio e all'elaborazione che si desiderano effettuare per l'applicazione, e necessita quindi di un' ulteriore fase di elaborazione dei segnali acquisiti (si rimanda alla sezione 3.3).

3.3 Filtraggio dell'immagine

Una fase importante dell' elaborazione dell'immagine risiede nel filtraggio.

Questo infatti è un processo che permette di far passare alla fase successiva di elaborazione soltanto l'informazione necessaria ai fini del riconoscimento e dell'interpretazione dell'oggetto di interesse all'interno dell'immagine.

Il passo preliminare da compiere, prima di procedere con il filtraggio vero e proprio dell'immagine, è quello di trasformare il formato ($YUV420P$) delle immagini provenienti dalla Webcam in un formato privo di informazioni inutili ai fini della elaborazione e in un formato più "leggero" dal punto di vista del contenuto informativo e quindi di esecuzione più rapida; si è scelto il formato in toni di grigio, il quale sfrutta uno spazio in memoria di 320x240 byte (per un solo canale, la scala di grigio), invece dei 320x240 byte (per tre canali, quello della luminosità (Y), e quello dei colori (U, V) in totale 320x240x1.5 byte).

Questa parte di programma, piuttosto articolata, è rilevabile nel listato allegato (si rimanda alla sezione 5.1). L'immagine elaborata da questa parte del programma e riprodotta sul PC è riportata nella figura 6.

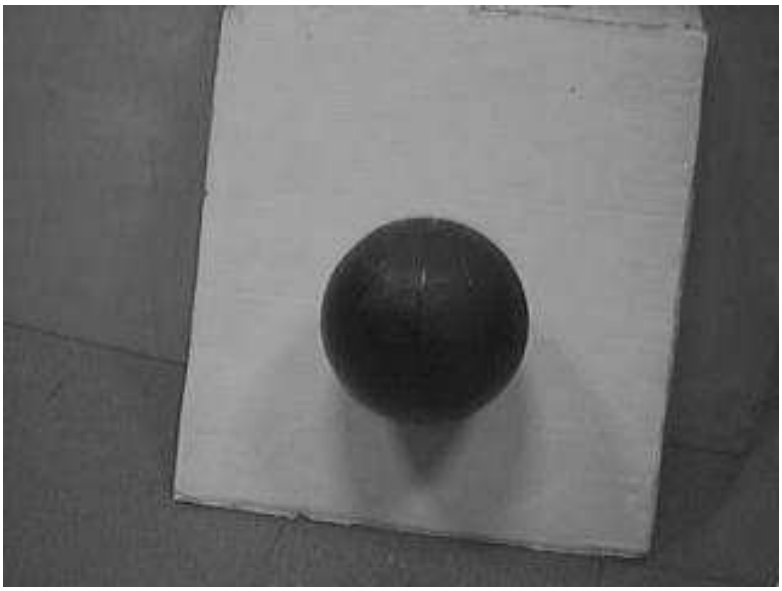

Figura 6: Immagine convertita in toni di grigio

Dopo aver operato la trasformazione di *Palette*, l'immagine residente in memoria è pronta per essere filtrata.

Il filtro utilizzato per primo, è il filtro di *Sobel* o del gradiente, il quale permette di mettere in risalto nell'immagine sorgente i contorni, ed in particolare il contrasto tra gli elementi contenuti nell'area dell'immagine.

Il filtro opera come segue:

vengono create delle maschere 3x3, nelle quali sono contenuti i valori caratterizzanti la luminosità dei pixel adiacenti ad un determinato pixel. In base a questi valo-

ri, ed operando su tutta l'immagine, viene calcolato per ogni pixel un valore del gradiente e questo viene confrontato con una determinata soglia: se risulta che il gradiente si trovi al di sopra di tale soglia, il valore di luminosità del pixel viene posto a 255 (bianco), altrimenti il valore di luminosità resta invariato. La parte essenziale del programma relativo al filtraggio dell'immagine e il risultato di questa fase riprodotta al PC sono riportate di seguito.

Sobel(greyscale);

La funzione *Sobel( )* effettua il filtraggio per mezzo del filtro di *Sobel*, e modifica il contenuto dell'array *greyscale*; il corpo della funzione è il seguente:

```
void Sobel(unsigned char *greyscale)
{
  int j;
  int gradG;
  int deltaXG, deltaYG;

  unsigned char *imageInG = greyscale, *imageOutG = NULL;

  imageOutG = malloc(307200 * sizeof(unsigned char));

  if (!imageOutG) {
            fprintf(stderr,imageOut alloc err\n);
            return;
            }

  for (j = grab_width; j < (grab_height-1)*grab_width; j++)
  {
    deltaXG = 2*imageInG[j+1] + imageInG[j-grab_width+1]
            + imageInG[j+grab_width+1] - 2*imageInG[j-1]
```

```
            - imageInG[j-grab_width-1] - imageInG[j+grab_width-1];

    deltaYG = imageInG[j-grab_width-1] + 2*imageInG[j-grab_width]
            + imageInG[j-grab_width+1] - imageInG[j+grab_width-1]
            - 2*imageInG[j+grab_width] - imageInG[j+grab_width+1];

    gradG = (abs(deltaXG) + abs(deltaYG)) / 3;

    if (gradG > 255) // soglia del gradiente...
      gradG = 255;

    imageOutG[j] = (unsigned char) gradG;
    }

  memcpy(greyscale, imageOutG, 307200); // greyscale è l'immagine filtrata

  free(imageOutG);
}
```

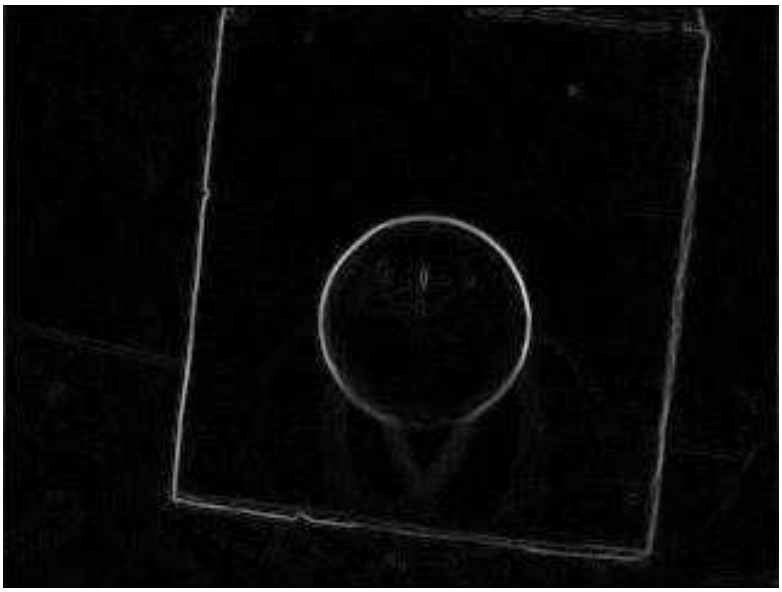

Figura 7: Immagine filtrata con il filtro di *Sobel* (o del gradiente)

Risulta chiaro che il filtraggio con il filtro di *Sobel* per-

mette di eliminare molta informazione inutile dall'immagine originale, e ció che ora è contenuto in memoria può passare alla fase successiva di filtraggio.

Per rendere meno oneroso il carico di lavoro attribuito alla *CPU* e ridurre i tempi di elaborazione del programma si è ritenuto opportuno filtrare l'immagine, precedentemente filtrata con *Sobel*, in modo tale da ottenere una immagine in bianco e nero contenente esclusivamente i contorni degli oggetti d'interesse per l'elaborazione.

Il filtraggio appena descritto opera nel modo seguente:

data una soglia di grigio opportuna, si controlla se il pixel attuale ha un valore di luminosità superiore alla soglia; in caso affermativo il pixel viene posto a 255 (bianco), in caso contrario a 0 (nero).

Questo processo, oltre a mettere in evidenza i contorni dell'oggetto di interesse, permette anche di eliminare tutti quei pixel che non sono utili ai fini del riconoscimento del *Landmark*, e quindi parte del disturbo proveniente dall'immagine originale, rendendo affidabile e sicuro il sistema di rilevamento e la successiva fase di elaborazione dei dati di orientamento e di navigazione.

La parte di programma che realizza tale ulteriore elaborazione risulta:

convertBW(greyscale,grab_width,grab_height,60);

La funzione opera il filtraggio sull'array *greyscale*, che contiene l'immagine precedentemente filtrata con *Sobel* e

di dimensioni *grab_width* x *grab_height*, con una soglia pari al valore definito dall'ultimo parametro passato alla funzione (in questo caso 60). Il corpo della funzione è il seguente:

```
void convertBW(unsigned char *greyscale, int width, int height, int
soglia)
{
   int s;

   for(s=0;s<width*height;s++)
   {
     if(greyscale[s] > (unsigned char) soglia)
       greyscale[s] = (unsigned char) 255; // bianco per disegno contorni
     else greyscale[s] = 0;
   }
}
```

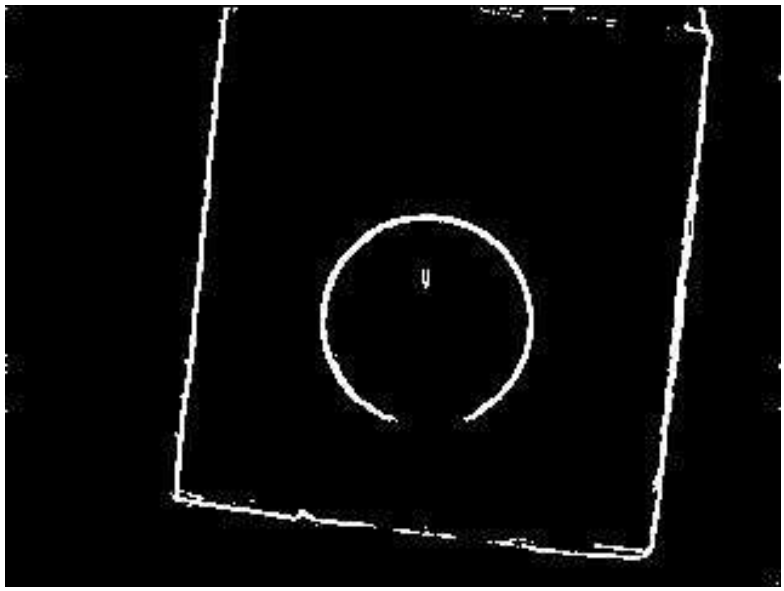

Figura 8: Immagine convertita in bianco e nero

L'immagine ora contenuta in memoria, e riportata nella figura 8, è pronta per passare alla fase successiva di elaborazione.

3.4 Elaborazione dell'immagine

Il processo di elaborazione dell'immagine consiste nel riconoscimento, a partire dall'immagine filtrata, dell'oggetto di interesse (*Landmark*) contenuto nell'immagine.

Questo processo è il più oneroso dal punto di vista di calcoli che la *CPU* deve effettuare. Ad ogni modo la velocità di elaborazione del processore ha permesso di ottenere risultati soddisfacenti in questo senso.

Nella fase di elaborazione la funzione addetta al riconoscimento delle circonferenze opera nel modo seguente:

partendo da un pixel distanziato dal bordo superiore e laterale di una distanza predefinita[9], l'algoritmo ricerca le possibili circonferenze con centro nel pixel corrente escludendo automaticamente le circonferenze con raggio inferiore ad una determinata soglia o superiore ad un'altra soglia predefinita, considerando anche i pixel in prossimità dei bordi dell'immagine.

Inoltre l'algoritmo, nella ricerca delle circonferenze, esclude anche i risultati erronei derivanti dalla eventuale presenza nell'immagine filtrata di tratti curvilinei di contorno, non facenti parte del *Landmark*. L'algoritmo di

[9]La distanza non è inferiore al raggio della circonferenza più piccolo che si può ricercare nella figura, altrimenti la circonferenza uscirebbe dall'area dell'immagine.

riconoscimento sviluppato all'interno del programma è riportato a seguito.

```
int circle_recog()
{
   int x1,y1,r1,x2,flag,count1 = 0;
   int count2 = 0;
   float th = 0.0;
   flag = 0;
   for(x1 = 40 ; x1 <= grab_width/2 ; x1+=2)
    for(y1 = 70 ; y1 < (grab_height - 70) ; y1+=2)
     for(r1 = 34 ; r1 <= x1 - 10 ; r1++)
     if( ( ( (y1 < grab_height/2) && (y1 >= r1) )
     || ( (y1 >= grab_height/2) && (grab_height - y1) >= r1) )
     && (r1 < 65) )
     {
      x2 = grab_width - x1;
      count1 = 0;
      count2 = 0;
      flag = 0;
      for(th = 0.0 ; th < 2*PI ; th+= 0.0349066) // in radianti 180 cicli
       {
        if( BW[x1 + (int) (r1 * cos(th))][y1 - (int) (r1 * sin(th))]
            == 255 )
          count1++;
        if( r1 <= (grab_width - x1) )
        if( BW[x2 + (int) (r1 * cos(th))][y1 - (int) (r1 * sin(th))]
            == 255 )
          count2++;
        if( th > 0.349066 && count1 < 6 && count2 < 6 )
          th = 2*PI;
        if( count1 > 65 ) // soglia per l'individuazione delle circonferenze
          {
            flag = 1;
            th = 2*PI;
            xc = x1;
            yc = y1;
            rc = r1;
            r1 = x1 = grab_width;
            y1 = grab_height;
```

```
            count1 = count2 = 0;
          }
        if( count2 > 65 ) // soglia per l'individuazione delle circonferenze
          {
            flag = 1;
            th = 2*PI;
            xc = x2;
            yc = y1;
            rc = r1;
            r1 = x1 = grab_width;
            y1 = grab_height;
            count2 = count1 = 0;
          }
        } // fine ciclo for di theta (th)
      } // fine if ...
  if(flag == 0) return(0); // circonferenza non individuata...
  else return(1); // circonferenza individuata...
}
```

Il risultato dell'elaborazione, visualizzato sul monitor del PC, ha fornito l'immagine elaborata[10] riportata in figura 9.

3.5 Interpretazione dei dati

I dati provenienti dall'elaborazione devono essere interpretati in modo da fornire indicazioni su come debba ruotare la torretta del Robot per portarsi a centrare il *Landmark*.

L'algoritmo di riconoscimento della circonferenza aggiorna i parametri che riguardano la posizione nell'im-

[10]Sull'immagine riportata in figura 9, è presente una croce, disegnata dal programma di prova durante l'elaborazione, che corrisponde al centro calcolato per la circonferenza individuata.

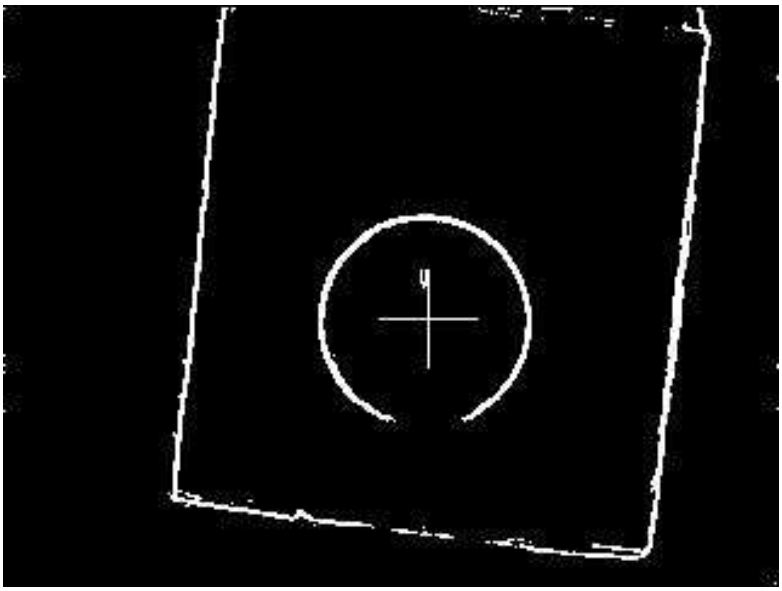

Figura 9: Immagine finale elaborata (circonferenza individuata)

magine del centro del cerchio individuato ed il parametro riguardante il valore del raggio della circonferenza.

Utilizzando come dati la variazione della distanza della coordinata x del centro della circonferenza con la coordinata x del centro dell'immagine, operando una opportuna stima[11] dell'angolo di cui devono ruotare la torretta e le ruote per centrarsi con il *Landmark*, si trasmette al Robot l'istruzione che permette alla torretta e alle ruote di ruotare di un angolo tale da centrare il *Landmark*.

La sezione di programma relativo all'algoritmo di interpretazione dei dati applicati al sistema di navigazione del Robot risulta:

[11] La stima viene effettuata considerando che ad una maggiore variazione di distanza del centro della circonferenza dal centro dell'immagine corrisponde una rotazione maggiore della torretta e delle ruote

```
if(enter)
{
  if( (grab_width/2 - xc) > 0 && (grab_width/2 - xc) <= 35 )
    sign = 1.2;
  if( (grab_width/2 - xc) > 35 && (grab_width/2 - xc) <= 80 )
    sign = 2.8;
  if( (grab_width/2 - xc) > 80 && (grab_width/2 - xc) <= 140 )
    sign = 3.5;

  if( (grab_width/2 - xc) < 0 && (grab_width/2 - xc) >= -35 )
    sign = -1.2;
  if( (grab_width/2 - xc) < -35 && (grab_width/2 - xc) >= -80 )
    sign = -2.8;
  if( (grab_width/2 - xc) < -80 && (grab_width/2 - xc) >= -140 )
    sign = -3.5;

  if( (grab_width/2 - xc) >= -16 && (grab_width/2 - xc) <= 16 )
    sign = 0;

  mv(MV_IGNORE,MV_IGNORE,MV_PR,sign*14,MV_PR,sign*14);
}
```

3.6 Navigazione del Robot

Il programma per la navigazione del Robot viene eseguito parallelamente a quello per il riconoscimento del *Landmark*. Inoltre viene mandato in esecuzione il programma *Nserver*, che rappresenta il *link* tra Robot e software, in modo da permettere la connessione e la comunicazione tra PC e Robot. Lo schema a blocchi del programma è riportato in figura 10.

Per la navigazione, il Robot utilizza come sensori i sonar solidali con la torretta, e i dati forniti dal pro-

gramma di elaborazione delle immagini provenienti dalla Webcam[12].

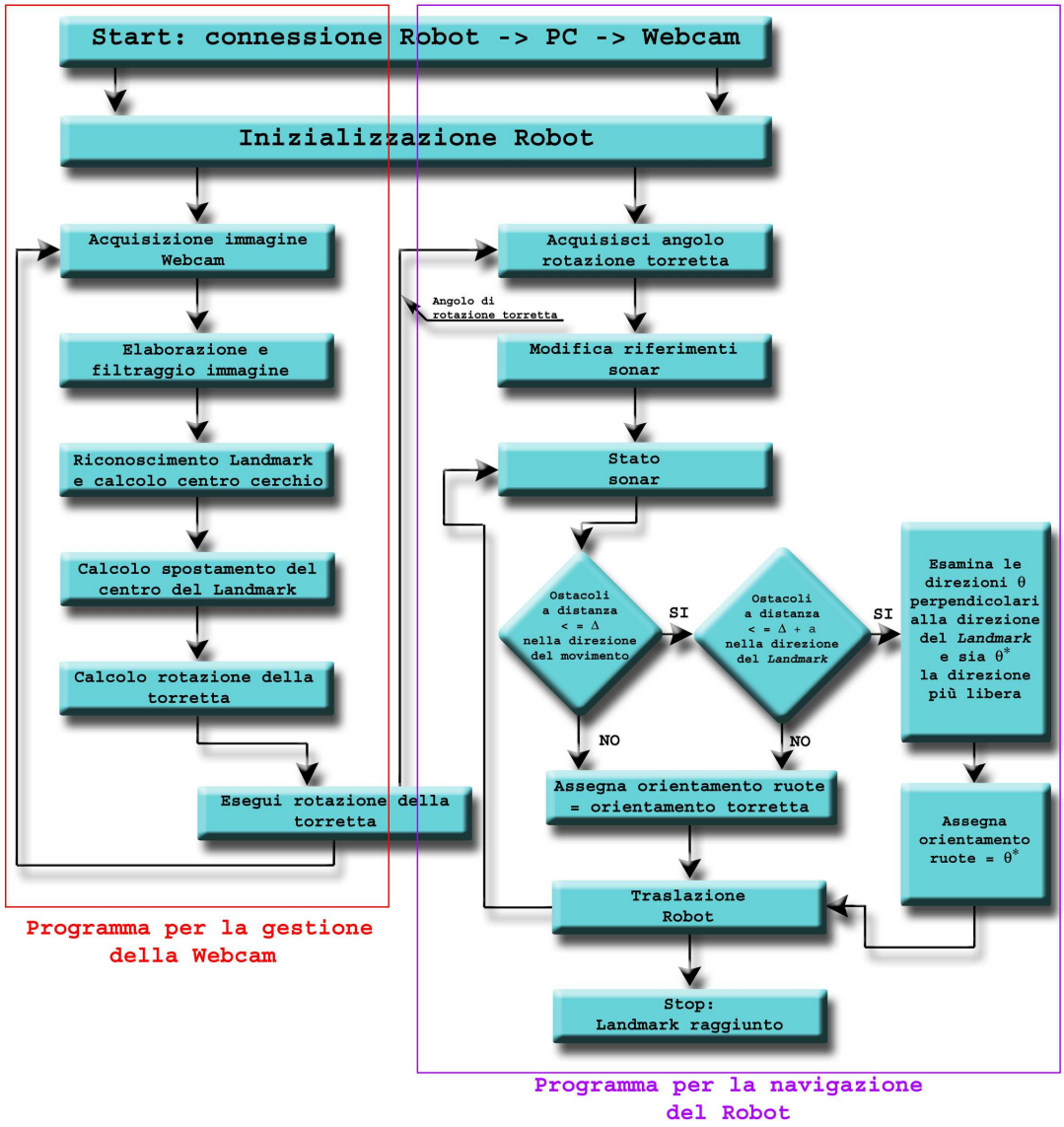

Figura 10: Schema di funzionamento del software descritto

Il programma di navigazione del Robot in primo luo-

[12]I dati sono l'angolo di cui è stata ruotata la torretta rispetto alla posizione precedente, l'angolo di cui sono state ruotate le ruote del Robot e quindi la direzione che deve seguire il Robot verso il *Landmark*.

go opera la inizializzazione del Robot e del collegamento fra questo ed il PC. Conclusa la fase di inizializzazione il programma acquisisce l'angolo di cui è ruotata la torretta del Robot per aggiornare i parametri relativi alla navigazione (inseguimento del *Landmark*).

La rotazione della torretta sulla quale è posizionata la Webcam, e con essa la rotazione dei sonar, ha richiesto una modifica dei riferimenti di questi sensori in modo da adeguarsi all'algoritmo di *obstacle avoidance*, mantenendo prioritari i cinque sonar che scoprono un angolo di 90 gradi (22.5x4) simmetricamente all'asse di allineamento della Webcam, che gli consentono di considerare comunque prioritaria la direzione di navigazione verso il *Landmark* in movimento anche durante la fase di aggiramento dell'ostacolo[13].

L'algoritmo di *obstacle avoidance* opera come segue:

si selezionano i 5 sonar frontali (nel senso della direzione di movimento del Robot e che individuano un angolo di indagine di 90 gradi), e si aggiornano iterativamente i valori della distanza letta da essi. Se la distanza raggiunge o scende al di sotto di una soglia Δ predefinita, il Robot arresta il suo movimento e orienta le proprie ruote

[13]Questa parte del software sviluppato non ha dato tuttavia risultati soddisfacenti in quanto la risoluzione di 22.5 gradi per ogni direzione discriminata dal sistema di *obstacle avoidance* non compatibile con la risoluzione che è applicata alla torretta e comandata dal sistema di acquisizione del *Landmark* (1/10 di grado). Nelle prove di esecuzione dell'applicazione eseguita interponendo degli ostacoli, si è infatti riscontata una perdita di direzione del sistema, che non poteva essere corretto efficacemente con gli angoli di rotazione della torretta per assegnare i nuovi riferimenti al sistema dei sonar (la correzione avrebbe richiesto una integrazione delle rotazioni eseguite dalla torretta).

nella direzione dell'ostacolo più distante tra quelli rilevati dai 5 sonar frontali; il comando dell'orientamento delle ruote rimane ancora assegnato ai sonar fino a quando i 3 sonar interni(SONAR_15, SONAR_0, SONAR_1), che individuano un angolo di 22.5x2 = 45 gradi nella direzione di puntamento del *Landmark* seguito dalla torretta[14] è libero da ostacoli[15]. Al verificarsi di questa condizione l'orientamento delle ruote viene nuovamente comandato dalla direzione calcolata con i dati elaborati dal software di gestione della Webcam. Naturalmente i sensori abilitati continueranno a gestire la navigazione del Robot fino a quando non si ristabilisce la condizione sopra illustrata.

A *Landmark* raggiunto, il programma si disconnette dal Robot.

Per la parte di programma contenente l'elaborazione dei dati di orientamento della torretta e di navigazione del Robot si rimanda alla sezione 5.2.

È da notare a questo punto una limitazione imposta dal sistema Robot-Webcam:

poiché la Webcam è posizionata sulla parte frontale del Robot, sulla torretta, essa vincola i sonar, solidali con la stessa torretta, a ruotare quando i dati elaborati richiedono una rotazione: questo si traduce in una perdita di riferimento degli ostacoli (letti dai sonar). Ma soprattutto, essendo i sonar distanziati l'uno dall'altro di 360/16

[14]La webcam è posizionata in allineamento con il SONAR_0.

[15]Condizione distanza rilevata $> \Delta + a$, dove con a è assegnato un margine necessario ad evitare il ricorrere della condizione $\leq \Delta$ per l'aggiramento sicuro dell'ostacolo.

gradi (22.5 gradi), e quindi con una risoluzione pari a 22.5 gradi, lo step di rotazione imposto dai sonar non è compatibile con le rotazioni calcolate dal programma per inseguire il *Landmark* con la Webcam posta sulla torretta (lo sensibilità di rotazione della torretta è di 1/10 di grado).

In questo modo nello sviluppo pratico dell'applicazione eseguito nel laboratorio di Automatica della Facoltà di Ingegneria Informatica dell'Università di "Tor Vergata",

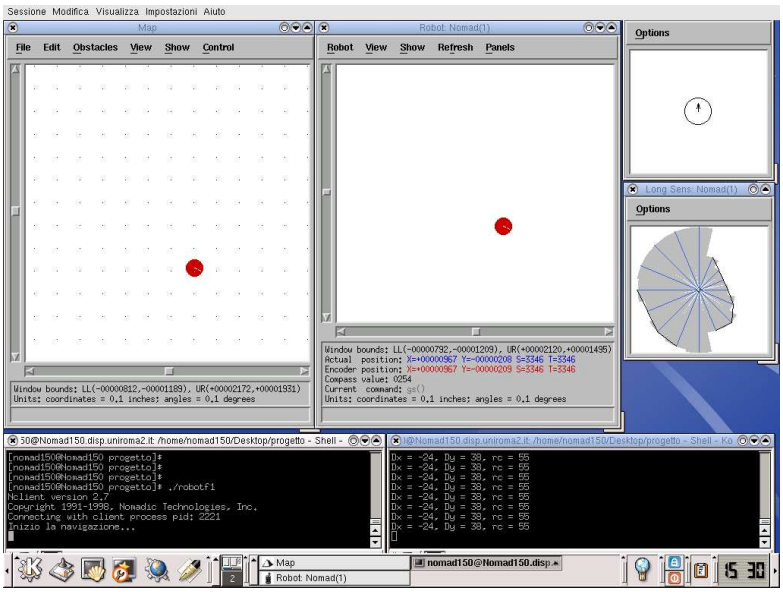

Figura 11: Prova di esecuzione del programma

si è potuto verificare la navigazione del sistema Robot-Webcam verso un *Landmark* costituito da una sfera che veniva spostata dall'operatore fino al raggiungimento della stessa (in figura 11 è riportata la schermata per una prova di esecuzione del programma); nella fase di naviga-

zione, il Robot aggiornava in continuazione la rotazione della torretta in modo da mantenere la Webcam puntata verso il *Landmark*, dirigendosi contemporaneamente verso di esso.

CAPITOLO 4

Considerazioni finali

4 Considerazioni finali

4.1 Considerazioni sul sistema

Nell'applicazione eseguita sono emerse due problematiche relative all'hardware utilizzato, che hanno limitato in modo sostanziale la possibilità di un approccio più completo ed articolato della navigazione del Robot *Nomad 150* integrata con un sistema di rilevamento delle immagini, costituito dalla Webcam Philips *ToUcam Pro 740*:

1. i tempi di elaborazione dell'immagine;
2. la posizione della Webcam sull'unica torretta contenente anche i sensori ad ultrasuoni che guidano il Robot *Nomad 150*.

Per i tempi di elaborazione più lunghi ci si riferisce a quelli del filtraggio e del riconoscimento delle circonferenze; essi, limitando l'aggiornamento delle immagini, hanno anche limitato la velocità con la quale il *Landmark* poteva essere seguito dal Robot e quindi spostato: infatti, essendo oltretutto il campo visivo della Webcam abbastanza limitato, sperimentalmente è accaduto che per via di questi ritardi nell'elaborazione, e quindi nell'aggiornamento delle immagini, il *Landmark* non fosse riconosciuto, a causa del fatto che era uscito dal campo visivo della Webcam.

Inoltre la correzione dei parametri di luminosità e di contrasto, operati automaticamente dalla Webcam, han-

no causato, anche se in rari casi, per via della lentezza con la quale avveniva la transizione da determinati valori di luminosità ad altri, il mancato riconoscimento del *Landmark*.

Come già detto la Webcam è solidale alla torretta del Robot e questo ha determinato delle limitazioni sostanziali.

Essendo infatti la torretta solidale con i sonar, oltre che con la Webcam, una rotazione della stessa torretta, imposta dai comandi impartiti dal software di gestione della Webcam, ha causato conflitto con i comandi dati al Robot dal software per la navigazione. Infatti la rotazione della torretta per inseguire il *Landmark* determina anche la rotazione dei sonar, e cambia quindi il loro riferimento per il riconoscimento degli ostacoli utilizzato nel software per la navigazione. Pertanto non è stato possibile realizzare un'applicazione integrata tra i due sistemi di navigazione compatibile con l'hardware approntato. Il problema poteva essere superato se il Robot fosse stato dotato di una propria torretta a disposizione della sola Webcam, alla quale impartire i comandi specifici di orientamento per l'inseguimento del *Landmark*, lasciando al sistema dei sonar il compito di evitare gli ostacoli.

Le prestazioni del sistema possono quindi essere migliorate intervenendo sull'hardware, con l'impiego di una doppia torretta (quella del Robot e una dedicata per la Webcam) e utilizzando inoltre una telecamera più adat-

ta all'acquisizione e al processamento delle immagini, che possa adattarsi ai cambiamenti di luminosità dell'ambiente in modo più veloce e soprattutto con un campo visivo più ampio rispetto a quello della Webcam utilizzata; con ciò, si ritiene sia possibile incrementare ulteriormente le prestazioni del sistema.

Dal punto di vista del software le modifiche che possono essere effettuate sono in parte collegate all'hardware. È infatti possibile modificare i parametri di luminosità, contrasto e bilanciamento del bianco (*whiteness*) per controllare via software le variazioni ambientali di luminosità. Inoltre è stato necessario, al fine di ottenere buoni risultati in termini di tempo di elaborazione, utilizzare come risoluzione la 320x240 pixel[16] per l'acquisizione ed il processamento delle immagini. Infine, l'algoritmo utilizzato per il riconoscimento del *Landmark* che, a causa della quantità di dati da elaborare, rappresenta la parte più pesante del programma sotto l'aspetto dei tempi di esecuzione, potrebbe essere modificato con risultati di elaborazione più rapidi impiegando il metodo della trasformata di *Hough* per il riconoscimento della circonferenza, o metodo basato su *colorized hysteresis thresholding* [8] per il riconoscimento di un cerchio di colore uniforme.

[16]Come detto la Webcam Philips *ToUcam Pro 740* permette di raggiungere risoluzioni di 640x480 pixel.

Riferimenti bibliografici

[1] D.H.Ballard e C.M.Brown [1982], *Computer Vision*, Prentice-Hall.

[2] P.J.McKerrow [1991], *Introduction to Robotics*, Addison-Wesley Publishing Company.

[3] M.D.Adams, H.Hu, P.J.Probert [1990], *Toward a Real Time Architecture for Obstacle Avoidance and Path Planning in Mobile Robots* in Proceedings of the 1990 IEEE International Conference on Robotics and Automation, pp 584-589.

[4] J.Borenstein, L.Feng, H.R. Everett, D. Wehe, *Mobile Robot Positioning - Sensors and Techniques*, Journal of Robotic Systems, Special issue on mobile robots. Vol 14 No 4, pp 231-249.

[5] K.S. Fu, R.C. Gonzales, C.S. Lee [1989], *Robotica*, McGraw Hill, New York.

[6] Nomadic Technologies Inc. [1997], *Nomad 150 Hardware Manual*, Mountain View, CA, USA.

[7] A. Fefè [1995], *Sistemi di visione e loro applicazione alla pianificazione del moto di robot mobili*, Università degli studi di Roma - Tor Vergata.

[8] D.Coleman, D.Creel, D. Drinka, N. Holifield, W. Mantzel, A. Saidi, M. Zuffoletti [2002], *Implementation of an Autonomous Aerial Reconnaissance*

System, University of Texas at Austin, IEEE Robot Team Aerial Robotics Project.

[9] I.R.Nourbakhsh, D.Andre, C.Tomasi, M.R.Genesereth [1996], *Obstacle Avoidance Via Depth From Focus*, Computer Science Department, Stanford University.

[10] B.Dirks [1999], *V4L - Video 4 Linux API Specification.*

[11] Nemosoft Unv [2001], *Video4Linux API additions for Philips USB Webcams.*

CAPITOLO 5

Listato

5 Listato

5.1 Codice sorgente per il software di controllo della Webcam

```
//
// Programma per il controllo della webcam e della torretta
// del robot
//
// compilare con:
// gcc -o webcam webcamfinale.c ‘imlib2-config –cflags‘
// ‘imlib2-config –libs‘ Nclient.o
//

#include <X11/Xlib.h>
#include <Imlib2.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <linux/videodev.h>
#include <math.h>

#include pwc-ioctl.h // header file per la webcam philips...
#include Nclient.h // header file per il robot Nomad 150...

#define PI 3.141592654
// conversione da decimi di inch a centimetri
#define COEFF_DIST 0.254
// conversione da decimi di grado a radianti
#define COEFF_ANGOLO 0.0017453


//
// dichiarazione costanti e variabili globali
//

  int xc,yc,rc;
```

```
  const char *temp_file = webcam.jpg;
  const char *salva_ext = jpg;

  int grab_width = 320;
  int grab_height = 240;
  int grab_delay = 0;
  int grab_quality = 100;

  int cam_contrast = 70;
  int cam_brightness = 30;
  int cam_hue = 50;
  int cam_colour = 50;
  int cam_whiteness = 60;
  int cam_framerate = 30;

  static struct video_mmap grab_buf;
  struct video_picture cam_pic;
  struct video_capture vicap;
  struct video_buffer buffer;
  struct pwc_probe probe;
  struct pwc_leds LED;
  int type,a,webcam,frame,i,grab_size;
  int IsPhilips;

  unsigned char *grab_data = NULL;
  unsigned char BW[320][240];

  Imlib_Image immagine,immagine2;

//
// prototipi delle funzioni
//

  int
    circle_recog();

  void
    convertBW(unsigned char *greyscale, int width,
                      int height, int soglia);
```

```
void
  Sobel(unsigned char *greyscale);

Imlib_Image
  convert_yuv420p_to_imlib2(unsigned char *mem,
                                          int width, int height);

Imlib_Image
  convert_rgb24_to_imlib2(unsigned char *mem,
                                        int width, int height);

void
  convert_imlib2_to_rgb24(Imlib_Image im, unsigned char *dest,
                                        int width, int height);

void
  convert_rgb24_to_greyscale (unsigned char *src,
                                          int n, unsigned char *dest);

void
  convert_imlib2_to_greyscale(Imlib_Image src, unsigned char *dest,
                                          int width, int height);

Imlib_Image
  convert_greyscale_to_imlib2(unsigned char *src,
                                          int width, int height);

int
  save_image(Imlib_Image image, char *file);

void
  salva_jpeg(Imlib_Image im);

  //void convert_rgb24_to_greyscale24 (unsigned char *src,

  //void swap_rgb24(unsigned char *mem, int n);

//
// funzione main()
//
```

```
int main(void)
{
  struct video_window vwin;
  struct video_capability grab_cap;
  struct video_mbuf vid_mbuf;


  // apre il dispositivo webcam...

  webcam = open(/dev/video0, O_RDWR);
  if(webcam<0)
  {
    printf(Errore:  non c'e' alcuna webcam connessa!\n);
    exit(-1);
  }

  // controlla se e' una webcam philips

  IsPhilips = 0;
  if (ioctl(webcam, VIDIOCGCAP, &grab_cap) < 0)
  {
    printf(Webcam non philips!!!\n);
    return(0);
  }
  if (sscanf(grab_cap.name, Philips %s webcam, &type) < 1)
  {
    // Webcam ancora non trovata, si prova con il probe
    if (ioctl(webcam, VIDIOCPWCPROBE, &probe) == 0)
      if (!strcmp(grab_cap.name,probe.name))
        IsPhilips = 1;
  }
  else
    IsPhilips = 1;

  if(IsPhilips)
    printf(E' una webcam philips!!!\n);

  // controlla il LED

  // Fa brillare il LED, 1 secondo di intervallo
```

```
  // durante l'acquisizione delle immagini
  LED.led_on = 200;
  LED.led_off = 800;
  ioctl(webcam, VIDIOCPWCSLED, &LED);


//————————————————————————————
// CAPACITA' DELLA WEBCAM
//————————————————————————————

  if (-1==ioctl(webcam,VIDIOCGCAP,&grab_cap))
  {
    // prende i valori contenuti in VIDIOCGCAP,
    // cioè le video capabilities
    perror(ioctl VIDIOCGCAP);
    exit(-1);
  }
  printf(-----------------------------------\n);
  printf(-----------------------------------\n);
  printf(nome -> %s\n, grab_cap.name);
  printf(tipo -> %d\n, grab_cap.type);
  printf(max larghezza -> %d\n, grab_cap.maxwidth );
  printf(max altezza -> %d\n, grab_cap.maxheight);
  printf(min larghezza -> %d\n, grab_cap.minwidth );
  printf(min altezza -> %d\n, grab_cap.minheight );
  printf(-----------------------------------\n);

//————————————————————————————
// FINESTRA PER LA CATTURA
//————————————————————————————

  vwin.x = 0;
  vwin.y = 0;
  vwin.width = grab_width;
  vwin.height = grab_height;
  vwin.clipcount = 0;
  vwin.chromakey = 1;
  vwin.flags = VIDEO_CLIPMAP_SIZE;

  int shutter = -1;
  int gain = -1;
```

```
ioctl(webcam, VIDIOCGWIN, &vwin);
if (vwin.flags & PWC_FPS_MASK)
{
  // Setta il nuovo framerate
  vwin.flags &= ~PWC_FPS_FRMASK;
  vwin.flags |= (cam_framerate << PWC_FPS_SHIFT);
}

// spegne lo snapshot mode
vwin.flags &= ~(PWC_FPS_SNAPSHOT);

if (ioctl(webcam, VIDIOCSWIN, &vwin) < 0)
  perror(provo a settare funzioni extra per la pwc\n);

if (ioctl(webcam, VIDIOCPWCSAGC, &gain) < 0)
  perror(provo a settare il guadagno\n);

if (ioctl(webcam, VIDIOCPWCSSHUTTER, &shutter) < 0)
  perror(provo a settare lo shutter\n);

if (vwin.flags & PWC_FPS_FRMASK)
  printf(Framerate corrente:  %d fps\n,
            (vwin.flags & PWC_FPS_FRMASK) >> PWC_FPS_SHIFT);
else
  return(0);


//------------------------------
// PROPRIETÀ DELL'IMMAGINE
//------------------------------

cam_pic.contrast = 65535 * ((float) cam_contrast / 100);
cam_pic.brightness = 65535 * ((float) cam_brightness / 100);
cam_pic.hue = 65535 * ((float) cam_hue / 100);
cam_pic.colour = 65535 * ((float) cam_colour / 100);
cam_pic.whiteness = 65535 * ((float) cam_whiteness / 100);
if (-1==ioctl( webcam, VIDIOCSPICT, &cam_pic ))
{
  // setta i valori per le proprietà dell'immagine
  perror(ioctl VIDIOCSPICT);
```

```
  exit(-1);
 }

 // ottiene i precedenti valori
 ioctl( webcam, VIDIOCGPICT, &cam_pic );

 printf("-----------------------------------\n");
 printf("luminosita' -> %d \n", cam_pic.brightness/256 );
 printf("tonalita' -> %d\n", cam_pic.hue/256);
 printf("colore -> %d\n", cam_pic.colour/256 );
 printf("contrasto -> %d \n", cam_pic.contrast/256 );
 printf("bianco -> %d\n", cam_pic.whiteness/256 );
 printf("profondita' -> %d\n", cam_pic.depth );
 printf("palette -> %d \n", cam_pic.palette );
 printf("-----------------------------------\n");

//--------------------------------
// MAPPING DEL BUFFER
//--------------------------------


 if (-1==ioctl(webcam,VIDIOCGMBUF,&vid_mbuf))
 {
  perror("ioctl VIDIOCGMBUF"); // informazioni sul memory map buffer
  exit(-1);
 }

 printf("-----------------------------------\n");
 // memoria totale da mappare in byte
 printf("size -> %d\n",vid_mbuf.size);
 printf("frame -> %d\n",vid_mbuf.frames); // frames
 printf("-----------------------------------\n");

 grab_buf.format = VIDEO_PALETTE_YUV420P;
 grab_buf.width = grab_width;
 grab_buf.height = grab_height;
 grab_buf.frame = 0; // avendo 2 frame, si puo' scegliere lo 0 e l'1

 grab_size = vid_mbuf.size;
 // mappa il video buffer
 grab_data = mmap(0,grab_size, PROT_READ |
```

```
                    PROT_WRITE, MAP_SHARED, webcam, 0);

//-------------------------------------
// SETTA I BUFFER PER LA CATTURA
//-------------------------------------


  if(ioctl(webcam,VIDIOCMCAPTURE,&grab_buf)<0)
    perror(ioctl VIDIOCMCAPTURE);

  if (-1 == ioctl(webcam,VIDIOCSYNC,&grab_buf))
    perror(ioctl VIDIOCSYNC);
  else
    printf(Immagine catturata...\n);

    //DATA32 *grab_data32;

  unsigned char *greyscale;

  SERV_TCP_PORT=7019;
  float distanza;
  int l,k,z = 0;
  int enter = 0; // non e' ancora entrato nel ciclo for...

  float sign = 0.0;

  greyscale = malloc(307200 * sizeof(unsigned char));

  if(!connect_robot(1))
    exit(0);

  printf(Inizio riconoscimento circonferenza...\n);

  for( ; ; )
  {
    k = l = z = 0;
    ioctl(webcam,VIDIOCMCAPTURE,&grab_buf);
    ioctl(webcam,VIDIOCSYNC,&grab_buf);
    immagine = convert_yuv420p_to_imlib2(grab_data,
                        grab_width,grab_height);
    //imlib_context_set_image(immagine);
```

```
//sleep(1);
//salva_jpeg(immagine);
convert_imlib2_to_greyscale(immagine,greyscale,
                                        grab_width,grab_height);
Sobel(greyscale);
// ora in greyscale c'e' un'immagine in bianco e nero
convertBW(greyscale,grab_width,grab_height,60);
//Imlib_Image immagine3;
//immagine3 = convert_greyscale_to_imlib2(greyscale,
// grab_width,grab_height);
//sleep(1);
//salva_jpeg(immagine3);

for(k = 0 ; k < grab_height ; k++)
  for(l = 0 ; l < grab_width ; l++)
  {
    BW[l][k] = greyscale[z];
    z++;
  }

if(!circle_recog()) // riconoscimento circonferenze...
{
  //printf(Circonferenza persa...\n);
  enter = 0;
}

if(enter)
{
  if( (grab_width/2 - xc) > 0 && (grab_width/2 - xc) <= 35 )
    sign = 1.2;
  if( (grab_width/2 - xc) > 35 && (grab_width/2 - xc) <= 80 )
    sign = 2.8;
  if( (grab_width/2 - xc) > 80 && (grab_width/2 - xc) <= 140 )
    sign = 3.5;

  if( (grab_width/2 - xc) < 0 && (grab_width/2 - xc) >= -35 )
    sign = -1.2;
  if( (grab_width/2 - xc) < -35 && (grab_width/2 - xc) >= -80 )
    sign = -2.8;
  if( (grab_width/2 - xc) < -80 && (grab_width/2 - xc) >= -140 )
    sign = -3.5;
```

```
      if( (grab_width/2 - xc) >= -16 && (grab_width/2 - xc) <= 16 )
        sign = 0;

      mv(MV_IGNORE,MV_IGNORE,MV_PR,sign*14,MV_PR,sign*14);
    // distanza dal centro xc,yc, raggio rc
    printf(Dx = %d, Dy = %d, rc = %d\n,grab_width/2
                - xc,grab_height/2 - yc,rc);
    }

    enter = 1; // e' entrato nel for
  } // fine for( ; ; )
  free(greyscale);
} // fine main()

//
// funzione per il riconoscimento delle circonferenze...
//


int circle_recog()
{
  int x1,y1,r1,x2,flag,count1 = 0;
  int count2 = 0;
  float th = 0.0;
  flag = 0;

  for(x1 = 40 ; x1 <= grab_width/2 ; x1+=2)
    for(y1 = 70 ; y1 < (grab_height - 70) ; y1+=2)
      for(r1 = 34 ; r1 <= x1 - 10 ; r1++)
      if( ( ( (y1 < grab_height/2) && (y1 >= r1) )
      || ( (y1 >= grab_height/2) && (grab_height - y1) >= r1) )
      && (r1 < 65) )
      {

        x2 = grab_width - x1;

        count1 = 0;
        count2 = 0;
        flag = 0;
        // for di 180 cicli
```

```
  for(th = 0.0 ; th < 2*PI ; th+= 0.0349066)
  {
    if( BW[x1 + (int) (r1 * cos(th))][y1 -
          (int) (r1 * sin(th))] == 255 )
      count1++;

    if( r1 <= (grab_width - x1) )
      if( BW[x2 + (int) (r1 * cos(th))][y1
          - (int) (r1 * sin(th))] == 255 )
        count2++;

    if( th > 0.349066 && count1 < 6 && count2 < 6 )
      th = 2*PI;

    // soglia per l'individuazione delle circonferenze
    if( count1 > 65 )
    {
      flag = 1;
      th = 2*PI;
      xc = x1;
      yc = y1;
      rc = r1;
      r1 = x1 = grab_width;
      y1 = grab_height;
      count1 = count2 = 0;
    }
    // soglia per l'individuazione delle circonferenze
    if( count2 > 65 )
    {
      flag = 1;
      th = 2*PI;
      xc = x2;
      yc = y1;
      rc = r1;
      r1 = x1 = grab_width;
      y1 = grab_height;
      count2 = count1 = 0;
    }
  } // fine ciclo for di theta (th)
} // fine if ...
```

```
    if(flag == 0) return(0); // circonferenza non individuata...
    else return(1); // circonferenza individuata...
}


//
// conversione da toni di grigio a bianco e nero
// (con valore di soglia)
//


void
  convertBW(unsigned char *greyscale,
                    int width, int height, int soglia)
{
  int s;

  for(s=0;s<width*height;s++)
  {
    if(greyscale[s] > (unsigned char) soglia)
      greyscale[s] =
        (unsigned char) 255; // bianco per disegno contorni
    else greyscale[s] = 0;
  }
}


//
// implementazione del filtro di Sobel
//


void
  Sobel(unsigned char *greyscale)
{
  int j;
  int gradG;
  int deltaXG, deltaYG;

  unsigned char *imageInG = greyscale, *imageOutG = NULL;
```

```
  imageOutG = malloc(307200 * sizeof(unsigned char));

  if (!imageOutG)
  {
    fprintf(stderr,imageOut alloc err\n);
    return;
  }

  for (j = grab_width; j < (grab_height-1)*grab_width; j++)
  {
    deltaXG = 2*imageInG[j+1] + imageInG[j-grab_width+1]
              + imageInG[j+grab_width+1] - 2*imageInG[j-1]
              - imageInG[j-grab_width-1]
              - imageInG[j+grab_width-1];

    deltaYG = imageInG[j-grab_width-1] + 2*imageInG[j-grab_width]
              + imageInG[j-grab_width+1] - imageInG[j+grab_width-1]
              - 2*imageInG[j+grab_width] - imageInG[j+grab_width+1];

    gradG = (abs(deltaXG) + abs(deltaYG)) / 3;

    if (gradG > 255) // soglia del gradiente...
      gradG = 255;

    imageOutG[j] = (unsigned char) gradG;
  }

  // in greyscale c'\'e l'immagine filtrata
  memcpy(greyscale, imageOutG, 307200);

  free(imageOutG);
}


//
// conversione da YUV420P a ARGB
//


Imlib_Image
  convert_yuv420p_to_imlib2(unsigned char *src,
```

```
                                        int width, int height)
{
  int line, col;
  int y, u, v, yy, vr = 0, ug = 0, vg = 0, ub = 0;
  int r, g, b;
  unsigned char *sy, *su, *sv;
  Imlib_Image im;
  DATA32 *data, *dest;

  im = imlib_create_image(width, height);
  imlib_context_set_image(im);
  data = imlib_image_get_data();
  dest = data;

  sy = src;
  su = sy + (width * height);
  sv = su + (width * height / 4);

  for (line = 0; line < height; line++)
  {
    for (col = 0; col < width; col++)
    {
      y = *sy++;
      yy = y << 8;
      u = *su - 128;
      ug = 88 * u;
      ub = 454 * u;
      v = *sv - 128;
      vg = 183 * v;
      vr = 359 * v;

      if ((col & 1) == 0)
      {
        su++;
        sv++;
      }

      r = (yy + vr) >> 8;
      g = (yy - ug - vg) >> 8;
      b = (yy + ub) >> 8;
```

```
      if (r < 0)
        r = 0;
      if (r > 255)
        r = 255;
      if (g < 0)
        g = 0;
      if (g > 255)
        g = 255;
      if (b < 0)
        b = 0;
      if (b > 255)
        b = 255;

// canale Blue è agli 8 bit meno significativi,
// Green dagli 8 ai 16, Blue dai 16 ai 24
// e ALPHA dai 24 ai 32
      *dest = (r << 16) | (g << 8) | b | 0xff000000;
      dest++;
    }
    if ((line & 1) == 0)
    {
      su -= width / 2;
      sv -= width / 2;
    }
  }
  imlib_image_put_back_data(data);
  return im;
}


//
// conversione da RGB24 a ARGB
//


Imlib_Image
  convert_rgb24_to_imlib2(unsigned char *mem,
                                    int width, int height)
{
  Imlib_Image im;
  DATA32 *data, *dest;
```

```
  unsigned char *src;
  int i;

  im = imlib_create_image(width, height);
  imlib_context_set_image(im);
  data = imlib_image_get_data();

  dest = data;
  src = mem;
  i = width * height;
  while (i--)
  {
    *dest = (src[2] << 16) | (src[1] << 8) | src[0] | 0xff000000;
    dest++;
    src += 3;
  }

  imlib_image_put_back_data(data);

  return im;
}


//
// conversione da ARGB a RGB24
//


void
  convert_imlib2_to_rgb24(Imlib_Image im,
              unsigned char *dest, int width, int height)
{

  DATA32 *data;
  int i;

  imlib_context_set_image(im);
  data = imlib_image_get_data();
  for(i = 0;i < 921600; i+=3)
  {
    dest[i] = (unsigned char) (data[0]);
```

```
    dest[i+1] = (unsigned char) (data[0] >> 8);
    dest[i+2] = (unsigned char) (data[0] >> 16);
    data++;
  }
}


//
// converte da Imlib2 (ARGB) a scala di grigi (8 bit)
//


void
convert_imlib2_to_greyscale(Imlib_Image src,
        unsigned char *dest, int width, int height)
{
  int c;
  DATA32 *data;

  imlib_context_set_image(src);
  data = imlib_image_get_data();

  for(c = 0; c < width*height; c++)
    dest[c] =
    (int) rint ((unsigned char)(data[c]) * 0.3 +
    (unsigned char)(data[c] >> 8) * 0.59 +
    (unsigned char)(data[c] >> 16) * 0.11);
}


//
// converte da RGB24 a scala di grigi (8 bit)
//


void
  convert_rgb24_to_greyscale (unsigned char *src,
      int n, unsigned char *dest)
{
  int c;
  int cc = 0;
```

```
  for (c = 0; c < n*3; c += 3)
  {
    dest[cc] =
    (int) rint ((src[c]) * 0.3 +
    (src[c + 1]) * 0.59 +
    (src[c + 2]) * 0.11);
    cc++;
  }
}


//
// converte da scala di grigi (8 bit)
// ad immagine in scala di grigi in ARGB
//


Imlib_Image
  convert_greyscale_to_imlib2(unsigned char *src,
                              int width, int height)
{
  Imlib_Image im;
  DATA32 *dest;
  int i, c;
  c = width * height;

  im = imlib_create_image(width, height);
  imlib_context_set_image(im);
  dest = imlib_image_get_data();

  for(i = 0; i < c; i++)
  {
    *dest = (src[i] << 16) | (src[i] << 8) | src[i] | 0xff000000;
    dest++;
  }

  imlib_image_put_back_data(dest);

  return im;
}
```

```
//
// archivia l'immagine corrente in un file in formato jpeg
//


void
  salva_jpeg(Imlib_Image im)
{
  char buffer[1028];
  char date[128];
  time_t t;
  struct tm *tm;
  struct stat st;

  time(&t);
  tm = localtime(&t);
  strftime(date, 127, %Y-%m-%d_%H%M%S, tm);

  do {
        snprintf(buffer, sizeof(buffer), webcam_%s.%s,
                        date, salva_ext);
      } while (stat(buffer, &st) == 0);

  save_image(im, buffer);
}


//
// salva l'immagine in un file
//


int
  save_image(Imlib_Image image,
                  char *file)
{
  Imlib_Load_Error err;

  imlib_save_image_with_error_return(file,&err);
```

```
if ((err) || (!image))
{
  switch (err)
  {
    case IMLIB_LOAD_ERROR_FILE_DOES_NOT_EXIST:
    printf("Errore %s - Il file non esiste", file);
    break;
    case IMLIB_LOAD_ERROR_FILE_IS_DIRECTORY:
    printf("Errore %s - Il file specificato e' una directory",
            file);
    break;
    case IMLIB_LOAD_ERROR_PERMISSION_DENIED_TO_READ:
    printf("Errore %s - Non posso leggere il file", file);
    break;
    case IMLIB_LOAD_ERROR_UNKNOWN:
    case IMLIB_LOAD_ERROR_NO_LOADER_FOR_FILE_FORMAT:
    printf("Errore %s - Non esiste un lettore Imlib2 per
    il formato di file", file);
    break;
    case IMLIB_LOAD_ERROR_PATH_TOO_LONG:
    printf("Errore %s - Il percorso e' troppo lungo", file);
    break;
    case
    IMLIB_LOAD_ERROR_PATH_COMPONENT_NON_EXISTANT:
    printf("Errore %s - Il percorso non esiste", file);
    break;
    case
    IMLIB_LOAD_ERROR_PATH_COMPONENT_NOT_DIRECTORY:
    printf("Errore %s - Il percorso non e' una directory",
             file);
    break;
    case
    IMLIB_LOAD_ERROR_PATH_POINTS_OUTSIDE_ADDRESS_
    SPACE:
    printf("Errore %s - Il percorso punta fuori lo spazio",
              file);
    break;
    case IMLIB_LOAD_ERROR_TOO_MANY_SYMBOLIC_LINKS:
    printf("Errore %s - Troppi livelli di collegamenti simbolici",
              file);
    break;
```

```
    case IMLIB_LOAD_ERROR_OUT_OF_MEMORY:
    printf(Errore %s - Fine della memoria, file);
    break;
    case IMLIB_LOAD_ERROR_OUT_OF_FILE_DESCRIPTORS:
    printf(Errore %s - Fine dei descrittori del file, file);
    break;
    case IMLIB_LOAD_ERROR_PERMISSION_DENIED_TO_WRITE:
    printf(Errore %s - Non posso scrivere sul file, file);
    break;
    case IMLIB_LOAD_ERROR_OUT_OF_DISK_SPACE:
    printf(Errore %s - Non posso scrivere -
         spazio su disco insufficiente, file);
    break;
    default:
    printf(Errore %s - Errore sconosciuto (%d).
    Provo a continuare,file, err);
    break;
   }
   return 0;
  }
  return 1;
}

/*
//
// converte da RGB24 a scala di grigi (24 bit !!!)
//


void
  convert_rgb24_to_greyscale24 (unsigned char *src,
                    int n, unsigned char *dest)
{
  int c;
  int cc = 0;
  for(c=0;c<n;c+=3)
  {
    dest[c] = dest[c+1] = dest[c+2] = (int) rint ((src[c])
                * 0.3 + (src[c + 1]) * 0.59 +
                (src[c + 2]) * 0.11);
  }
```

```
}
*/

/*

//
// converte da GBR24 a RGB24
//


void
  swap_rgb24(unsigned char *mem, int n)
{
  unsigned char c;
  unsigned char *p = mem;
  int i = n ;
  while (–i)
  {
    c = p[0];
    p[0] = p[2];
    p[2] = c;
    p += 3;
  }
}
*/
```

5.2 Codice sorgente per il software per la navigazione del Robot

```
//
// Programma per la navigazione del robot
//
// da compilare con:
// gcc -o robot robot.c Nclient.o
//

#include <stdio.h>
#include <string.h>
#include <fcntl.h>
```

```
#include <sys/stat.h>
#include <sys/mman.h>
#include <sys/ioctl.h>
#include <math.h>

#include "Nclient.h" // header file per il robot Nomad 150...

#define PI 3.141592654
#define COEFF_DIST 0.254 // conversione da decimi di inch a centimetri
#define COEFF_ANGOLO 0.0017453 // conversione da decimi di grado a
radianti

//
// funzioni per la gestione e la navigazione del Robot
//


// la seguente funzione legge la posizione del robot

void LEGGI_ENCODER(double *Stato)
{
  get_rc();
  // converte da decimi di grado a radianti
  Stato[0] = State[STATE_CONF_STEER]*COEFF_ANGOLO;
  // converte le X da decimi di pollici a centimetri
  Stato[1] = State[STATE_CONF_X]*COEFF_DIST;
  // converte le Y da decimi di pollici a centimetri
  Stato[2] = State[STATE_CONF_Y]*COEFF_DIST;
  // converte da decimi di grado a radianti
  Stato[3] = State[STATE_CONF_TURRET]*COEFF_ANGOLO;
}


// la seguente funzione attende l'inizio e la conclusione del movimento

void ATTENDI(int asse)
{
  if(State[asse] == 0) // attende che abbia inizio il movimento
    gs();

  while(State[asse] != 0) // attende che sia completato il movimento
```

```
    gs();
}

// la seguente funzione restituisce il valore minimo dei 3 sonar frontali

int MIN_DISTANZA(int offset)
{
  int i,min = 255;

  if( offset >= 0 )
  {
    // seleziona i 3 sonar frontali
    for(i=0;i<2;i++)
    if( STATE_SONAR_0 + i + offset < STATE_SONAR_15 + 1 )
    {
      if(State[STATE_SONAR_0 + i + offset] < min)
      min = State[STATE_SONAR_0 + i + offset];
    }
    else
      if(State[STATE_SONAR_0 + i + offset - 16] < min)
        min = State[STATE_SONAR_0 + i + offset - 16];

    for(i=15;i<=15;i++)
      if( STATE_SONAR_0 + i + offset < STATE_SONAR_15 + 1 )
      {
        if(State[STATE_SONAR_0 + i + offset] < min)
          min = State[STATE_SONAR_0 + i + offset];
      }
      else
        if(State[STATE_SONAR_0 + i + offset - 16] < min)
          min = State[STATE_SONAR_0 + i + offset - 16];
  }

  if( offset < 0 )
  {
    // seleziona i 3 sonar frontali
    for(i=0;i<2;i++)
    {
      if( STATE_SONAR_0 + i + offset >= STATE_SONAR_0 )
      {
        if(State[STATE_SONAR_0 + i + offset] < min)
```

```
      min = State[STATE_SONAR_0 + i + offset];
    }
    else
      if(State[STATE_SONAR_0 + i + offset + 16] < min)
        min = State[STATE_SONAR_0 + i + offset + 16];
  }
  for(i=15;i<=15;i++)
  {
    if( STATE_SONAR_0 + i + offset >= STATE_SONAR_0 )
    {
      if(State[STATE_SONAR_0 + i + offset] < min)
        min = State[STATE_SONAR_0 + i + offset];
    }
    else
      if(State[STATE_SONAR_0 + i + offset] < min)
        min = State[STATE_SONAR_0 + i + offset];
  }
 }

 return(min);
}

// la seguente funzione e' addetta alla scelta della direzione migliore da
seguire
// restituisce l'angolo di cui deve ruotare il robot per non incontrare ostacoli

int SCELTA_DIREZIONE()
{
 int i,dir,max = 0;
 int l,r;

 for(i = STATE_SONAR_0;i<=STATE_SONAR_4;i++)
  if( (State[i] > max) )
  {
    max = State[i];
    dir = i - STATE_SONAR_0;
  }

 for(i = STATE_SONAR_12;i<=STATE_SONAR_15;i++)
  if( (State[i] > max) )
  {
```

```
    max = State[i];
    dir = i - STATE_SONAR_0;
   }

  if( (l = (STATE_SONAR_0 + dir + 1) ) > STATE_SONAR_15 )
   l = STATE_SONAR_0;
  if( (r = (STATE_SONAR_0 + dir - 1) ) < STATE_SONAR_0 )
   r = STATE_SONAR_15;

  if( State[l] < 15 || State[r] < 15 )
  {
   if( State[l] > State[r] )
    dir = dir + 1;
   else
    dir = dir - 1;
  }

  // sceglie la rotazione a destra o a sinistra...
  if(dir > 8)
   dir = dir - 16;

  // data la soglia di 20 inch, letti dai sonar, sceglie la direzione migliore
  if(max > 20)
   return(dir * 225); // si moltiplica per 22,5 gradi, cioe' 360 gradi/16
sonar
  else
   return(-1);
}

// la seguente funzione controlla che vi sia una direzione libera da seguire
// nella direzione del sonar 0

int SCELTA_DIR_LIBERA(int soglia)
{
  if( (State[STATE_SONAR_0] > soglia)
   && (State[STATE_SONAR_1] > soglia - 20)
   && (State[STATE_SONAR_15] > soglia - 20)
   && (State[STATE_SONAR_14] > soglia - 35)
   && (State[STATE_SONAR_2] > soglia - 35))
   return(1);
  else
```

```
    return(-1);
}

//
// funzione main()
//


int main(void)
{
  SERV_TCP_PORT=7019;
  int angolo,min,i = 0;
  int flag = 0;
  int h = 0;
  int enter = 0; // non e' ancora entrato nel ciclo for...

  if(!connect_robot(1))
    exit(0);

  conf_tm(1); // setta il timeout del robot in secondi
  zr(); // allinea le ruote e la torretta allo zero dei bumper
  ac(200,300,300); // setta le accelerazioni
  sp(100,300,300); // setta le velocita'

  init_sensors();
  printf(Inizio la navigazione...\n);

  for( ; ; )
  {
    vm(50,0,0);

    // finché si trova a più di 28 inch si muove
    while( (min = MIN_DISTANZA(h)) > 28 )
    {
      gs();
      h = (int) ((State[STATE_CONF_STEER]
        - State[STATE_CONF_TURRET])/225);
      if( ((State[STATE_CONF_STEER]
        - State[STATE_CONF_TURRET]) % 225) > 112)
        h = h + 1;
      if( ((State[STATE_CONF_STEER]
```

```
    - State[STATE_CONF_TURRET]) % 225) < -112)
    h = h - 1;

  if( h > 15 )
    h = h - 15;
  if( h < -15 )
    h = h + 15;
}

st();
if( (angolo = SCELTA_DIREZIONE()) != -1)
{
  pr(0,angolo,0);
  ATTENDI(STATE_VEL_STEER);
}
else
  exit(0);

vm(50,0,0);

while( (min = MIN_DISTANZA(h)) > 9 && flag == 0)
{
  gs();
  h = (int) ((State[STATE_CONF_STEER]
    - State[STATE_CONF_TURRET])/225);
  if( ((State[STATE_CONF_STEER]
    - State[STATE_CONF_TURRET]) % 225) > 112)
    h = h + 1;
  if( ((State[STATE_CONF_STEER]
    - State[STATE_CONF_TURRET]) % 225) < -112)
    h = h - 1;

  if( h > 15 )
    h = h - 15;
  if( h < -15 )
    h = h + 15;

  if( SCELTA_DIR_LIBERA(50) == 1 )
    flag = 1;
}
```

```
if( flag == 1 )
{
  if( (angolo = 3600 - (State[STATE_CONF_STEER]
    -State[STATE_CONF_TURRET])) > 1800 )
    angolo = -(State[STATE_CONF_STEER]
      - State[STATE_CONF_TURRET]);
  if( angolo > 1800 )
    angolo = angolo - 3600;
  mv(MV_IGNORE,MV_IGNORE,MV_PR,angolo,MV_IGNORE,MV_IGNORE);
  ATTENDI(STATE_VEL_STEER);
  flag = 0;
}

if( (min = MIN_DISTANZA(h)) <= 9 ) // se la distanza e' minore di 9
{                                  // inch cerca un'altra direzione
  st();

  gs();

  if( (angolo = SCELTA_DIREZIONE()) != -1 )
  {
    pr(0,angolo,0);
    ATTENDI(STATE_VEL_STEER);
  }
  else
    exit(0);

  vm(50,0,0);

  while( (min = MIN_DISTANZA(h)) > 8 && flag == 0)
  {
    gs();
    h = (int) ((State[STATE_CONF_STEER]
      - State[STATE_CONF_TURRET])/225);
    if( ((State[STATE_CONF_STEER]
      - State[STATE_CONF_TURRET]) % 225) > 112)
      h = h + 1;
    if( ((State[STATE_CONF_STEER]
      - State[STATE_CONF_TURRET]) % 225) < -112)
      h = h - 1;
```

```
    if( h > 15 )
      h = h - 15;
    if( h < -15 )
      h = h + 15;

    if( SCELTA_DIR_LIBERA(50) == 1 )
      flag = 1;
  }

  if( flag == 1 )
  {
    if( (angolo = 3600 - (State[STATE_CONF_STEER]
      -State[STATE_CONF_TURRET])) > 1800 )
      angolo = -(State[STATE_CONF_STEER]
        -State[STATE_CONF_TURRET]);
    if( angolo > 1800 )
      angolo = angolo - 3600;
    mv(MV_IGNORE,MV_IGNORE,MV_PR,angolo,MV_IGNORE,MV_IGNORE);
    ATTENDI(STATE_VEL_STEER);
    flag = 0;
  }

if( (min = MIN_DISTANZA(h)) <= 9 ) // se la distanza e' minore di 9
{                                  // inch cerca un'altra direzione
  st();

    gs();

    if( (angolo = SCELTA_DIREZIONE()) != -1 )
    {
      pr(0,angolo,0);
      ATTENDI(STATE_VEL_STEER);
    }
    else
      exit(0);

    vm(50,0,0);

    while( (min = MIN_DISTANZA(h)) > 5 && flag == 0)
    {
      gs();
```

```
        h = (int) ((State[STATE_CONF_STEER]
          - State[STATE_CONF_TURRET])/225);
        if( ((State[STATE_CONF_STEER]
          - State[STATE_CONF_TURRET]) % 225) > 112)
          h = h + 1;
        if( ((State[STATE_CONF_STEER]
          - State[STATE_CONF_TURRET]) % 225) < -112)
          h = h - 1;

        if( h > 15 )
          h = h - 15;
        if( h < -15 )
          h = h + 15;

        if( SCELTA_DIR_LIBERA(50) == 1 )
          flag = 1;
      }

      if( flag == 1 )
      {
        if( (angolo = 3600 - (State[STATE_CONF_STEER]
          -State[STATE_CONF_TURRET])) > 1800 )
          angolo = -(State[STATE_CONF_STEER]
          -State[STATE_CONF_TURRET]);
        if( angolo > 1800 )
          angolo = angolo - 3600;
        mv(MV_IGNORE,MV_IGNORE,MV_PR,angolo,MV_IGNORE,MV_IGNORE);
        ATTENDI(STATE_VEL_STEER);
        flag = 0;
      }
    } // fine primo if
  } // fine secondo if
 } // fine for( ; ; )
} // fine main()
```