UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Elettronica per l'Automazione

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica
(Prof. Riccardo Cassinis)

Tobor ricerca e afferra lattine

Elaborato di esame di: **Michele Cagno, Valerio Manenti, Maria Panteghini**

Consegnato il: **7 settembre 2001**

Sommario

Il robot deve riconoscere quattro diversi tipi di lattine: coca-cola, pepsi, schweppes, henninger; deve raggiungerne una a scelta dell'utente e afferrarla. Il riconoscimento viene effettuato tramite l'analisi cromatica delle immagini acquisite tramite la telecamera posta sul robot. Il raggiungimento si basa sulla ricognizione di un punto significativo della lattina da cercare, le cui coordinate vengono utilizzate dal robot per la sua movimentazione.

1. Introduzione

Lo scopo principale di questo elaborato è il riconoscimento di oggetti tramite l'elaborazione di un'immagine acquisita da una telecamera di basso costo: questo comporta la creazione di algoritmi che siano il più possibile robusti e adatti al riconoscimento degli oggetti scelti, la cui caratteristica saliente, quella cioè che li distingue dal mondo esterno e da altri oggetti simili, è il colore. Per questo motivo abbiamo deciso di lavorare all'individuazione delle lattine, che sono tutte di uguale forma e che vengono riconosciute soprattutto tramite il loro colore predominante: rosso per la coca-cola, blu per la pepsi, giallo per la schweppes, ecc.

Il meccanismo di base consiste nei seguenti passi:

- acquisizione di un'immagine dalla telecamera posta sul robot con un'angolazione di circa 30° rispetto alla posizione orizzontale

Fig 1: La lattina da cercare è la birra Henninger

- filtraggio della stessa per l'individuazione dei possibili punti sopra soglia, cioè di quelli che molto probabilmente appartengono alla lattina che il robot sta cercando, per ottenere un'immagine in bianco e nero in cui nero sta per 'sopra soglia'

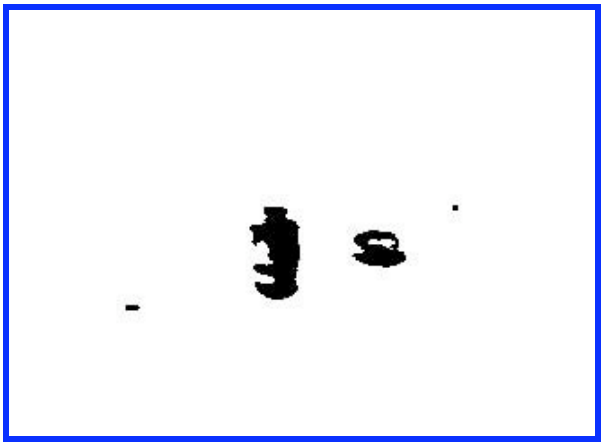


Fig 2: Individuazione dei punti sopra soglia

- elaborazione passabasso dell'immagine in bianco e nero allo scopo di eliminare i disturbi

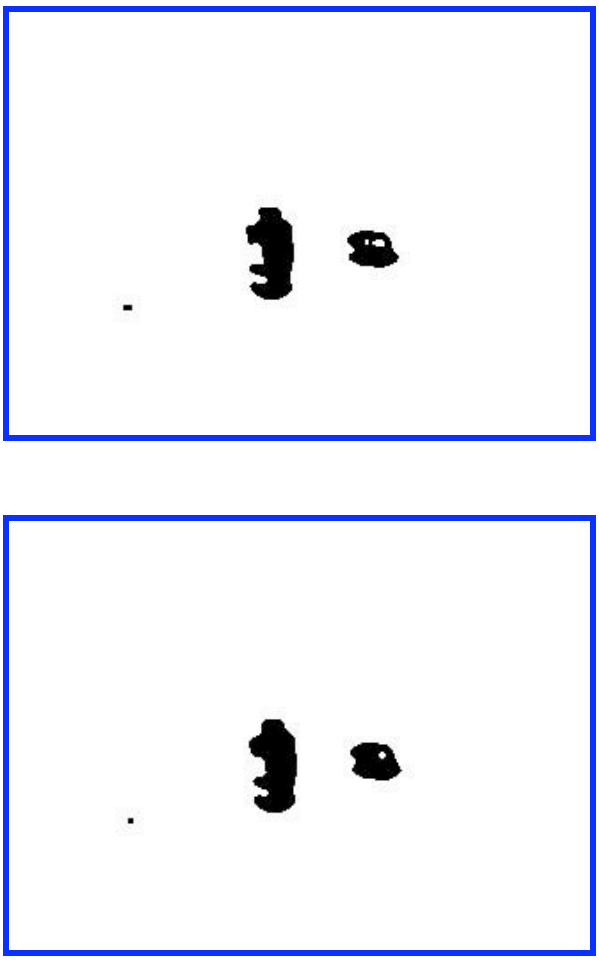


Fig 3: Passi successivi dell'elaborazione passabasso

- ricerca del probabile baricentro della lattina con l'individuazione delle due coordinate bar_x e bar_y che serviranno al robot per spostarsi

Fig 4: Individuazione del "baricentro"

- raggiungimento da parte del robot della lattina prescelta.

2. Il problema affrontato

Come previsto, il problema più difficile sul quale ci siamo imbattuti, è stato il riconoscimento del colore. Come si può vedere dalle immagini, gli oggetti scelti, le lattine, sono particolarmente brutte da riconoscere per svariati motivi. Anzitutto la loro non uniformità: la presenza di scritte e loghi aggiunge molto rumore e toglie punti significativi; inoltre i colori presenti su un determinato tipo di lattina si ritrovano molto facilmente su altri tipi di lattina, generando possibili confusioni tra una lattina e l'altra. In secondo luogo il materiale del quale sono composte, l'alluminio, riflette la luce in maniera tale da distorcere molto i colori acquisiti. A tutto questo si aggiungono le condizioni ambientali, in modo particolare l'illuminazione è costituita da un forte neon, il quale emette luce ad una certa frequenza ovviamente diversa rispetto a quella di acquisizione della telecamera. Ad esempio, la taratura di una certa lattina con delle soglie RGB effettuata nella mattinata, già nel primo pomeriggio della stessa giornata non erano più validi a causa della variazione della luce solare entrante nel laboratorio. Un primo tentativo di soluzione è stato quella di migliorare il mondo nel quale opera il robot. Abbiamo spento il neon e acceso il faretto giallo ad incandescenza in dotazione nel laboratorio. Purtroppo le immagini acquisite in questo modo erano veramente brutte: la particolare forma della lattina non permette un'illuminazione uniforme se la luce proviene dall'alto. Abbiamo provato a montare due faretti in modo da avere una luce frontale diretta, ma anche in questa direzione abbiamo fallito. Ci siamo allora imposti il seguente non facile obiettivo: rendere più robusto l'algoritmo anziché agire sul mondo nel quale lavora il robot.
Un secondo problema, completamente diverso, è stato quello di dover ricercare la lattina prescelta con un'opportuna movimentazione del robot, evitando nel frattempo scontri contro pareti o altri oggetti presenti nell'ambiente. Per spostare il robot all'inizio usavamo istruzioni trovate sul manuale come ad esempio sfSetPosition,

istruzioni di alto livello che semplicemente muovono il robot nella posizione e angolo desiderati occupandosi loro di tutti i dettagli. L'uso di queste istruzioni di alto livello però non permette ai vari behaviour di essere eseguiti in parallelo, rendendo in tal modo Tobor completamente cieco verso gli ostacoli. L'unico modo è di usare istruzioni di livello più basso come SETCOMVEL e SETCOMRVEL che settano direttamente velocità angolare e traslazionale del robot. Comandare il robot in questo modo è più complicato, va tenuto particolarmente conto del tempo di risposta di Tobor a comandi di questo tipo.
Abbiamo avuto dei problemi anche con i sonar: spesso restituiscono risultati fasulli ad esempio distanze superiori ai cinque metri. Questo per i rimbalzi che il segnale uscente dal sonar subisce prima eventualmente ritornare da dove è partito.

3. La soluzione adottata

Il primo banale provvedimento adottato per il riconoscimento delle lattine, è stato quello di acquisire sempre due volte un'immagine nel punto in cui è stata riconosciuta una lattina, allo scopo di evitare al robot il raggiungimento di una lattina inesistente qualora l'immagine risultasse troppo distorta.

Per quanto riguarda il riconoscimento del colore, sono stati adottati tre criteri:

1. livello di rosso, verde e blu presente in un colore; a tale scopo, un pixel, per essere riconosciuto come sopra soglia, cioè appartenente alla lattina desiderata, doveva avere valori RGB all'interno di intervalli prefissati (min_rosso-max_rosso, min_verde-max_verde, min_blu-max_blu; si tratta di variabili globali il cui valore viene assegnato di volta in volta al momento del caricamento dei parametri della lattina da trovare)
2. percentuale di rosso, verde e blu del pixel in esame; questo controllo è più raffinato e rende l'analisi indipendente dalla luminosità, permettendo di caratterizzare meglio l'oggetto richiesto (ad esempio, la coca-cola deve avere una minima percentuale di rosso piuttosto alta, indipendentemente dal fatto che il valore Red dei parametri RGB sia elevato o meno); le variabili globali min_per_rosso-max_per_rosso, min_per_verde-…,ecc. contengono le soglie consentite per la lattina prescelta
3. prodotto dei valori RGB; si è visto che a volte le lattine erano riconosciute meglio in base alla loro luminosità complessiva, cioè al prodotto dei loro valori RGB; la lattina di birra, ad esempio, era particolarmente luminosa e anche questa caratteristica doveva essere sfruttata per distinguerla dalle altre. (Le variabili globali utilizzate sono state min_prodotto e max_prodotto.)

Ognuno di questi tre controlli può essere abilitato o no secondo la lattina richiesta: alcune lattine più facili da riconoscere non li utilizzano tutti e tre. I controlli abilitati vengono messi in AND logico all'interno della funzione ApplicaSoglie.

3.1. Riconoscimento della birra

Fig 5: Birra e Schweppes accostate

L'immagine nella figura precedente mostra un limite fisico invalicabile: parecchi punti, più della metà, della lattina gialla (schweppes) sono bianchi quindi indistinguibili dalla lattina bianca di birra.
Questo è un caso che capita frequentemente: la lattina bianca è molto facile da riconoscere ma è altrettanto facile riconoscere come bianco un oggetto che non lo è. Il problema è stato risolto anzitutto applicando un filtro passabasso di potenza più elevata rispetto alle altre lattine, e poi aumentando il numero di punti minimi necessario affinché l'oggetto individuato venga riconosciuto come lattina.

Fig 6: La stessa immagine dopo la funzione ApplicaSoglie e il filtraggio passabasso

Fig 7: “Baricentro” ricavato

3.2. Acquisizione di una Coca-cola

Come si può notare, viene acquisito anche il rosso presente sulla lattina pepsi, senza però essere riconosciuto come lattina per via delle sue ridotte dimensioni.

Fig 8: Immagine iniziale

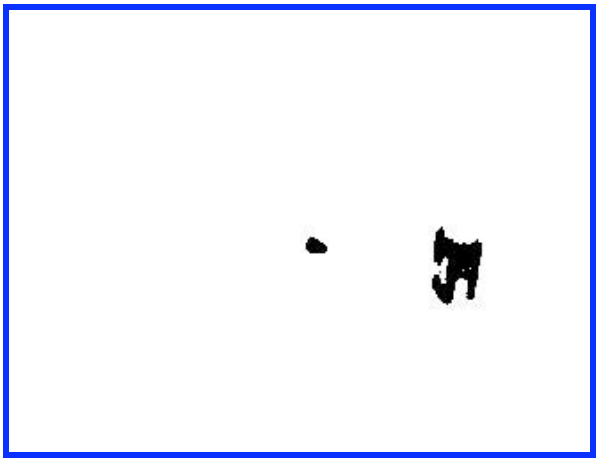


Fig 9: Punti sopra soglia

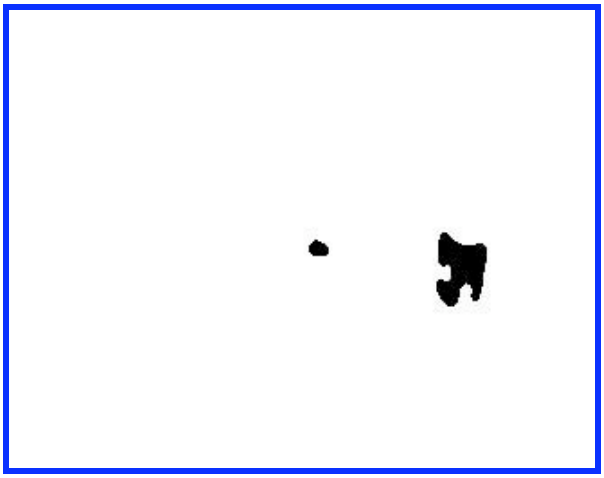


Fig 10: Applico i due filtri successivi passa basso

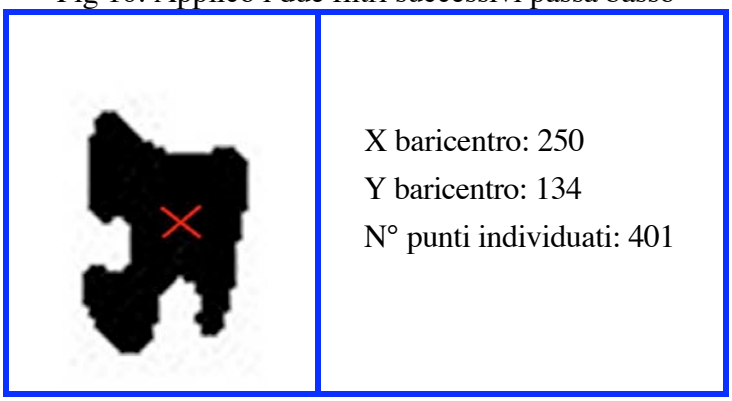


Fig 11: Risultato dell'elaborazione

3.3. Acquisizione di una Pepsi

Fig 12: Immagine acquisita

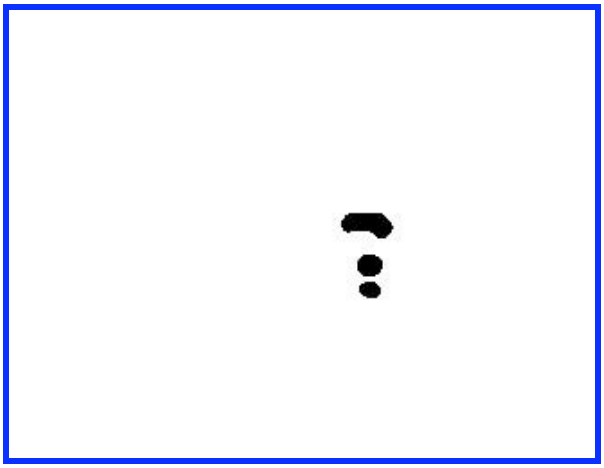


Fig 13: Immagine elaborata

3.4. Finestramento dell'immagine

Nel corso delle varie esperienze si è visto che talvolta lo sfondo costituiva un ostacolo, potendo causare falsi riconoscimenti poiché non uniforme. Pertanto si è deciso di finestrare l'elaborazione dell'immagine tagliando le parti troppo lontane e quelle troppo ai lati, migliorando anche l'efficienza dell'elaborazione stessa, che lavora con meno punti.

Fig 14: Immagine finestrata

3.5. Acquisizione di una Schweppes

Fig 15: Ricerca della Schweppes

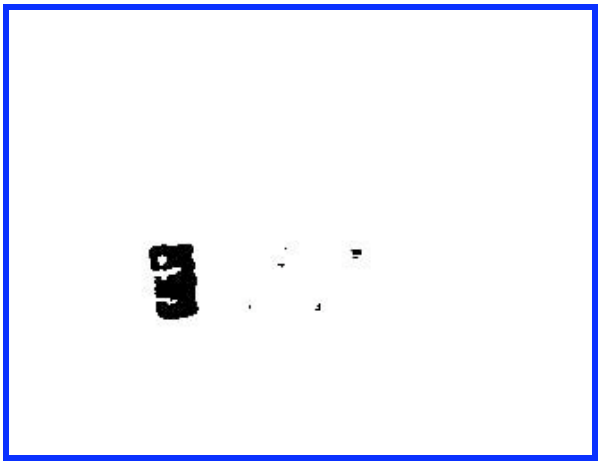


Fig 16: L'immagine dopo la funzione ApplicaSoglie

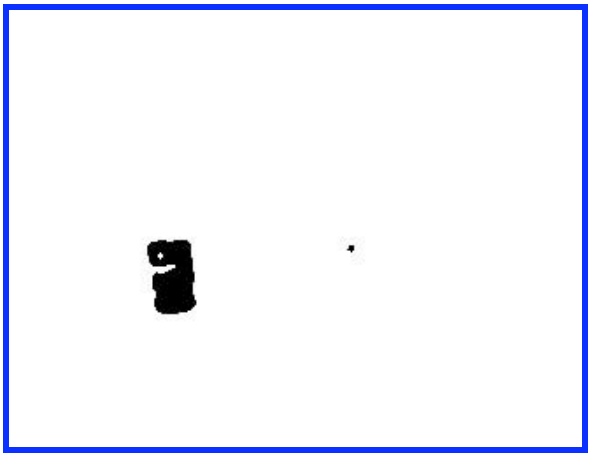


Fig 17: Dopo un passo di integrazione

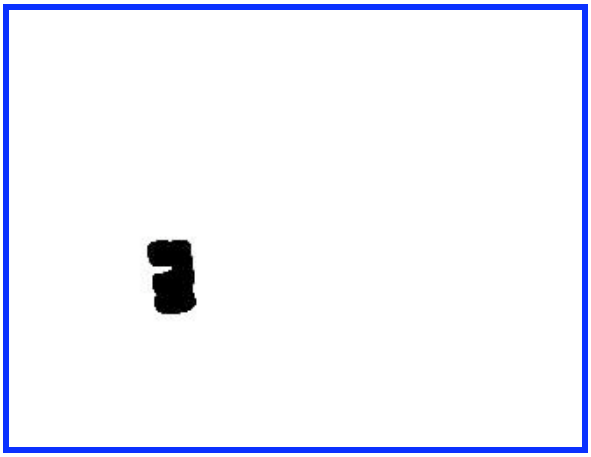


Fig 18: Dopo il secondo passo di integrazione

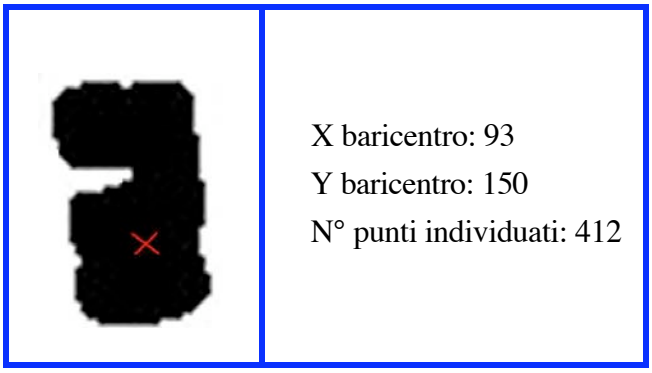


Fig 19: “Baricentro” individuato

3.6. Filtraggio passa basso

Presentiamo un'elaborazione particolarmente significativa che mette in luce l'utilità dei due filtri passa basso. Tobor deve afferrare la coca cola.

Fig 20: Immagine acquisita

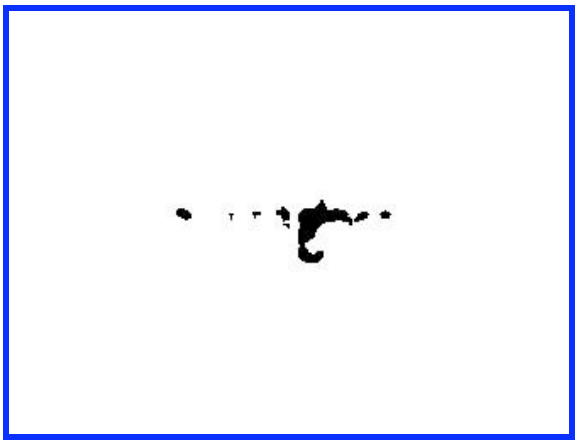
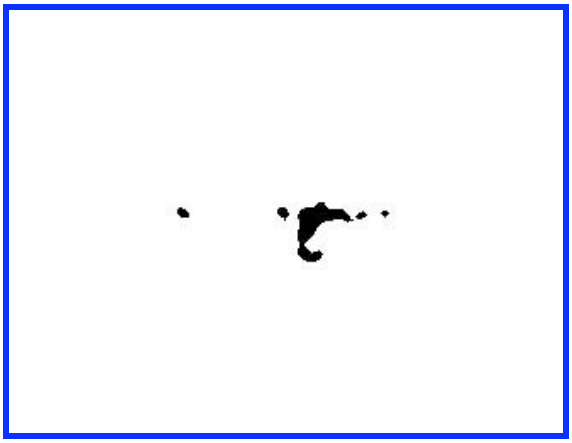


Fig 21: Punti sopra soglia[1]



[1] L'immagine contiene molti disturbi

Fig 22: Risultato dopo un primo filtraggio[2]

Fig 23: Immagine finale dopo il secondo filtraggio[3]

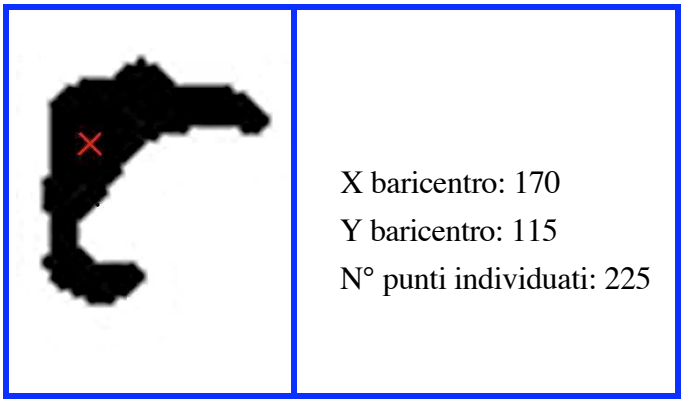


Fig 24: Coordinate del "Baricentro"

3.7. Disturbi

A volte è possibile che vengano riconosciute correttamente lattine anche alla presenza di immagini molto confuse, come in questo caso:

[2] Con il primo filtro attenuo i rumori
[3] Alcuni punti significativi isolati della coca cola a causa dei filtri sono stati eliminati, tuttavia ora i disturbi sono minimi: abbiamo raggiunto un compromesso accettabile.

Fig 25: Immagine acquisita non nitida

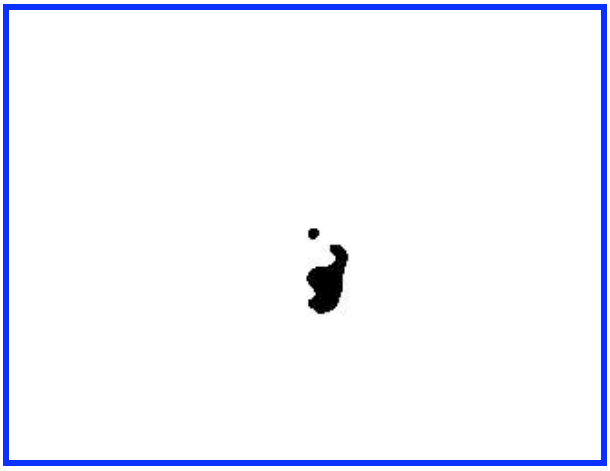


Fig 26: Riconoscimento corretto della lattina cercata, la Pepsi

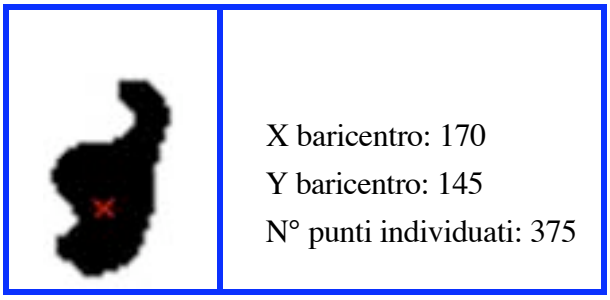


Fig 27: “Baricentro”

3.8. Ricerca del baricentro

Per la ricerca del baricentro, e quindi della posizione, della lattina, si utilizza un rettangolo di 17x25 pixel, approssimativamente la dimensione di una lattina vista alla massima distanza consentita alla telecamera. Quest'ultima viene posizionata su ogni

pixel dell'immagine già filtrata passabasso, e viene contato il numero di punti sopra soglia che cadono all'interno della sua area: quella che contiene più punti è quella che molto probabilmente contiene anche il baricentro[4], che viene quindi fissato a metà della finestra. La funzione che svolge questo è `TrovaBaricentro`. Per evitare riconoscimenti fasulli nel caso che nell'immagine non sia presente la lattina viene fissato un numero minimo di punti (tramite la variabile globale `NUMERO_MINIMO_DI_PUNTI`) consentiti per poter affermare che quella è effettivamente la lattina cercata. Comprensibilmente alle lattine più facili da identificare è stato assegnato un numero minimo di punti maggiore. Per rendere più leggera l'elaborazione è stata ideata anche una modalità di ricerca di tipo approssimato, che evita di far la somma dei punti coperti dalla finestra nel caso in cui il primo punto non sia sopra soglia; essa viene utilizzata quando il robot è in movimento.

3.9. Algoritmo di filtraggio

L'algoritmo viene applicato due volte, la prima con una potenza configurabile secondo il tipo di lattina in modo da minimizzare la perdita di punti significativi nel caso in cui la lattina sia difficile da individuare: bisogna filtrare poco se la lattina si vede male. La seconda passata invece può essere eseguita a potenza leggermente superiore, il cui valore è comune per tutte le lattine.

```
void FiltraPassaBasso ( void )
{   int x , y , i , j , xx , yy , somma;
      for (  x=0  ;  x<WIDTH  ;  x++)
        for (  y=0  ;  y<HEIGHT  ;  y++)
              mat2[ y ][ x ] = SOTTO_SOGLIA;
      for ( x = 2  ;  x < WIDTH – 2  ;  x++  )
        for (  y = SOGLIA_NERO  ;  y < HEIGHT – 2  ;  y++  )
              if (mat[y][x])
                  for ( i = y - 1 ; i <= y + 1 ; i++ )
                    for ( j = x - 1 ; j <= x + 1 ; j++ )
                        if ( mat2[i][j] == SOTTO_SOGLIA )
                          {   somma=0;     /* con -2 +2 prende 25 punti */
                              for (  yy = i – 2 ;  yy <= i + 2  ;  yy++ )
                                for (  xx = j – 2  ;  xx <= j + 2  ;   xx++ )
                                                  somma += mat[ yy ][ xx ];
                            somma/=25;
                             mat2[i][j]=(somma >=SOGLIA_PASSA_BASSO) ? SOPRA_SOGLIA : SOTTO_SOGLIA;
                          }
}
```

3.9.1. Spiegazione dettagliata

L'algoritmo di filtraggio passabasso è stato pensato per essere efficiente, nonostante sia composto da sei cicli FOR nidificati tra loro. Non vengono esaminati tutti i punti dell'immagine di partenza, ma solo due categorie di punti:

1. tutti i punti neri sopra soglia dell'immagine di partenza

[4] Se il baricentro non è stato trovato, le coordinate del baricentro vengono settate di default a (0,0)

2. tutti gli 8 punti intorno ai punti neri scovati al precedente passo

I due cicli for più esterni servono per scandire tutti i punti del passo 1, i due cicli for interni servono per completare il passo 2. Infine, un `IF` più interno serve per evitare di esaminare ulteriormente punti già analizzati.

A tutti e soli questi punti trovati si applica l'algoritmo classico di filtraggio passa-basso: per ogni punto considero la media del quadrato di 5x5 punti che ha come centro il punto in esame. Se la media supera SOGLIA_PASSA_BASSO (variabile a seconda del tipo di lattina) si crea un punto nero, cioè sopra soglia, nella nuova immagine altrimenti lo si lascia bianco.

Ad esempio, per la coca cola, SOGLIA_PASSA_BASSO è fissata al 33% mentre per la birra è fissata ad un valore più alto (56%), questo perché la birra viene vista meglio rispetto alla coca-cola: si può quindi filtrare con una potenza maggiore senza il rischio di perdere punti significativi.

3.10. Movimentazione del robot

Dopo vari tentativi si è ritenuto opportuno sovrapporre, accanto agli algoritmi di ricerca spaziale delle lattina, meccanismi di prevenzione degli urti forniti da saphira (e quindi già adeguatamente testati) come il behaviour `sfAvoidCollision`, che viene abilitato in parallelo all'avvio del programma.

La ricerca consiste nell'analisi a 360° dello spazio intorno al robot; quando viene individuata una lattina, il robot si ferma e riacquisisce ulteriormente l'immagine: se era effettivamente una lattina, viene invocata la funzione `Raggiungi` che approssima a grandi linee la posizione reale della lattina con l'altezza del baricentro trovato e sposta tobor verso di essa, infine completando i movimenti necessari: richiudere le pinze in prossimità della lattina, girarsi di 180° gradi per riportare la lattina, riaprire le pinze ed emettere una segnalazione sonora di successo.

Nel caso in cui la lattina non venga trovata durante il giro completo da fermo, il robot individua una posizione casuale, controlla tramite i sonar che la distanza dai muri sia accettabile (né troppo grande né troppo piccola) e si sposta di una quantità di spazio predefinita.

La distanza rilevata dai sonar non deve essere troppo piccola altrimenti significa che il suo "sguardo" è orientato verso un ostacolo o una parete. Non deve essere nemmeno troppo grossa altrimenti il valore del sonar non è significativo, probabilmente a causa di qualche rimbalzo, ed è meglio direzionarsi lungo un'altra diversa direzione casuale.

Una volta trovata una direzione ammissibile, il robot si sposta un poco in quella direzione, si ferma e ricomincia da capo la ricerca. Dopo un certo numero di questi tentativi, da noi fissato a 15, il robot si ferma e segnala che non sono presenti lattine del tipo richiesto.

4. Modalità operative

Per la realizzazione dell'elaborato è stato utilizzato il programma `grab.c`, il quale si occupa dell'elaborazione a basso livello delle immagini.
Le funzioni, scritte in linguaggio C, sono rese visibili a Saphira tramite la funzione `sfAddEvalFn`; avvenuta la compilazione, per avviare il programma, è sufficiente dare due comandi:

1. Caricamento dei parametri della lattina che si vuole ricercare; al momento sono disponibili quattro lattine diverse:

   → `coc()` per Coca-cola

   → `pep()` per Pepsi

   → `bir()` per birra Henninger

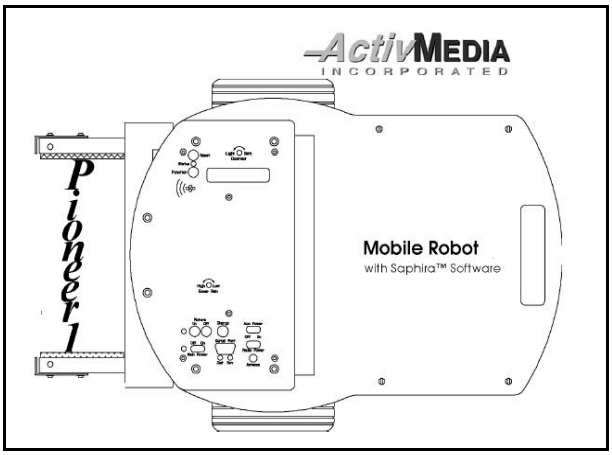
   → `sch()` per Schweppes

2. → `vai()` per far partire il programma.

E' disponibile inoltre un'ulteriore istruzione *salva(),* richiamabile direttamente da saphira, la quale salva su disco l'immagine acquisita dalla telecamera e le sue successive elaborazioni.

4.1. Componenti necessari

- Il robot Tobor, presente in laboratorio:



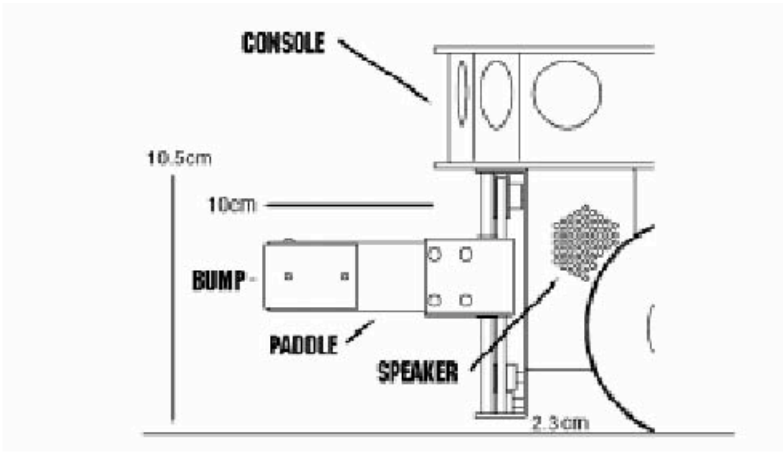


Figure 1-2 Pioneer Gripper side view.

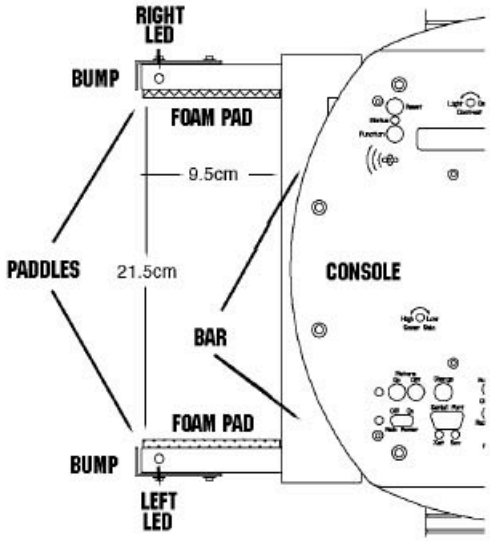


Figure 1-3. Pioneer Gripper top view

- Il pacchetto software Saphira.

4.2. Modalità di installazione

Si compila il file sorgente lattine.c con il makefile presente; viene generato il file lattine.so il quale viene poi caricato da Saphira. A questo punto ci si deve connettere al robot. Non devono essere caricati da Saphira nessun behavior di default, poiché il programma *lattine.so* contiene già al suo interno la primitiva di caricamento del behavior con gli opportuni parametri. Si consiglia, pertanto, di analizzare il file di configurazione di Saphira per rimuovere eventuali file di autoesecuzione contenenti behavior non desiderati.

4.3. Modalità di taratura

Modificando le funzioni void pepsi(void), void cocacola(void), void birra(void), void schweppes(void) è possibile correggere i parametri da noi settati per venire incontro a condizioni al contorno diverse.

5. Conclusioni e sviluppi futuri

A nostro avviso, il programma funziona particolarmente bene nella parte del riconoscimento delle lattine: aggiungendo funzioni simili a quelle elencate nel paragrafo 4.3, è possibile riconoscere altre lattine individuando i parametri opportuni mediante l'analisi di diverse acquisizioni con la telecamera. La parte di movimentazione può essere nettamente migliorata introducendo un minimo di intelligenza che controlli gli spostamenti del robot: ad esempio evitando di spostarsi in una direzione già esaminata. Volendo, per migliorare l'utilità pratica del programma, si potrebbero memorizzare le coordinate di partenza del robot, in modo tale che una volta afferrata la lattina desiderata, questa venga riportata indietro dal suo "padrone" che l'aveva richiesta.

Ritornando al problema principale che abbiamo nel nostro piccolo cercato di affrontare, vale la pena di spendere ancora due parole. Il programma funziona molto bene ma solo nel laboratorio dove è stato sviluppato, su di un tappeto verde circondato da cartoni neri! Già la presenza dei cartoni neri, che in realtà neri non lo sono affatto per la telecamera (anzi, quando illuminati contengono diversi punti vicini al bianco) non ci ha permesso di lavorare con la lattina del chinotto. I punti neri del cartone vengono spesso e volentieri visti come marrone assolutamente identico a quello presente sulla lattina, e non c'è modo di andare oltre basandosi sull'esame del solo colore. A questo punto bisogna lavorare sulla forma dei punti sopra-soglia acquisiti e solo irrobustendo gli algoritmi da questo punto di vista sarà possibile riconoscere anche lattine "scomode" oppure semplicemente far funzionare tobor non su di un bel tappeto verde, ma su un pavimento qualsiasi, magari rosso, lo stesso rosso della coca-cola. L'unico accorgimento del nostro programma verso questa direzione è stato quello di contare i punti sopra soglia solo all'interno di finestre di pixel di dimensioni uguali a quelle della generica lattina. E' già un qualcosa, perché permette di filtrare un disturbo orizzontale, ma un disturbo verticale grande quasi come una lattina porta ugualmente ad un falso riconoscimento. Lungo questa direzione si può lavorare ancora molto: nel nostro piccolo abbiamo solo toccato con mano uno dei più grandi problemi della robotica industriale: il riconoscimento di oggetti.

6. Complementi

Riportiamo di seguito le funzioni di saphira che abbiamo utilizzato, seguite da commenti sintetici e da esempi chiari in modo da ridurre il tempo necessario per una futura rielaborazione di tale progetto. I commenti non vogliono sostituire la documentazione originale, ma solo dare una spinta alla comprensione per futuri programmatori estranei alle fasi di progettazione che vogliono modificare gli algoritmi utilizzati od aggiungere funzioni al lavoro svolto.

- **void sfMessage (char *str)**
  void sfSMessage (char *str, ...)

`sfMessage` visualizza nella finestra principale di Saphira la stringa `str`.

`sfSMessage` permette come sfMessage di visualizzare a video la stringa str, inoltre mostra il contenuto delle variabili specificate.

Esempi: sfMessage("Non riesco ad aprire il file!");
sfSMessage("immagine %s salvata." , filename);

- **`int sfStartThread(void *fn, void *arg)`**
  `int sfTaskSuspended(char *iname)`
  `void sfResumeTask (char *iname)`
  `void sfRemoveTask (char *iname)`

`sfStartThread`: i threads rappresentano un modo per eseguire più task parallelamente; sono identificati da un unico valore non negativo restituito dalla funzione `sfStartThread` quando sono attivati. Se per qualche ragione il thread non può partire, per esempio a causa di risorse del sistema insufficienti, allora la funzione ritorna –1. `fn` è un puntatore alla funzione implementata in linguaggio C e `arg` è il nome che assegniamo a tale task.

`sfTaskSuspended`: disabilita temporaneamente l'istanza di un micro-task di nome `iname`. Questo micro-task può essere un'activity un behavior o semplicemente un micro-task.

`sfResumeTask`: riabilita l'istanza di un micro-task precedentemente sospeso.

`sfRemoveTask`: rimuove l'istanza del micro-task di nome `iname`.

Esempi: sfStartThread (avvicinati , "parti");
sfTaskSuspended("parti");
sfResumeTask("parti");
sfRemoveTask("parti");

- **`int sfSonarRange(int num)`**

`sfSonarRange`: restituisce la distanza rilevata espressa in mm dal sonar specificato come parametro da `num`. Tobor possiede 7 sonar, tra i quali:

- 0 indica quello più a sx
- 3 quello frontale
- 6 il sonar più a dx.

Esempio: sfSonarRange(3)

- **void sfSetPosition(int dist)**
  int sfDonePosition(int dist)
  void sfSetDHeading(int dhead)
  int sfDoneHeading(int ang)
  void sfPause(int ms)

`sfSetPosition`: Sposta il robot di una distanza pari a `dist` espressa in mm, tale parametro può essere positivo ed il robot avanza oppure negativo ed in tal caso indietreggia.

`SfDonePosition`: verifica se un precedente comando di spostamento traslazionale è terminato. Il parametro `dist` indica la tolleranza con la quale possiamo considerare l'operazione conclusa. Il parametro è espresso in mm e per un robot Pioneer è necessario indicare almeno 100 mm.

`sfSetDHeading`: Ruota il robot di un numero di gradi pari a `dhead` espresso in gradi sessagesimali, tale parametro può essere positivo ed il robot ruota in senso orario oppure negativo ed in tal caso il robot ruota in senso antiorario.

`sfDoneHeading`: verifica se un precedente comando di spostamento rotazionale è terminato. Il parametro `ang` indica la tolleranza con la quale possiamo considerare l'operazione conclusa. Il parametro è espresso in gradi e per un robot Pioneer è necessario indicare almeno 10 gradi.

`sfPause`: esprime una pausa nelle operazioni di `ms` millisecondi.

Esempi:
```
sfSetPosition ( 600+s/10 );
  while ( !sfDonePosition(100) ) sfPause ( 500 );

sfSetDHeading ( 25.0*atan( x1/y1 ) );
  while ( !sfDoneHeading(10) ) sfPause ( 100 );
```

Table 3-1. Gripper motor controls and state switches

Name	I/O port	Meanings
G.motor	OD0	rotate 1=down; 0=up
G.motor	OD1	1=enable
top	ID0	0= G at top
carry	ID1	0=paddles closed, G 4-5cm above floor
open	ID2	0=paddles fully open
inner b-b	ID3	1=object inside
outer b-b	ID4	1=object inside
bumps	ID5	0=paddle tip touching object

Il movimento della pinza del robot è generato da un unico motore pilotato attraverso 2 uscite digitali:

1. OD0 controlla la direzione di rotazione del motore
2. OD1 abilita e disabilita il motore

Le 3 funzioni seguenti permettono di far eseguire dei comandi al robot voluti dal programmatore:

- **void sfRobotComInt (int com, int arg)**
  void sfRobotComStrn (int com, char *str, int n)
  void sfRobotCom2Bytes(int com, int b1, int b2)

`sfRobotComInt: com` rappresenta il numero del comando che si vuol far eseguire al robot e `arg` è il parametro che indica la dimensione dell'operazione.

`SfRobotComStrn: com` è il numero del comando, `str` è una pascal-type string e `n` rappresenta la lunghezza della stringa.

`SfRobotCom2Byte: com` è il numero del comando, `b1` è una maschera che indica i bits che si vogliono modificare e `b2` rappresenta il valore che si vuole assegnare a questi bits.

Nome	Numero	Parametri	Valori	Descrizione
SfCOMVEL	**11**	**unsigned int**	**mm/sec [-400 , 400]**	**Setta la velocità di traslazione**
SfCOMRVEL	**21**	**unsigned int**	**gradi/sec [-200 , 200]**	**Setta la velocità di rotazione**
SfCOMDIGOU	**30**	**Integer**	**bits 0-7**	**Setta i bits delle uscite**

T				digitali
SfCOMSAY	16	unsigned int		Emette segnale acustico

```
sfRobotComInt(  sfCOMRVEL  , 80 );
sfRobotComInt( sfCOMVEL , 0 );
sfRobotComStrn( sfCOMSAY , "\010\002\010\000\010\002" , 6 );
                /* emette un segnale acustico
sfRobotCom2Bytes(  sfCOMDIGOUT , 0x3 , 0x2  );
/* rilascia la pinza:  OD0: 0 apre pinza
                       OD1: 1 motore che pilota la pinza abilitato*/
sfRobotCom2Bytes( sfCOMDIGOUT , 0x3 , 0x3 );
/* chiude la pinza:  OD0: 1 chiude pinza
                     OD1: 1 motore che pilota la pinza abilitato*/
```

Table 4-1. Pioneer I/O ports and connections

Label	Connector/Pin	Use
Digital Inputs		
ID0	G7	Gripper top switch
ID1	G9	Gripper carry switch
ID2	G11	Gripper open switch
ID3	G13	Paddle inner break beam
ID4	G15	Paddle outer break beam
ID5	N5	Paddle bump switches
ID6	N3	User push-button switch
ID7	N1	User slide switch
Digital Outputs		
OD0	G17	Gripper motor rotation direction
OD1	G19	Gripper motor enable
OD2	G21	User LED (amber)
OD3	G23	Right paddle and front LEDs
OD4	G25	Left paddle and front LEDs
OD5	N13	Speaker
OD6	N11	User servo motor logic
OD7	N9	User servo motor logic
Miscellaneous		
A/D	N7,G1	Analog-to-digital input
IDT	G3	Timer input
ODT	N15	Timer output
Logic Power		
Ground	N2,4,6,8,14,16; G2,4,10,12,14,16,18,20,22,24,26	
+5 VDC	N10,12; G6,8	

- **void sfStartBehavior(behavior *b, char *iname, int timeout, int priority,int suspended, ...)**

  `void sfTerminate(char *iname)`

`sfStartBehavior`: attiva il behavior specificato da `b`, puntatore alla struttura behavior. `Iname` è il nome dell'istanza del behavior.

`sfTerminate`: Disabilita l'istanza del behavior attivo

Esempi:

sfStartBehavior(sfAvoidCollision, `tipo di behavior`

"LattineB", `nome dell'istanza del behavior`

```
                    0,  timeout:  0 = no timeout
                    0,  priorità: 0 = massima priorità
                    0,  se 1 = parte in suspended state
                  1.5,  sensibilità frontale ( 0.5 → 3 )
                  0.5,  sensibilità laterale ( 0.5 → 3 )
                 10.0,  velocità con cui si allontana
                         dall'ostacolo(4→10)
flakey_radius+20.0,  distanza di sicurezza attorno al
                         robot in mm
      sfRIGHTTURN );  direzione predefinita di rotazione
```

sfTerminate("LattineB");

- **int sfAddEvalFn (char *name, void *fn, int rtype, int nargs, ...)**

`sfAddEvalFn` rende visibile al Colbert la funzione specificata come parametro.`name` è il nome della funzione visibile da Colbert e `fn` è un puntatore di funzione che punta alla funzione implementata in linguaggio C. Il tipo di valore restituito dalla funzione è `rtype`, e il numero di parametri da passare alla funzione è `nargs`. I successivi elementi sono i tipi di parametri della funzione.

OSS. Il tipo di valore restituito, `rtype`, is the C index of a Colbert type; i tipi predefiniti sono:

Type	C index
int	sfINT
float	sfFLOAT
void	sfVOID
string	sfSTRING
act	sfACTIVITY
behavior	sfBEHAVIOR
void *	sfPTR

Esempio: sfAddEvalFn("salva" , SalvaTutteLeImmagini , sfVOID , 0);

7. Listati

7.1. File principale: “Lattine.c”

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "grab.h"
#include "grab.c"
#include "saphira.h"

#define dim 230400    /*   > 320 * 240 * 3 <   */
#define WIDTH 320
#define HEIGHT 240

#define SOPRA_SOGLIA 255
#define SOTTO_SOGLIA 0
/* definisce la soglia del filtro passa basso all'ultimo passaggio da D a E */
#define ULTIMA_SOGLIA 113
/* range di valori accettabili dal sonar */
#define MASSIMO_SONAR 3500
#define MINIMO_SONAR   1000

#define TRUE 1
#define FALSE 0
/* Massimo valore contenuto nel file .ppm */
#define MAXVAL_PPM         255


/*  Numero di righe dello sfondo che non analizziamo */
#define SOGLIA_NERO    80
#define TRIANGOLO       100
#define LATERALE         25
```

```
#define BRIGHTNESS      128
#define HUE             128
#define CONTRAST        128
#define COLOR           128

#define ESATTO           1
#define APPROSSIMATO     0

/* massimi tentativi di ricerca */
#define MAX_TENTATIVI  15

#define OFFSET_RAGGIUNGIMENTO           180
#define COEFFICIENTE_RAGGIUNGIMENTO     6

unsigned char  mat[ HEIGHT ][ WIDTH ],
               mat2[ HEIGHT ][ WIDTH ];
/* Definizione dei prototipi di funzione */
void salvaimmagine ( unsigned char , char );   /** 1 **/
void FiltraPassaBasso ( void );                /** 2 **/
void SalvaTutteLeImmagini ( void );            /** 3 **/
void grab ( unsigned char , int , int  );      /** 4 **/
void TrovaBaricentro( int , int , int  );      /** 5 **/
void ApplicaSoglie ( unsigned char , int  );   /** 6 **/
void coordinate ( int , int , int  , int  );   /** 7 **/
void avvicinati1 ( void );                     /** 8 **/
void avvicinati ( void );                      /** 9 **/
int raggiungi ( void );                        /** 10 **/
EXPORT void sfLoadInit ( void );               /** 11 **/
void uscita ( void );                          /** 12 **/
void FermaRobot ( void );                      /** 13 **/
void pepsi ( void );                           /** 14 **/
void coccola ( void );                         /** 15 **/
```

void birra (void); **/** 16 **/**
void schweppes (void); **/** 17 **/**

```
/* Definizione delle variabili globali utilizzate */
int   min_rosso  =  0,     max_rosso  = 0,
      min_verde  =  0,     max_verde  = 0,
       min_blu   =  0,      max_blu   = 0,

   min_per_rosso = 0,    max_per_rosso  = 100,
   min_per_verde = 0,    max_per_verde = 100,
   min_per_blu   = 0,    max_per_blu   = 100,

   MINIMO_NUMERO_DI_PUNTI = 100,
   SOGLIA_PASSA_BASSO = 85,   /* 1/3 dei punti trovati deve essere sopra soglia */

   CONTROLLO_PERCENTUALE = FALSE,
   CONTROLLO_PRODOTTO = FALSE;

unsigned long min_prodotto = 0L  ,  max_prodotto = 17000000L;   /* 255^3  */
```

salvaimmagine : salva l'immagine in formato ppm (portable pixel map) su disco.
Parametri in ingresso: → *mem* : puntatore al primo primo byte dell'area di memoria riservata per memorizzare i colori (R,G,B) dei pixel dell'immagine
filename : nome del file che memorizzerà l'immagine su disco
Valore restituito: nessuno
Variabili: → *ppm* : puntatore di tipo file

void salvaimmagine (unsigned char mem , char filename) **/** 1 **/**

```
{
  FILE *ppm;
  if (  (ppm=fopen( filename , "wb" )  ) == NULL)
                                          {
```

```
                                        sfMessage( "Non riesco ad aprire il file!" );
                                        return;
                                        }
  fprintf( ppm , "P6\n" );
  fprintf( ppm , "%d %d\n %d\n" , WIDTH , HEIGHT , MAXVAL_PPM );
  fwrite( mem , dim , 1 , ppm );
  fclose( ppm );
  sfSMessage( "immagine %s salvata." ,  filename );
}
```

FiltraPassaBasso : costruisce la matrice mat2 filtraggio di mat

Parametri in ingresso: → nessuno

Valore restituito: → nessuno

Variabili: → *x , y , i , j , xx, yy*: indici

somma : memorizza la media dei colori dei punti contenuti nel quadrato 5x5

```
void FiltraPassaBasso ( void )   /** 2 **/
 {
  int x , y , i , j , xx , yy , somma;

  for (  x=0  ;  x<WIDTH  ;  x++)
     for (  y=0  ;  y<HEIGHT  ;  y++)
                  mat2[ y ][ x ] = SOTTO_SOGLIA;

  for ( x = 2  ;  x < WIDTH – 2  ;  x++  )
   for (  y = SOGLIA_NERO  ;  y < HEIGHT – 2  ;  y++  )

     if (mat[y][x])
       for ( i = y - 1 ; i <= y + 1 ; i++ )
         for ( j = x - 1 ; j <= x + 1 ; j++ )
           if ( mat2[i][j] == SOTTO_SOGLIA )
             {
               somma=0;      /* con -2 +2 prende 25 punti */
                for (  yy = i – 2 ;  yy <= i + 2  ;  yy++ )
```

```
                for (  xx = j – 2  ;  xx <= j + 2  ;   xx++ )
                   somma += mat[ yy ][ xx ];
             somma/=25;
             mat2[i][j]=(somma >=SOGLIA_PASSA_BASSO) ? SOPRA_SOGLIA
                                                     : SOTTO_SOGLIA;
          }
 }
```

SalvaTutteLeImmagini : richiama la funzione *coordinate* per salvare l'immagine acquisita dalla telecamera ("AAAAAAAA.ppm"). La funzione coordinate richiama a sua volta la funzione *ApplicaSoglie* per salvare l'immagine finestrata ("BBBBBB.ppm"). Successivamente SalvaTutteLeImmagini salva tre immagini su disco, ossia l'immagine contenente i punti sopra_soglia ("CCCCCCC.ppm"), l'immagine in uscita dal filto passa basso ("DDDDDDD.ppm") e l'ultima ottenuta in uscita ad un secondo filtro passa basso ("EEEEEEE.ppm").

Parametri in ingresso: → nessuno

Valore restituito: → nessuno

Variabili: → *buffer* : puntatore al primo primo byte dell'area di memoria riservata per memorizzare i colori (R,G,B) dei i pixel dell'immagine

i , *j* : indici

a , *b* : variabili di supporto necessarie per poter richiamare la funzione coordinate

```
void SalvaTutteLeImmagini ( void )    /** 3 **/
{
 unsigned char *buffer;
 int i , j , a , b;

 coordinate( &a , &b , TRUE , ESATTO );
 buffer = malloc(  WIDTH * HEIGHT * 3  );
 for (  i=0 ; i<WIDTH ; i++  )
  for (  j=0 ; j<HEIGHT ; j++  ){   *(buffer+ (i+j*WIDTH)*3 )      =   255-mat[j][i];
                                    *(buffer+ (i+j*WIDTH)*3+1 )   =   255-mat[j][i];
```

```
                                *(buffer+ (i+j*WIDTH)*3+2 )  =  255-mat[j][i];     }

  salvaimmagine(  buffer , "CCCCCCC.ppm"  );
  free( buffer );
  buffer = malloc(  WIDTH * HEIGHT * 3 );
  FiltraPassaBasso();
  for (  i=0 ; i<WIDTH ; i++  )
    for (  j=0 ; j<HEIGHT ; j++  ){  *(buffer+ (i+j*WIDTH)*3 )      =  255-mat2[j][i];
                                *(buffer+ (i+j*WIDTH)*3+1 )  =  255-mat2[j][i];
                                 *(buffer+  (i+j*WIDTH)*3+2  )  =  255-mat2[j][i];
}
  salvaimmagine( buffer , "DDDDDDD.ppm" );
  for (  i=0 ; i<WIDTH ; i++  )              /* ora copio mat2 in mat */
    for (  j=0 ; j<HEIGHT ; j++  )
                        mat[j][i] = mat2[j][i];

  SOGLIA_PASSA_BASSO=ULTIMA_SOGLIA;
  FiltraPassaBasso();
   for ( i=0 ; i<WIDTH ; i++ )
     for ( j=0 ; j<HEIGHT ; j++ ){   *(buffer+ (i+j*WIDTH)*3 )      =  255-mat2[j][i];
                                *(buffer+ (i+j*WIDTH)*3+1 )  =  255-mat2[j][i];
                                *(buffer+ (i+j*WIDTH)*3+2 )  =  255-mat2[j][i];   }
  salvaimmagine(  buffer , "EEEEEEE.ppm"  );
  free( buffer );
}
```

grab : acquisisce l'immagine a basso livello tramite il frane grabber

Parametri in ingresso: → *cop* : un puntatore a un area di memoria
lum : luminosità
contr : contrasto

Valore restituito: → nessuno
Variabili: → *i , j* : indici
device : dispositivo di acquisizione
channel : canale del dispositivo 1 composito, 2 Svideo
format : formato dell'immagine (GRAB_MODE_PAL op GRAB_MODE_NTSC)
bpp : profondità di colore
p : un puntatore a un area di memoria.

```
void grab ( unsigned char cop , int lum , int contr )   /** 4 **/
{
 int device = 0,
     channel = 1,
     format = GRAB_MODE_NTSC,
     bpp = 24,
     i,j;
 unsigned char *p;

 grab_open(  WIDTH , HEIGHT , device , channel , format , bpp );  /* Apertura del frame grabber */

 grab_set_attr( lum , HUE , contr , COLOR  );   /* Setta i parametri di acquisizione della telecamera. */

 p = grab_capture();                              /* Acquisizione immagine */

 if( p == NULL ){   sfMessage( "Buffer dell'immagine non valido." );
                    grab_close();
                    uscita();
                    exit(0);      }
```

```
 for( i=0 ; i<HEIGHT ; i++ )
   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 */
}
```

TrovaBaricentro : effettua i due filtraggi dell'immagine acquisita e finestrata successivamente calcola l'eventuale baricentro della lattina individuata.

Parametri in ingresso: → *bar_x , bar_y* : coordinate del baricentro
precisione : ESATTO : calcola le coordinate del baricentro in maniera esatta, viceversa assume valore APPROSSIMATO.

Valore restituito: → nessuno

Variabili: → *x , y, i , j* : indici
cont : n° di punti sopra_soglia individuati in ciascuna finestra 17x25
cont_max: max valore di cont
DIM_FINESTRA_X , DIM_FINESTRA_Y : dimensioni della finestra utilizzata per il riconoscimento del baricentro dell'oggetto

void TrovaBaricentro(int bar_x , int bar_y , int precisione) / 5 **/**

```
{
int cont , cont_max,
       x , y,
        i , j;

const int    DIM_FINESTRA_X = 17,
             DIM_FINESTRA_Y = 25;

 *bar_x = *bar_y = 0;
```

```
 /* minimo numero di punti affinché venga riconosciuta una lattina */
cont_max = MINIMO_NUMERO_DI_PUNTI;
 FiltraPassaBasso();

 for (  i=0 ; i<WIDTH ; i++  )          /* copio mat2 in mat */
  for (  j=0 ; j<HEIGHT ; j++  )     mat[ j ][ i ] = mat2[ j ][ i ];

 SOGLIA_PASSA_BASSO = ULTIMA_SOGLIA;
 FiltraPassaBasso();

for( j=SOGLIA_NERO ;  j<HEIGHT-DIM_FINESTRA_Y ;  j++ )

 for(  i=LATERALE ;  i<WIDTH-DIM_FINESTRA_X-LATERALE  ;  i++ )

 {   /*  se voglio una misura approssimata del baricentro, scarto a priori
        tutte le finestre il cui primo punto non è sopra soglia */

      if (  precisione == APPROSSIMATO && mat2[ j ][ i ] == SOTTO_SOGLIA  )
            continue;
      cont = 0;
      for ( x=0 ;  x<DIM_FINESTRA_X ; x++ )
       for ( y=0 ;  y<DIM_FINESTRA_Y ; y++ )
          if (  mat2[ j+y ][ i+x ] == SOPRA_SOGLIA  ) cont++;
          if (  cont >= cont_max  ) {  cont_max = cont;
                                       *bar_x = i+DIM_FINESTRA_X/2;
                                       *bar_y = j+DIM_FINESTRA_Y/2;    }
  }
  sfSMessage( "cont ---> %d bar_x ---> %d bar_y ---> %d" ,  cont_max  , *bar_x  ,
*bar_y  );

}
```

ApplicaSoglie : assegna i valori alla matrice mat, se sono sopra soglia o sotto soglia
passaggio dall'immagine di partenza B a quella in bianco e nero C

Parametri in ingresso: → *mem* : un puntatore a un area di memoria
SALVA : luminosità

Valore restituito: → nessuno

Variabili: → *x ,y* : indici
per_rosso , per_verde , per_blu : percentuale di rosso, verde e blu contenute nel punto sotto esame
rosso , verde , blu : colori dei punti dell'immagine finestrata
OK : TRUE se supera la soglia del colore e FALSE se il punto non supera un ulteriore controllo richiesto
somma : rosso + verde + blu
buffer,inizio : puntatori a un area di memoria.

```
void ApplicaSoglie( unsigned char mem , int SALVA )      /** 6 **/
{
  int  x , y ,
       per_rosso , per_verde , per_blu ,
       rosso , verde , blu ,
       OK;
  double somma;
  unsigned char *buffer , *inizio;


  if ( SALVA ) { inizio = buffer = malloc(  WIDTH * HEIGHT * 3  );

                    if (  inizio == NULL  ) {  sfMessage( "Memoria insufficiente" );
                                               uscita();
                                               exit(0);          }
                 }

 for( y=0 ; y<HEIGHT ;  y++ )
  for( x=0 ; x<WIDTH ;  x++ )
   {
    OK = FALSE;
    rosso = verde = blu = 0;
```

```
if (   y > SOGLIA_NERO   &&    y < x + 240 – TRIANGOLO   &&
       y < 320 - x  + 240 - TRIANGOLO &&     x > LATERALE  &&
       x < ( 320- LATERALE)    )
                              {
                                /* se sono nella zona permessa, guardo i colori */
                                rosso =  *(mem+( y *WIDTH +x  )*3  );
                                verde =  *(mem+( y *WIDTH +x  )*3+1 );
                                blu  =  *(mem+( y *WIDTH +x  )*3+2 );      }

if ( SALVA ){   *(buffer)     = rosso;
                *(buffer+1) = verde;
                *(buffer+2) = blu;
                 buffer += 3;             }

if (   rosso >= min_rosso  &&  rosso <= max_rosso   &&
       verde >= min_verde  &&  verde <= max_verde  &&
        blu   >= min_blu     &&   blu    <= max_blu              )

            {  /* ho passato la soglia */

      OK = TRUE;
             if ( CONTROLLO_PRODOTTO )
                 if (   rosso*verde*blu  >  max_prodotto   ||
                           rosso*verde*blu  <  min_prodotto        )    OK = FALSE;

             if (  CONTROLLO_PERCENTUALE  )
                   {
                    somma = (double) (rosso+verde+blu );
                    per_rosso  = (  (double)rosso/somma ) *100;
                    per_verde  = (  (double)verde/somma ) *100;
                    per_blu      = (  (double)blu/soma ) *100;

                    if (  per_blu      <  min_per_blu       ||
                         per_blu     >  max_per_blu       ||
                         per_rosso  <  min_per_rosso    ||
                         per_rosso  >  max_per_rosso   ||
                        per_verde  <  min_per_verde  ||
```

```
                    per_verde  >  max_per_verde      )
                                                    OK = FALSE;
                }
            }  /* chiude: ho passato la soglia  */

        mat[y][x] = (  OK == TRUE  )  ?  SOPRA_SOGLIA  :  SOTTO_SOGLIA;
    }

 if ( SALVA ) {     salvaimmagine( inizio , "BBBBBB.ppm" );
                    sfMessage( "Ho salvato l'immagine finestrata \n" );
                    free( inizio );                                    }
}
```

coordinate : acquisisce l'immagine (se specificato la salva su disco), applica le soglia e trova il baricentro

Parametri in ingresso: → *bar_x* , *bar_y* : coordinate del baricentro
SALVA : se TRUE salva l'immagine acquisita dalla telecamera
precisione : indica il metodo con cui individuare il baricentro; può assumere due valori ESATTO ed APPROSSIMATO

Valore restituito: → nessuno

Variabili: → *inizio* : puntatori a un area di memoria.

void coordinate (int bar_x , int bar_y , int SALVA , int precisione) / 7 **/**

```
{
unsigned char *originale;

 originale = ( unsigned char* ) malloc( dim );
 if (  originale == NULL  ){     sfMessage( "Memoria insufficiente" );
                                 exit(0);                              }
```

```
grab(  originale , BRIGHTNESS , CONTRAST  );
ApplicaSoglie(  originale , SALVA  );

if ( SALVA ) salvaimmagine(  originale , "AAAAAAAA.ppm"  );

TrovaBaricentro(  bar_x  ,  bar_y  ,  precisione  );

free( originale );
}
```

avvicinati1 : attiva il thread avvicinati che chiameremo "parti".

Parametri in ingresso: → nessuno
Valore restituito: → nessuno
Variabili: → nessune

```
void avvicinati1 ( void )        /** 8 **/
 {
  sfStartThread( avvicinati , "parti" );
 }
```

avvicinati : gestisce le operazioni necessarie all'individuazione e all'avvicinamento della lattina desiderata. Richiama infine la funzione *raggiungi* per afferrarla

Parametri in ingresso: nessuno
Valore restituito: → nessuno
Variabili: → *i , j :* indici
orario : indica il senso di rotazione del robot
bar_x , bar_y : coordinate del baricentro
iterazioni : contatore
timeout : n° di iterazioni necessarie per far ruotare il robot di un giro completo
s1 , s2 : memorizzano due letture successive del sonar frontale

s : media di s1 e s2
tentativi : n° di tentativi effettuati alla ricerca della lattina

```
void avvicinati ( void )          /** 9 **/
{
int   i , j , orario,
      bar_x , bar_y,
      iterazioni , timeout,
      s1 , s2 , s,
      tentativi=0;

random();                  /* inizializza i numeri casuali */
sfRobotCom2Bytes(  sfCOMDIGOUT , 0x3 , 0x2  );  /* rilascia pinza */

while (  tentativi < MAX_TENTATIVI ) {      /* ciclo principale */
 tentativi++;
/* acquisisce il frame e calcola il  baricentro */
 coordinate(  &bar_x  ,  &bar_y ,  FALSE  ,  APPROSSIMATO  );


 /* per sicurezza acquisisco l'immagine ulteriormente */
 if ( bar_x==0 && bar_y==0 )
              coordinate (  &bar_x ,  &bar_y  ,  FALSE , APPROSSIMATO  );

 if (  bar_x  ||  bar_y  ) if ( raggiungi() ) {     /* se lo ha preso */
                                                uscita();
                                                return;              }

 if (   sfSonarRange( 0 )  >= sfSonarRange( 6 )   ) orario = TRUE;
              else orario = FALSE;

 sfSMessage( "Faccio partire il robot" );
```

```
sfRobotComInt( sfCOMRVEL , (orario) ? 80 : -80 );
iterazioni = 0;
timeout = 40;        /* un giro */
   do{
      iterazioni++;
      coordinate( &bar_x , &bar_y , FALSE , APPROSSIMATO );

      if ( bar_x || bar_y ){
                              sfMessage( "HO TROVATO L'OGGETTO" );
                              if ( raggiungi() ) {  /* se lo ha preso */
                                                    uscita();
                                                     return;                }
                              sfRobotComInt( sfCOMRVEL , (orario) ? 80 : -80 );
                              }
      sfSMessage( "iterazione n:%d" , iterazioni );
      }
      while ( iterazioni < timeout );

 FermaRobot();

 /* ho fatto tutto un giro, adesso mi sposto in una direzione casuale */
 sfMessage( "ho fatto un giro completo" );

do {

   if ( sfSonarRange(0) >= sfSonarRange(6) ) orario = TRUE;  else orario = FALSE;

   sfTaskSuspended( "parti" );

   sfRobotComInt( sfCOMRVEL , (orario) ? 90 : -90 );

   sfSMessage( "mi sposto in direzione casuale" );

   sfPause( 1100+( rand() % 3300 ) );

   sfRobotComInt( sfCOMRVEL , 0 );
```

```
    sfResumeTask( "parti" );

    sfSMessage( "fine direzione casuale" );

    s1 = sfSonarRange( 3 );
    s2 = sfSonarRange( 3 );
    s = ( s1+s2 ) / 2;     /* calcolo la media di due rilevazioni del sonar per diminuire
                              l'errore dovuto ad una acquisizione del sonar sbagliata */

    sfSMessage( "distanza trovata %d %d %d" , s1 , s2 , s );

    if (  s > MASSIMO_SONAR  ) sfMessage( "direzione sbagliata. riprovo!" );
    if (  s < MINIMO_SONAR   ) sfMessage( "troppo vicino alla parete. provo un altra
direzione" );

   } while (  s > MASSIMO_SONAR  ||  s < MINIMO_SONAR  );

 sfMessage( "direzione esatta. mi sposto." );

 sfTaskSuspended( "parti" );

 sfSetPosition( 600+s/10 );
 while (  !sfDonePosition(100)  ) sfPause ( 500 );

 sfResumeTask( "parti" );
 FermaRobot();
 }  /* fine ciclo principale */

 sfMessage( "Ho esaurito i tentativi e non ho trovato nulla!" );
 uscita();
 return;
}
```

raggiungi : viene richiamata ogni qualvolta si rilevata la lattina desiderata.

Parametri in ingresso: → nessuno

Valore restituito: → TRUE : se il robot ha preso la lattina
FALSE : se ad una analisi più accurata si conclude che l'oggetto non è la lattina cercata.

Variabili: → *x1* : punti mancanti rispetto al centro in orizzontale
y1 : punti mancanti rispetto al fondo in verticale
bar_x , bar_y : coordinate del baricentro

```
int raggiungi ( void )   /** 10 **/
{
  float x1,y1;
  int bar_x,bar_y;
  FermaRobot();
  coordinate( &bar_x  , &bar_y , FALSE ,ESATTO );
  if ( !bar_x  &&  !bar_y )
              {  sfMessage( "Ho ricontrollato in modo esatto e non era un immagine" );
                 return FALSE;    }
  SalvaTutteLeImmagini();
  sfTaskSuspended( "parti" );
  x1= WIDTH/2 - bar_x;
  y1= HEIGHT - bar_y;
  sfSetDHeading( 25.0*atan( x1/y1 )  );
  while (  !sfDoneHeading(10)  ) sfPause ( 100 );
  sfPause( 1000 );
  sfSetPosition( OFFSET_RAGGIUNGIMENTO  +  y1  *
                        COEFFICIENTE_RAGGIUNGIMENTO );

  while (  !sfDonePosition(100)  ) sfPause ( 500 );
  sfPause( 1100 );
  sfRobotCom2Bytes( sfCOMDIGOUT , 0x3 , 0x3 );   /* chiude  la pinza */
  sfPause(3000);
  sfSetPosition( -400 );
  while (  !sfDonePosition(100)  ) sfPause ( 500 );
```

```
  sfPause( 1000 );

 /* destra >= sinistra */
  if (  sfSonarRange(0) >= sfSonarRange(6)  ) sfSetDHeading( 180 );
  else sfSetDHeading( -180 );

  while ( !sfDoneHeading(10) ) sfPause ( 100 );
   sfSMessage( "X: %d  Y:%d  TH:%d   ax: %d" ,(int)(sfRobot.ax),(int)(sfRobot.ay)
                                                ,(int)(sfRobot.th),(int)(sfRobot.x) ) ;
  sfPause( 1000 );
  sfRobotCom2Bytes( sfCOMDIGOUT , 0x3 , 0x2 );   /* rilascia la pinza */
   /* emette un segnale acustico */
 sfRobotComStrn( sfCOMSAY , "\010\002\010\000\010\002" , 6 );
 return TRUE;            /* ho preso l'oggetto */
}
```

sfLoadInit : attiva il behavior sfAvoidCollision e i task "salva","vai","pep","sch","coc"e "bir".

Parametri in ingresso: → nessuno

Valore restituito: → nessuno

Variabili: → nessune

EXPORT void sfLoadInit (void) / 11 **/**

```
{
sfStartBehavior(sfAvoidCollision,"LattineB",0,0,0,1.5,0.5,10.0,
flakey_radius+20.0,sfRIGHTTURN );
sfAddEvalFn( "salva" , SalvaTutteLeImmagini , sfVOID , 0 );
sfAddEvalFn( "vai"    ,      avvicinati1        , sfVOID , 0 );
sfAddEvalFn( "pep"    ,        pepsi             , sfVOID , 0 );
sfAddEvalFn( "sch"    ,      schweppes           , sfVOID , 0 );
sfAddEvalFn( "coc"    ,      cocacola            , sfVOID , 0 );
sfAddEvalFn(  "bir"   ,        birra             , sfVOID , 0 );
}
```

uscita : disabiltita il behavior attivo e i task “parti”,”evita” e “Stop”. Infine ferma sia il movimento traslatorio che rotazionale del robot

Parametri in ingresso: → nessuno

Valore restituito: → nessuno

Variabili: → nessune

```
void uscita ( void )      /** 12 **/
 {
  sfTerminate( "LattineB" );
  sfRemoveTask( "parti" );
  sfRemoveTask( "evita" );
  sfRemoveTask( "Stop" );
  sfRobotComInt( sfCOMVEL , 0 );
  sfRobotComInt( sfCOMRVEL , 0 );
 }
```

FermaRobot : ferma sia il movimento traslatorio che rotazionale del robot.

Parametri in ingresso: → nessuno

Valore restituito: → nessuno

Variabili: → nessune

```
void FermaRobot ( void )      /** 13 **/
{
  sfRobotComInt( sfCOMVEL , 0 );
  sfRobotComInt( sfCOMRVEL , 0 );
  sfPause( 400 );
}
```

pepsi : assegna i valori alle variabili globali necessari per riconoscere le lattine della pepsi.

Parametri in ingresso: → nessuno

Valore restituito: → nessuno

Variabili: → nessune

```
void pepsi ( void )      /** 14 **/
```

```
{
 min_rosso =  30  ; max_rosso  = 180;
 min_verde = 50  ; max_verde = 180;
 min_blu    = 85  ; max_blu    = 255;
 sfMessage( "voglio la pepsi!" );
 MINIMO_NUMERO_DI_PUNTI = 80;
 SOGLIA_PASSA_BASSO = 85;
CONTROLLO_PRODOTTO = FALSE;
CONTROLLO_PERCENTUALE = TRUE;
 min_per_rosso  =  0;   max_per_rosso  =  100;
 min_per_verde =  0;   max_per_verde =  33;
 min_per_blu    = 38;  max_per_blu   =  100;
}
```

cocacola : assegna i valori alle variabili globali necessari per riconoscere le lattine della cocacola.

Parametri in ingresso: → nessuno

Valore restituito: → nessuno

Variabili: → nessune

void cocacola (void) / 15 **/**

```
{
 min_rosso = 150 ; max_rosso =  255;
 min_verde = 60   ; max_verde = 180;
 min_blu    = 60   ; max_blu    = 170;
 sfMessage( "voglio la coca-cola!" );
 MINIMO_NUMERO_DI_PUNTI=80;
 SOGLIA_PASSA_BASSO=85;
 CONTROLLO_PRODOTTO=TRUE;
  min_prodotto=0L;
  max_prodotto=8500000L;
 CONTROLLO_PERCENTUALE=TRUE;
  min_per_rosso  = 40;   max_per_rosso  = 100;
  min_per_verde  = 0;     max_per_verde  = 32;
  min_per_blu    = 0;      max_per_blu   = 100;
}
```

birra : assegna i valori alle variabili globali necessari per riconoscere le lattine della birra "henninger".

Parametri in ingresso: → nessuno

Valore restituito: → nessuno

Variabili: → nessune

```
void birra ( void )     /** 16 **/
{
 min_rosso = 245 ; max_rosso = 255;
 min_verde = 245 ; max_verde = 255;
 min_blu   = 245 ; max_blu   = 255;
 sfMessage( "voglio la birra!" );
 MINIMO_NUMERO_DI_PUNTI = 380;
 SOGLIA_PASSA_BASSO = 142;
 CONTROLLO_PRODOTTO = TRUE;
 min_prodotto = 14000000L;
 max_prodotto = 16600000L;
 CONTROLLO_PERCENTUALE = TRUE;
 min_per_rosso = 31;   max_per_rosso = 35;
 min_per_verde = 31;   max_per_verde = 35;
 min_per_blu   = 31;   max_per_blu  = 35;
}
```

schweppes : assegna i valori alle variabili globali necessari per riconoscere le lattine della schweppes.

Parametri in ingresso: → nessuno

Valore restituito: → nessuno

Variabili: → nessune

```
void schweppes ( void )    /** 17 **/
{
 min_rosso = 200 ; max_rosso = 255;
 min_verde = 140 ; max_verde = 255;
 min_blu   =  0  ; max_blu   = 190;
 sfMessage( "voglio la Schweppes!" );
```

```
 MINIMO_NUMERO_DI_PUNTI = 90;
 CONTROLLO_PRODOTTO = TRUE;
 min_prodotto = 6900000L;
 max_prodotto = 17000000L;
 CONTROLLO_PERCENTUALE = FALSE;
 SOGLIA_PASSA_BASSO = 85;
}
```

7.2. File di supporto

7.2.1. Grab.h

```
/* grab.h - Version 0.1
   definizione delle funzioni per il grabbing di immagini
   (c) 2000 Stefano Inelli, Alessandro Bonometti, Danilo Duina   */

#ifndef GRAB_H
#define GRAB_H
#define GRAB_MODE_PAL 0
#define GRAB_MODE_NTSC 1

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

 /* Apertura del frame grabber
    Parametri: width, height: dimensioni dell'immagine da acquisire (Es. 320,240)
               device: dispositivo di acquisizione (0) (NON IMPLEMENTATO)
               channel: canale del dispositivo 1 composito, 2 SVideo
               format: formato dell'immagine (GRAB_MODE_PAL op
                                              GRAB_MODE_NTSC)
               bpp: profondità di colore (24) (NON IMPLEMENTATO)
    Valore di ritorno: 0 apertura effettuata correttamente  */

/*** A ***/
 int grab_open( int width , int height , int device , int channel , int
format, int bpp )

 /* Acquisizione dell'immagine
    Restituisce un puntatore a un area di memoria di dimensioni width*height*3
   Valore di ritorno: NULL capture NON effettuata correttamente  */
 unsigned char* grab_capture( void )     /*** B ***/

 /* Chiusura del frame grabber
```

```
   Valore di ritorno: 0 chiusura effettuata correttamente  */
int grab_close( void )   /*** C ***/

/* Permette di impostare e leggere i parametri indicati
   RANGE: 0 - 255
   Valore di ritorno: 0 impostazione effettuata correttamente  */
int grab_set_attr( int brightness , int hue ,  int contrast , int colour )
/*** D ***/

/*** E ***/
int grab_get_attr( int *brightness ,  int *hue ,  int *contrast ,
                   int *colour )

/* Permette di scrivere su filename.ppm il contenuto del buffer   */
int grab_write_ppm( char *filename )       /*** F ***/

/* Funzioni utilizzate per il debug
   Restituiscono le impostazioni delle strutture video4linux   */
int grab_get_vcap( void )      /*** G ***/
int grab_get_vp( void )        /*** H ***/

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

7.2.2. Grab.c

```
/* grab.c - Version 0.1
   Implementazione delle funzioni della libgrab
  (c) 2000 Stefano Inelli, Alessandro Bonometti, Danilo Duina   */

#include <stdio.h>
```

```
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <sys/mman.h>
#include <linux/videodev.h>
#include "grab.h"

#define GRAB_ERR_CHANNEL 1
#define GRAB_ERR_ISCLOSED 2
#define GRAB_ERR_ISOPEN 3
#define GRAB_ERR_GPICT 4
#define GRAB_ERR_SPICT 4
#define V4L_DEVICE_0 "/dev/video0"
#define V4L_DEVICE_1 "/dev/video1"
#define V4L_DEVICE_2 "/dev/video2"
#define V4L_DEVICE_3 "/dev/video3"

static struct video_capability vcap;
static struct video_channel    vc;
static struct video_picture    vp;
static struct video_mmap       mm;

static int grab_width;
static int grab_height;

static int fd;
static unsigned char* buf;
static int grabber_isopen = 0;
/*** A ***/
int grab_open( int width , int height , int device , int channel ,
                    int format, int bpp )
{
  int i;

  if(  grabber_isopen  ) {   perror( "Grabber is already OPEN" );
```

```
                                    return GRAB_ERR_ISOPEN;          }
grab_width = width;
grab_height = height;
/* Apertura del dispositivo in lettura scrittura  */
fd = open( V4L_DEVICE_0 , O_RDWR  );
if(  fd <= 0  ) {  perror( "open" );
                   return 10;           }

/* Acquisizione Video capability del dispositivo  */
if(  ioctl( fd , VIDIOCGCAP ,  &vcap ) < 0  ) {  perror("VIDIOCGCAP");
                                                 return 10;     }

fprintf(  stderr , "Video Capture Device Name : %s\n" , vcap.name );
/* Controllo dei canali disponibili  */
for(  i=0 ; i<vcap.channels ; i++  )
  {
   vc.channel = i;
   if(  ioctl( fd , VIDIOCGCHAN , &vc  ) <0 ) {  perror("VIDIOCGCHAN");
                                                 return 10;        }
   fprintf(  stderr , "Video Source (%d) Name : %s\n" , i , vc.name  );
  }

/* Setting del buffer di mmap  */
buf    =    (unsigned    char*)    mmap(0    ,    grab_height*grab_width*3    ,
PROT_READ|PROT_WRITE,  MAP_SHARED ,  fd , 0 );

if(  (int)buf < 0  )  {   perror( "mmap" );
                          return 10;     }

/* Setting del canale   */
vc.channel  = channel;
vc.norm  = format;

if(  ioctl( fd , VIDIOCSCHAN , &vc ) < 0  ) {  perror( "VIDIOCSCHAN" );
                                               return GRAB_ERR_CHANNEL;   }

if(  ioctl( fd , VIDIOCGCHAN , &vc ) < 0  ) {  perror( "VIDIOCGCHAN" );
```

```
                                                    return GRAB_ERR_CHANNEL;   }
  fprintf(  stderr , "\nSelected Video Source (%d) Name : %s\n" , channel , vc.name  );
  fprintf(  stderr , "Video Format : %s\n\n" , format   ?  "NTSC"  :  "PAL"  );

  if (  ioctl( fd , VIDIOCGPICT ,  &vp )  < 0  ) {   perror( " VIDIOCGPICT " );
                                                      close(fd);
                                                      return GRAB_ERR_GPICT;      }
  grabber_isopen = 1;
  return 0;
}
```

unsigned char* grab_capture(void)   /* B ***/**

```
{
  int cycle = 0;
  const int cyclemax = 10;

  if( grabber_isopen )
    {
       mm.frame  = 0;
       mm.height = grab_height;
       mm.width  = grab_width;
       mm.format = VIDEO_PALETTE_RGB24;

       if(  ioctl( fd , VIDIOCMCAPTURE , &mm  ) <0 )
                          {  perror( " VIDIOCMCAPTURE " );
                             return NULL;              }
      while ( (cycle <  cyclemax) && (ioctl(fd, VIDIOCSYNC, &mm.frame)<0) )
                        cycle++;
    }
  else   {  perror( "Grabber is CLOSED" );
             return NULL;      }
   if( cycle >= cyclemax ) {    perror( "No image timeout" );
                                return NULL;        }
  return buf;
}
```

```
int grab_close( void )   /*** C ***/
 {
 if( grabber_isopen )
   {
    munmap(  buf  ,  grab_height * grab_width * 3  );
    if(  close(fd) !=  0  )
         perror( "Error in Close fd" );
    grabber_isopen = 0;
   }
 else
   return GRAB_ERR_ISCLOSED;
 return 0;
}
int grab_set_attr( int brightness , int hue ,  int contrast , int colour )
/*** D ***/
 {   vp.brightness =  brightness*256;
     vp.hue          =  hue*256;
     vp.contrast    =  contrast*256;
     vp.colour      =  colour*256;
 if (  ioctl( fd , VIDIOCSPICT ,  &vp )  < 0   ) {  perror( " VIDIOCSPICT " );
                                                    close(fd);
                                                    return GRAB_ERR_SPICT;    }
 return 0;
}
/*** E ***/
int grab_get_attr( int *brightness ,  int *hue ,  int *contrast ,
                   int *colour )
{  if (  ioctl( fd , VIDIOCGPICT , &vp ) < 0 )  {   perror( " VIDIOCGPICT " );
                                                    close(fd);
                                                   return GRAB_ERR_GPICT;     }
   *brightness = vp.brightness/256;
   *hue        = vp.hue/256;
   *contrast   = vp.contrast/256;
   *colour     = vp.colour/256;
 return 0;
}
```

int grab_write_ppm(char filename) /* F ***/**

```
{  int i , j;
   FILE *fp;
   fp = fopen( filename , "w" );
   fprintf(  fp , "P6\n %d %d\n 255\n" , grab_width , grab_height  );
   fflush( fp );
   for(  i=0 ; i<grab_height ; i++ )
    for(  j=0 ; j<grab_width ; j++ ){  fwrite(  buf + (i * grab_width + j)*3+2 , 1, 1, fp  );
                                       fwrite(  buf + (i * grab_width + j)*3+1 , 1, 1, fp  );
                                       fwrite(  buf + (i * grab_width + j)*3  , 1, 1, fp   ); }
 fclose(fp);
 return 0;
}
```

int grab_get_vcap(void) /* G ***/**

```
{ if(  ioctl( fd , VIDIOCGCAP , &vcap ) <0  ) {   perror( " VIDIOCGCAP " );
                                                  return 10;          }
  fprintf(  stderr , "Capability name     : %s\n"    , vcap.name  );
  fprintf(  stderr , "Capability type     : %d\n"     , vcap.type  );
  fprintf(  stderr , "Capability channels  : %d\n"   , vcap.channels  );
  fprintf(  stderr , "Capability audios    : %d\n"    , vcap.audios  );
  fprintf(  stderr , "Capability maxwidth  : %d\n" , vcap.maxwidth  );
  fprintf(  stderr , "Capability maxheight : %d\n" , vcap.maxheight  );
  fprintf(  stderr , "Capability minwidth  : %d\n"  , vcap.minwidth  );
  fprintf(  stderr , "Capability minheight : %d\n"  , vcap.minheight  );
  return 0;
}
```

int grab_get_vp(void) /* H ***/**

```
{   if (  ioctl( fd , VIDIOCGPICT , &vp )  <  0  ){  perror( " VIDIOCGPICT " );
                                                     close(fd);
                                                     return GRAB_ERR_GPICT;     }
    fprintf(  stderr , "Pictures brightness : %d\n" , vp.brightness  );
    fprintf(  stderr , "Pictures hue        : %d\n"     ,  vp.hue  );
    fprintf(  stderr , "Pictures colour     : %d\n"   , vp.colour  );
    fprintf(  stderr , "Pictures contrast   : %d\n"   , vp.contrast  );
    fprintf(  stderr , "Pictures whiteness  : %d\n" , vp.whiteness  );
```

```
    fprintf(  stderr , "Pictures depth      : %d\n"    , vp.depth  );
    fprintf(  stderr , "Pictures palette    : %d\n"    , vp.palette  );
    return 0;
}
```

7.3. Makefile

```
# Makefile per compilare lattine.c come file shared object caricabile dal saphira
SHELL = /bin/sh
#################################################################
SRCD = ./
OBJD = ./
INCD = $(SAPHIRA)/handler/include/
LIBD = $(SAPHIRA)/handler/obj/
COLBERT = $(SAPHIRA)/colbert/

# find out which OS we have
include $(SAPHIRA)/handler/include/os.h
CFLAGS =  -g -w -D$(CONFIG) $(PICFLAG)
CC = gcc
INCLUDE = -I$(INCD) -I$(X11D)include
#################################################################
 all: lattine.so
        touch all
 clean:
        rm *.o *~
$(OBJD)lattine.o: $(SRCD)lattine.c $(INCD)saphira.h
        $(CC) $(CFLAGS) -c $(SRCD)lattine.c $(INCLUDE) -o $(OBJD)lattine.o
lattine.so: $(OBJD)lattine.o
        $(LD) $(SHARED) $(OBJD)lattine.o -o lattine.so
```

8. Bibliografia

[1] Kurt G. Konolige, PhD : “Saphira Manual Version 6.1”, Novembre 1997.

[2] ActiveMedia ROBOTICS : “Saphira Manual Version 6.2”, Agosto 1999.

[3] ActiveMedia ROBOTICS : “Pioneer 1 Gripper & Experimenter’s Manual version 1.2”, Agosto 1997.

Indice