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

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica Mobile
(Prof. Riccardo Cassinis)

Parcheggio di Speedy: modulo visione

Elaborato di esame di:

Mauro Bianchi, Paolo Cucchi, Alberto Forino, Francesco Lombardo, Matteo Sacco

Consegnato il:

27 giugno 2006

Sommario

L'elaborato svolto costituisce una delle due parti del progetto più ampio finalizzato alla guida del robot ActivMedia Pioneer "Speedy" nella sua stazione di ricarica. Si è affrontato il problema dell'interpretazione delle immagini acquisite mediante una videocamera posta a bordo del robot stesso. Le informazioni ottenute dall'elaborazione vengono poi messe a disposizione del modulo che si occupa della navigazione e del controllo dei movimenti del robot finalizzati al parcheggio nella stazione di ricarica.

1. Introduzione

Per consentire al robot di riconoscere la stazione di ricarica e di entrarvi correttamente, l'ambiente di lavoro è stato opportunamente strutturato introducendo due elementi ben identificabili: una linea gialla diretta verso la stazione e un landmark artificale verde posto all'inizio di tale linea che permette di raggiungere la posizione di partenza per la vera e propria operazione di parcheggio. La stazione inoltre presenta due guide che facilitano la fase finale di parcheggio del robot, anche in caso di un ingresso non perfettamente allineato, e due fermi metallici che permettono a Speedy di arrestarsi nella posizione desiderata.

Figura 1: Dettaglio della stazione di ricarica e dei due oggetti da riconoscere

L'operazione di parcheggio all'interno della stazione di ricarica è stata suddivisa in due fasi principali:

- ricerca del marker verde attraverso l'esplorazione casuale dell'ambiente
- avvicinamento alla stazione di ricarica seguendo la traiettoria indicata dalla linea gialla

Inizialmente infatti il robot esplora l'ambiente con movimento casuale alla ricerca del landmark di colore verde, sufficientemente grande da poter essere rilevato dalla webcam già a una discreta distanza. Una volta individuato e raggiunto tale punto, il robot ruota su se stesso alla ricerca della striscia gialla che lo guida fino alla stazione di ricarica. In questa fase il compito del robot è quello di seguire tale linea correggendo man mano il proprio orientamento. Una volta che il robot è giunto alla stazione e le ruote hanno toccato gli appositi fermi meccanici, mandando in stallo i motori, il programma termina e il robot si ferma.

Figura 2: Speedy sul marker verde e nella posizione finale di ricarica

La soluzione del problema è stata raggiunta grazie a due attività complementari, svolte dai due gruppi cui è stato assegnato il progetto: l'elaborazione delle immagini acquisite mediante la videocamera e il controllo dei movimenti del robot sulla base delle informazioni ottenute. In particolare il gruppo che si è occupato della visione, ha elaborato le immagini acquisite in modo da riconoscere l'oggetto verde e la striscia gialla, calcolandone la relativa posizione. Il gruppo che si è occupato della navigazione ha utilizzato queste informazioni sulla posizione dei blob per la navigazione del robot e per prendere decisioni sui comportamenti e sui movimenti da compiere per raggiungere l'obiettivo.

Le problematiche affrontate in questa documentazione hanno riguardato la prima attività, ovvero l'acquisizione e l'elaborazione delle immagini tramite la webcam, per ricavarne le informazioni da fornire poi al gruppo che si è occupato della navigazione.

2. Il problema affrontato

Per rendere possibile la corretta esecuzione della procedura di parcheggio bisogna in primo luogo rilevare e localizzare il landmark verde e in seguito la linea gialla che conduce alla stazione di ricarica. Il progetto svolto si propone perciò di affrontare le problematiche relative al riconoscimento di questi due oggetti all'interno delle immagini acquisite attraverso la telecamera. In questo ambito trovano applicazione le tecniche di segmentazione del colore (ossia il partizionamento dell'immagine in regioni omogenee) e riconoscimento della forma: in prima istanza è stato necessario scegliere un'opportuna rappresentazione dei colori presenti nell'immagine, e in seguito stabilire le condizioni secondo le quali individuare un punto appartenente all'oggetto da ricercare, dal quale eventualmente estendere la selezione nel suo vicinato di colore simile ("region growing"). I principali problemi emersi nell'implementare ciò sono causati dalla variazione delle condizioni al contorno. Ad esempio l'accensione di luci nel laboratorio e la proiezione di ombre sugli oggetti da riconoscere possono condurre a falsi riconoscimenti. Occorre pertanto effettuare dei controlli che possano garantire, con un opportuno grado di sicurezza, i riconoscimenti corretti anche quando le mutate condizioni esterne vanno ad influenzare i parametri. In particolar modo è stato necessario effettuare dinamicamente la correzione del guadagno della telecamera, in modo che l'acquisizione risentisse il meno possibile degli effetti delle modificazioni ambientali.

Riguardo al riconoscimento della forma, vi sono due elementi che contribuiscono a definire l'aspetto dell'oggetto: la posizione relativa rispetto al robot e l'orientamento da cui viene visto. Per gli scopi del presente progetto non si sono però manifestati problemi particolari, data la semplicità degli oggetti da individuare e conseguentemente dei controlli da effettuare. Essi si limitano ad analizzare le dimensioni piuttosto che la morfologia nel suo complesso.

2.1. Rappresentazione dell'immagine

Tra i numerosi metodi per specificare i colori attraverso coordinate numeriche, ossia come punti appartenenti al cosiddetto "spazio dei colori", alcuni si prestano meglio di altri all'elaborazione attraverso il calcolatore. Ad esempio la struttura della rappresentazione RGB, che prevede tre coordinate

corrispondenti all'intensità di rosso, verde e blu, presenta degli inconvenienti quando si desidera individuare una classe di punti con colore simile. I volumi coinvolti in relazioni di questo tipo sono infatti di forma conica e non possono essere ricondotti a semplici soglie sulle componenti. Ciò è invece possibile nelle rappresentazioni basate sulla separazione fra luminanza e crominanza, dovute a A. H. Munsell: in questa categoria rientra il sistema scelto in questo progetto, cioè l'HSI (Hue – Saturation – Intensity). Esso è stato scelto proprio in quanto consente una più semplice ed efficace elaborazione in termini di soglie costanti, esprimibili con semplici istruzioni condizionali nel codice.

Le coordinate relative ad un colore nello spazio HSI corrispondono ai suoi valori di tonalità (Hue), saturazione (Saturation) ed intensità (Intensity) . Tonalità e saturazione sono le componenti relative alla crominanza, mentre l'intensità rappresenta la luminanza. Per valori crescenti di S il colore migliorerà in purezza, diminuendo la sua tendenza verso il grigio. Al crescere di I si avrà un aumento della luminosità del colore, con tendenza al nero per valori minimi di questa coordinata. H è invece una componente angolare ed è da intendersi come una rotazione antioraria attorno all'asse I. Essa produce l'alternarsi delle varie tinte, riassunte nella tabella seguente:

Hue	Tinta
0° o 360°	Rosso
60°	Giallo
120°	Verde
180°	Ciano
240°	Blu
300°	Magenta

In figura è possibile confrontare i due spazi di colore appena descritti.

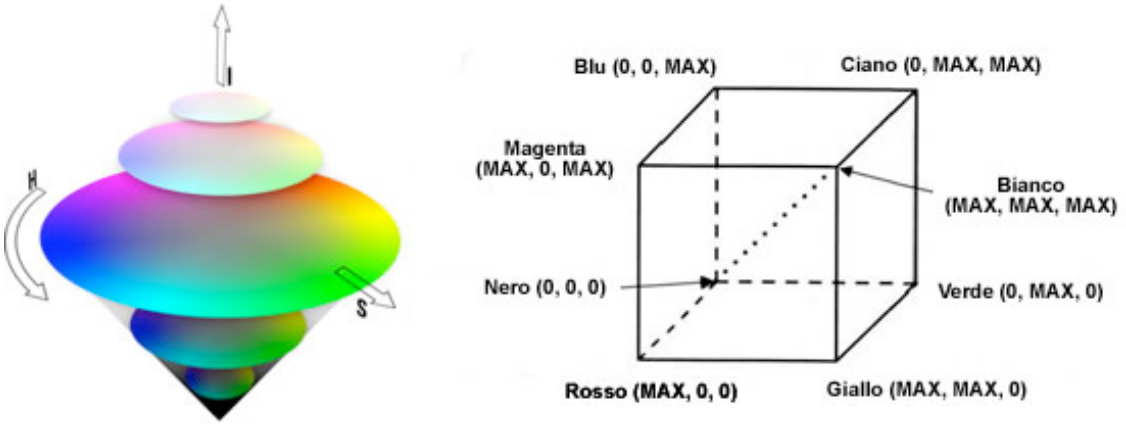

Figura 3: Confronto tra gli spazi di colore HSI e RGB

La trasformazione tra coordinate RGB a HSI è ottenibile attraverso un semplice algoritmo dovuto a Gonzalez e Woods, ma la trattazione esula dagli scopi del presente rapporto: basti sapere che le librerie utilizzate forniscono una funzione per la trasformazione tra i due spazi di colore della struttura dati che descrive un'immagine.

2.2. Condizioni di riconoscimento

Lavorando nello spazio HSI è stato possibile imporre semplici valori di soglia per il riconoscimento dei colori cercati. Si sono caratterizzate le due classi di colore "verde marker" e "giallo linea" specificando per ciascuna una n-upla di sei valori soglia: due per ognuna delle dimensioni dello spazio di colore. Nelle condizioni si sono poi utilizzati dei valori misurati durante prove sperimentali e si sono applicate delle tolleranze distinte per le singole componenti. Le problematiche affrontate in sede di definizione di

queste condizioni sono tutte riconducibili alla variabilità della luminosità dell'ambiente e alla presenza di eventuali ombre sulla linea gialla. Per quanto riguarda il landmarker, i problemi causati dalle ombre proiettate sulla sua superficie verde sono minimi rispetto a quelli relativi alla linea: quest'ultima ne è molto più soggetta in quanto di colore chiaro. Sono state condotte numerose prove sperimentali per testare il riconoscimento in svariate condizioni di illuminazione: solo luce al neon del laboratorio, accensione di uno o di entrambi i fari a incandescenza presenti in laboratorio, proiezione di ombre sulla linea. Queste problematiche sono state riscontrate sia nell'identificare la superficie di interesse, sia nella fase preliminare di individuazione del seme, punto di partenza del region growing.

L'elaborazione attraverso Matlab dei valori di HSI corrispondenti ai punti di immagini acquisite in diverse situazioni di luce e ombra hanno consentito lo svolgimento di uno studio empirico, teso a fornire opportuni valori di soglia. A latere è stato possibile anche trarre conclusioni in merito all'utilizzo delle tre componenti dello spazio di colore. Infatti è emerso come fra di esse la più discriminante per gli scopi in esame sia H, in quanto evidenzia delle discontinuità molto nette tra la linea e il colore del pavimento circostante. Viceversa le informazioni sull'intensità I risultano quelle meno significative, visto che i suoi valori sono distribuiti in maniera tale da rendere indistinguibili i due colori. Filtrando opportunamente i dati acquisiti, cioè azzerando le componenti dei punti da considerare di colore differente da quello della linea, è stato possibile determinare opportune configurazioni di valori soglia e tolleranze in grado di costituire un compromesso applicabile nelle diverse condizioni ambientali.

Nelle figure che seguono è possibile osservare due delle situazioni ambientali di prova considerate e i grafici delle elaborazioni relative. Nella prima serie si ha assenza di ombre, nella seconda invece esse sono presenti in quantità rilevante. La componente studiata è quella più significativa, ossia la H.

Figura 4: Immagine catturata nella situazione priva di ombre

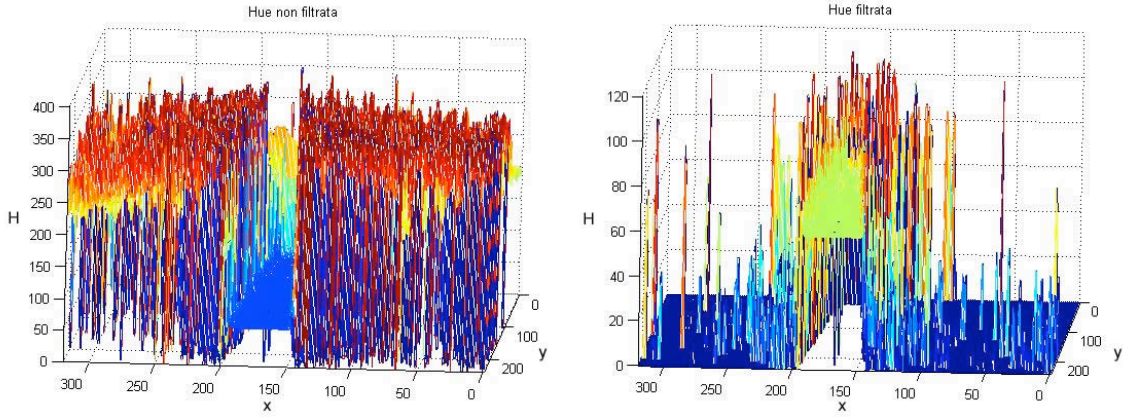

Figura 5: Grafici in falsi colori della componente H prima e dopo l'applicazione del filtro

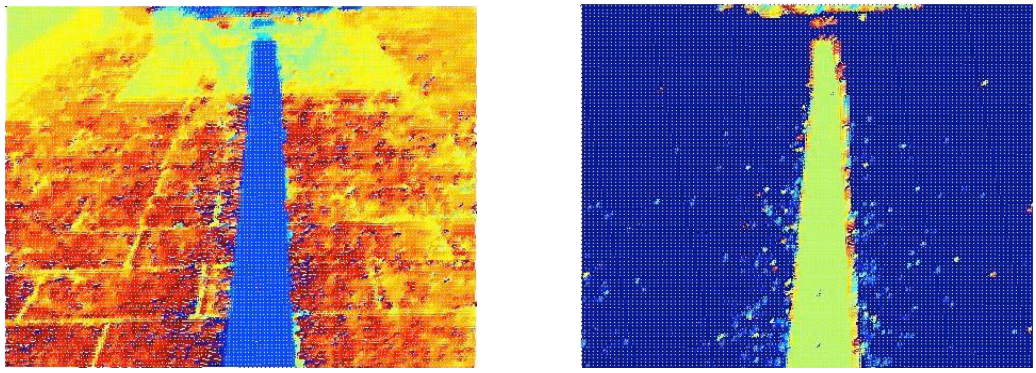

Figura 6: Gli stessi grafici proiettati sulle due dimensioni delle coordinate spaziali

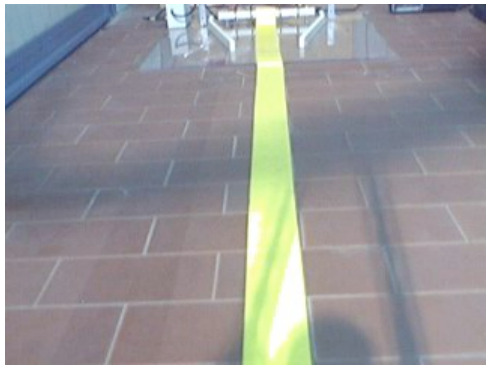

Figura 7: Immagine catturata nella situazione con presenza di ombre

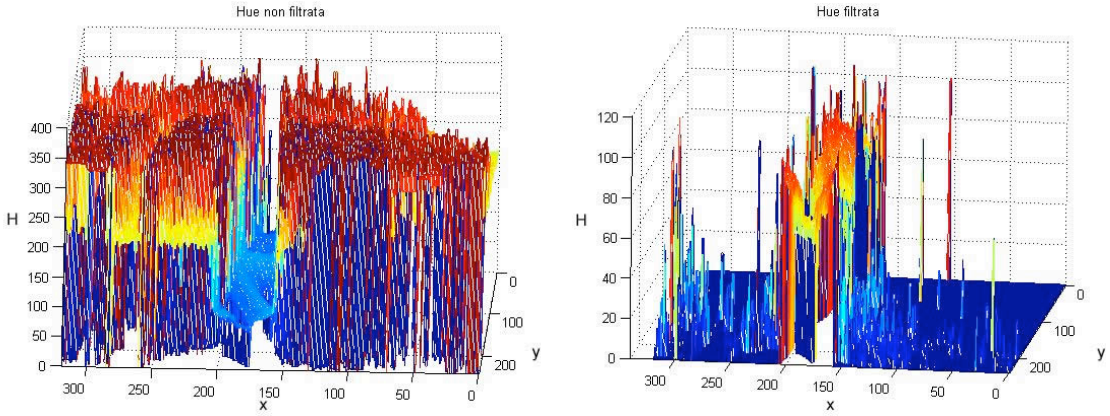

Figura 8: Grafici in falsi colori della componente H prima e dopo l'applicazione del filtro

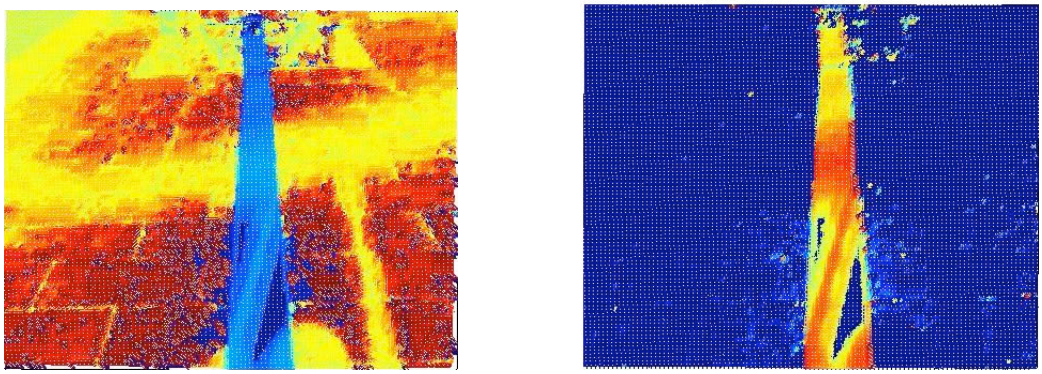

Figura 9: Gli stessi grafici proiettati sulle due dimensioni delle coordinate spaziali

3. La soluzione adottata

Per elaborare le immagini acquisite mediante la videocamera USB Logitech Quickcam Pro 4000 si è fatto ricorso alle librerie open source VisLib (versione 1.8), che mettono a disposizione una serie di metodi per l'acquisizione delle immagini e la successiva elaborazione delle stesse. Il paragrafo 3.1 introdurrà queste librerie focalizzandosi sui metodi utilizzati per risolvere il problema della ricerca del rettangolo e della linea colorata descritto nel capitolo precedente.

3.1. Le VisLib ActivMedia

Le librerie VisLib sono reperibili liberamente dal sito del progetto open source omonimo [6]. La libreria è compatibile con i sistemi GNU\Linux e le librerie "Video for Linux" [7] che offrono un ampio supporto per l'acquisizione video in generale su tali piattaforme, sia sotto forma di driver sia sottoforma di tool software per l'acquisizione e l'elaborazione dei fotogrammi. Grazie alle librerie VisLib è possibile visualizzare immagini a colori da una telecamera collegata al sistema ed elaborare le immagini in base alle informazioni di interesse.

Le funzioni principali messe a disposizione da tale libreria permettono, oltre all'acquisizione e alla visualizzazione di fotogrammi, la rilevazione di zone di colore, la rilevazione di forme geometriche e la possibilità di effettuare il tracking delle stesse.

Per un corretto funzionamento le librerie necessitano di un kernel versione 2.0.35 o superiore (quella utilizzata per il progetto è la 2.6.15) e di un ambiente grafico a 8, 16 o 24 bit, anche se lavorare con il primo è fortemente sconsigliato.

Le strutture dati principali utilizzate dalla libreria sono:

- vlPixel → rappresenta il singolo pixel dell'immagine
- vlImageFormat → descrive il formato dell'immagine che può essere (NONE, RGB, HSI, NRG, GRAY, BINARY). Quello utilizzato per la realizzazione dell'elaborato è il formato HSI (tonalità, saturazione, intensità). L'immagine fornita dalla telecamera è nel formato RGB e quindi prima di elaborarla è stata convertita nel formato HSI con una funzione apposita messa a disposizione dalla libreria stessa.
- vlImage → è l'immagine vera e propria formata da un array di vlPixel, da un vlImageFormat oltre che dalle informazioni sulle dimensioni dell'immagine (heigth, width)

Oltre a queste strutture dati principali la libreria ne definisce altre utili per l'elaborazione e la visualizzazione delle immagini quali vlMask per l'operazione di convoluzione, vlWindow per definire aree sull'immagine e vlPoint per definire un punto di un immagine con determinate caratteristiche. La struttura vlObject è invece definita per descrivere un'area sull'immagine convessa inscritta in un rettangolo, individuata tramite un offset su ogni riga della quale viene definita la lunghezza.

Le funzioni messe a disposizione dalla libreria ritornano il valore 0 se eseguite con successo o -1 altrimenti; questo permette sempre di controllare il regolare funzionamento della libreria. Oltre alle funzioni di inizializzazione e arresto per i moduli di interesse (Display, Grab, Motion o tutti), VisLib mette a disposizione anche numerose funzioni per l'utilizzo delle strutture dati e per l'elaborazione dei dati dell'immagine acquisita.

Per la visualizzazione della telecamera sono a disposizione funzioni per la gestione di finestre di visualizzazione e funzioni per la gestione di eventi (click del mouse, selezione di un punto, selezione di un'area rettangolare) durante la visualizzazione. Per l'acquisizione di un'immagine esiste la funzione vlGrabImage. Sono inoltre presenti funzioni per la conversione fra tutti i formati prima descritti.

Relativamente all'elaborazione la libreria fornisce funzioni per semplici operazioni fra immagini (ridimensionamento, sottrazione di aree) e anche per operazioni più complesse quali, ad esempio, quella di filtro su diversi parametri e di sfumatura.

La libreria infine mette a disposizione anche funzioni per il rilevamento e la gestione di eventi di movimento (motion detection).

Le funzioni maggiormente utilizzate nell'elaborato, oltre a quelle di inizializzazione, creazione e distruzione oggetti, verrano descritte più approfonditamente di seguito:

- int vlObjectExtractHsi (vlImage *pic, vlPoint *point, int toleranceHue, int toleranceSat, int toleranceInt, vlObject *object)

La funzione effettua l'estrazione di un oggetto dall'immagine passata partendo ad analizzare l'immagine dal punto passato. Tutti i pixel contigui con i valori di H, S, e I dentro la tolleranza passata alla funzione rispetto ai valori di H, S e I del punto passato verranno aggiunti all'oggetto.

Funzione utilizzata nelle funzioni di ricerca marker verde e striscia gialla, dopo aver individuato il punto da cui iniziare l'analisi.

- int vlGrabImage (vlImage *image)

La funzione è parte del modulo Grab che deve essere precedentemente inizializzato. Richiede un immagine alla telecamera che viene memorizzata nella variabile image passata.

Viene utilizzata nel ciclo infinito di acquisizione immagini, prima della ricerca dei blob colorati.

- vlRgb2Hsi(pic, window, pic2)

Tramite questa funzione si converte l'immagine passata (pic) o parte di questa (window) dal formato generato dalla telecamera RGB nel formato con cui questa viene elaborata HSI e l'immagine generata viene salvata in pic2.

Viene utilizzata dopo ogni acquisizione (vlGrabImage)

- vlGrabSetGain(int gain)

La funzione non fa parte della VisLib ma è stata aggiunta per poter modificare il valore di guadagno della telecamera; il valore passato viene impostato come guadagno della telecamera.

Viene utilizzata dopo l'acquisizione ma solo se il valore di luminosità non rientra in un range prestabilito

Sono state utilizzate anche altre funzioni per visualizzare a video l'immagine della telecamera ed evidenziare i blob corrispondenti agli oggetti identificati, in modo da verificare il corretto funzionamento del programma.

Ognuna di queste funzioni è descritta in dettaglio nella documentazione della VisLib.

3.2. La struttura architetturale

Per consentire di lavorare distintamente sui programmi di elaborazione delle immagini e di navigazione si è deciso di far comunicare i due processi via socket. A questa conclusione si è dovuti giungere in corso d'opera, dopo aver lavorato inizialmente tramite FIFO, per conformarsi con le mutate modalità di comunicazione implementate nell'altro modulo del progetto.

In questo modo il programma di acquisizione opera come un server che risponde alle richieste del programma di controllo del movimento riguardo la posizione relativa dei due oggetti marker.

Dopo le prime prove utilizzando un solo thread principale si è notato che le informazioni relative alla posizione, elaborate al momento della richiesta del client, potevano essere rese disponibili al secondo modulo solo con un inevitabile ritardo dovuto ai tempi di computazione, risultando quindi poco utili ai fini del controllo del movimento. Per questo motivo si è scelto di suddividere il programma di visione in due thread : il server vero e proprio e il thread per l'acquisizione e l'interpretazione delle immagini.

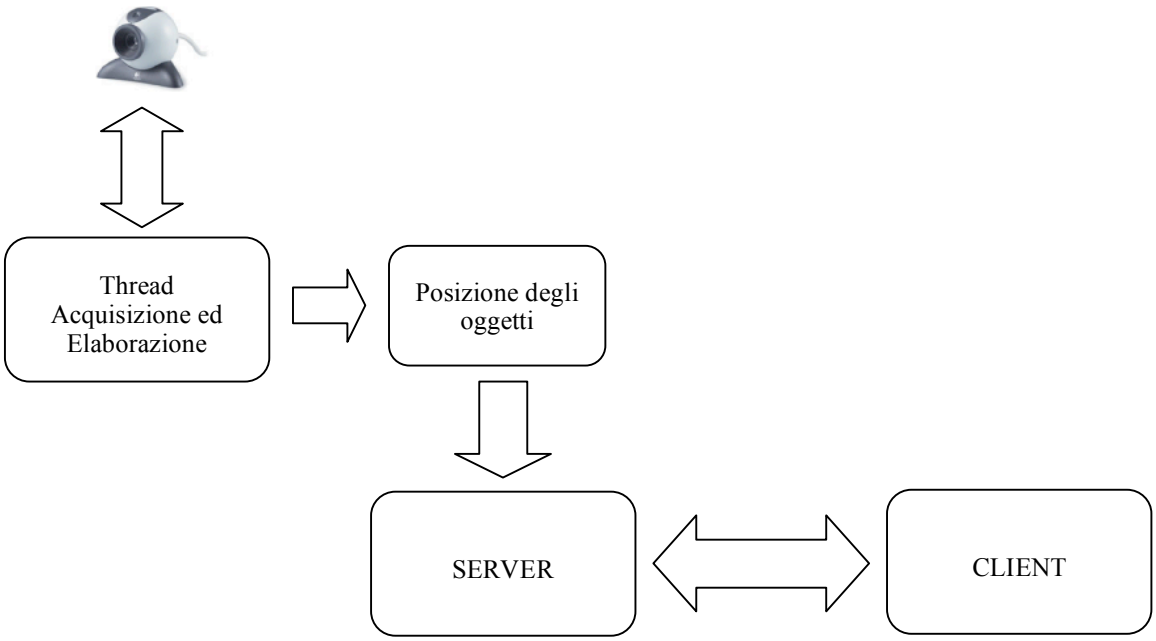

Figura 10: Schema della struttura architetturale

Così facendo l'acquisizione e l'interpretazione delle immagini avviene continuamente ed indipendentemente dal server, di conseguenza la posizione relativa degli oggetti è costantemente aggiornata all'ultimo fotogramma. Il server, quando riceve una richiesta, risponde prelevando le informazioni da una zona di memoria condivisa, evitando quindi il ritardo che si avrebbe facendo partire da capo una nuova acquisizione.

3.3. L'algoritmo di acquisizione ed interpretazione

Per la soluzione del problema dell'acquisizione delle immagini dalla videocamera e dell'elaborazione delle stesse si sono utilizzate alcune funzioni messe a disposizione dalle librerie VisLib, introdotte nel paragrafo 3.1. Nello schema di Figura 11 viene riportato il diagramma di flusso che illustra ad alto livello i passi compiuti dal thread di elaborazione delle immagini.

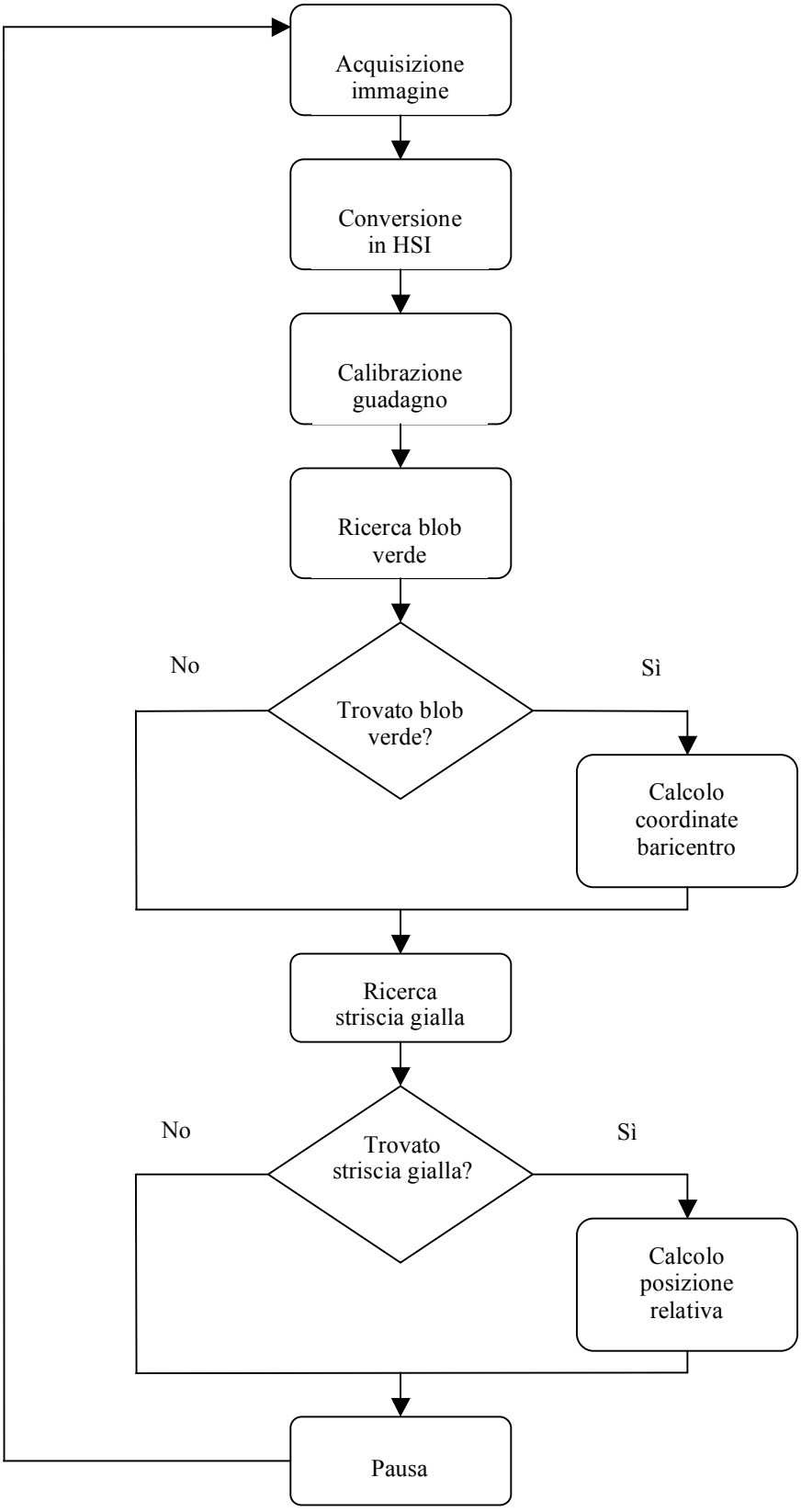
Acquisizione immagine
Conversione in HSI
Calibrazione guadagno
Ricerca blob verde
Trovato blob verde?
No
Sì
Calcolo coordinate baricentro
Ricerca striscia gialla
Trovato striscia gialla?
No
Sì
Calcolo posizione relativa
Pausa

Figura 11: Algoritmo elaborazione immagine

Acquisendo le immagini a intervalli di tempo regolari, si è innanzitutto utilizzata una funzione di conversione per lavorare con il sistema di rappresentazione dei colori HSI, che è risultato migliore del sistema RGB per la definizione dei blob. E' stato poi effettuato un controllo sulla luminosità media dell'immagine per poi procedere ad un'eventuale regolazione del guadagno della videocamera in modo da adattarsi a condizioni di luce diverse. Si è infatti notato che condizioni di illuminazione variabili rendevano difficile il riconoscimento degli oggetti, in particolare quello della striscia gialla che, se illuminata da una luce forte, tende a diventare bianca nell'immagine catturata dalla videocamera e quindi non più riconoscibile. Il controllo introdotto consente di regolare il guadagno in funzione della luminosità e permette di risolvere tali problemi.

Il riconoscimento del marker verde viene fatto scandendo (da sinistra a destra e dall'altro verso il basso) i pixel dell'immagine fino a che se ne trova uno le cui caratteristiche di colore siano all'interno di un certo intervallo di tonalità, saturazione e luminosità stabilito. Trovato il pixel viene invocata una funzione messa a disposizione delle librerie VisLib che consente, tramite la tecnica del region growing, di identificare un blob a partire da tale punto, considerando i pixel adiacenti che hanno caratteristiche di colore simili. Una volta che è stato evidenziato, vengono effettuati dei controlli sulla sua dimensione in modo da filtrare i blob troppo piccoli o addirittura singoli punti dell'immagine, che avevano caratteristiche simili all'oggetto da riconoscere. Se il blob supera questo controllo viene considerato valido e ne vengono calcolate le coordinate del baricentro. Questi dati sono poi salvati in variabili globali e verranno trasmessi al client che ne fa richiesta. In caso contrario, il programma riprenderà con la ricerca di un altro punto a partire dall'ultimo pixel dell'immagine analizzato. Se non è stato trovato niente che soddisfi i requisiti di dimensione e colore, vengono settati dei valori convenzionali per le relative variabili globali in moda da indicarne l'assenza.

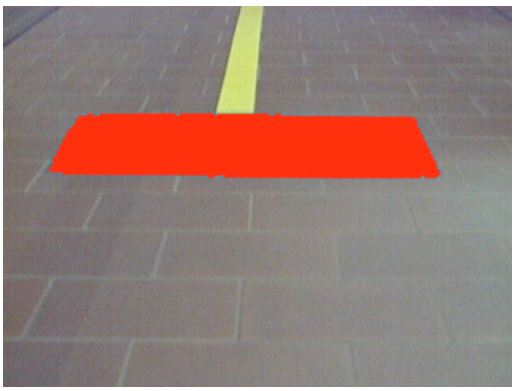

Figura 12: Riconoscimento del marker verde con colorazione del blob corrispondente

Per filtrare ulteriormente i blob si era pensato di introdurre un controllo più raffinato sulla morfologia degli oggetti individuati. Tuttavia questa idea è stata accantonata perché in questo modo sarebbe stato scartato anche un blob corretto, ma corrispondente ad un oggetto da identificare che nell'immagine compariva in modo incompleto (e quindi di forma diversa da quella prevista). Per velocizzare invece la scansione dei pixel dell'immagine, non sono stati scanditi tutti i singoli punti della matrice, ma si è proceduto con un certo passo (nelle prove effettuate si è scelto il valore 5), dal momento che gli oggetti da riconoscere hanno una certa dimensione e quindi si estendono per un numero abbastanza elevato di pixel. Dopo alcune prove sperimentali, si è misurata una capacità di elaborazione con frequenza massima intorno alle 10 immagini al secondo, compresi 20 millisecondi di ritardo introdotti in ciascuna iterazione per non sovraccaricare il processore.

Il riconoscimento della striscia gialla avviene in maniera simile a quanto visto per il landmark verde con la differenza che in questo caso viene scandita una sola linea della matrice dell'immagine, posta nella regione inferiore dell'immagine stessa. Questa scelta è motivata dal fatto che quando il robot sta seguendo la striscia, questa parte dalla parte bassa dell'immagine e si estende in altezza per un certo tratto. Risulta quindi sufficiente scandire una linea posta nella parte inferiore dell'immagine per rilevare la striscia (se presente ovviamente). Inoltre questo rende la ricerca più veloce e migliora le prestazioni. Una volta individuato il blob che rappresenta la striscia gialla, viene calcolata la posizione relativa (da -

50 a +50) della base della striscia rispetto all'asse centrale dell'immagine. Questa informazione è utile al programma di navigazione del robot perché consente di stabilire se il robot risulti o meno allineato con la striscia gialla.

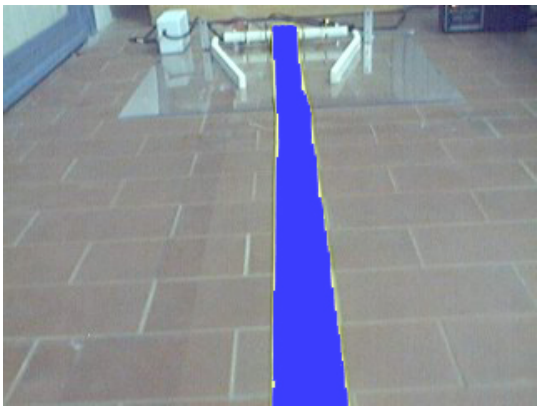

Figura 13: Riconoscimento della striscia gialla con colorazione del blob

3.4. Il protocollo Client-Server

Per consentire il dialogo con il programma che si occupa del movimento del robot si è definito un semplice protocollo di comunicazione che utilizza i socket.

Tale protocollo basato su TCP/IP prevede che il server resti in ascolto sulla porta 20000 e risponda a tre tipologie di messaggi provenienti dal client:

- *Richiesta di posizione del blob corrispondente al marker verde,* codificata come "0";
- *Richiesta di posizione relativa della striscia gialla,* codificata come "1";
- *Fine della comunicazione,* codificata come "2";

In base alla tipologia di richiesta il server risponderà con i valori opportuni, cioè con le coordinate del centro del blob riferito al landmark verde identificato nell'immagine corrente [x,y] per la richiesta "0", o con la posizione relativa del centro dell'ultima riga della striscia gialla per la richiesta "1". Tale posizione viene calcolata in base alla coordinata orizzontale del centro dell'ultima riga della striscia scalata ai valori compresi tra -50 e 50 (rispettivamente oggetto completamente a sinistra o destra nell'immagine).

Dal momento che il server è finalizzato esclusivamente a fornire le informazioni che consentono l'ingresso nella stazione di ricarica, si è deciso di prevedere anche la terminazione dello stesso a conclusione del parcheggio, una volta ricevuto dal client il messaggio "2". Tale decisione è motivata anche dal fatto che l'utilizzo della videocamera può avvenire solo in modo esclusivo.

3.5. Accorgimenti adottati

3.5.1. Regolazione del guadagno della webcam

Durante le numerose prove effettuate sul riconoscimento degli oggetti attraverso il loro colore è sorto il problema delle condizioni di illuminazione ambientale notevolmente variabili. Infatti a seconda delle diverse luci che venivano accese in laboratorio si ottenevano immagini con intensità di colore anche molto differenti. Ciò rendeva difficile stabilire dei valori assoluti di H, S e I da associare ai vari colori.

La soluzione adottata in questo caso è stata quella di cercare di avere delle immagini con intensità media costante e su tale valore impostare le soglie per i riconoscimenti.

Si è quindi implementato un regolatore proporzionale del guadagno della webcam che consente di avere una intensità media della luminosità costante (fatta eccezione per i transitori). Per ogni immagine acquisita viene quindi stimata una media dell'intensità dell'immagine e in base alla differenza con l'intensità media desiderata si sceglie di aumentare o diminuire proporzionalmente il guadagno della webcam.

3.5.2. Regolazione delle tolleranze e delle soglie per i colori

Un altro problema sorto durante le prove effettuate in laboratorio è stato quello delle ombre sulla striscia gialla.

Le ombre infatti facevano sì che il riconoscimento di aree adiacenti dello stesso colore (nel nostro caso il giallo) si interrompesse tra zone di luce e di ombra sulla linea gialla, e ciò poteva portare ad una identificazione errata del blob e quindi ad elaborare informazioni sulla posizione relativa anche molto imprecise.

Tale problema è stato risolto in parte con la regolazione del guadagno della webcam ed in parte impostando valori delle tolleranze diversi per le varie componenti di colore. In particolare si è osservato che l'algoritmo implementato migliorava il suo comportamento imponendo delle tolleranze abbastanza strette sulla componente tinta (hue) e imponendo invece più elasticità su saturazione ed intensità.

In questo modo si è riusciti a risolvere gran parte delle problematiche relative alle ombre che tuttavia rimangono nei casi di forti contrasti illuminazione-ombra.

3.5.3. Controlli morfologici

Talvolta nella fase di individuazione del marker verde o della striscia gialla si verificavano degli errori.

Solitamente ciò accadeva in due situazioni particolari: nei transitori della regolazione del guadagno dove i colori dell'immagine erano falsati rispetto alle soglie impostate, o per la presenza effettiva di elementi nell'immagine con caratteristiche di colore simili a quelle degli oggetti (ad esempio le fughe del pavimento in alcuni punti e la striscia gialla). Per ovviare a questo problema o per lo meno limitarne gli effetti si è implementato un controllo morfologico sugli oggetti individuati.

In particolare tale controllo viene effettuato sulle dimensioni minime che tali oggetti possono avere.

Per la striscia gialla le dimensioni minime nell'immagine sono state impostate ad una altezza e larghezza di 20 pixel dato che tale striscia va riconosciuta solo da vicino. Per l'altro blob invece si è fissata la larghezza minima a 20 pixel e si è impostata l'altezza a 10 pixel per consentire di riconoscerlo anche a distanza.

Non sono stati implementati controlli sulla dimensione massima dei due marker, poiché possono assumere dimensioni notevoli in alcune inquadrature.

Questi filtri morfologici dunque hanno consentito non tanto di riconoscere l'oggetto in base alle sue caratteristiche di forma, quanto piuttosto a scartare tutti quegli oggetti individuati per sbaglio in base alla sola verifica del colore.

Tali test potranno essere ulteriormente perfezionati in un futuro, anche se già solo con questo semplice controllo hanno ridotto notevolmente il numero di errori.

4. Modalità operative

L'applicativo che permette il parcheggio nella stazione di ricarica di Speedy può essere fornito in due modi differenti. Si segnala immediatamente che in entrambi i casi è richiesto che sul calcolatore che si utilizza siano installate le librerie Aria e le librerie VisLib. Queste ultime sono necessarie solo in fase di compilazione, mentre le librerie Aria sono necessarie anche al momento dell'esecuzione per il linking dinamico.

I due modi in cui l'applicazione può essere fornita sono:

- Tramite due file eseguibili (per sistema operativo Linux)

- Tramite l'insieme dei sorgenti dell'applicativo

Il caso più semplice è sicuramente il primo, in cui l'utente non deve di fatto fare alcuna operazione di installazione. I due file eseguibili possono essere lanciati direttamente dalla console di comando, e sono:

- main (all'interno della sottodirectory client)
- visione (all'interno della sottodirectory visione)

Essi corrispondono ai due processi client e server che comunicano tramite socket per la soluzione dell'intero problema. Il file visione è l'eseguibile del processo server che appena è lanciato rimane in attesa di richieste di connessione (ne accetta al massimo una), mentre il file main è l'eseguibile del processo client che si connette prima al robot e poi al processo server.

Per come è costruito, il client cerca di connettersi al server non appena ha stabilito la connessione con il robot: in caso di fallimento in questa fase, il client termina l'esecuzione. È quindi necessario avviare prima il processo server attraverso il comando "./visione/visione", e immediatamente dopo il processo client attraverso il comando "./client/main".

Nel caso in cui vengano forniti i sorgenti del progetto i due file eseguibili non sono immediatamente disponibili. In questo caso è necessaria una compilazione preliminare, per i cui dettagli si rimanda al paragrafo "Modalità di installazione".

4.1. Componenti necessari

Il progetto è stato realizzato e testato completamente su un sistema operativo di tipo Linux, e quindi è richiesta una distribuzione di tale sistema per eseguire l'applicativo.

Come già accennato in precedenza è inoltre necessario avere installate anche:

- la suite Aria
- la libreria VisLib (versione 1.8 o superiore)

Il motivo di questa necessità risiede nel fatto che i file eseguibili richiedono di poter accedere ad alcuni shared object (file con estensione ".so") che vengono linkati dinamicamente e che compongono le librerie suddette.

Si assume che le librerie Aria e VisLib siano installate rispettivamente all'interno della directory "/usr/local/Aria/" e della directory "/usr/local/vislib-1.8/".

4.2. Modalità di installazione

L'applicativo che permette il parcheggio del robot Speedy viene fornito tramite l'insieme dei suoi file sorgenti, scritti nei linguaggi C e C++, assumendo che l'utilizzatore abbia a disposizione un sistema operativo di tipo Linux. Prima di iniziare l'esecuzione del progetto, è necessario quindi compilare i file sorgenti: per farlo è sufficiente lanciare il comando "make" all'interno della directory in cui sono contenuti i sorgenti. Più precisamente, i file sorgenti sono raggruppati in due distinte directory: la prima, denominata client, contiene la parte relativa alla gestione del movimento (il processo client), mentre la seconda, denominata visione, contiene i sorgenti relativi al modulo omonimo. L'utente deve invocare due volte il comando make: la prima all'interno della directory client per compilare la parte relativa al movimento, la seconda all'interno della directory visione, per compilare la parte relativa alla gestione della videocamera.

Dopo questa operazione, all'interno delle due directory vengono creati i due file eseguibili main e visione che possono essere lanciati dalla console di comando.

4.3. Avvertenze

Eventuali condizioni di luce sfavorevoli (luce spenta o forte illuminazione) potrebbero rendere difficile il corretto riconoscimento degli oggetti. Per ottenere i migliori risultati si consiglia di eseguire le dimostrazioni all'interno dell'ARL con le luci appese al soffitto accese ed eventualmente accendendo anche il faretto a incandescenza posto sull'armadio sopra la docking station.

Si consiglia infine di non variare l'illuminazione durante l'esecuzione del programma, poiché durante il transitorio della regolazione del guadagno della videocamera potrebbero verificarsi problemi di riconoscimento (soprattutto della linea gialla).

5. Conclusioni e sviluppi futuri

L'obiettivo dell'elaborato è stato raggiunto e il modulo di visione sviluppato opera bene in condizioni di luce normale. Ciò consente, congiuntamente con il modulo sviluppato dall'altro gruppo, di far arrivare Speedy correttamente alla sua stazione di ricarica.

Nonostante gli accorgimenti adottati, in presenza di illuminazioni eccessive o particolari (ad esempio luci colorate) il programma può comunque dare dei problemi. La mancanza di un controllo sofisticato sulla morfologia può talvolta far sì che oggetti lontani vengano scambiati per il marker verde. Tali effetti hanno comunque un impatto marginale poiché quando il robot si avvicina a oggetti erroneamente identificati come landmark spesso tali oggetti non vengono più riconosciuti come marker dato che avvicinandosi cambiano il loro colore (ad esempio mattonelle illuminate con riflessi che da lontano sembravano verdi).

Eventuali sviluppi futuri potrebbero prevedere un controllo più sofisticato della morfologia degli oggetti (rischiosa però quando essi sono visibili solo parzialmente), e una definizione dello spazio di colore ancora più accurata di quella che è stata implementata per il giallo della striscia e il verde del marker in modo da risolvere anche gli ultimi problemi di forte luce-ombra.

L'efficienza del riconoscimento dell'appartenenza di un pixel ad una classe di colori può essere migliorata utilizzando quanto suggerito nell'articolo di Bruce, Balch e Veloso [1]. L'approccio classico consiste nell'utilizzare in maniera ingenua un insieme di confronti fra i valori e le soglie massime e minime per ciascuna componente, operando così sei diverse operazioni di confronti: ciò può essere molto inefficiente su processori a pipeline con esecuzione speculativa delle istruzioni. La proposta alternativa prevede invece di utilizzare una decomposizione della condizione multidimensionale in tre funzioni booleane, una per ogni componente, e con la condizione memorizzata in tre vettori con un numero di elementi binari pari al numero dei possibili valori per quella componente. In questo modo l'appartenenza ad una classe di colore può essere espressa con operazioni di AND bit a bit fra queste funzioni.

Per quanto riguarda invece il riconoscimento di regioni connesse si potrebbe cercare di applicare i diversi algoritmi presenti in letteratura per modificare le funzioni già presenti nelle VisLib, confrontando le prestazioni al fine di migliorarne efficacia ed efficienza. In particolare si potrebbe pensare di utilizzare, in fase di pre-elaborazione, opportuni filtri con lo scopo di fornire robustezza alla segmentazione ed in seguito tecniche basate su stimatori statistici per determinare l'appartenenza o meno dei singoli pixel alla figura da riconoscere. In questi ambiti si segnala l'algoritmo sviluppato da Comaniciu e Meer [2] per il filtraggio, e lo stimatore dell'ellissoide a volume minimo (MVE) [3]. Ancora più indicato, visto che è specificatamente concepito per la navigazione di robot autonomi e presenta ridotti tempi di esecuzione, è l'algoritmo di Adami e Mosma [4]. Si tratta di un metodo di region growing, in analogia a quanto esposto in questa sede.

Un'alternativa meno articolata consiste in un'evoluzione delle tecniche adottate in questo progetto, specificamente mirata ad affrontare le difficoltà introdotte dalla presenza di ombre. Si può infatti pensare di definire come ammissibile l'unione di due regioni dello spazio di colore: l'una centrata in un punto con terna HSI misurata in completa assenza di ombra, l'altra in un punto corrispondente invece ad una situazione di ombra. La forma più indicata per tali regioni è costituita da un parallelepipedo, il che agevola anche l'espressione e la verifica delle condizioni.

Inoltre dato che i due moduli sviluppati sono destinati a lavorare sulla stessa macchina e che il thread che si occupa dell'acquisizione e dell'elaborazione è indipendente, si può pensare di integrare i due moduli evitando così la comunicazione via socket.

Bibliografia

[1] Bruce, J., Balch, T., Veloso, M.: *"Fast and cheap color image segmentation for interactive robots"*, School of Computer Sience, Carnegie Mellon University, Pittsburgh.

[2] Comaniciu, D., Meer, P.: "Robust analysis of feature spaces: color image segmentation", in *Proc. IEEE Conference of computer vision and pattern recognition*, San Juan, 1997.

[3] Rousseeuw, P. J., Leroy, A. M.: *"Robust regression and outlier detection"*, Wiley, New York, 1987.

[4] Adami, N., Mosma, P.: *"Metodologie di autolocalizzazione per robot autonomi basate sulle caratteristiche di visione cromatica degli insetti*", Tesi di Laurea in Ingegneria Elettronica, Università degli Studi di Brescia, 1998.

[5] Gönner, C., Rous, M., Kraiss, K.: *"Real-time adaptive colour segmentation for the RoboCup Middle Size League"*, Technical University of Aachen

[6] http://sourceforge.net/projects/VisLib/

[7] http://www.exploits.org/v4l/

[8] http://en.wkipedia.org/wiki/HSI_color_space

Indice