POLITECNICO DI MILANO
Corso di Laurea in Ingegneria Informatica
Dipartimento di Elettronica e Informazione

UN SISTEMA PER IL TRACKING VISIVO DI PERSONE CON TELECAMERA IN MOVIMENTO

AI & R Lab
Laboratorio di Intelligenza Artificiale
e Robotica del Politecnico di Milano

Relatore: Prof. Andrea BONARINI
Correlatore: Ing. Matteo MATTEUCCI

Tesi di Laurea di:
Simone Mattia Antonio LAURENZANO, matricola 674269
Matteo MERLIN, matricola 668045

Anno Accademico 2006-2007

Il calcolatore è straordinariamente veloce, accurato e stupido.
L'uomo è incredibilmente lento, impreciso e creativo.
L'insieme dei due costituisce una forza incalcolabile.
[Albert Einstein]

Errare è umano, ma per fare veramente casino ci vuole la password di root...
[Anonimo]

Sommario

Il lavoro in oggetto si colloca nell'area della robotica mobile e della visione artificiale attraverso la progettazione di un apparato robotico mobile di visione a singola camera. Il robot disporrà della capacità di identificare e seguire oggetti in movimento, con particolare interesse verso il tracking di figure umane.

Tale progetto può essere considerato all'interno del programma a più ampio respiro, nato all'interno del laboratorio di Intelligenza Artificiale e Robotica (AiRLab) presso il Politecnico di Milano, che consiste nell'attrezzatura di un robot per lo svolgimento di compiti di assistenza personale. Applicazioni di questo tipo richiedono che il robot sia dotato di una camera motorizzata, in modo da seguire il proprio target senza dover sempre muovere l'intero veicolo; che sia in grado di seguire persone indipendentemente da come si presenta il loro aspetto e che tutte queste operazioni vengano svolte in real-time con una buona reattività, a causa della dinamicità del contesto.

La ricerca è quindi partita dal basso secondo un approccio progettuale bottom-up: si è realizzato il veicolo/robot, si è quindi motorizzato la telecamera e infine si è dotato il robot di tutto il software necessario a svolgere le routine di base. Nel seguito è stato realizzato il software adibito al tracking e al controllo di più alto livello.

Le caratteristiche del soggetto da seguire verranno identificate e memorizzate durante una prima fase di set-up, poiché il sistema da noi progettato non possiede a priori una rappresentazione interna del modello da identificare. Durante la prima fase verrà quindi creato un modello estraendo dalla regione di interesse caratteristiche salienti utili al riconoscimento e all'inseguimento dell'oggetto nel contesto delle immagini successive. Durante il tracking tuttavia il modello necessiterà di modifiche: i cambiamenti di luce e

la natura non statica o rigida del target renderanno infatti necessario un'aggiornamento continuo del modello nel tempo poiché la sua rappresentazione varierà sensibilmente.

Il sistema così realizzato verrà utilizzato per indagare l'interazione uomo-macchina di un robot autonomo impiegabile in situazioni di servizio come: guidare persone all'interno di costruzioni, trasportare carichi oppure effettuare consegne all'interno di edifici.

Ringraziamenti

Innanzi tutto desidero ringraziare Simone, per avermi sopportato durante questi lunghi mesi di preparazione della tesi (ed anche prima), senza il quale inoltre questa tesi sarebbe stata, con buona probabilità, alquanto differente. Non posso poi evitare di ringraziare Veronica, e Giambattista, per l'aiuto fornitomi e per avermi dato gli spunti che mi hanno permesso di imparare i diversi argomenti utili alla progettazione e alla realizzazione di questo lavoro. Chiaramente è doveroso ringraziare i miei genitori ed i miei famigliari, per avermi permesso di arrivare a questo traguardo incoraggiandomi e credendo in me.

È pur inevitabile un caloroso ringraziamento a tutte le persone conosciute all'AirLab, con cui ho avuto modo di condividere successi e problemi, incertezze e conoscenze, in particolare Paolo, Simone C., Ettore, Silvia, Ahmed, Michele F., Gianmaria, Davide, etc...

Infine un altrettanto sentito ringraziamento va al professor Andrea Bonarini e all'ingegner Matteo Matteucci per avermi garantito una certa autonomia, pur aiutandomi nei momenti di necessità, e per avermi fornito materiale e conoscenze senza le quali questo progetto ora non esisterebbe.

Matteo

Un primo grazie va a Matteo per avermi ...sopportato durante questo non breve periodo, avere a disposizione un amico disposto ad interminabili discussioni sul nulla e talvolta sul meta del meta del meta di qualcosa (oltre ad una infinita collezione di sfondi) è un contributo impagabile (Molla il freno a mano teo!).

In secondo luogo un grazie va al professor Bonarini e all'ingegner Matteucci, non capita spesso di avere un sorriso, vero, da chi ha l'incombenza di giudicarti (siete stati ...umani).

Non sarei mai sopravvissuto alla seconda settimana di università senza tutti gli amici che mi hanno circondato lungo l'asse Milano, Casorate Primo e Pavia. A chi tra questi fa parte della schiera degli ingegneri dedico una

mia massima “l’ingegnere o impazzisce o molla”, spero che noi si mantenga alto l’onore della categoria.

In ordine sparso citerò: direttamente dall’airlab, il capo Paolo, l’attendente il buon Fanetti, la nostra gradita presenza femminile Silvia, Simone the Little One, il dispensatore di saggezza Ettore, Ahmed delle piramidi, l’instancabile Michele, Giammaria l’arredatore di interni, Ciccio Zainetto Cartella, lo storico gruppo Lace, Bedda, Vince e Guse, l’olandese Francesco low energy, luca less, quel metallaro del legna e Fulvio il terzo moschettiere piacentino, etc...

Agli amici di lunga data voglio solo dire che di loro non avrei mai potuto fare a meno, scusate, ma non sono in grado di fare un lungo elenco, con inevitabili errori di precedenze e omissioni, abbiate con me ancora un pò di infinita pazienza.

L’abbraccio più grande va ai miei genitori, sorelle, nonni e a tutti i miei parenti, che mi hanno permesso di arrivare alla meta dei miei studi, credendo in me in ogli istante, e dandomi la possibilità di arrivare con tanta serenità all’inizio di un nuovo percorso.

Per concludere, nella modesta speranza di diventare, da grande, un “aggettivo” vi saluto e ringrazio tutti.

Simone

Indice

Capitolo 1

Introduzione

1.1 Inquadramento generale

In molte moderne applicazioni robotiche è indispensabile che il robot sia in grado di tenere traccia dello spostamento di oggetti interessanti nella sua area operativa. Il fine è quello di non costituire un pericolo per le persone e di essere sempre più un mezzo di ausilio e supporto nella vita dell'uomo. La capacità di muoversi in un ambiente non predisposto alla sua presenza, non strutturato ed in cui le informazioni iniziali sono scarse è compito ancora più arduo. Inoltre, al crescere del grado di indipendenza richiesto al robot, cresce il numero di competenze necessarie a condurre un progetto razionale ed efficace. Esistono allo stato attuale alcuni prototipi in grado di operare in ambienti antropici come: uffici [1, 51], ambiti sanitario/assistenziali [56, 48], situazioni museali [54], e grandi magazzini [6, 24]. Consideriamo quindi fondamentale per un robot di questo tipo la capacità di riconoscere oggetti, in particolare persone, e navigare in maniera sicura nel loro stesso ambiente.

Il problema del tracciamento di oggetti mobili o persone in ambienti popolati è un argomento trattato in letteratura da diversi lavori. Questo obiettivo è realizzato raccogliendo e interpretando le informazioni fornite dai sensori di bordo e combinando tecniche di visione artificiale con rilevazioni effettuate da sensori sonar e infrarossi. In letteratura sono stati sviluppati vari algoritmi e varie architetture per ottenere questo risultato, ma tutti si basano sullo stesso principio: estrarre un modello della generalizzazione di oggetti da seguire, compararlo con la percezione dei sensori e adattare il modello per riutilizzarlo al passo successivo. Per generare un modello affidabile e coerente, negli anni sono state esplorate diverse tecniche e nella maggior parte dei casi sono state utilizzate tecniche di visione artificiale.

Alcune di esse sono basate sull'identificazione del colore della pelle [27], altre sul colore dei vestiti [56, 9], altre ancora sono basate sul riconoscimento di texture notevoli [26, 56] e sull'estrazione dei contorni [49, 28]. Infine esistono approcci basati sulla rilevazione di suoni [16] o che prevedono l'applicazione sulla persona di target artificiali [42, 20].

La nostra soluzione si basa sull'integrazione di diversi metodi. In questo lavoro abbiamo combinato un rapido metodo di riconoscimento di colori con un algoritmo per corner-detection, un algoritmo per il flusso ottico di semplici feature e un ulteriore algoritmo per il matching di template. Alcuni di questi algoritmi sono più veloci di altri, la politica adottata è stata di utilizzare algoritmi computazionalmente più complessi sulla preselezione realizzata da quelli più veloci o semplicemente applicati in precedenza. Ad ogni ciclo di elaborazione, ovvero ad ogni frame, la rappresentazione interna del target da seguire viene quindi aggiornata in base al risultato degli algoritmi eseguiti, in modo da aggiornare il modello, rendendolo robusto alle deformazioni e ai cambiamenti di luce. Proprio l'interazione di metodi eterogenei rappresenta il contributo innovativo finalizzato alla soluzione del problema di tracking.

Lo scopo della tesi è quindi quello di realizzare un robot mobile su ruote dotato di camera motorizzata a due DOF e un algoritmo di visione basato sulle librerie opensource di visione artificiale openCV[1]. L'algoritmo di visione permetterà all'utente di selezionare una persona nel campo visivo del robot, e al robot di distinguerla e seguirla in un ambiente chiuso.

Inoltre è nostra intenzione testare l'effettivo funzionamento dell'algoritmo sul robot mobile preparato al fine di condurre dei test sul campo. Tali test sono fondamentali non solo per renderci conto delle reali performance della nostra soluzione, ma anche per valutare l'impatto dell'interazione tra un potenziale assistente robotico e l'ambiente umano. I test sono stati condotti sempre su target in movimento, ma in diverse situazioni: prima più semplici ovvero con il robot fermo; in seguito con la sola telecamera in movimento e infine con l'intero veicolo camera e target in movimento. Si è notato che all'aumentare della complessità del sistema e del coordinarsi dei movimenti tra base mobile e telecamera il software di riconoscimento ha incontrato qualche difficoltà in più nell'inseguimento del target. Comunque, nei test condotti tra interno ed esterno del laboratorio, abbiamo ottenuto buoni risultati, sia con luce artificiale, sia con luce solare, sia in presenza di alternanza delle due.

[1] http://sourceforge.net/projects/opencvlibrary/

1.2 Breve descrizione del lavoro

Come spesso accade nella robotica sperimentale la realizzazione di un robot risulta un'esperienza pluridisciplinare. Nella progettazione e nella realizzazione bisogna tenere conto di elementi di meccanica, elettrotecnica, elettronica, programmazione, intelligenza artificiale e nel nostro caso anche degli elementi specifici della visione artificiale. Possiamo quindi collocare il nostro progetto a cavallo tra la robotica mobile e la visione artificiale.
Per quello che riguarda l'aspetto di robotica mobile si è realizzato un veicolo dotato di sensori di distanza e prossimità che, rilevando gli ostacoli, permettono a questo di identificare spazi liberi per il movimento. Si sono quindi affrontati i basilari problemi di navigazione di un veicolo mobile.

Per quanto riguarda invece la visione artificiale ci siamo occupati di strategie per il tracciamento di oggetti non rigidi. In particolare si è indagato il tracciamento di oggetti in movimento tramite camera mobile montata su piattaforma mobile, affrontando quindi problematiche di object detection e object tracking. Il campo di analisi risulta complesso proprio in funzione del fatto che non esistono punti di riferimento fissi nell'immagine e che non esiste un modello rigido da identificare. Scendendo nello specifico dal punto di vista elettronico e meccanico ci siamo occupati della messa a punto del veicolo a ruote (con architettura differential drive), che comprende un resistente telaio con due potenti motori dotati di encoder e la relativa scheda di controllo con interfaccia di comunicazione seriale. In seguito di siamo occupati della motorizzazione di una camera tramite servomotori, della realizzazione della apposita scheda di controllo su cui è montato un pic 16F877A, nonchè del software necessario a quest'ultima. Alla fine di questa prima fase abbiamo ottenuto una piattaforma mobile alta circa un metro in grado di muoversi anche a velocità sostenute, dotata di una torre motorizzata a due gradi di libertà alla cui sommità è montata una camera con risoluzione 640x480 che realizza filmati a 15 fps. A questa configurazione sono stati aggiunti un sensore di distanza ad ultrasuoni e tre sensori di prossimità ad infrarossi. La combinazione di questi sensori con l'apparato di visione ha reso la navigazione del robot sicura e robusta agli imprevisti.

Ultimato l'apparato elettronico ed il protocollo di comunicazione tra pc e scheda, ci siamo dedicati alla realizzazione del software per il processing delle immagini. Nello sviluppo di questo software ci siamo appoggiati sulle librerie di visione openCV.

Il problema di tracking di una persona è stato affrontato da diversi punti di vista e attraverso diverse tecniche: tracciamento del colore, identificazione di feature notevoli, flusso ottico, selezione e riconoscimento di template nel-

l'immagine e infine identificazione di geometrie notevoli tipiche della figura umana come ad esempio l'identificazione del volto. Dapprima si sono testati gli algoritmi separatamente, in seguito è stata realizzata un'interfaccia che permetta di integrare tra loro le varie funzionalità e garantisca un utilizzo trasparente di tutti i settaggi degli algoritmi implementati.

L'interazione di queste diverse tecniche per la visione artificiale, applicate ad un sistema fisico, costituisce il contributo creativo grazie al quale si sono ottenute buone prestazioni per l'obiettivo prefissato. I test hanno evidenziato un comportamento affidabile e robusto, il sistema è infatti resistente a parziali occlusioni dell'oggetto da seguire. Inoltre, i risultati sono buoni anche nel caso di repentini cambi di direzione sia a basse che ad alte velocità.

Avendo condotto test in condizioni ottimali per la ripresa (frame rate elevato, luminosità costante, immagini nitide ad alto contrasto e basso rumore), il sistema non perde l'obiettivo del tracking e non lo confonde con elementi dello sfondo. Ciò non toglie che possano esistere situazioni più problematiche per l'impiego del sistema. Ad esempio al diminuire dell'area su cui è calcolato il modello da inseguire, la probabilità di errore aumenta, poiché il sistema potrebbe non essere in grado di estrapolare un contenuto informativo sufficiente da piccole regioni.

Ad ogni modo il sistema consiste in una parte di un progetto sperimentale molto più ampio. Vi sono diversi passi che possono essere sviluppati per aumentarne il grado di integrazione in un ambiente popolato. Va indubbiamente affrontato il problema della navigazione e della autolocalizzazione utilizzando sensoristica più complessa, che potrebbe prevedere scanner laser o una seconda camera, oppure adottando soluzioni di tipo visual slam. Infine va migliorato il livello di interazione tra la macchina e l'uomo affinché tale rapporto diventi il più naturale possibile. In questa ottica la comprensione del linguaggio naturale, argomento di dibattimento ancora aperto e in continua evoluzione, fornirebbe a sistemi come questo una capacità strategica nell'interattivita con esseri umani. L'idea di camminare al fianco di un simpatico e instancabile assistente robotico ha quindi compiuto i sui primi passi e non è più lontana dall'essere a portata di mano.

Lo scopo di questa tesi consisterà nell'indagare un aspetto di questo disegno. Nel seguito mostreremo come si è realizzato il robot mobile su ruote, la camera motorizzata a due DOF e l'algoritmo di visione e inseguimento di oggetti basato su openCV.

1.3 Struttura della tesi

La tesi è strutturata nel modo seguente.

Nel capitolo due si mostra lo stato dell'arte includendo i principali lavori noti in letteratura nell'ambito della robotica mobile e della visione artificiale. In particolare, si parla di interazione uomo-macchina nell'ambiente umano e relative applicazioni commerciali e non. Quindi si presenta il problema del tracking nella visione artificiale ed infine si illustrano le librerie di visione disponibili nel panorama open source.

Nel capitolo tre si illustra l'impostazione del problema di ricerca, si considerano le problematiche da affrontare e si analizzano gli algoritmi noti valutandone il funzionamento e le caratteristiche di interesse al nostro scopo.

Nel capitolo quattro si chiarisce come sono stati utilizzati gli algoritmi descritti nel capitolo precedente e come il software del sistema sfrutta l'unione di diverse soluzioni per raggiungere l'obiettivo prefissato.

Nel capitolo cinque si mostra il progetto dell'architettura hardware del sistema in maniera modulare, descrivendo i singoli componenti, il loro interfacciamento ed il funzionamento del firmware.

Nel capitolo sei si descrivono i test effettuati e se ne presentano i risultati, commentati nell'ottica di dimostrare la corretta funzionalità del sistema.

Nel capitolo sette si riportano le conclusioni, riassumendo le valutazioni critiche e le prospettive future di ciò che è stato realizzato per mostrare come proseguire nell'ambito della ricerca intrapresa.

Capitolo 2

Stato dell'arte

2.1 Interazione uomo-macchina

Il campo di maggiore impiego dei robot è indubbiamente quello industriale, in questo campo, infatti, esistono innumerevoli prodotti commerciali professionali. Solitamente, tali robot si muovono in ambienti molto strutturati o addirittura predisposti alla loro presenza, raramente è a loro richiesto di interagire direttamente con delle persone. Tuttavia anche in questo campo, soprattutto per complessi compiti di assemblaggio, alcuni progetti si muovono verso l'obiettivo di una più intensa cooperazione tra la macchina e l'operatore umano [30].

Un altro campo in cui la cooperazione tra uomo e macchina è ormai molto sviluppato, ed in cui esistono diversi prodotti commerciali, è quello biomedico: sempre più spesso in sala operatoria i semplici strumenti sono sostituiti da complessi robot guidati dalla mano del chirurgo in grado di aumentare le possibilità dell'operatore umano [44]. Anche in questo caso però si tratta di robot disegnati per specifici ambienti, e per lo più non autonomi.

Esistono innumerevoli esempi di robot che svolgono compiti "per noi", non esistono invece robot commerciali in grado di vivere "con noi" e di svolgere mansioni per noi naturali ma estremamente complesse per una macchina. Non si considera la robotica per intrattenimento o i prototipi non commerciali costruiti da istituti di ricerca o da grandi aziende spesso al solo scopo promozionale. Tra queste mansioni ci sono sicuramente la comprensione del linguaggio naturale, anche non verbale, e la capacità di

vivere nel nostro stesso contesto senza essere un pericolo o un impaccio, ma al contrario essendo di aiuto e supporto.

2.2 La visione artificiale

In quest'ambito consideriamo lo strumento della visione come uno degli strumenti più potenti ed economici da fornire ad un robot: oltre ad essere il mezzo che noi stessi usiamo massicciamente per creare una rappresentazione della realtà, è ormai disponibile un vasto insieme di apparecchiature a basso costo e buona affidabilità per realizzare sistemi di visione artificiale di interesse.

Purtroppo, ancora oggi i sistemi visivi di molti esseri viventi hanno prestazioni irraggiungibili da qualsiasi macchina. Ad esempio basti pensare alla potenza di calcolo necessaria per gestire un sistema visivo complesso come quello dell'uomo. Non a caso ben il 30% circa dei neuroni (circa 60 miliardi) che compongono un cervello umano è dedicato all'elaborazione delle informazioni visive. Dai numerosi tentativi compiuti dall'uomo per comprendere i principi alla base della visione naturale sono emersi negli anni altrettanti progressi per tentare di imitarla a livello computazionale.

I sistemi di visione in tempo reale richiedono una potenza di calcolo elevata che, fino a non molti anni fa, era difficilmente garantita con la tecnologia a disposizione. L'accesso a calcolatori sempre più potenti ha mitigato il problema, ed ora, anche semplici PC di uso comune dal costo contenuto permettono di realizzare elaborazioni in tempo reale con soddisfacente rapidità. Questo ha portato ad un utilizzo sempre più diffuso degli algoritmi di visione e allo sviluppo di tecniche e librerie dedicate al settore della computer vision.

Nonostante il continuo sviluppo, il confronto con il contesto reale risulta ancora una sfida. La notevole quantità delle informazioni percettive provenienti da questi sistemi va infatti interpretata e portata verso un livello superiore di astrazione. Questi livelli comprendono sistemi di controllo ad alto livello, di pianificazione del moto e moduli di intelligenza artificiale che a loro volta si sono rivelati computazionalmente molto onerosi. Se la somma dei tempi superasse una certa soglia, le richieste operative potrebbero non essere soddisfatte. Risulta necessario quindi trasformare il flusso video in una rappresentazione logica basata su alcune caratteristiche, strettamente dipendenti dall'applicazione che si sta sviluppando.

2.3 Il tracking

Il problema del riconoscimento e del tracciamento di un oggetto, tramite visione artificiale, è un argomento trattato in diversi lavori e da diversi gruppi di ricerca in questi anni. Prenderemo ispirazione da questi lavori per realizzare il nostro scopo e analizzeremo le varie soluzioni considerandone pregi e difetti. I diversi approcci proposti in letteratura possono essere divisi tra quelli a basso livello veloci e robusti, ma carenti nell'effettuare raffinate classificazioni dei soggetti, e quelli ad alto livello in grado di tracciare figure complesse, purtroppo lenti e poco robusti. Tra quelli a basso livello possiamo considerare il blob tracking, la sottrazione dello sfondo e gli svariati corner detector, mentre tra quelli a più alto livello annoveriamo le tecniche basate su template matching e riconoscimento di forme complesse, come ad esempio il riconoscimento del viso umano.

2.3.1 Tracking per camere fisse e sottrazione dello sfondo

Alcune delle soluzioni utilizzate per realizzare l'inseguimento e il riconoscimento di persone e oggetti vengono dal campo della video sorveglianza per evitare intrusioni indesiderate o per la sorveglianza del traffico [10]. In generale si tratta di telecamere stazionarie posizionate in punti strategici di edifici o incroci pericolosi, ma è il caso di segnalare alcuni veicoli mobili indipendenti dotati di telecamera di recente commercializzazione come OFRO e prodotti simili venduti anche in Italia [33].

Nel caso di camere stazionarie, ovvero ferme, il problema di tracciamento è più semplice: la camera non segue l'uscita di scena del soggetto e in alcuni casi è possibile considerare lo sfondo come elemento stabile dell'immagine. Allo stesso tempo la camera non è in grado di posizionarsi in modo vantaggioso per identificare la figura intera ed in alcuni casi vengono considerate solo alcune porzioni della figura intera come il volto, gli occhi [13] o piedi [23].

Altri lavori riguardanti camere stazionarie derivano dalla ricerca nel campo della realtà virtuale e degli effetti di computer graphic. In questi casi il soggetto si muove su sfondi statici e talvolta uniformi, l'identificazione del target è realizzata tramite tecniche di sottrazione dello sfondo [45]. Si considera infatti sfondo qualsiasi pixel invariato tra un frame e il successivo e per sottrazione di immagine tra due frame successivi si costruisce un immagine che contiene i soli pixel che sono variati tra un'immagine e la successiva. Combinando diverse immagini di questo tipo si ottiene quella che è definita la motion history image o MHI. In questi approcci è spesso richiesto che la

persona obiettivo dell'operazione di tracking non sia occlusa alla vista della camera, ovvero che che nessun oggetto si frapponga tra target e camera [47] Questo tipo di architettura non è funzionale alle nostre esigenze poiché il robot deve seguire la persona e contemporaneamente muoversi verso di essa rimanendone ad una ragionevole distanza.

2.3.2 Il colore, blob detection

Il primo e più diffuso approccio all'object tracking consiste nell'utilizzare caratteristiche cromatiche nell'aspetto del target, che lo distinguono dallo sfondo. La tecnica si avvale dell'identificazione di un colore o di un istogramma colore caratteristico da confrontare con l'immagine. Il confronto isola i pixel corrispondenti al modello, che sono enfatizzati in base al loro grado di corrispondenza. I primi lavori in questa direzione consideravano un singolo colore sulla base del quale segmentare l'immagine. Approcci più moderni invece prevedono di calcolare un completo istogramma colore e di confrontarlo con l'immagine, in questo modo la classificazione risulta più precisa, in grado di apprezzare diverse sfumature di colore e non solo una singola tinta. Dopo il confronto il risultato è solitamente sottoposto ad algoritmi di clustering in modo da identificare le zone a massima densità di pixel corrispondenti al colore.

Come esempio degli innumerevoli lavori in questo campo possiamo citare [38, 27, 8] per l'uso del colore allo scopo di ricercare il colore del viso. Citiamo invece [36] per l'utilizzo del colore nel tessuto dei vestiti unito a test statistici per l'identificazione di generici blob colore ed infine [4] per l'utilizzo di blob multipli. Di particolare interesse è inoltre l'approccio di [50] che combina l'utilizzo dell'istogramma colore con l'identificazione di contorni (edge) nell'immagine. Tale approccio irrobustisce il tracciamento del target, ma aumenta inevitabilmente il carico computazionale dell'algoritmo. Più recente è invece l'interessante lavoro di [32] che tratta il problema di estrarre il movimento della persona dal confronto di due immagini da camere indipendenti non calibrate.

La più forte limitazione di questi approcci riguarda la necessità che la persona debba indossare colori differenti dallo sfondo o, come nel caso di riconoscimento del volto, debba presentarsi frontalmente alla camera. Inoltre, i cambiamenti di luce causano seri problemi agli algoritmi basati sul riconoscimento del colore. Una parziale soluzione è proposta da [11]: un sistema robusto alle parziali occlusioni è in grado di sopperire anche alla luce diretta del sole, poiché alcune zone dell'oggetto illuminato scompaiono in seguito alla forte luminosità.

2.3.3 Corner e feature tracking

Ogni sistema di tracciamento basato su punti caratteristici dell'immagine deve poter contare su un affidabile algoritmo di estrazione dei corner.

Deve essere tenuto in considerazione il fatto che non tutti i punti dell'immagine sono zone affidabili da inseguire. Ad esempio, una superficie liscia senza texture evidenti non può dare informazioni sul proprio spostamento. Il bordo di un oggetto invece permette di rilevare lo spostamento solo nella direzione perpendicolare al bordo stesso. Infine in un punto angoloso (o corner), dove è presente un netto cambio di luminosità, possono essere rilevati gli spostamenti su entrambi gli assi.

Un considerevole numero di algoritmi è stato presentato durante gli anni, ed è possibile una classificazione in due gruppi: il primo comprende gli algoritmi che estraggono i bordi nell'immagine e cercano i punti di curvatura massima o dove i bordi si incontrano; nel secondo gruppo troviamo invece gli algoritmi che ricercano gli angoli analizzando direttamente i livelli di grigio dell'immagine. Quest'ultimo approccio ha garantito i risultati migliori: segue ora un elenco dei metodi più usati.

Kitchen-Rosenfeld Questo metodo è uno dei primi riportati in letteratura, [31], ed è stato usato come benchmark in tutti i seguenti studi sull'argomento. L'algoritmo calcola un valore di angolatura data dal gradiente locale di intensità e dalla variazione di direzione del gradiente.

Harris-Stephens Un altro corner detector è quello sviluppato da Harris e Stephens in [14], il quale calcola un operatore d'interesse in base all'autocorrelazione delle immagini. La dimensione della maschera di convoluzione determina il bilanciamento tra la precisione nella localizzazione dei punti e la quantità di rumore escluso.

Smith (SUSAN) Smith in [52] ha sviluppato un estrattore di feature che non utilizza le derivate spaziali, garantendo così un'alta velocità di esecuzione. In esso, ogni pixel dell'immagine è il centro di una piccola maschera circolare. I valori di grigio di ogni pixel all'interno di questa maschera sono confrontati col valore centrale (il nucleo) e quelli aventi intensità simile sono assunti come facenti parte della stessa zona dell'immagine.

Shi-Tomasi L'algoritmo proposto in [15] non parte da una definizione a priori di una buona zona da tracciare, ma parte dall'algoritmo di tracking che (si suppone) verrà utilizzato per estrarre feature ottime da

inseguire. Con questo metodo di selezione le feature estratte saranno quindi le migliori da dare in input al modulo di tracciamento.

2.3.4 Flusso ottico

I primissimi approcci con il flusso ottico risalgono agli anni '50 contestualmente alle prime ricerche nel campo della visione artificiale. In prima istanza è possibile definire il flusso ottico come la rappresentazione delle traiettorie seguite dai punti dell'oggetto tracciato durante il suo movimento rispetto alla telecamera. Questo è quello che viene solitamente visualizzato calcolando il flusso tra due immagini successive.

Si tratta di un insieme di vettori proiettati sul piano immagine, di alcuni punti dell'oggetto, in movimento rispetto all'osservatore [3, 2].

Tali vettori hanno intensità proporzionale allo spostamento dei punti, rapportato all'intervallo di tempo che intercorre tra l'acquisizione di due frame successivi, rappresentano quindi delle velocità. Dall'osservazione del flusso ottico nella scena è possibile fare considerazioni sulla forma di oggetti rigidi basandosi sulla distribuzione spaziale e temporale dello stesso. Infatti la tecnica è spesso usata con telecamere fisse per il riconoscimento di auto, pedoni e passanti [18, 17]. Oltre a considerazioni sulla forma, è possibile fare anche alcune osservazioni sull'orientamento degli oggetti: infatti angoli, contorni e zone di occlusione presentano gradienti di velocità apparentemente considerevoli rispetto ai punti interni alla superficie. L'analisi e la stima del movimento tramite l'utilizzo di camere in movimento risulta però molto più complessa dell'analisi fatta con sensori stazionari.

Esistono due metodi principali per affrontare il problema: la stima basata sulla luminosità, ovvero Gradient Based, e quella basata su corrispondenze discrete, ovvero Matching. Nella prima soluzione il flusso ottico è rilevato dall'interpretazione della variazione di luminosità dell'immagine al passare del tempo. In questo caso, si ottengono vettori di flusso densi, ovvero ad ogni pixel dell'immagine corrisponde un vettore velocità. Esempio di tale approccio sono [2, 39, 25]. La seconda soluzione è invece basata sulle corrispondenze discrete tra elementi costanti di due immagini successive. Tipicamente queste caratteristiche, che vengono considerate permanenti nel tempo, sono: corner, profili o particolari pattern. Questi algoritmi si prestano ad essere implementati su architetture di tipo piramidale dove, ad ogni livello della piramide si analizzano i movimenti a diverse risoluzioni. Le traiettorie trovate ad un certo livello guidano la risoluzione del livello inferiore [40]. Esistono casi di hardware sviluppato ad hoc per la determinazione del flusso ottico mediante un algoritmo di tale classe [3].

Esempi di questo approccio sono forniti dalle ricerche di [35] in cui punti appartenenti alla figura vengono isolati assumendo che questa si muova in modo differente dallo sfondo e da [19] in cui viene calcolato il flusso del centro dell'immagine per estrarre informazioni sulla velocità dell'oggetto da inseguire. Le principali limitazioni all'utilizzo di questo approccio vengono dal fatto che in generale è molto influenzato da errori dovuti alle rotazioni fuori dal piano dell'immagine. La deriva di questi algoritmi li rende adatti al tracking di brevi percorsi ma non affidabili sul lungo termine o a repentini cambi di direzione a meno di appropriate correzioni e filtri.

2.3.5 Template Matching

Il template matching consiste in un confronto diretto tra l'immagine da esaminare e il modello dell'oggetto che si vuole individuare, ovvero il template. La selezione del template deve essere realizzata in modo da facilitarne il ritrovamento. Solitamente vengono scelte particolari zone dell'immagine contenenti caratteristiche facilmente rintracciabili. Il modello viene utilizzato in fase di matching mediante una traslazione su tutte le porzioni dell'immagine, per ognuna delle quali viene valutato il grado di somiglianza con il template [29]. Vengono quindi presi in considerazione tutti i frame della sequenza, frame diversi dal sorgente del template, applicando tecniche statistiche di correlazione.

Questa tecnica si dimostra estremamente flessibile e potente per seguire oggetti con piccole distorsioni tra un frame e il successivo. Per rendere il template robusto alle grandi variazioni nel tempo o adattarlo al tracciamento di oggetti non rigidi esso è aggiornato di continuo. Le regole adottate per realizzare l'aggiornamento sono diverse.

Nel caso in cui il nuovo template sia notevolmente diverso dal precedente si procede ad una completa sostituzione del vecchio con il nuovo, perdendo così tutte le informazioni sul modello originale. Se le variazione tra vecchio e nuovo modello è di piccola entità, viene calcolato un nuovo modello attraverso il calcolo della media tra le due immagini. In questo modo si mantengono le informazioni sul modello originale e si migliora la stabilità del sistema in persenza di disturbi e rumore.

Se invece le variazioni sono molto piccole è conveniente continuare ad usare il template originale. Questa politica è particolarmente importante nei casi in cui la traslazione della template è inferiore al pixel, l'adattamento del modello renderebbe infatti irrecuperabile l'informazione sulle traslazioni subpixel.

Le tecniche di correlazione usate per il template matching consistono nel-

la ricerca della miglior corrispondenza (o della minor diversità) tra una parte di immagine caratteristica e la parte corrispondente nel frame successivo. Le tecniche di correlazione possono essere region-based [41] o feature-based [37]. Questi metodi, noti anche come tecniche di token tracking, sono comunemente estesi allo spazio 3-D per ricostruire il movimento tridimensionale degli oggetti.

2.3.6 Sistemi a visione multipla

Un ulteriore approccio consiste nello sfruttare fortemente le informazioni fornite da sistemi con visione stereoscopica o in generale multipla [12]. Per analogia a molti sistemi visivi naturali questi sistemi consistono nel posizionare due camere a distanza nota l'una dall'altra [22] in modo da acquisire due immagini contemporanee della stessa scena, tra le quali esistono lievi differenze e piccoli spostamenti relativi tra i vari oggetti. Le piccole differenze vengono utilizzate da funzioni di stima, che usano tecniche di triangolazione, per ottenere informazioni sulla profondità dei punti nella scena. La combinazione di informazioni odometriche e considerazioni geometriche sulla distanza e posizione fornite dalle camere, solitamente calibrate, permettono di isolare la persona dallo sfondo e di seguirla facendo anche delle previsioni sulla sua futura posizione. In altri casi, partendo da queste informazioni viene creata una mappa di zone occupate [12] utilizzata per determinare oggetti in movimento mentre, il tracking è ottenuto sfruttando un filtro di Kalman o analoghe tecniche [50].

Il sistema risulta efficiente e affidabile nell'estrarre informazioni di distanza dall'ambiente, utilizzando hardware a basso costo anche nel caso di applicazioni in tempo reale. Per semplificare i passaggi successivi all'acquisizione delle immagini stereo è consigliabile utilizzare camere con lenti di uguale lunghezza focale e sensori con pixel della stessa dimensione. Inoltre è fondamentale che le immagini non abbiano nessun disallineamento temporale, che causerebbe inevitabili errori nel calcolo delle corrispondenze tra le due immagini.

2.3.7 Sistemi basati su target artificiali

È inoltre possibile applicare sulla persona un oggetto artificiale in modo da seguire questo target predefinito. Applicando sulla persona RFID o emettitori di luce infrarosso e/o onde radio, il tracking risulta molto più semplice. Possono esser studiate soluzioni ad-hoc di estrema affidabilità ma comunque artificiali. Noi non ci occuperemo di questo approccio perché obbliga le persone a indossare specifici oggetti ed esula dunque dall'obiettivo di creare un

sistema che si adatti al contesto naturale. Citiamo soltanto per completezza i lavori di [34, 50, 42].

2.4 Altri sensori non basati su visione

2.4.1 Sonar

Spesso, e il nostro caso non fa eccezione, è necessario affiancare al sistema di visione alcuni sensori di distanza. L'utilizzo di tali sensori ha lo scopo di rendere la navigazione più sicura e migliorare la percezione dell'ambiente. Nel nostro caso sono stati utilizzati sensori sonar e ad infrarossi.

I sonar sono sensori che sfruttano la propagazione di onde ultrasoniche per determinare misure di distanza. Nelle applicazioni robotiche sono solitamente utilizzati sonar di tipo attivo ovvero che funzionano da emettitori di segnale non udibile dagli esseri umani, e che dopo l'emissione restano in attesa della sua riflessione. Alla ricezione dell'eco, grazie al calcolo del tempo di volo e della velocità di propagazione dell'onda nell'aria, è possibile risalire ad una misurazione della distanza.

Purtroppo, da soli, questi sensori sono inadatti al tracciamento a causa della scarsa precisione e della difficoltà di ottenere letture non affette da rumore. Spesso sono invece impiegati come sensori di contatto e prossimità [55], oppure nell'ambito della navigazione applicando tecniche probabilistiche [5].

2.4.2 Scanner laser

Discorso simile all'utilizzo di sensori sonar può essere fatto per l'utilizzo di scanner laser. Questi sensori hanno un funzionamento simile ai sonar, ma sfruttano la propagazione della luce ed in particolare di luce laser, non visibile (spesso di categoria 1 non nociva per l'uomo). Questi scanner ottengono misure di distanza, sfruttando la riflessione dell'impulso emesso su superfici opache. Solitamente i laser effettuano passate per un certo angolo di ampiezza restituendo una serie di misure, dalle quali è possibile ottenere le distanze dei vari oggetti nella scena. Purtroppo questi sensori, benché affidabili e precisi, hanno un costo ancora elevato, anche 100 volte superiore a quello di una camera, inoltre sono ingombranti e hanno un peso piuttosto elevato. Tali considerazioni li rendono adatti ad applicazioni più complesse e con a disposizione maggiori risorse.

Ciò non vuol dire che in letteratura non esistano applicazioni che fanno uso di tali sensori per il tracciamento di ostacoli, per la creazione di

mappe per la navigazione o per il tracciamento e inseguimento di persone. In questi casi la presenza di target è rilevata tramite la comparsa di minimi locali negli istogrammi di distanza rilevati. Il loro tracciamento è realizzato calcolando il flusso di questi minimi [21]. Purtroppo questi minimi possono essere causati non solo da persone ma anche da oggetti statici dell'ambiente, è quindi necessario confrontare successive misurazioni applicando filtri probabilistici. In particolare è possibile combinare i vantaggi offerti dalle tecniche di approssimazione della densità basata sui campioni con l'efficienza di filtri JPDAF (Joint Probabilistic Data Association Filter) svincolandosi dalla necessità di utilizzare filtri di Kalman, riuscendo cosi a rappresentare distribuzioni arbitrarie e non solo gaussiane [50].

2.5 Librerie per l'elaborazione di immagini

Tra tutte le librerie di visione esistenti sono state prese in considerazione esclusivamente le librerie Open Source con un buon livello di sviluppo e un grande numero di utenti. Inoltre per massimizzare la riusabilità del codice un requisito gradito sarebbe anche la portabilità, in modo da garantire che il codice possa esser eseguito su una qualsiasi macchina. Bisogna comunque considerare che avendo a disposizione una macchina con sistema operativo GNU/Linux e processore Intel, questo influenzerà indubbiamente la nostra scelta.

2.5.1 XVision2

Questo sistema è stato sviluppato alla Yale University tra il 1993 e il 1998, ora lo sviluppo continua alla Johns Hopkins University a cura del professor Greg Hager. È comunque frutto di un solo gruppo di sviluppatori, con alcuni collaboratori esterni, per la verità non molto numerosi. Questo influisce sul supporto e sulla buona documentazione che accompagnano la libreria, rendendoli scarsi e di difficile reperibilità. È sviluppata in C++ ed in modo da favorire la creazione di applicazioni interattive che rendano semplice l'utilizzo da parte dell'utente.

Il sistema è formato da una serie di classi: quelle di base definiscono le strutture elementari e l'interfaccia standard, quelle a più alto livello definiscono le funzionalità di edge tracking, region tracking, blob tracking ecc...

A suo favore si deve aggiungere che comprende un'interfaccia MatLab, utile in molti progetti e che garantisce ottime prestazioni nella segmentazione in tempo reale e nel tracciamento.

2.5.2 VXL

VXL è l'acronimo di Vision Understanding Environment e si tratta di una collezione di più librerie scritte in C++. L'obiettivo di queste librerie è di essere leggere e portabili. Questo sistema è stato sviluppato da ricercatori di General Electrics e da diverse università tra cui Oxford, Manchester, Leuven e Otago. Il sistema, di struttura modulare, si compone di cinque librerie principali e di varie librerie aggiuntive dedicate a specifici problemi. Inoltre sono supportate varie piattaforme, compilatori e applicazioni di vario genere. Questo progetto è supportato da una larga comunità che fa ben sperare nella sua robustezza e vitalità. Purtroppo per la sua stessa natura, che vede un elevato numero di contributi di differenti soggetti, soffre di una lacunosa documentazione.

Il grosso problema nell'utilizzo di queste librerie è proprio la difficoltà di individuare le funzionalità e caratteristiche adeguate al lavoro che si sta svolgendo data anche la vastità del progetto.

2.5.3 TINA

TINA, che non è l'acronimo di "Tina Is No Acronym", è parte del programma IST finanziato dalla comunità europea. È composta da una serie di librerie di base con funzioni per la geometria, per operazioni di basso livello sulle immagini digitali e specifiche per l'analisi di immagini mediche. Su questa base si sviluppano diversi progetti. L'obiettivo è quello di fornire funzionalità a tutto campo nell'analisi di immagini: manipolazione, statistica, funzionalità ad alto livello per la visione artificiale e funzionalità per l'ambito sanitario

Tina però, almeno allo stato attuale, non fornisce il supporto e le funzionalità a noi necessari.

2.5.4 LTI-Lib

Si tratta di una libreria ad oggetti sviluppata in C++ all'università di Aachen e fornisce strutture dati comuni al campo della visione artificiale, oltre ad alcuni algoritmi e alcune funzionalità di classificazione. È stata testata sia in ambienti Linux che WindowsNT ma non ha però dimostrato un grado di efficienza accettabile al nostro contesto, dinamico e real time

2.5.5 CMVision

CMVision è sviluppata alla Carnegie Mellon University da James Bruce. La libreria è progettata per tutte quelle applicazioni robotiche in tempo reale

che sfruttano principalmente le informazioni sul colore. Questo progetto è piuttosto popolare nell'ambito della robotica e non è disegnato per un preciso hardware, il che lo rende anche portabile. L'idea di CMVision è quella di segmentare l'immagine in zone di colore uniforme avvalendosi di tecniche di sogliatura.

È pensata per situazioni indoor focalizzate su specifici problemi ed è stata utilizzata soprattutto nell'ambito RoboCup.

2.5.6 Gandalf

Gandalf è una libreria C per lo sviluppo di applicazioni di visione artificiale. Contiene numerosi algoritmi di uso comune, in special modo per l'ottimizzazione. È strutturata in modo da permettere un efficiente uso della memoria, riconfigurando dinamicamente le strutture, sfrutta molta elaborazione di basso livello, evitando astrazioni di livello superiore per ottenere vantaggi dal punto di vista della velocità computazionale.

È stata utilizzata per lo sviluppo di Mokey, uno strumento professionale per la postproduzione video. La mancanza di funzionalità di alto livello adeguate ai requisiti del sistema, tuttavia, ha portato a scartare questa libreria.

2.5.7 OpenCV

OpenCV[1] è una collezione di funzioni C/C++ supportata da Intel per la visione artificiale. Questa libreria è orientata principalmente alla visione in tempo reale utilizzando delle routine ad alte prestazioni con basso spreco di memoria.

Queste prestazioni sono ottenute anche grazie alla ottimizzazione per architetture Intel, realizzata sfruttando tra l'altro la tecnologia MMX di cui la gran parte di processori Intel è ad oggi dotata. Questo tipo di ottimizzazione rappresenta tuttavia una limitazione alla portabilità del codice su macchine con architetture differenti. Allo stesso tempo, considerando l'hardware a nostra disposizione, l'ambito di applicazione attuale e futuro e il fatto che il 90% delle macchine del laboratorio monta architettura Intel, più che uno svantaggio, questo tipo di ottimizzazione, è sembrato un vantaggio. OpenCV è inoltre compatibile con IPL (Image Processing Library) sempre di Intel: una libreria commerciale non open source che effettua operazioni a basso livello su immagini digitali. Gli algoritmi ad alto livello sono invece

[1] Non si confonda OpenCV con l'omonimo progetto che si riferisce ad un sistema open source per la creazione di curriculum vitae on line.

il focus di OpenCV che comprende infatti: tecniche di calibrazione, feature detection, tracciamento tramite flusso ottico, analisi sulla geomentria dei contorni, identificazione del movimento, ricostruzione 3D, riconoscimento dei gesti della mano e segmentazione dell'immagine. La presenza di queste funzioni ad alto livello permette di affrontare il problema ad un livello di astrazione alta fin dalle prime fasi del progetto.

Infine, il progetto è fiorentemente e attivamente sviluppato da Intel con il supporto di una comunità di utenti ampia e attiva. L'effetto di questo interesse si concretizza in una documentazione ufficiale non sempre eccellente, ma completata da vivaci forum e mailing list. Tra le librerie prese in esame è questa quella che presenta la migliore documentazione e i migliori test ed esempi che coprono molte delle funzionalità. Per queste ragioni è stata scelta nello sviluppo del presente progetto.

Capitolo 3

Impostazione del problema di ricerca

In questo capitolo verranno presentati gli strumenti e le tecniche utilizzati per comporre il sistema, dei quali vengono illustrate le principali caratteristiche e le ragioni della loro scelta. Nei capitoli successivi ne verrà invece mostrato l'utilizzo e come sono state sfruttate le loro modalità di interazione, sia per quanto riguarda la parte software (Capitolo 4) sia per quanto riguarda l'hardware (Capitolo 5).

3.1 Identificazione di caratteristiche notevoli dell'immagine

L'obiettivo della ricerca è quello di costruire un sistema in grado di osservare il movimento di un oggetto o persona, in modo tale da poterli seguire. Il primo problema consiste quindi nell'interpretare le informazioni fornite dal sensore di visione, isolando caratteristiche utili ad identificare l'area di interesse nella scena.

Le caratteristiche notevoli di un immagine permettono di quantificare alcune caratteristiche di un oggetto differenti rispetto all'intera immagine. Si tratta di estrarre, da alcune grandezze dell'immagine, attraverso funzioni più o meno complesse, zone o punti distinguibili dal resto della scena.

Vengono utilizzate tecniche di estrazione di caratteristiche a basso e alto livello.

3.1.1 Caratteristiche a basso livello

Per caratteristiche a basso livello si intendono le singolarità indipendenti dal contesto dell'immagine, applicabili quindi ad una qualsiasi immagine valida. L'identificazione di questi elementi è associata ad un'elevata rapidità di calcolo, anche grazie a funzioni particolarmente ottimizzate. Esse sono generiche e quindi molto adattabili, ma proprio a causa della loro flessibilità non assolvono completamente al compito di identificare univocamente uno specifico oggetto nella scena.

Nel caso in esame si sono scelte le seguenti funzionalità.

Ricerca di blob colore: si tratta di caratterizzare, attraverso l'analisi di ogni singolo pixel (sono infatti spesso definite pixel level), il colore di punti o zone di interesse, coinvolgendo allo stesso tempo metodi di sogliatura e clustering.

Individuazione dei contorni: vengono considerate le feature calcolate attraverso la segmentazione dell'immagine tramite algoritmi di individuazione dei contorni (Local Feature). Poiché ben si prestano all'utilizzo in ambienti chiusi, caratterizzati da contorni netti e distinti.

Corrispondenza di aree note: dette anche tecniche di "template matching", si tratta di campionare una zona dell'immagine particolarmente significativa per poi ricercarla nei fotogrammi successivi. Con questa tecnica è possibile caratterizzare in modo ancora più robusto un'immagine campione, ottenendo il grado di correlazione del template iniziale con aree delle medesime dimensioni, all'interno di una sequenza di frame successivi.

3.1.2 Caratteristiche di alto livello

Per contro le caratteristiche ad alto livello assolvono meglio all'identificazione univoca di soggetti nella scena, ma sono più onerose dal punto di vista computazionale. Sono infatti calcolate sull'intera immagine attraverso analisi di distribuzioni matematiche e statistiche, talvolta con l'impiego di classificatori. Esse sono dipendenti dal contesto dell'immagine e dall'applicazione considerata, ovvero la loro presenza è subordinata all'ambito in cui viene effettuata la ripresa. Sono dette ad alto livello perché la loro applicazione deve essere comunque coadiuvata dalla presenza di tecniche di basso livello provenienti dal gruppo elencato nel paragrafo 3.1.1.

Una loro interessante applicazione consiste nella rilevazione di oggetti specifici, proprio per questo rivestono un ruolo particolarmente importante

nel sistema realizzato, come punto di riferimento nella rilevazione di volti umani.

Nell'applicazione progettata verranno utilizzate tecniche appartenenti ad entrambi i gruppi, partendo da quelle a basso livello fino ad arrivare all'utilizzo di quelle ad alto livello.

3.2 Blob detection

Nell'area della visione artificiale, "blob detection" si riferisce all'insieme delle modalità con cui si possono individuare punti, o regioni in un immagine, dotati di luminosità maggiore o minore delle aree circostanti. Vi sono due classi principali di riconoscitori di blob:

- Metodi differenziali basati su espressioni derivative
- Metodi basati su estremi locali di intensità nell'ambiente

Utilizzando una più recente terminologia, questi operatori sono anche conosciuti come riconoscitori di punti o regioni di interesse.

Vi sono diversi motivi per cui la rilevazione di blob è stata considerata. Una principale ragione è poter disporre di informazioni accessorie, riguardo a regioni di un immagine, che non possono essere ottenute da rilevatori di contorno o di punti salienti. Pertanto nell'applicazione la rilevazione di blob viene usata per riconoscere regioni di interesse, utilizzate nelle successive elaborazioni.

Come si può notare in figura 3.1, queste regioni possono mostrare la presenza di oggetti o parti di essi nel dominio dell'immagine, con applicazione nel riconoscimento o tracciamento di oggetti. In altre applicazioni, come l'analisi di istogramma, i descrittori di blob possono essere usati per rilevare punti di picco, con applicazione nella segmentazione. Un altro utilizzo comune dei riconoscitori di blob è come principale primitiva per l'analisi e il riconoscimento di texture in una immagine.

Recentemente i descrittori di blob sono stati sempre più utilizzati per segnalare la presenza di caratteristiche ad alto contenuto informativo, per il riconoscimento di oggetti basato su statistiche locali dell'immagine. A corredo delle informazioni fornite da un rilevatore di blob, vi è anche la possibilità di segnalare variazioni di dimensione e di inclinazione degli oggetti.

Figura 3.1: esempi di blob detection per il colore rosso in due scene una di esterno e l'altra di interno

3.3 Definizione e uso di contorni o punti angolosi

Un edge, ovvero contorno, è una zona dell'immagine che presenta una forte discontinuità nel gradiente di luminosità in una sola direzione, si tratta di confini e margini tra zone diverse appartenenti alla medesima immagine.

Per analogia, un corner, ovvero un punto angoloso, è invece una zona dell'immagine che presenta una forte discontinuità nel gradiente di luminosità in più di una direzione.

L'importanza dei corner sta nella loro buona resistenza ai cambiamenti di illuminazione, caratteristica particolarmente utile per operazioni di tracking in un contesto dinamico. È inoltre preferibile utilizzare, come punti caratteristici, dei corner e non degli edge in funzione della rilevazione del moto tramite flusso ottico.

3.3.1 "Aperture Problem"

Gli edge, infatti, produrrebbero delle condizioni di ambiguità nel momento in cui si decidesse di valutare il moto lungo direzioni diverse dalla normale allo spigolo rilevato. Il problema, noto anche come "aperture problem", è chiaramente inquadrato nei due esempi mostrati in figura 3.2.

Nel primo esempio, malgrado la figura e il suo perimetro compiano

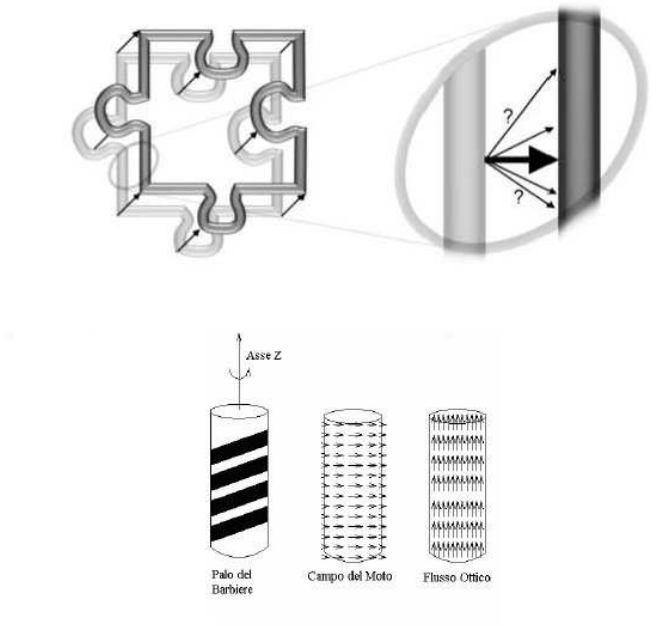


Figura 3.2: Casi notevoli del problema di apertura

un movimento rigido ben evidente, nella zona ingrandita (lato destro in figura) si può notare come sia impossibile identificare univocamente un generico punto di edge che potrebbe corrispondere ad infiniti punti dopo lo spostamento.

Il secondo esempio si riferisce all'illusione ottica del "palo del barbiere", in questo caso gli spigoli disegnati in colore scuro, ruotando sull'asse verticale, danno l'illusione di muoversi all'infinito verso l'alto. Anche in questo caso, nonostante il campo del moto si disponga in un senso, l'inseguimento dei punti di edge rende assolutamente incoerente il calcolo del flusso ottico.

Al contrario, per un corner la direzione del moto è univocamente determinata, infatti le informazioni su tali punti permetteranno di identificarli più facilmente in modo univoco durante la fase di tracking.

3.4 Estrazione dei contorni

Le principali tecniche di estrazione dei contorni (edge) sono basate sull'analisi del gradiente di luminosità dell'immagine. Il gradiente consiste nel vettore delle derivate parziali lungo le direzioni principali:

$$\nabla I[x,y] = \left[\frac{\delta I[x,y]}{\delta x}, \frac{\delta I[x,y]}{\delta y}\right]$$

Consideriamo un immagine in scala di grigio come una matrice di scalari a due dimensioni. I contorni sono definiti anche come le regioni dell'immagine in cui la superficie vista dalla telecamera presenta una discontinuità nella profondità oppure nell'orientamento della normale alla superficie.

Queste discontinuità sono rese molto bene dal passaggio alle derivate parziali che identificano la variabilità del segnale lungo gli assi dell'immagine. In corrispondenza di forti variazioni, la derivata assume valori di modulo elevato; laddove invece il segnale è costante, il modulo della derivata si annulla.

Il calcolo delle derivate, come da definizione, può essere sviluppato utilizzando il semplice rapporto incrementale.

$$\frac{\delta I[x,y]}{\delta x} = \lim_{\Delta x \to 0} \frac{I[x,y] - I[x+\Delta x, y]}{\Delta x} \Rightarrow I[x,y] - I[x+1,y]$$

$$\frac{\delta I[x,y]}{\delta y} = \lim_{\Delta y \to 0} \frac{I[x,y] - I[x, y+\Delta y]}{\Delta y} \Rightarrow I[x,y] - I[x,y+1]$$

Tuttavia, a causa di problemi di simmetria rispetto al punto di applicazione e di robustezza al rumore, il calcolo di tali derivate viene effettuato con tecniche diverse, che utilizzano filtri digitali come operatori (filtri di Prewitt e quello di Sobel, vedi figure 3.3).

3.4.1 Prewitt

Per calcolare in modo rapido il gradiente orizzontale, verticale e diagonale, Prewitt ha proposto queste maschere:

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

-1	-1	0
-1	0	1
0	1	1

1	1	0
1	0	-1
0	-1	-1

Le prime due maschere calcolano il gradiente in orizzontale e verticale, mentre la terza e quarta enfatizzano le componenti in diagonale dell'immagine. La risposta della maschera è calcolata secondo le modalità usate per il filtraggio spaziale.

3.4.2 Sobel

Sobel propone invece una versione un pò migliore di quella di Prewitt, dando più peso al pixel centrale:

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

-2	-1	0
-1	0	1
0	1	2

2	1	0
1	0	-1
0	-1	-2

Tali filtri producono immagini in cui i contorni sono ben definiti, ma purtroppo non direttamente utilizzabili per l'estrazione dei segmenti. Essi compongono infatti edge estremamente frammentati e, a causa del rumore dal quale sono particolarmente influenzati, evidenziano anche una non trascurabile presenza di segmenti spuri. Nel caso in cui però non si sia interessati a considerazioni geometriche sui contorni, ma sia di interesse procedere nella direzione di estrarre corner caratteristici dall'immagine, per utilizzarli come feature notevoli di una zona inquadrata, tali considerazioni sono da valutare in secondo piano ed in funzione delle elaborazioni successivamente effettuate. Tali tecniche di estrazione dei contorni sono infatti solamente la base di quasi tutte le tecniche di riconoscimento di corner notevoli (ovvero di Local Feature).

3.5 Rilevatore di Harris per i corner

L'estrattore di corner che è stato utilizzato è quello di Harris, noto anche come Harris Corner Detector. Si basa sulla definizione di Local Structure Matrix C_s:

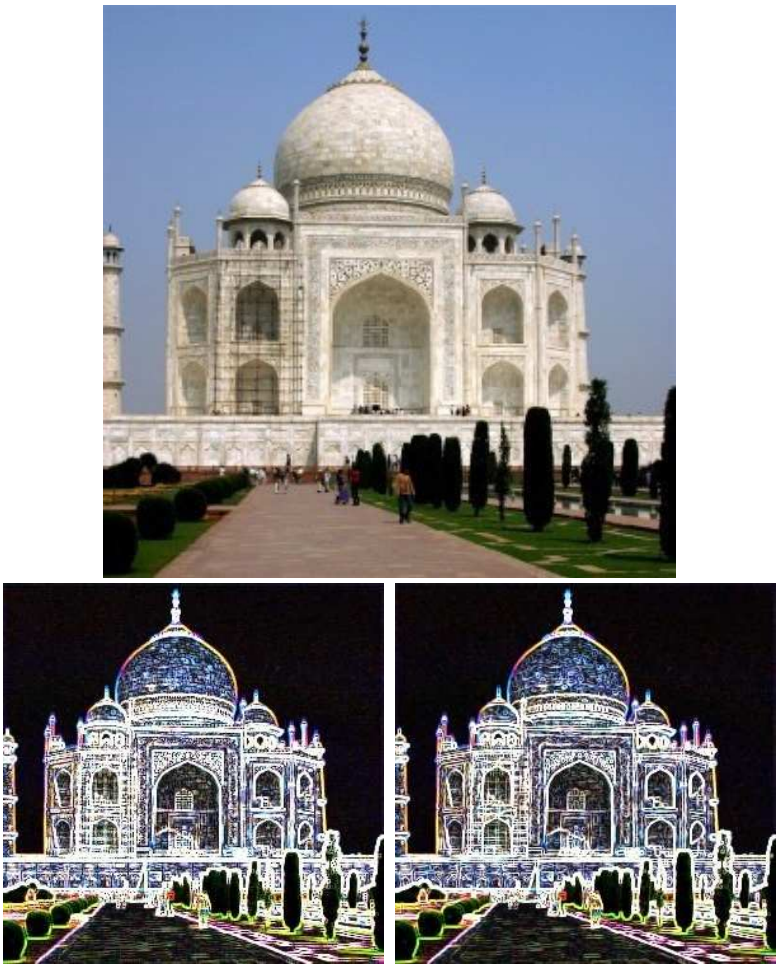
$$C_s = \omega_G(r, \gamma) * F$$

$$\text{F}= \begin{vmatrix} f_x^2 & f_x & f_y \\ f_x & f_y & f_t^2 \end{vmatrix}$$

La matrice C_s è formata dai prodotti delle derivate parziali lungo le due direzioni dell'immagine, calcolati lungo le due direzioni principali per ogni punto. Tale matrice è per definizione simmetrica e definita positiva, è possibile quindi diagonalizzare tale elemento nella forma:

$$C_s= \begin{vmatrix} \gamma_1 & 0 \\ 0 & \gamma_2 \end{vmatrix} \text{con } \gamma_1 \geq \gamma_2 \geq 0.$$

In cui gli autovalori γ_1 e γ_2 sono non negativi.



(a) Filtro di Prewitt (b) Filtro di Sobel

Figura 3.3: L'applicazione dei filtri di Prewitt e Sobel

Questa matrice viene poi combinata con un filtro gaussiano ω_g o con filtri più leggeri dal punto di vista computazionale attraverso l'operatore di convoluzione.

Tramite l'analisi punto per punto di entrambi gli autovalori della matrice C_s è possibile fare alcune considerazioni sull'immagine.

Se gli autovalori sono entrambi nulli allora la porzione d'immagine considerata è perfettamente uniforme e il punto in esame non appartiene né ad un edge né ad un corner.

Se invece il punto appartiene ad uno spigolo (edge), l'autovalore γ_1 è positivo mentre l'autovettore γ_2 è nullo, questo implica che ci troviamo in un

Figura 3.4: Un esempio di corner identificati dall'estrattore di Harris

massimo locale della variazione di luminosità ma lungo una sola direzione. In particolare la variazione massima ha andamento perpendicolare all'autovettore associato a γ_1. Se infine il punto appartiene ad un corner, entrambi gli autovettori risultano essere positivi e non nulli.

Si osservi inoltre che un corner non è altro che l'incontro di due spigoli, quindi un corner può essere considerato un doppio punto di edge in cui sia γ_1 che γ_2 non sono nulli, bensì di modulo rilevante.

Sulla base delle precedenti osservazioni, Harris definisce una funzione in grado di misurare l'intensità del corner trovato.

$$H(x,y) = detC_s - \alpha(traceC_s)^2$$

$$detA = \prod_i^N \gamma_i$$

$$traceA = \sum_i^N \gamma_i$$

da cui si ottiene :

$$H = \gamma_1\gamma_2 - \alpha(\gamma_1 + \gamma_2)^2$$

dove α è un parametro e $H \geq 0$ se $0 \leq \alpha \leq 0.25$.

Viene quinidi rilevato un corner quando $H(x, y) \geq H_t$.

H_t è il parametro di soglia che insieme ad α determina la robustezza del corner. In particolare, per valori di α piccoli (generalmente $\alpha \leq 0,10$), si otterranno pochi corner, ma molto definiti e robusti ai cambiamenti di luce, mentre per valori più alti, si otterranno molti più corner ma meno affidabili. Inoltre, durante l'estrazione dei contorni di Harris, è importante tenere traccia della posizione relativa dei corner estratti. È infatti auspicabile che tali punti mantengano una minima distanza.

L'impostazione di tali parametri va fatta per via euristica, sulla base di risultati sperimentali. Attraverso un'operazione di sogliatura è quindi possibile ottenere i migliori corner nell'immagine. A tale proposito, per estrarre i migliori N corner nell'immagine, vengono isolati i corner che presentano i maggiori autovalori e che si trovano ad una minima distanza tra loro (OpenCV offre una funzione specifica che sfrutta proprio l'estrattore di Harris).

3.6 Flusso ottico

Dalla analisi di immagini 2D si estrae come parametro del movimento il flusso ottico, ovvero un'approssimazione del moto in 3D nell'immagine bidimensionale.

Le zone in movimento sono infatti date dalla proiezione del movimento 3D dei punti nella scena osservata sul sensore, tramite il sistema camera. Poiché solo il movimento apparente può essere estratto da una sequenza di immagini 2D, sono necessarie ulteriori assunzioni a priori per un analisi quantitativa. Alcune semplificazioni vanno fatte sui cambi di luminosità, sulle proprietà degli oggetti e sulle relazioni tra il movimento nella scena 3D e la sua proiezione sul sensore 2D. Un semplice esempio è in grado di chiarire il concetto.

Si consideri una sfera rigida con superficie omogenea riflettente, in rotazione attorno ad un'asse passante per il centro della stessa. Se la superficie non presenta texture evidenti e l'illuminazione rimane costante, il risultato di optical flow è nullo, ovvero la sfera risulta apparentemente ferma. Se invece una sorgente di luce si muove attorno ad essa, gli apparenti cambiamenti di illuminazione sarebbero erroneamente attribuiti ad un movimento della sfera.

Le sequenze di immagini reali, fortunatamente, non offrono spesso situazioni di questo tipo, ma presentano comunque alcune problematiche, meno

speculative ma pur sempre importanti. I problemi più frequentemente incontrati nell'analisi di sequenze reali riguardano le sovrapposizioni di movimenti multipli, le occlusioni, i cambi di illuminazione e i movimenti non rigidi.

Per questo è sconsigliabile l'uso diretto delle informazioni di optical flow, ed è quindi consigliabile utilizzare queste informazioni grezze in maniera qualitativa piuttosto che quantitativa. Tali problemi non impediscono comunque di utilizzare il flusso ottico, dato che, nella maggior parte dei casi, le problematiche accennate non coinvolgono l'intera area dell'immagine. È quindi possibile l'applicazione di tale tecnica anche nel caso di sistemi di tracciamento con camera in movimento, come il presente.

La conoscenza a priori di alcune feature degli oggetti presenti nella scena aiuta a raffinare la tecnica, ad esempio l'estrazione dei bordi dell'immagine può aiutare a definire meglio i margini delle zone in movimento, che non sempre risultano affidabili con il calcolo del flusso ottico.

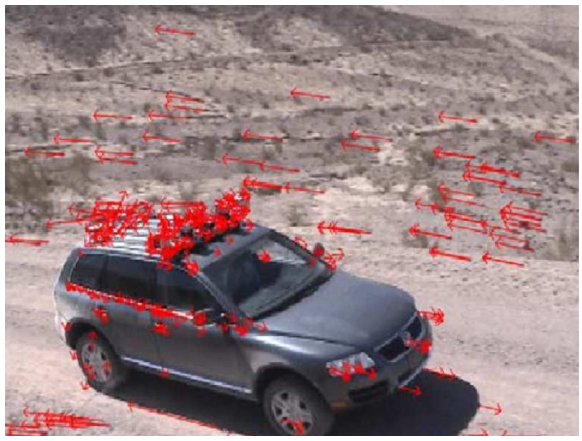


Figura 3.5: Un esempio di calcolo del flusso ottico per alcuni corner in un immagine in movimento

3.6.1 Lucas-Kanade

Limitando il calcolo del flusso ottico a punti di specifico interesse, selezionati a partire da requisiti specifici, si focalizza l'inseguimento solo nelle porzioni di immagine che effettivamente apportano informazioni sul movimento. I punti estratti dalle immagini, ovvero i corner, sono quindi ricercati nella sequenza di frame consecutivi da un algoritmo iterativo.

La procedura che viene considerata è quella descritta da Lucas-Kanade

[53], proposta nel 1981. Si tratta di un algoritmo iterativo di ottimizzazione non-lineare, basato sulla minimizzazione della somma dei quadrati.

I principali passi dell'algoritmo con riferimento alla figura 3.6 sono:

1. Filtra l'immagine I con la finestra W(x; p) per calcolare I(W(x;p))
2. Calcola l'errore dell'immagine T(x) - I(W(x;p))
3. Filtra il gradiente ∇ I con W(x;p)
4. Valuta lo Jacobiano $J\frac{\partial W}{\partial p}$ in (x;p)
5. Calcola la massima discesa di $\nabla I \frac{\partial W}{\partial p}$
6. Calcola la matrice Hessiana usando l'equazione $H = \sum_x \left[\nabla I \frac{\partial W}{\partial p}\right]^T \left[\nabla I \frac{\partial W}{\partial p}\right]$
7. Calcola $\sum_x \left[\nabla I \frac{\partial W}{\partial p}\right]^T [T(x) - I(W(x;p))]$ per aggiornamento dei parametri interni
8. Calcola $\Delta p = H^{-1} \sum_x \left[\nabla I \frac{\partial W}{\partial p}\right]^T [T(x) - I(W(x;p))]$
9. Aggiorna il parametro p sommando Δp appena calcolato
10. I processo procede fino a che $|\Delta p| \leq \epsilon$

L'obiettivo é quello di minimizzare la differenza residua tra due finestre W di pixel, una nell'immagine J e una nell'immagine I. La minimizzazione è calcolata derivando la funzione di differenza ed espandendola successivamente con la serie di Taylor.

$$\epsilon = \int_W [J(x) - I(x - d)]^2 dx$$

La derivazione di questa equazione può essere realizzata in diversi modi, per maggiori informazioni sulla tecnica utilizzata si rimanda a [53].

Al termine della minimizzazione della funzione di differenza è necessario controllare l'effettivo allineamento delle due porzioni di immagine, basandosi appunto sul calcolo della differenza residua. Le due componenti chiave di questo algoritmo di tracking sono l'accuratezza e la robustezza.

L'accuratezza si riferisce alla precisione locale nel matching tra le feature: intuitivamente una piccola finestra W(x; p) permette di non tralasciare dettagli contenuti nell'immagine. La robustezza è invece collegata alla sensibilità del tracking rispetto ai cambiamenti di luminosità e dalla quantità di moto da rilevare nella sequenza di immagini.

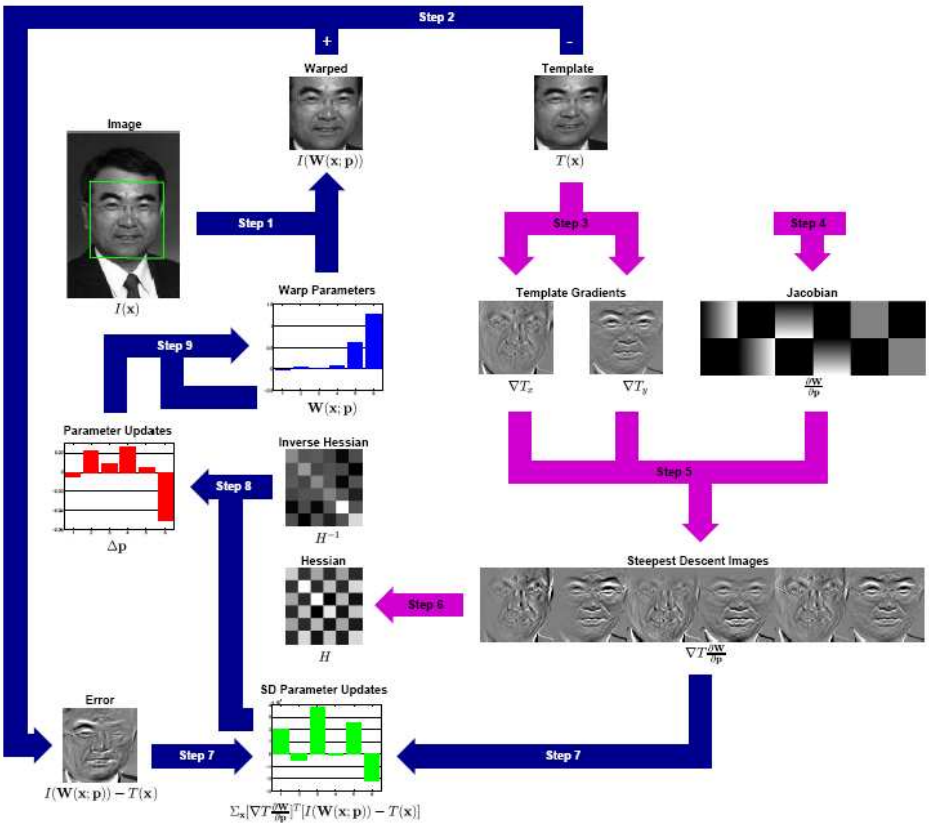


Figura 3.6: I passi dell'algoritmo di Lucas-Kanade

È auspicabile arrivare quindi ad un buon compromesso tra i termini di precisione e robustezza, soprattutto in relazione alla scelta delle dimensioni della finestra di integrazione, in modo tale da rilevare comunque movimenti ampi dei corner senza abbassare troppo la precisione nell'inseguimento. La soluzione adottata per ovviare a questo problema sta nella rappresentazione piramidale delle immagini all'interno dell'algoritmo di Lucas-Kanade.

3.6.2 Rappresentazione piramidale

La rappresentazione piramidale di un'immagine si ottiene partendo dalla risoluzione e dalle dimensioni iniziali tramite due passaggi.

Il primo passo consiste nell'applicare un filtro gaussiano con deviazione standard pari al valore di scala. Ciò permette di ottenere una riduzione continua nella risoluzione dell'immagine, applicando contemporaneamente anche un efficace filtraggio al rumore dei pixel nella scena. Il secondo pas-

so consiste invece nel selezionare un pixel ogni n (tipicamente ogni due), in modo da ottenere un'immagine di dimensioni ridotte rispetto alla precedente (tipicamente un quarto dell'area originale). La dimensione del filtro applicato è detta finestra di riduzione (reduction window), il valore di n è chiamato invece fattore di riduzione (reduction factor).

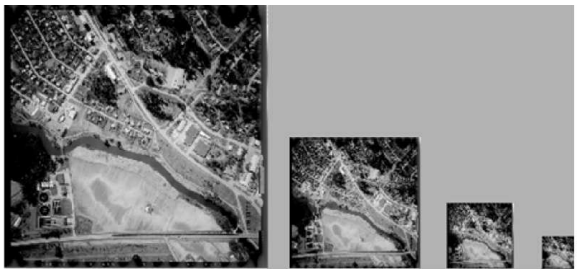


Figura 3.7: Rappresentazione piramidale a quattro livelli di una ripresa aerea

Successive applicazioni di questo processo generano una pila di immagini a risoluzioni e dimensioni decrescenti, che è possibile immaginare come in figura 3.8. L'altezza della piramide supera raramente i due o tre piani, infatti oltre un certo livello le immagini diventano estremamente piccole, e non avrebbe significato procedere oltre appesantendo il calcolo, dato che, ad ogni livello è necessario ripetere l'algoritmo.

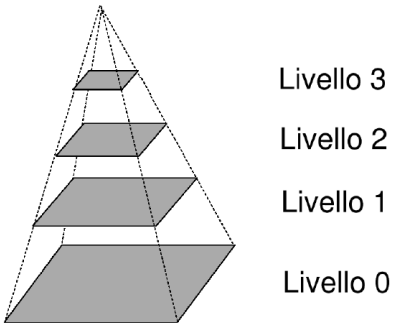


Figura 3.8: La piramide di immagini si parte da una finestra di integrazione piú ampia per arrivare a una piú stretta

Il flusso ottico è quindi calcolato al livello più alto della piramide, per l'immagine più piccola, e successivamente la stima ottenuta è utilizzata nel

livello sottostante. Partendo da questa stima iniziale, il livello di accuratezza viene migliorato passando al livello sottostante e propagando il raffinamento del flusso ottico verso il basso. Ad ogni passaggio si ripete l'algoritmo di Lucas-Kanade fino a raggiungere il livello zero, ossia l'immagine originale, determinando così definitivamente la stima del flusso.

Al livello più alto della rappresentazione piramidale è inoltre soddisfatto il vincolo che impone a gruppi di pixel adiacenti di avere lo stesso flusso. Procedendo poi verso i successivi livelli, la funzione ϵ di differenza, calcolata partendo da già accurate stime iniziali, converge velocemente verso la sua minimizzazione. In conclusione l'accoppiamento dell'algoritmo di Lucas-Kanade insieme alla rappresentazione piramidale garantiscono non solo un accurato calcolo del flusso ottico, ma anche un ottimo compromesso tra accuratezza e robustezza. L'accuratezza è in grado di scendere a livello subpixel, mentre il filtraggio gaussiano garantisce la robustezza al rumore.

3.6.3 Difetti dell'algoritmo di flusso

Come accennato nella sezione 3.6, lavorando con sequenze di immagini acquisite da una telecamera nel mondo reale, non è sempre possibile rispettare tutti i vincoli per l'ottimo funzionamento dell'algoritmo di inseguimento.

Le prestazioni sono infatti fortemente influenzate dalle condizioni di ripresa. È necessario mantenere un frame rate elevato, in modo da garantire spostamenti relativi dei corner non troppo ampi, che porterebbero a rilevazioni errate. Inoltre il modello di Lucas-Kanade adottato ammette solo traslazioni delle feature tra immagini consecutive, non considerando effetti di scalatura e rotazione. Questo influisce sui bruschi cambiamenti di prospettiva per i quali il sistema non è in grado di fornire un comportamento adeguato. Sarebbe possibile introdurre nel sistema ulteriori parametri, in modo da tenere conto di questi tipi di trasformazioni, ma il carico elaborativo andrebbe a discapito dell'utilizzo dell'applicazione in un contesto real-time. Infine, come è ovvio, è preferibile lavorare con telecamere in grado di produrre immagini con bassi livelli di rumore, con sequenze nitide, ad alto contrasto e con illuminazione costante.

Il difetto che crea i maggiori problemi, ed effettive ripercussioni sulle prestazioni del sistema, riguarda i corner inseguiti non correttamente. Un cattivo inseguimento avviene nel momento in cui una feature è classificata come corretta, ma in realtà si posiziona in un punto dell'oggetto o dello sfondo diverso da quello in cui era stata selezionata.

Ciò avviene per corner deboli, tipicamente rilevati in zone a basso gradiente, in cui texture e colore risultano pressochè uniformi. In questo caso

il tracciamento non sarà affidabile e probabilmente si discosterà dal moto dell'oggetto di cui faceva parte, andando a fermarsi su punti dello sfondo o su punti non appartenenti all'oggetto.

Un secondo caso di cattivo tracciamento riguarda le situazioni di occlusione, che si verificano nel momento in cui la porzione dell'oggetto al quale la feature era associata non è più visibile, per esempio perché l'oggetto ha subito un rotazione su un suo asse o per la presenza di altri corpi occludenti. Se la ricerca di Lucas-Kanade trova nelle vicinanze una regione con gradiente simile, la feature non viene scartata ma verrà meno la corrispondenza tra i corner di due frame successivi.

3.7 Corrispondenza di Template

Le tecniche di template matching si basano sulla localizzazione di un'area, campionata da un immagine sorgente, in sequenze di immagini successive tramite tecniche di correlazione.

Si tratta di riconoscere un oggetto per il quale si è memorizzato un template, ovvero verificare che nell'immagine sia presente un'istanza dello stesso. Siccome non si conoscono a priori le regioni in cui l'istanza può presentarsi, è necessario confrontare il modello con tutte le sottoparti dell'immagine che hanno le stesse dimensioni del template.

A questo scopo, l'immagine modello viene fatta scorrere sequenzialmente sull'intera immagine, valutando per ogni possibile posizione la similarità tra il template e la regione in esame.

3.7.1 Fasi dell'algoritmo

La tecnica di template matching si svolge in tre principali fasi.

Estrazione del template

Nella prima fase viene estratto il template ovvero una porzione dell'immagine originale. Questa zona va scelta con attenzione, deve infatti essere un sottoinsieme stretto e significativo dell'immagine totale, in modo da facilitare l'identificazione di zone simili a quella data. Inoltre la zona campionata deve avere dimensioni né troppo grandi né troppo piccole. Tale compromesso deriva dal fatto che zone ridotte difficilmente possono contenere informazioni sufficienti alla loro identificazione univoca. Inoltre costringono l'algoritmo a una lunga serie di confronti tra molte zone dell'immagine con il template. Al contrario, ampie zone dell'immagine risultano troppo generiche e costringeranno l'algoritmo a poche iterazioni, ma piuttosto lente.

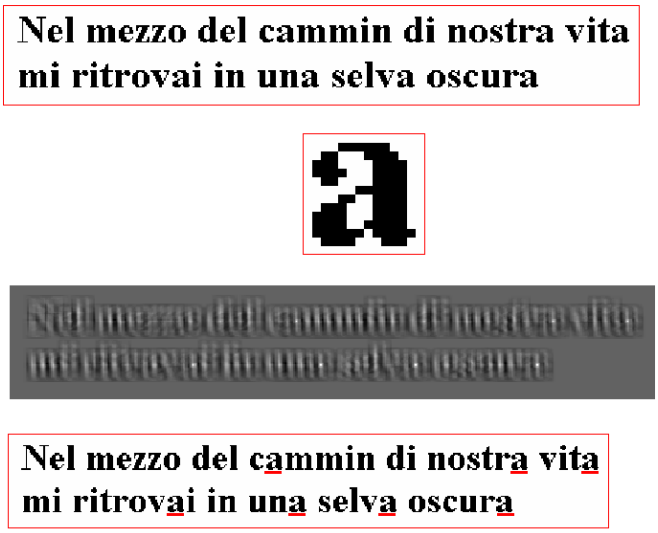


Figura 3.9: Un esempio di template matching, dall'alto verso il basso: immagine di un testo sorgente, ingrandimento del template (la lettera "a"), risultato dell'algoritmo ovvero matching table e identificazione delle zone a maggiore corrispondenza.

Dimensionamento della matching table

La fase successiva alla scelta del modello consiste nel ritrovamento di zone nell'immagine che appaiono simili al template. Riassumendo, data un'immagine di dimensioni W x H e un template di dimensioni w x h, l'output dell'algoritmo consiste in una tabella di scalari di dimensioni W-w+1 x H-h+1, detta tabella di matching. In questa matrice ogni pixel alla locazione (x,y) caratterizza la corrispondenza tra il template e una zona rettangolare dell'immagine delle stesse dimensioni, avente l'origine nelle coordinate (x,y). La "somiglianza" può essere calcolata utilizzando diverse tecniche, solitamente vengono utilizzati i metodi di:

- Square difference, normalizzato e non
- Cross correlation, normalizzato e non
- Correlation coefficent, normalizzato e non

In questo caso si è utilizzato il calcolo del coefficiente di correlazione normalizzato. Possiamo scrivere con I(x,y) la luminosità del pixel in (x,y), appartenente all'immagine, e con T(x,y) la luminosità del pixel nella locazione (x,y) del template. Il coefficiente di correlazione normalizzato si

presenta come:

$$R(x,y) = \frac{\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1}\tilde{T}(x',y')\tilde{I}(x+x',y+y')}{\sqrt{\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1}\tilde{T}(x',y')\sum_{y'=0}^{h-1}\sum_{x'=0}^{w-1}\tilde{I}(x+x',y+y')^2}}$$

in cui $\tilde{T}(x',y') = T(x',y') - \overline{T}$

e $\tilde{I}(x+x',y+y') = I(x+x',y+y') - \overline{I}(x,y)$ dove $\overline{I}$ e $\overline{T}$ indicano i valori medi di luminosità dei pixel nell'immagine e nel template.

Ricerca della corrispondenza

L'ultima fase consiste nella localizzazione del punto di massimo nella matching table e nell'aggiornamento del modello, ovvero nell'adattamanto del template alle situazioni dinamiche di tracking. Per la ricerca del massimo valore all'interno della tabella di matching è possibile analizzare in modo completo l'intera tabella, identificando il punto di massimo globale. In alternativa è possibile affidarsi ad una ricerca locale, partendo da una soluzione locale, che spesso coincide con la posizione originaria del template nel frame sorgente. La ricerca locale risulta più veloce ed efficiente, nonché robusta, per la soluzione di tracking real-time poiché evita di commettere grossi errori di localizzazione. Questo vale soprattutto nel caso di movimenti non estremamente rapidi rispetto al frame rate.

3.7.2 Problematiche e adattamenti

I principali problemi di questa tecnica riguardano la sua applicazione in caso di oggetti deformabili che compiono movimenti non rigidi. L'implementazione considerata non risulta robusta agli effetti di scalatura e rotazione degli oggetti seguiti. In questo caso, sarebbe possibile introdurre dei termini che tengano conto delle deformazioni degli oggetti. In particolare si potrebbe applicare al template una funzione che simuli l'effetto di scalature e rototraslazioni, ripetere il passo di matching fino ad ottenere diverse matching table e identificare tra tutte le matrici il punto di massimo assoluto. Tale adattamento è però praticabile, fino in fondo, solo in casi in cui non sia richiesto di agire in tempo reale.

3.8 Face detection

Il riconoscimento di volti è un esempio di riconoscimento ad alto livello. Si tratta di uno dei più sviluppati esempi di individuazione di pattern. Se per la descrizione di semplici figure come triangoli, quadrati e cerchi, trovare una rappresentazione matematica è molto semplice, nel caso di figure complesse come un volto, la descrizione non è banale.

3.8.1 Caratteristiche "haar-like"

Una tecnica per il riconoscimento dei volti, disponibile in openCV, sfrutta la rilevazione di oggetti di tipo "Haar-Like". Il nome deriva dal fatto che le feature ricercate sono simili alle "haar wavelets" (teorizzate nel 1909 da Alfred Haar). Le caratteristiche "Haar-Like" utilizzate sono quelle riportate in figura 3.10. Il rilevatore di oggetti che le sfrutta è stato proposto da Viola [43] e ottimizzato da Lienhart [46].

3.8.2 Classificatore e training

Innanzitutto, un classificatore (costituito da una sequenza in cascata di classificatori boosted, basati su caratteristiche haar-like) viene addestrato con alcune centinaia di esempi. Si utilizzano due gruppi di immagini, il cui ordine di grandezza è proporzionale all'accuratezza desiderata per l'identificazione: il primo gruppo rappresenta il campione positivo, ovvero immagini che mostrano l'oggetto da individuare (nel nostro caso volti umani), mentre il secondo gruppo è il campione negativo, dove vengono raccolte le immagini che non raffigurano l'oggetto di interesse.

3.8.3 "Cascade" e "boosting"

Il classificatore è definito "in cascata", o "cascade", perché utilizza un insieme di classificatori più semplici, applicati in maniera sequenziale finché uno dei essi dà esito negativo, oppure vengono tutti quanti superati con successo. La parola "boosted" significa invece che ognuno di questi classificatori è costruito basandosi sul contributo pesato di quattro tecniche di boosting. Le tecniche supportate dalle librerie di visione sono: Discrete Adaboost (DAB), Real Adaboost (RAB), Gentle Adaboost (GAB), Logitboost (LB). I classificatori base sono semplici alberi decisionali dotati di almeno due foglie.

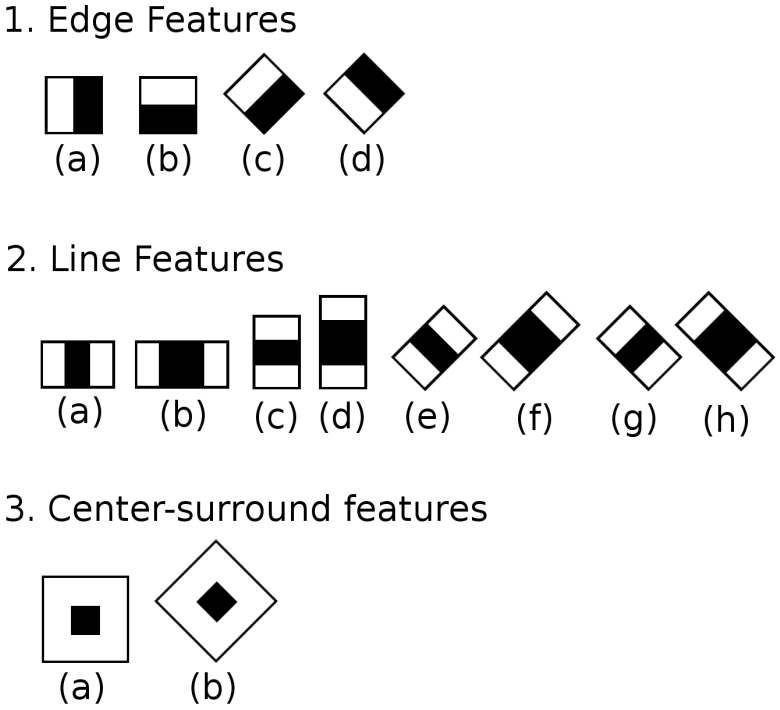


Figura 3.10: I gruppi di feature utilizzati dal classificatore di oggetti haar-like

3.8.4 Calcolo della corrispondenza delle feature

Ogni caratteristica utilizzata dal classificatore è definita per forma, dimensioni e posizione nella regione di interesse. Ad esempio nel caso della caratteristica 2c (figura 3.10), la corrispondenza è valutata attraverso la differenza fra la somma dei valori di luminosità dei pixel di immagine coperti da tutta la caratteristica e la somma dei pixel coperti dalla sola banda nera.

$$Correspondence_{i,k} = w_{i,k,1} * RectSum_{i,k,black,white}(I) + w_{i,k,2} * RectSum_{i,k,white}(I)$$

Dove (i,k) sono le coordinate di un generico punto nella Haar-feature e $I = brightness$

Il risultato è quindi moltiplicato per il rapporto tra l'area totale della caratteristica usata e la banda nera in essa presente per compensarne le diverse dimensioni. Le formule utilizzate sono le seguenti.

$$w_{i,k,1} * Area_{i,k,black,white} + w_{i,k,2} * Area_{i,k,black,white} = 0$$

Per la Haar-feature 2-a si ottiene

$$w_{i,k,2} = -3w_{i,k,1}$$

mentre per la 3-b

$$w_{i,k,2} = -3w_{i,k,1}$$

3.8.5 Ricerca di oggetti o volti

Al termine della fase di addestramento il classificatore può essere applicato su una regione di un'immagine generica, che deve avere le stesse dimensioni delle immagini usate nel campione di training positivo. Esso restituisce un valore positivo o negativo a seconda della presenza o meno dell'oggetto ricercato (ad esempio il volto umano). Per cercare il volto si usa una finestra di ricerca che scorre iterativamente sull'intera immagine o su zone d'interesse. Il classificatore è costruito inoltre in modo tale da poter riconoscere oggetti ridimensionati, il che è più efficiente di riscalare l'immagine completa. Perciò per trovare un oggetto o volto di dimensioni sconosciute la scansione va ripetuta più volte a differenti fattori di scala.

Capitolo 4

Progetto logico dell'architettura software

In questo capitolo viene descritta la parte software del sistema per l'analisi delle immagini e l'inseguimento dell'obiettivo. Nella prima parte le problematiche risolte sono riportate descrivendo il funzionamento e l'interazione dei vari algoritmi utilizzati, già introdotti al capitolo 3. L'ultima parte del capitolo riguarda invece la descrizione dell'interfaccia di controllo del sistema ed il suo utilizzo, nonché una breve descrizione dei suoi principali componenti.

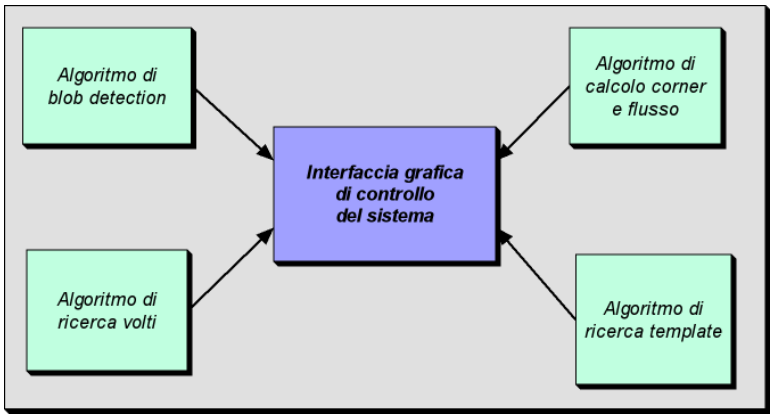


Figura 4.1: Rappresentazione modulare delle funzionalità software e interfaccia utente

4.1 Il rilevamento del blob colore

Uno dei primi problemi affrontati nella realizzazione dell'algoritmo è consistito nel rendere il sistema in grado di individuare una regione in movimento nell'immagine. (Per una rappresentazione grafica dell'algoritmo utilizzato si faccia riferimento alla figura 4.4).

4.1.1 L'individuazione dell'obiettivo

In un primo approccio al problema, la decisione di cosa rintracciare spetta all'utente, che manualmente seleziona la porzione di immagine raffigurante l'oggetto da inseguire. È anche possibile dire al sistema di individuare i volti presenti nell'immagine e selezionare automaticamente il busto della persona, la cui faccia sia stata riconosciuta per prima. Ad ogni modo, il risultato di questa prima operazione è lo stesso in entrambi i casi: l'oggetto da osservare viene circoscritto da una selezione rettangolare di cui sono note posizione e dimensioni.

Per quanto riguarda l'identificazione dell'oggetto nei fotogrammi successivi, diventa necessario poter valutare la distribuzione statistica del colore nella regione selezionata. Questo viene fatto attraverso il calcolo dell'istogramma colore, costruito utilizzando la coordinata HUE dello spazio colore HSV.

Lo spazio colore HSV

Per motivi algoritmici e di efficienza si è scelto di rappresentare l'immagine da filtrare nello spazio colore tridimensionale chiamato HSV (Hue, Saturation, Value). In questa rappresentazione del colore (vedi figura 4.2) le tre coordinate descrivono: la tonalità cromatica (Hue), il grado di saturazione (Saturation) e il valore di luminosità (Value).

Sfruttando tale rappresentazione è possibile filtrare l'immagine molto più efficacemente di quanto si possa fare utilizzando il più comune spazio colore RGB (Red, Green, Blue) dove le varie coordinate esprimono la quantità di colore primario presente nei vari pixel.

4.1.2 Sogliatura dell'immagine

L'algoritmo di riconoscimento colore calcola i valori estremi della selezione nelle coordinate dello spazio colore. Effettua in tal modo una sogliatura per poter escludere dall'immagine tutto ciò che si presenta con colore diverso dall'oggetto di interesse, che pertanto non può essere (a meno di sensibili variazioni di luce considerate più avanti) l'obiettivo da ricercare.

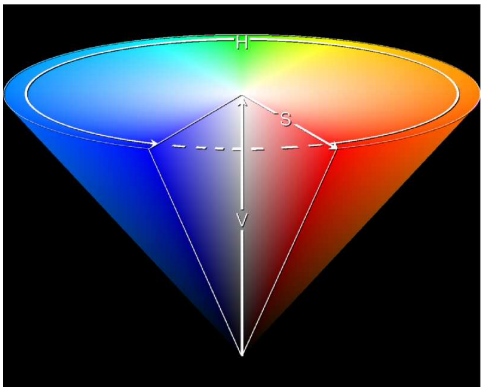


Figura 4.2: Rappresentazione conica dello spazio colore HSV

In un primo passo viene dunque costruita una maschera di filtraggio estrapolando dalla selezione effettuata i valori massimo e minimo di luminosità (V) e il valor minimo di saturazione (S). Questo permette di escludere di fatto tutto ciò che risulta al di sotto della saturazione minima o al di fuori dei valori di luminosità dell'area selezionata. Tale operazione di sogliatura risulta necessaria poiché, come evidenziato in figura 4.2, l'intero spazio HSV può essere rappresentato da un cono, in cui bassi valori di saturazione rendono i valori sull'asse H molto vicini tra loro e poco distinguibili. Similmente alti valori di luminosità V tendono a sbiadire completamente i colori virando verso il bianco, mentre bassi valori di V portano rapidamente verso il nero. In ognuna di queste tre situazioni l'individuazione dei valori di hue, che servirà nel calcolo dell'istogramma colore, risulta molto difficile se non impossibile ed è quindi auspicabile evitarle a priori, grazie alla sogliatura appena descritta.

4.1.3 Il calcolo dell'istogramma colore

A questo punto viene isolato il canale hue dal resto dell'immagine e ne viene calcolato (e visualizzato) l'istogramma colore. In esso l'asse orizzontale rappresenta un valore hue espresso in gradi da 0° a 180°, mentre il valore di ogni barra verticale indica il contributo percentuale di ogni valore di tonalità colore. Poiché l'istogramma viene calcolato sulla base della selezione, i colori presenti sono soltanto quelli all'interno della selezione.

La scala della coordinata HUE, che come si può notare osservando la figura 4.2, è normalmente espressa in gradi sessagesimali tra 0° e 360°, viene però riportata dal grafico in valori tra 0° e 180°. Questo è dovuto semplice-

mente ad una diversa scala utilizzata da OpenCV per motivi di velocità computazionale e, al di là di una leggera approssimazione nei valori, non comporta particolari problemi. (Sono infatti comunque presente tutti i valori di H, l'unica conseguenza è quella di disporre di una risoluzione raddoppiata rispetto alla scala sessagesimale). I valori percentuali delle barre sono visualizzati invece in scala normalizzata, considerando il colore dominante come massimo assoluto dell'istogramma.

L'istogramma permette di considerare la distribuzione di colore, espresso in coordinata hue, dell'oggetto a cui si è interessati. È finalizzato quindi al calcolo del frame di "backprojection", grazie al quale si potrà seguire il target iniziale nelle immagini successive.

4.1.4 Il frame di "backprojection"

Dopo aver calcolato l'istogramma colore, viene calcolato il frame di "backprojection" che raffigura la trasposizione della scena originale in uno spazio colore monodimensionale a toni di grigio. Viene costruito utilizzando l'istogramma colore, calcolato nella sezione 4.1.3, e attribuisce ai pixel dell'immagine un colore tanto più bianco quando più essi corrispondono alla distribuzione di colore riportata dall'istogramma. In termini statistici il valore di ogni pixel nel frame di "backprojection" è la probabilità che quel pixel nell'immagine originale corrisponda alla distribuzione data dall'istogramma.

Dal momento che l'istogramma è calcolato sull'oggetto di interesse, è dunque chiaro che i pixel mostrati in bianco appariranno in corrispondenza dei pixel dell'oggetto selezionato nell'immagine di partenza. Al contrario pixel colorati di grigio sempre più scuro indicheranno aree meno attinenti ai colori dell'obiettivo da ricercare. Per un esempio di quanto affermato si faccia riferimento alla figura 4.3, ove l'istogramma colore, costruito in base alla selezione del vestito blu nell'immagine originale, è mostrato accanto al relativo frame di "backprojection".

4.1.5 Il blob detector

Come già discusso nel capitolo 3 sezione 3.2, i rilevatori blob noti in letteratura si occupano di riconoscere regioni a maggiore o minore luminosità all'interno di immagini. Ecco perché il frame di backprojection è così importante: sarà l'input dell'algoritmo di riconoscimento blob utilizzato.

Prima di allora però il frame viene ulteriormente filtrato con la maschera costruita al punto 4.1.2, in modo da irrobustire ulteriormente la ricerca, eliminando a priori dall'immagine ciò che sicuramente non sarà l'oggetto

Figura 4.3: Esempio di immagine prima e dopo il calcolo del frame di "backprojection" corredata da istogramma colore e soglie di filtraggio S e V

da riconoscere e inseguire. Il frame risultante dal filtraggio viene finalmente utilizzato per l'esecuzione dell'algoritmo di rilevazione blob chiamato CamShift.

L'algoritmo CamShift

Si tratta di un algoritmo di tracciamento di oggetti noto in letteratura [8] e disponibile all'interno delle librerie openCV utilizzate. Il suo nome deriva da "Continuously Adaptative MeanSHIFT" poiché esso si avvale dell'utilizzo di un altro algoritmo, chiamato MeanShift, che ricerca il centro dell'oggetto e ne calcola dimensioni e orientamento, attraverso un certo numero

di approssimazioni successive. Il risultato di questo algoritmo è pertanto una struttura di tipo "connected component" che rappresenta l'oggetto da noi inizialmente selezionato attraverso le sue dimensioni, orientamento e posizione.

4.1.6 Controllo del robot e inquadratura della telecamera

Grazie alle informazioni sul blob da inseguire, è dunque possibile correggere la posizione della telecamera affinché il centro del blob trovato corrisponda al centro dell'inquadratura. Utilizzando il sensore di distanza ad ultrasuoni, montato accanto alla telecamera, sarà possibile avere anche una stima della distanza dell'oggetto e comandare al veicolo di avvicinarsi o allontarsi dall'obiettivo a seconda dei casi.

Sia il controllo della telecamera che il controllo del veicolo vengono realizzati attraverso due semplici controllori rispettivamente di tipo parabolico e lineare. A livello elettrico il comando della telecamera è gestito dalla scheda di controllo servi e lettura sensori, descritta nella sezione 5.2, mentre il movimento del veicolo è gestito dalla scheda AirBoard descritta nella sezione 5.1.2.

4.2 La rilevazione dei punti salienti

Poiché la semplice individuazione del blob potrebbe confondere le diverse regioni con lo stesso colore, è necessario isolare la regione di interesse attraverso l'utilizzo dei punti salienti. L'idea è di identificare piccole regioni caratteristiche all'interno del blob colore e tracciarne lo spostamento.

4.2.1 Ricerca dei buoni candidati

Per poter effettuare quanto descritto è necessario trovare dei punti o delle aree che risultino tracciabili con un buon grado di confidenza. Questo viene fatto attraverso l'utilizzo di una specifica funzionalità presente nelle librerie di visione utilizzate chiamata "GoodFeatureToTrack". Il nome lascia dunque intendere proprio la ricerca di regioni candidate a essere ritrovate nei fotogrammi successivi con un particolare grado di "affidabilità".

Tecnicamente vengono ricercati nell'immagine i punti di corner associati ad autovalori particolarmente elevati, avvalendosi pertanto dell'algoritmo di Harris per la rilevazione di contorni. La ricerca viene infine opportunamente filtrata e sogliata affinché i punti di corner rilevati non siano troppo vicini tra loro. Le feature vengono ricercate all'interno del blob di interes-

se, per limitare il tempo di esecuzione e conseguentemente non appesantire eccessivamente il sistema.

Poiché ad ogni nuovo fotogramma alcuni punti o regioni potranno essere esclusi a causa della diminuizione della loro affidabilità, è necessario ricalcolare un certo numero di nuovi punti candidati ad ogni frame. Il numero massimo di nuovi corner da ricercare influenza di molto le prestazioni del sistema e va quindi dimensionato con cura in funzione delle capacità del calcolatore. Sempre per lo stesso motivo è ovviamente necessario che il numero totale di punti salienti da considerare non ecceda mai un certo valore.

Ad ogni fotogramma vengono perciò ricercati n nuovi punti salienti, che sono aggiunti agli eventuali k punti salienti calcolati nei frame precedenti. La somma n+k continua a crescere finché non supera un valore deciso in fase di testing, nel qual caso il calcolo dei nuovi punti viene inibito finché i vecchi punti non decrescono sotto il massimo valore totale consentito.

4.2.2 Calcolo dello spostamento tramite flusso ottico

Terminata la fase di ricerca dei candidati, si dispone di un certo numero di punti di cui è possibile misurare lo spostamento all'interno dell'immagine tra due frame consecutivi. Viene pertanto utilizzata una funzione adibita al calcolo del flusso ottico attraverso una versione iterativa dell'algoritmo di Lucas-Kanade. Essa calcola le coordinate dei punti salienti nel frame corrente basandosi sulla posizione dei rispettivi punti nel frame precedente.

4.2.3 Esclusione dei punti non più validi

Il risultato viene filtrato dal sistema affinché si considerino i soli nuovi punti salienti che risultino ancora all'interno della regione di colore individuata dal blob detector.

Ciò permette innanzi tutto di escludere i punti salienti erroneamente considerati buoni candidati al frame precedente e costringe a cercare nuovi candidati con cui rimpiazzarli, aumentando il grado di affidabilità dell'insieme di punti salienti.

Data la natura dinamica del sistema, ad ogni frame verranno comunque ricalcolati nuovi candidati di cui tenere traccia ed esclusi i nuovi punti ottenuti non più validi allo scopo dell'applicazione.

4.2.4 Regione dei punti salienti su "backprojection"

L'insieme dei punti salienti trovati fino a questo punto viene dunque utilizzato per costruire una regione rettangolare nell'immagine che racchiude

il target. Considerato il punto saliente a coordinate minime e il punto a coordinate massime, viene costruito un rettangolo che circoscrive tutti i punti presenti e che verrà utilizzato per irrobustire il calcolo del frame di backprojection appena prima della rilevazione blob nel nuovo frame.

4.2.5 Irrobustimento del blob attraverso i punti salienti

Nel momento in cui l'algoritmo di blob detection ha calcolato il frame di "backprojection" (vedi sezione 4.1.4), viene controllata la presenza di punti salienti. Qualora ne esistano, una regione rettangolare che li contiene è stata definita e, a sua volta, è compresa nell'inquadramento del blob colore.

Proprio questa regione viene dunque utilizzata per filtrare ulteriormente il frame di "backprojection". Si costruisce infatti una nuova maschera dove la regione dei punti salienti è perfettamente trasparente (colore bianco), mentre la porzione di immagine restante è colorata con un valore al 25% di grigio.

Poiché la maschera viene sovrapposta al frame di "backprojection", ottenuto dall'istogramma colore filtrato, essa permette di trascurare parzialmente tutto ciò che non presenta punti salienti nell'immagine e fà si che l'algoritmo di CamShift consideri soprattutto la regione su cui persistono dei punti di corner. Dal momento che i corner sono calcolati soltanto nella regione di interesse, essi permettono infatti di filtrare efficacemente il frame di "backprojection" subito prima dell'algoritmo di blob detection.

Qualora nessun punto saliente sia più disponibile, il filtraggio con grigio al 25% non viene più effettuato fino alla presenza di almeno un punto saliente nella regione critica. (Il diagramma in figura 4.4 descrive schematicamente questo comportamento).

4.3 Individuazione automatica delle persone

Nell'ottica di una sempre più completa autonomia del sistema si è cercato un metodo per poter individuare automaticamente le persone. Si è pensato che un buon punto di partenza fosse la possibilità di seguire il busto di una persona, poiché in linea generale questo presenta più o meno lo stesso colore su tutta la superficie e spesso include scritte, loghi o disegni, che rappresentano degli ottimi punti salienti da considerare.

4.3.1 Problematiche del modello per individuare il busto

All'interno delle librerie utilizzate è presente una interessante funzionalità, chiamata "Haar-like object detection", che permette di rilevare oggetti di

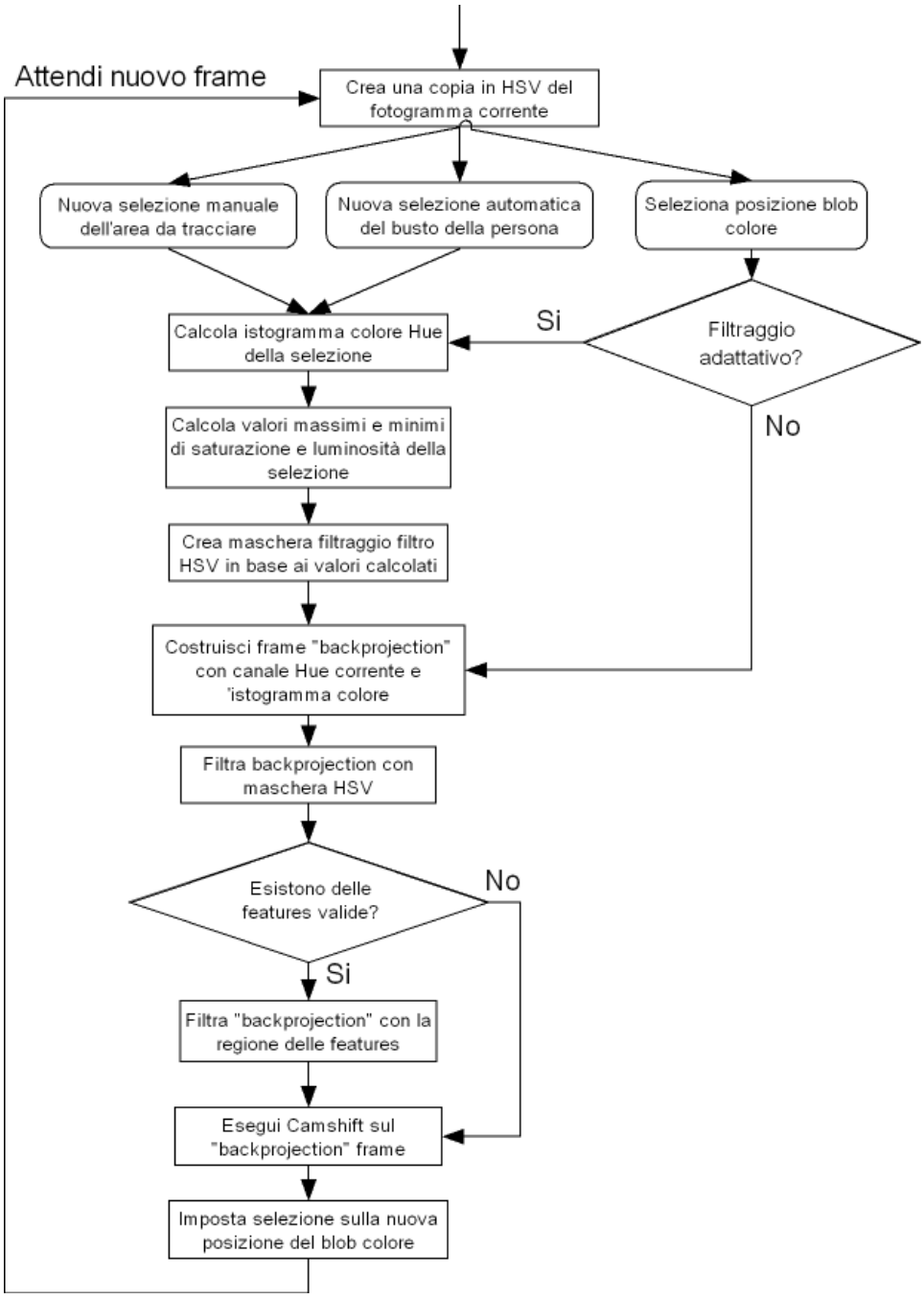


Figura 4.4: Diagramma di flusso della ricerca blob colore e interazione con la regione dei punti salienti

cui si è costruito un modello attraverso gli strumenti integrati. Per poter rilevare una maglietta era quindi necessario disporre di un enorme numero di immagini di magliette e di sfondi, con cui costruire il modello utlilizzato dal classificatore.

Gli esperimenti condotti con un centinaio di immagini e altrettanti sfondi non hanno però dato i risultati sperati. Al di là della probabile eccessiva limitazione del campione per ottenere risultati interessanti, si è notato che la rilevazione automatica di una maglietta risulta troppo generica e il rischio di ottenere falsi positivi, in questo caso ben più dannosi dei falsi negativi, era troppo elevato. Pertanto l'idea di costruire un classificatore per le magliette è stata abbandonata, decidendo invece di utilizzare un ben più efficace rilevatore di volti.

4.3.2 Rilevazione di volti

Sempre utilizzando la funzionalità di openCV per la rilevazione di oggetti "haar-like", si è trovato un buon modello già disponibile per la classificazione di volti umani in posizione frontale. L'affidabilità di tale modello è garantita dalla comunità di sviluppatori di OpenCV, dove si è scoperto che il modello è stato costruito utilizzando un vastissimo campione positivo e negativo (pare che l'ordine di grandezza sia intorno alle migliaia).

La selezione automatica di un volto però introduce problematiche dovute al fatto che una volta la persona venga inquadrata di spalle, essa non mostra più aree di pelle da poter tracciare e, poiché l'istogramma colore è stato calcolato proprio sulla pelle del volto del soggetto, il tracking può non essere efficace.

4.3.3 Selezione del busto attraverso il volto

Pertanto si è deciso di ricorrere ad alcuni semplici calcoli fisiometrici per poter selezionare il busto o la maglietta della persona, tornando così alla soluzione del problema iniziale, sulla base delle dimensioni e della posizione del volto rilevato. Tale operazione è impermeata di una evidente approssimazione ma è parsa comunque affidabile nei test condotti. La selezione del colore della maglietta viene infatti effettuato su un'area piccola, con maggiori probabilità di uniformità di colore, e viene poi utilizzata dal rilevatore blob come se si trattasse di una selezione manuale da parte dell'utente. A questo punto l'algoritmo di CamShift tende ad espandere ad ogni fotogramma la regione trovata, includendo zone dello stesso colore al di fuori della selezione inziale e isolando di fatto la maglia della persona. In seguito ven-

gono poi calcolati i punti salienti e si irrobustisce la rilevazione, esattamente come già descritto.

Tramite questo accorgimento si ha così la possibilità di seguire la persona sia di fronte che di spalle, e nella maggior parte dei casi anche di profilo, poiché il colore dominante della maglia non cambia sensibilmente.

4.4 Riconoscimento di aree note nell'immagine

Nonostante gli accorgimenti sin qui utilizzati può capitare, soprattutto durante il movimento del veicolo, che l'oggetto da inquadrare o il veicolo si muovano troppo rapidamente, facendo sì che il robot perda di vista l'obiettivo. Il risultato può portare all'arresto del veicolo e alla ricerca di un blob colore analogo a quello originario nell'immagine correntemente inquadrata. In tale situazione, il comportamento è però assai poco robusto in quanto la ricerca del blob colore mantiene esclusivamente le informazioni sull'istogramma colore iniziale. I punti salienti sono infatti ormai persi a causa della rapidità con cui l'obiettivo è uscito dalla scena. Se poi si considera che il ritrovamento di un blob errato fa ripartire il calcolo dei punti salienti all'interno di una regione che non dovrebbe essere tracciata, diventa necessario introdurre una nuova funzionalità che faccia fronte a questo problema.

Per quanto detto si è deciso così di implementare una funzionalità di riconoscimento di aree note, dette "template", in accoppiata alla ricerca del blob colore.

4.4.1 Calcolo del "template" iniziale

Come primo passo viene effettuata una copia del frame corrente, attuando una conversione dello spazio colore in scala di grigi. Questo è necessario in quanto le funzioni di tipo statistico a disposizione sono in grado di processare soltanto immagini monocanale, con l'effetto di ridurre i tempi di elaborazione rispetto ad immagini a colori.

Il template iniziale viene quindi calcolato sulla base della posizione dell'area di interesse al frame precedente ed eventualmente ne vengono corretti i parametri, affinché l'algoritmo possa funzionare correttamente. Esso viene infatti traslato di un certo valore se si trova troppo vicino ai bordi del frame, al fine di poter calcolare correttamente la finestra di ricerca.

A questo punto si ha il primo dei tre parametri necessari alla ricerca dei template, ovvero il template stesso, gli altri due sono la finestra di ricerca e la tabella di corrispondenza che verranno calcolati subito dopo.

4.4.2 La finestra di ricerca

La finestra di ricerca viene posizionata attorno al template in modo da creare una zona di ricerca più ampia del template stesso, ma ridotta rispetto alle dimensioni totali dell'immagine.

Tale valore è dimensionato in base alla capacità di elaborazione del calcolatore, poiché nonostante la ricerca all'interno dell'intera immagine sia più resistente ad ampi e veloci spostamenti, il calcolo diventerebbe troppo lento peggiorando le prestazioni del sistema.

La ricerca avviene quindi in un'area le cui dimensioni risultano aumentate di un fattore costante rispetto a quelle del template iniziale. Infine sia la finestra di ricerca sia il template vengono estrapolati da due fotogrammi differenti monocanale a stessa profondità di colore (32 bit per rappresentare tutte le tonalità di grigio).

4.4.3 Tabella di corrispondenza

Una volta completata la costruzione della finestra di ricerca si passa quindi al calcolo della tabella di corrispondenza. Questa è un immagine in cui ogni singolo punto rappresenta il grado di corrispondenza del template, posizionato sulla finestra di ricerca nel pixel alle stesse coordinate.

Data un'immagine di ricerca di dimensioni più grandi del template, la tabella di matching ha dimensioni comprese tra le due. Ogni pixel caratterizza infatti la somiglianza tra il template, traslato su ogni posizione della finestra di ricerca, e la finestra stessa.

Le dimensioni della tabella vengono aggiornate di continuo in seguito alla variazione delle dimensioni del template. Per maggiori dettagli sul dimensionamento si rimanda alla sezione 3.7.1.

In questo caso si utilizza l'immagine monocanale a tonalità di grigio, rappresentate in 32 bit poiché la funzione di matching si avvale di calcoli in virgola mobile, come il calcolo del coefficiente di correlazione normalizzato, che richiedono una buona precisione.

4.4.4 Ricerca nel frame successivo

Il template campionato nel frame iniziale viene confrontato così con i frame successivi. Si cerca dunque, nella tabella di corrispondenza, il pixel a maggiore luminosità, ovvero la massima correlazione che denota l'origine dell'area in cui si è spostato il template. L'affidabilità della ricerca è quindi rappresentata dal valore di correlazione trovato, compreso tra zero e uno.

Si considera la soglia di 0.5 per decidere se la ricerca ha avuto esito positivo o no e si procede all'aggiornamento della finestra di ricerca sulla base del nuovo template, ritrovato in maniera analoga a quanto riportato in sezione 4.4.2

4.4.5 Politiche di ricalcolo

Dal momento che il template originale potrebbe perdere di attendibilità in maniera definitiva a causa, ad esempio, di un cambio di illuminazione o di una variazione consistente nella prospettiva di inquadratura, diventa indispensabile determinare come ricalcolare il template iniziale. La scelta della politica è stata influenzata dal fatto che non ci interessa distinguere i movimenti con precisione sub-pixelare, per cui il calcolo è effettuato soltanto nel caso di grandi variazioni tra template e immagine. Si è deciso dunque di sostituire completamente il vecchio template, ricalcolandolo nuovamente sulla base della selezione inquadrata nel frame corrente.

Un buon indice di robustezza è inoltre il numero di fotogrammi per i quali non è stato necessario ricalcolare il template. Questo implica che l'oggetto da tracciare non ha subito rilevanti alterazioni da diverso tempo ed il modello è dunque particolarmente buono.

Attraverso l'utilizzo dell'indice di affidabilità si decide se ricalcolare completamente il template (affidabilità inferiore a 0.5) o proseguire con analisi real-time sulla robustezza del modello. Non sempre infatti i risultati dei due algoritmi di tracking su template e blob sono consistenti, ovvero i centri delle due regioni non sono sempre rispettivamente inclusi nell'altra regione.

In altre parole qualora i centri non appartengano ad entrambe le regioni è necessario decidere quale delle due regioni deve dominare l'altra in base alle seguenti alternative. Se il template non è stata ricalcolato da più di cento frame (corrispondenti a circa 7 secondi) vuol dire che è particolarmente robusto e vengono quindi ricalolati i punti salienti mantenendo immutato il template. D'altro canto in caso di scarsa robustezza del template, si decide di ricalcolare il modello utilizzando l'area identificata dal rilevatore di blob e rinforzata dai punti salienti. (Per il diagramma di flusso del calcolo dei template e la relazione con i punti salienti si veda la figura 4.5).

4.5 Comportamento finale, integrazione dei diversi algoritmi

A conclusione del capitolo è possibile a questo punto riepilogare i vari algoritmi e la loro interazione nell'applicazione finale del sistema di inseguimento

di oggetti o persone.

Supposto abilitato il movimento di telecamera e veicolo, il sistema si attiva con la selezione di una regione, che può essere effettuata in maniera manuale dall'utente o in modalità automatica attraverso l'individuazione di un volto.

Sulla base della selezione, vengono calcolati l'istogramma colore e la sogliatura da applicare all'immagine, per poter identificare l'oggetto da inseguire, la cui posizione viene aggiornata nei frame successivi grazie all'algoritmo di CAMshift.

Non appena sono trascorsi alcuni secondi per permettere al blob colore di espandersi su tutto l'oggetto da seguire, l'isolamento dell'oggetto viene rinforzato dal continuo ricalcolo di punti di corner nella sua area. Essi permettono di escludere dall'immagine tutti quegli oggetti che, nonostante aderiscano al modello dell'istogramma colore, non rappresentano l'oggetto da seguire.

La funzionalità di calcolo dei template entra in gioco sin dall'inizio per permettere al sistema di ritrovare l'oggetto di partenza, qualora questo venga perso. Alle volte può capitare che il template iniziale non sia particolarmente affidabile per svariati motivi: punti di corner scarsi, illuminazione variata sensibilmente, proporzioni modificate dal cambio di prospettiva; fanno sì che questo venga quindi ricalcolato.

Se il risultato della ricerca di un template non è in accordo con la ricerca del blob colore, si decide, in base alla persistenza del template, quale delle due regioni abbia la priorità. Ovvero si considera il numero di fotogrammi trascorsi da quando il template è stato ricalcolato e lo si interpreta come indice di robustezza del template: più frame sono passati senza ricalcolo, più affidabile sarà il template.

La telecamera si sposta così per inquadrare l'oggetto sempre nel centro dell'immagine, contemporaneamente il veicolo cerca di mantenersi ad un distanza costante dall'obiettivo, disponendo della possibilità di avanzare, indietreggiare, e ruotare in senso orario o antiorario. Il movimento finale di avvicinamento o allontamento viene realizzato tramite la somma vettoriale dei movimenti rettilinei e rotatori così da poter seguire obiettivi lungo una traiettoria anche non uniforme.

L'idea alla base di tutto il sistema è quindi quella di concatenare una serie di algoritmi per irrobustire il tracking nei casi critici. A tal proposito si consideri la figura 4.5. Ad ogni nuovo frame, se esiste una regione inquadrata (sia essa il risultato del precedente ciclo o una selezione manuale dell'utente) viene eseguita la serie di algoritmi elencata in precedenza: il calcolo dei punti di corner sulla regione inquadrata precede l'esecuzione del blob tracker, che

ne risulta rinforzato, per cercare l’oggetto da seguire. Il risultato è dunque la nuova posizione dell’oggetto in coordinate spaziali relative all’immagine.

A questo punto, la telecamera utilizza le coordinate appena citate per posizionarsi, in modo tale che l’oggetto rilevato sia nel centro dell’inquadratura e ne possa essere rilevata la distanza dal sonar. Si leggono allora i valori dei sensori di distanza sul veicolo e si decide il movimento del robot, in base alla somma di diversi contributi: angolo relativo tra telecamera e veicolo; scostamento della posizione dell’oggetto dal centro dell’asse x; distanza del robot dall’oggetto misurata dal sonar.

Il passo del ciclo termina infine con l’attuazione del movimento e il calcolo del tempo trascorso fino ad ora dalla fine del ciclo precedente. Quest’ultimo valore serve a calcolare la reattività del sistema in quanto, poiché in un ciclo viene elaborato un singolo fotogramma, il suo inverso rappresenta la velocità espressa in fotogrammi per secondo.

Il ciclo viene ripetuto infinitamente finché l’utente non ferma il robot, attraverso l’interfaccia o direttamente tramite il pulsante di spegnimento. Tuttavia non è necessario spegnere il robot in quanto esso è perfettamente autonomo. Infatti nel caso l’obiettivo si fermi, il robot si ferma anch’esso ad una distanza di sicurezza, pronto a riprendere l’inseguimento qualora il soggetto cominci a riallontanarsi dal centro dell’inquadratura.

4.6 Interfaccia Software “Servocamera”

L’interfaccia grafica del sistema, costruita grazie alle librerie open source “Gtk+ 2.0”, permette all’utente di gestire o monitorare il comportamento del robot. Per non penalizzare la quantità di funzionalità disponibili tramite interfaccia si è deciso per il momento di non automatizzare completamente l’applicazione lasciando all’utente il compito di selezionare alcuni controlli e di avviare il robot. Ad ogni modo la modularizzazione utilizzata durante la stesura del codice sorgente lascia comunque ampio spazio ad una successiva automatizzazione del robot al fine di ridurre l’utilizzo manuale dell’interfaccia e di conseguenza aumentare il grado di autonomia del sistema.

Come si può notare in figura 4.6, un gruppo di moduli si occupa di realizzare gli algoritmi di visione artificiale descritti nei precedenti paragrafi di questo capitolo. Si tratta in particolare degli algoritmi di visione artificiale: blob detection, ricerca corner, template matching, face detection (realizzati rispettivamente dai moduli blobdetect.h, flowfeat.h, template.h, facedetect.h).

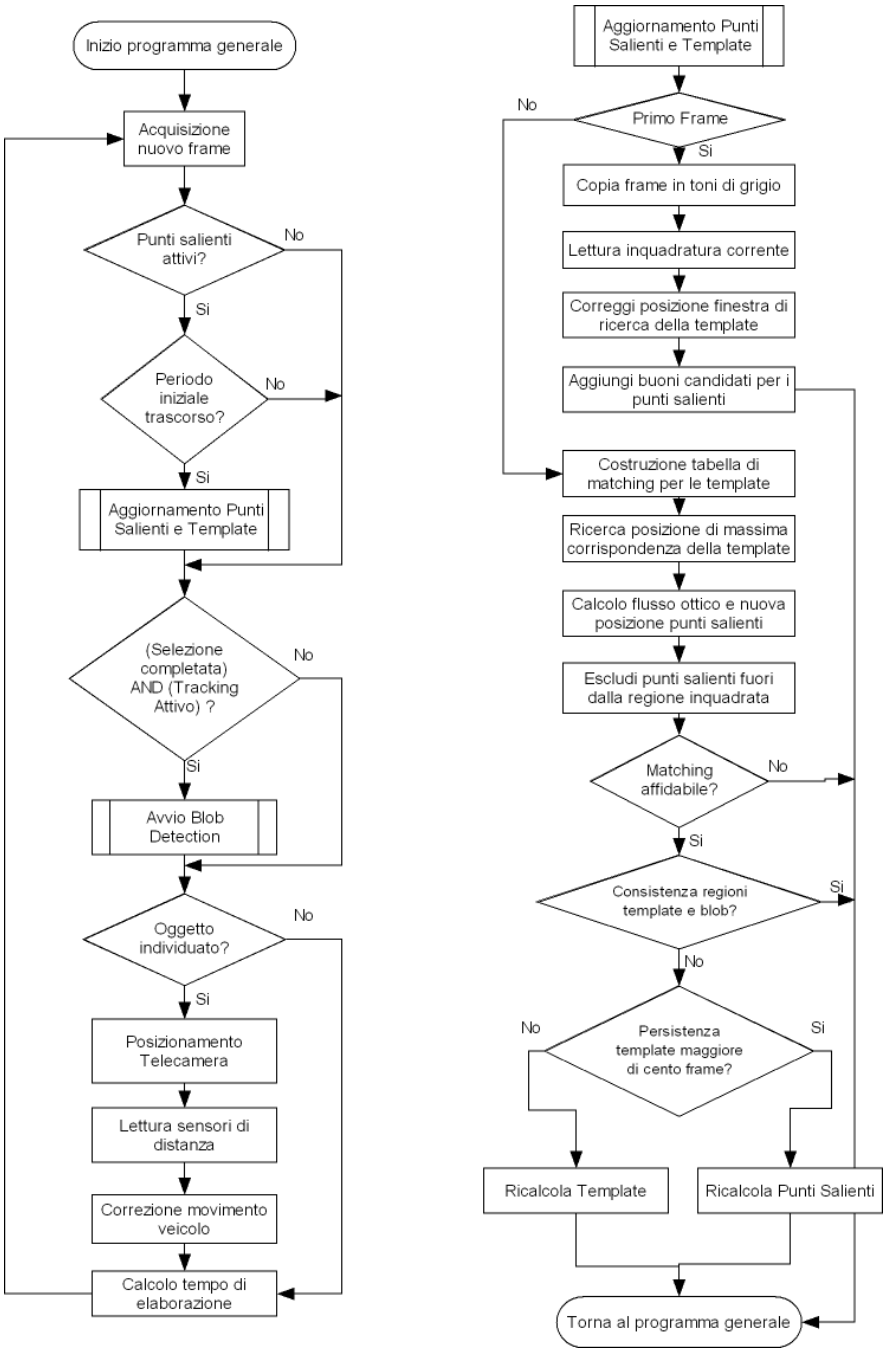


Figura 4.5: Diagramma di flusso del programma principale e subroutine per il calcolo di punti salienti con ricerca template

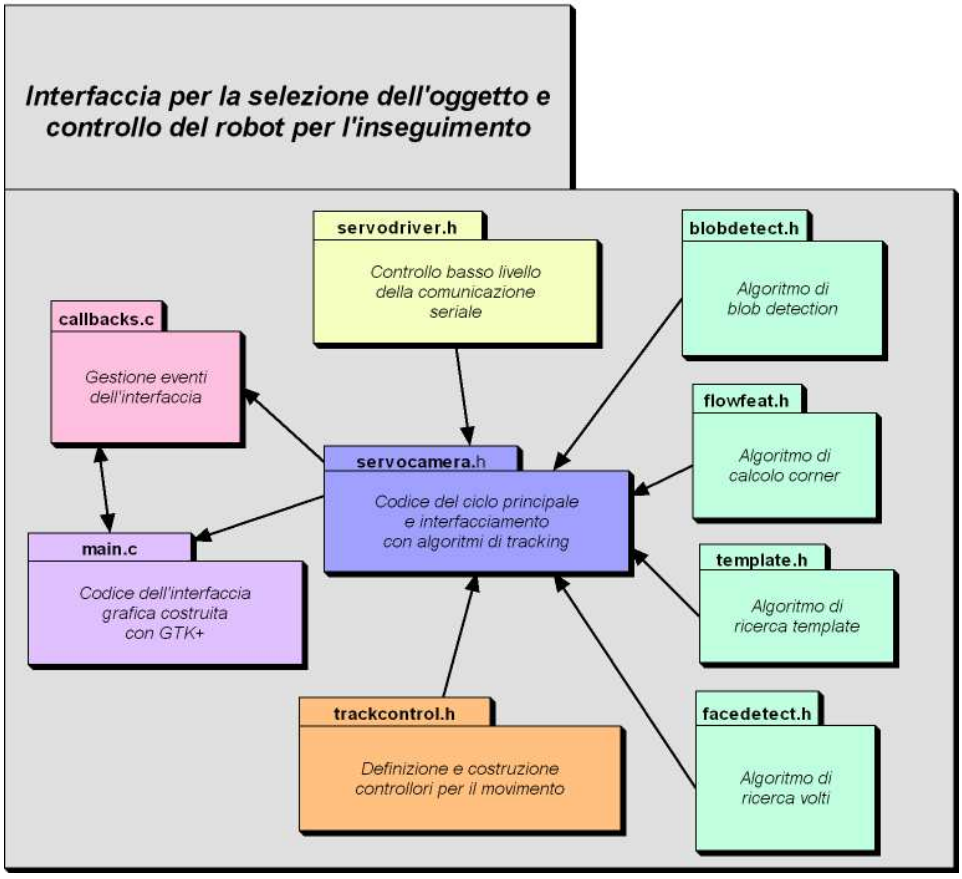


Figura 4.6: Moduli componenti l'interfaccia per l'utilizzo del robot

Il controllo della comunicazione seriale, predisposta al comando di movimento sia dei servomotori di posizionamento della telecamera che dei motori delle ruote del veicolo, è invece affidato ad un modulo separato (servodriver.h).

La gestione e implementazione dei controllori, utilizzati per il posizionamento dei servomotori e per il movimento del veicolo, è inclusa in un ulteriore modulo atomico (trackcontrol.h), che serve a garantire una certa trasparenza rispetto al resto dell'applicazione.

I moduli sino a questo momento elencati sono tutti inclusi nel modulo nucleo del sistema (servocamera.h) comprendente il comportamento a più alto livello. Esso infatti contiene il principale ciclo di funzionamento, che racchiude l'interazione tra loro dei vari algoritmi di visione ed il controllo del movimento di servomotori e robot.

Un'altra coppia di moduli si occupa invece della realizzazione e gestione dell'interfaccia. Si tratta in particolare di un modulo (callbacks.c) che collega gli eventi generati dai singoli componenti dell'interfaccia (bottoni, finestre, controlli numerici, etc. . .) alle funzionalità offerte dal sistema. L'ultimo modulo riportato (main.c) è invece predisposto alla semplice creazione dell'infrastruttura dell'interfaccia grafica e dei componenti presenti, nonché all'inizializzazione della comunicazione seriale e alla configurazione iniziale della telecamera FireWire. Rappresenta di fatto il modulo "starter" da cui parte l'esecuzione del sistema.

4.6.1 Utilizzo dell'applicazione

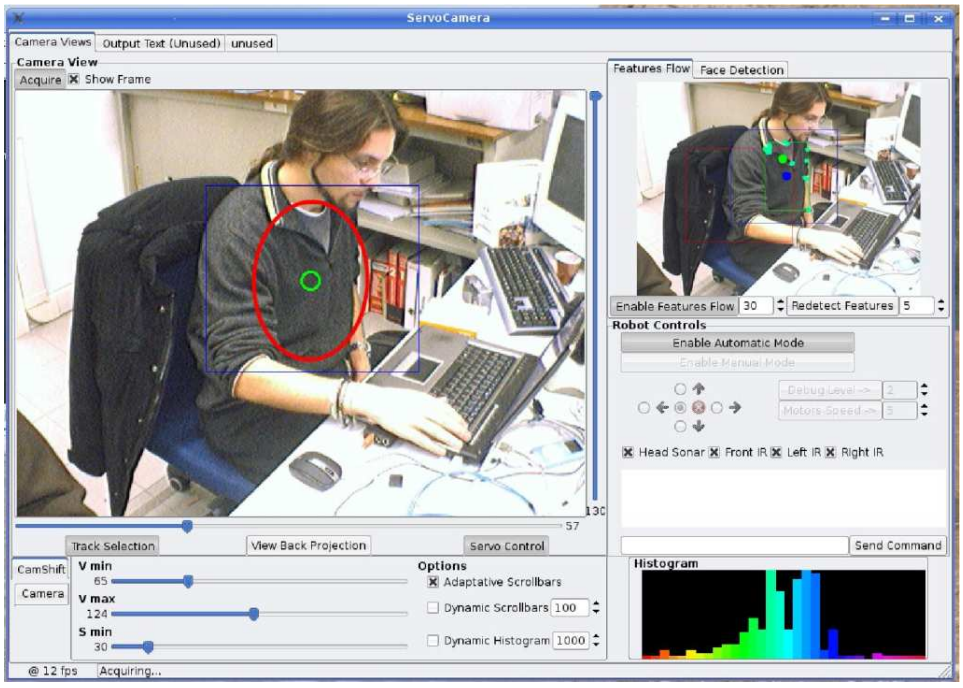


Figura 4.7: Interfaccia durante l'esecuzione con punti di corner attivati

Una volta avviata l'applicazione è necessario premere il tasto "Acquire" per poter accendere la telecamera e visualizzare il video acquisito. Il modulo di gestione degli eventi associa infatti a tale pulsante il ciclo principale di funzionamento del sistema. A questo punto è possibile selezionare l'area dell'immagine che il robot dovrà tracciare ed inseguire cliccando con un tasto qualsiasi del mouse sull'immagine video e trascinando il puntatore fino all'estremo inferiore destro dell'area da selezionare. Al termine della selezione

l’applicazione calcola l’istogramma colore (visualizzato in basso a destra nell’interfaccia) contenente le varie componenti di colore della porzione di immagine selezionata e attiva il movimento dei servomotori affinché la regione di interesse sia sempre inquadrata. (Vedi figura 4.7).

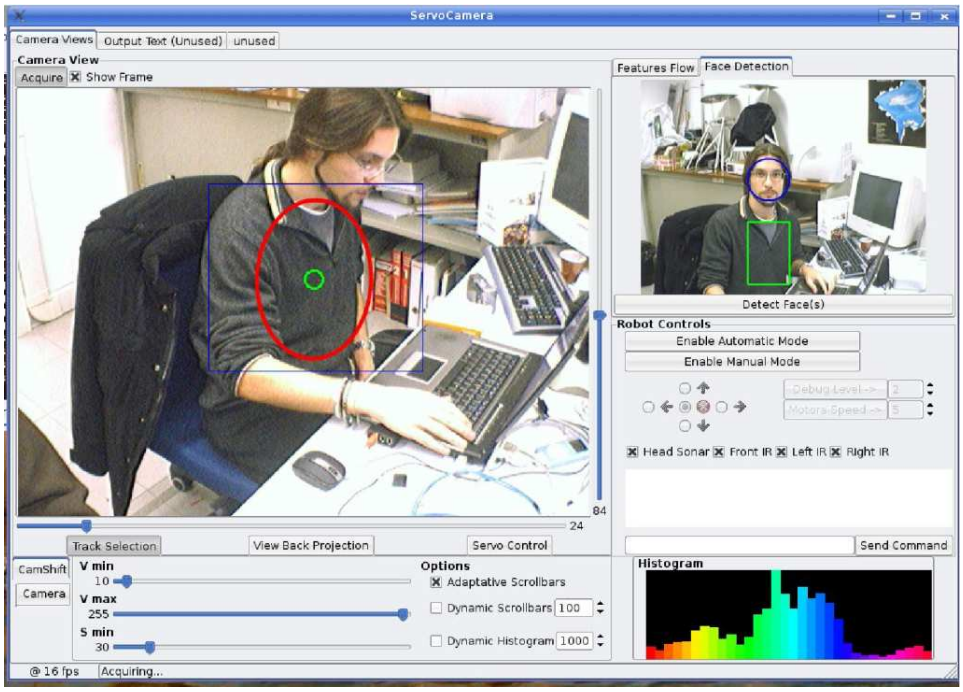


Figura 4.8: Interfaccia dopo l’esecuzione del rilevamento facciale

Per irrobustire l’algoritmo di inseguimento del colore è possibile abilitare l’utilizzo dei punti di corner premendo sul tasto “Enable Features Flow” a destra in alto e attendendo qualche istante affinché l’applicazione possa riconoscere i punti salienti presenti nella regione da osservare. L’abilitazione dei punti salienti è possibile sia prima che dopo la selezione della porzione di immagine da inseguire. Durante l’esecuzione del programma è poi possibile ricalcolare manualmente i punti di corner tramite l’apposito pulsante “Redetect Features”.

Il calcolo dei corner è collegato internamente all’applicazione con il calcolo dei template, di conseguenza abilitare o ricalcolare i punti di corner abilita o ricalcola anche il template su cui effettuare il successivo matching.

Proseguendo per abilitare il movimento del robot è necessario premere il pulsante “Enable Automatic Mode” che in funzione della posizione dell’obiettivo e della presenza o meno di ostacoli nelle prossimità del veicolo

permette al robot di seguire l'oggetto selezionato mantenendo una distanza prefissata. Il pulsante "Enable Manual Mode" permette invece un controllo manuale del robot attraverso i quattro movimenti: avanti, indietro, ruota in senso orario, ruota in senso antiorario, qualora ciò si rendesse necessario.

In alternativa alla selezione manuale di un obiettivo è possibile utilizzare la funzionalità di riconoscimento facciale presente nelle linguette in alto a destra: una volta premuto il tasto sinistro del mouse sulla linguetta "Face Detection" e sufficiente selezionare il pulsante "Detect Face" e l'applicazione cercherà tutte le facce presenti nella scena inquadrata. Il primo volto rilevato da sinistra viene così utilizzato come riferimento per inquadrare il busto della persona secondo semplici calcoli fisiometrici allo stesso modo in cui un utente avrebbe potuto effettuarne manualmente la selezione. (Vedi figura 4.8).

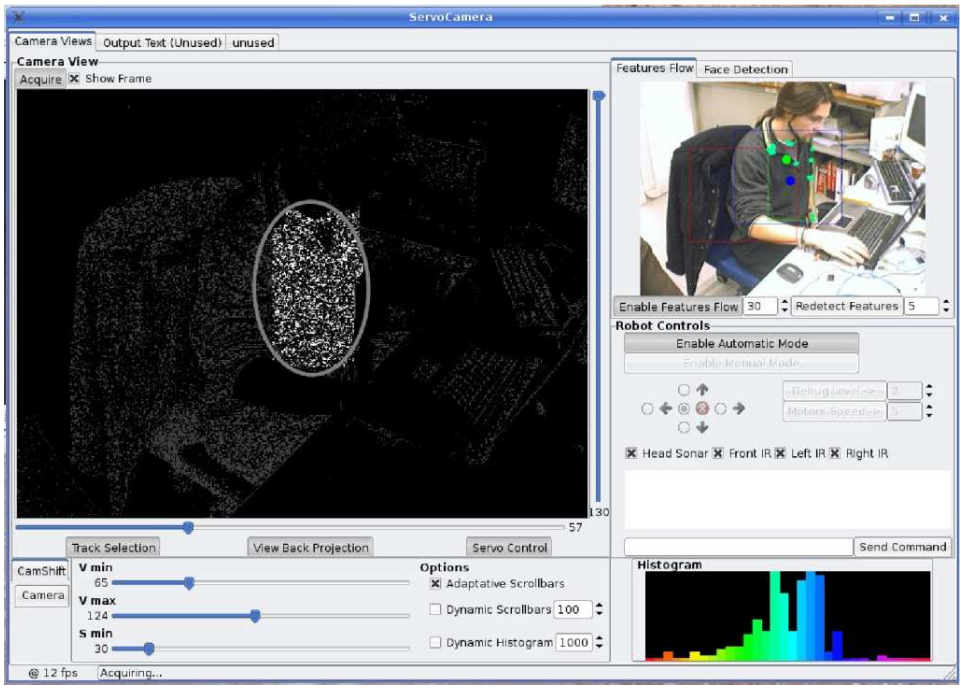


Figura 4.9: Interfaccia con visualizzazione filtraggio attivata

Per controllare il corretto filtraggio dell'immagine è poi possibile abilitare il pulsante "View Back Projection" che mostra l'immagine subito dopo l'applicazione del filtro di colore eventualmente irrobustisto dalla presenza di corner sulla regione di interesse. Tale immagine, riportata in figura 4.9 è ciò che viene inviato all'algoritmo di rilevazione blob colore centrale del sistema.

Oltre al calcolo dell'istogramma colore nella selezione effettuata è auspi-

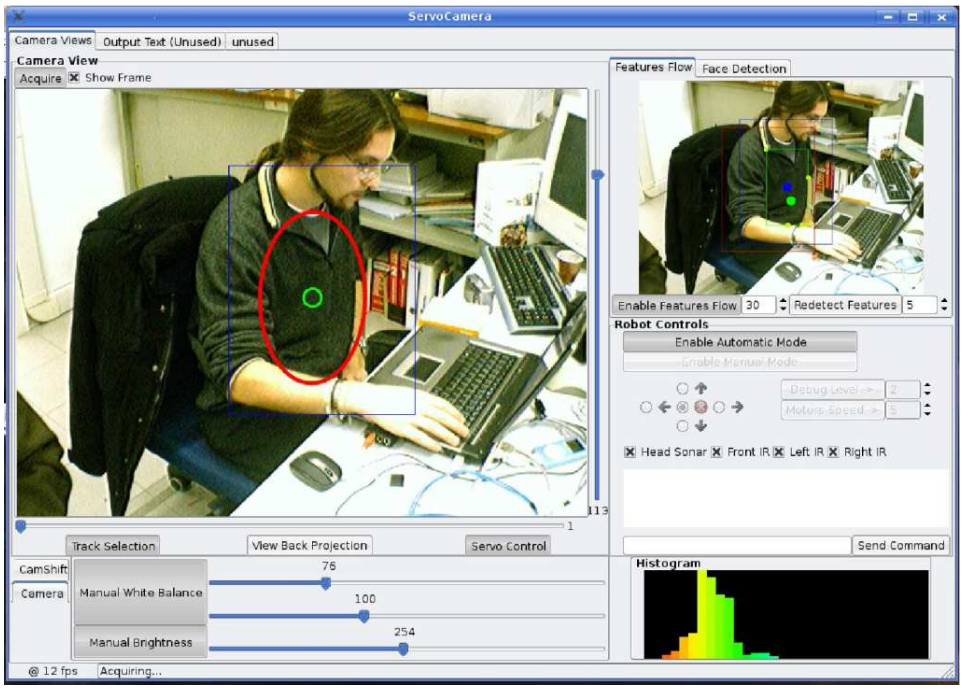


Figura 4.10: Interfaccia con controlli bilanciamento del bianco e luminosità manuali

cabile misurare le soglie di filtraggio in funzione dei valori minimi e massimi di valore colore (“Hue”) e valore minimo di saturazione (“Saturation”) all'interno dello spazio colore HSV (“Hue, Saturation, Value”). Poiché ciò irrobustisce sensibilmente l'isolamento del colore di interesse nell'immagine, la casella “adaptative scrollbars” è abilitata di default per effettuare tale calcolo nella costruzione del filtro.

Per quanto riguarda la resa dei colori dell'acquisizione, la telecamera utilizzata dispone di controlli automatici per il bilanciamento del bianco e per la luminosità dell'immagine. In situazioni di luce omogenea senza cambi di sorgente di illuminazione è però piú robusto utilizzare un bilanciamento ed una luminosità settati manualmente su un valore costante: ciò può essere effettuato regolando le apposite barre presenti nella sezione “Camera” attivabile tramite omonima linguetta. (Vedi figura 4.10).

Un'alternativa alle regolazioni automatiche della telecamera è rappresentata dalle caselle “Dynamic Scrollbars” e “Dynamic Histogram” che permettono di adattare il filtraggio colore sulla base del colore attualmente inquadrato e, a patto di regolare correttamente la velocità di adattamento (disponibile nella casella numerica accanto alla voce selezionata), possono

far fronte a cambi di illuminazione pur in presenza di controlli manuali della telecamera. Infine, una volta inizializzata correttamente l'applicazione e abilitati tutti i controlli automatici, si può disattivare la visualizzazione del video premendo sulla casella "Show Frame". Considerando che in un robot autonomo la visualizzazione non riveste un ruolo fondamentale questo consente di incrementare la velocità di elaborazione garantendo così una maggiore reattività e fluidità del sistema nell'inseguimento.

Capitolo 5

Architettura hardware del sistema

Questo capitolo descrive l'architettura hardware del sistema. Il sistema per l'analisi delle immagini e l'inseguimento di un obiettivo è costituito da un robot a due ruote motorizzato sul quale è montata una telecamera firewire. All'interno del robot è alloggiato un calcolatore, dotato di sistema operativo Linux, che esegue l'applicazione di controllo della visione e movimento del robot (per l'immagine completa del sistema vedi figura 5.1).

Il veicolo e la scheda di controllo AirBoard associata sono stati adattati alle nostre esigenze poiché provengono dal noto progetto Robocup[1] sviluppato dal Politecnico di Milano.

La prima sezione riporta un elenco descrittivo dei vari moduli presenti, mentre la seconda ne illustra il funzionamento.

5.1 Moduli hardware

5.1.1 Il veicolo

Il veicolo è composto da una struttura in profilati di alluminio dotata di due ruote affiancate ciascuna da un motoriduttore, un motore, ed un encoder di posizione che realizzano il movimento motorizzato secondo uno schema "differential drive".

Il nome dello schema si riferisce al fatto che il vettore di movimento del veicolo è pari alla somma dei movimenti indipendenti delle due ruote, posizionate ai lati del veicolo. Altre due ruote di diametro inferiore, vedi

[1] http://www.robocup.org

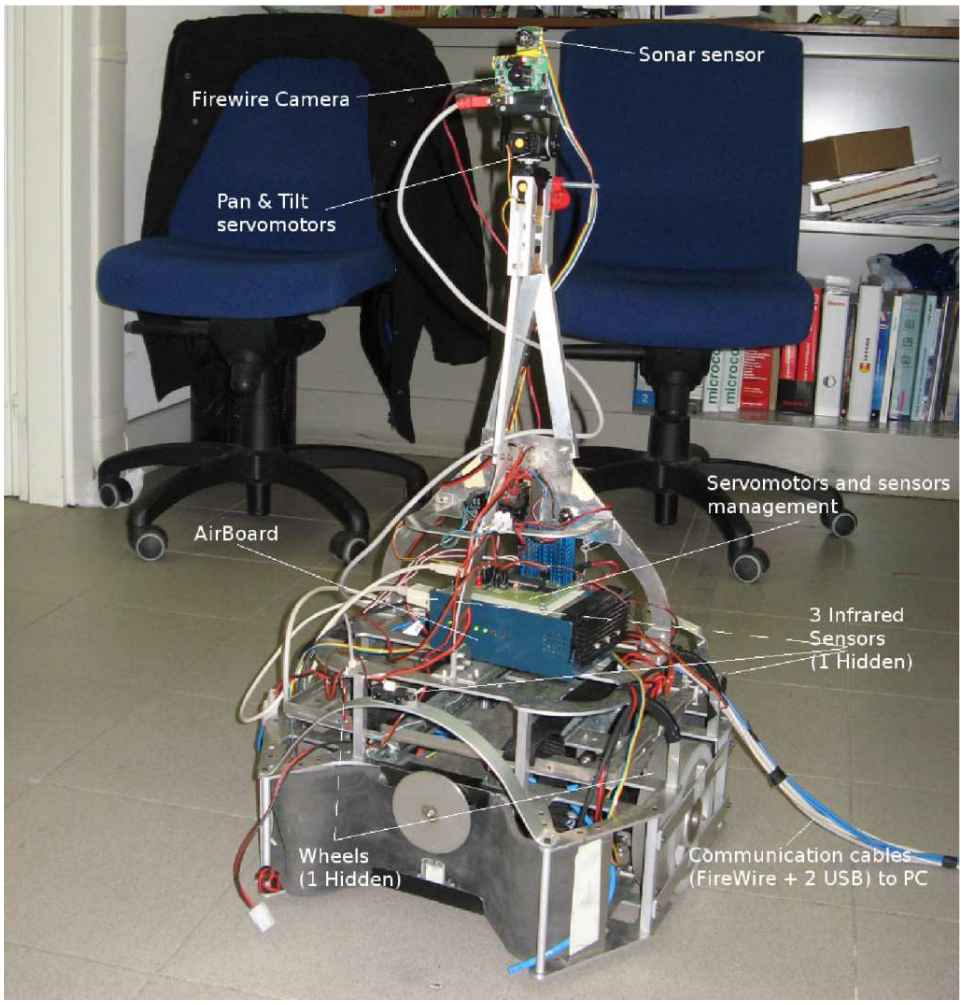


Figura 5.1: Immagine completa del robot

figura 5.2, sono poi posizionate nella parte frontale e nella parte posteriore dell'asse longitudinale del veicolo col mero scopo di aumentarne la stabilità durante la marcia.

Tra i motori e le due ruote principali sono stati collocati due motoriduttori per aumentare la coppia dei motori riducendone la velocità, poiché i motori non avrebbero sufficiente coppia per comandare direttamente la ruota associata.

La semplicità dei sistemi "differential drive" paga il prezzo di una non semplice gestione del moto rettilineo. Per garantire un movimento in linea retta è infatti necessario assicurarsi che entrambe le ruote mantengano la

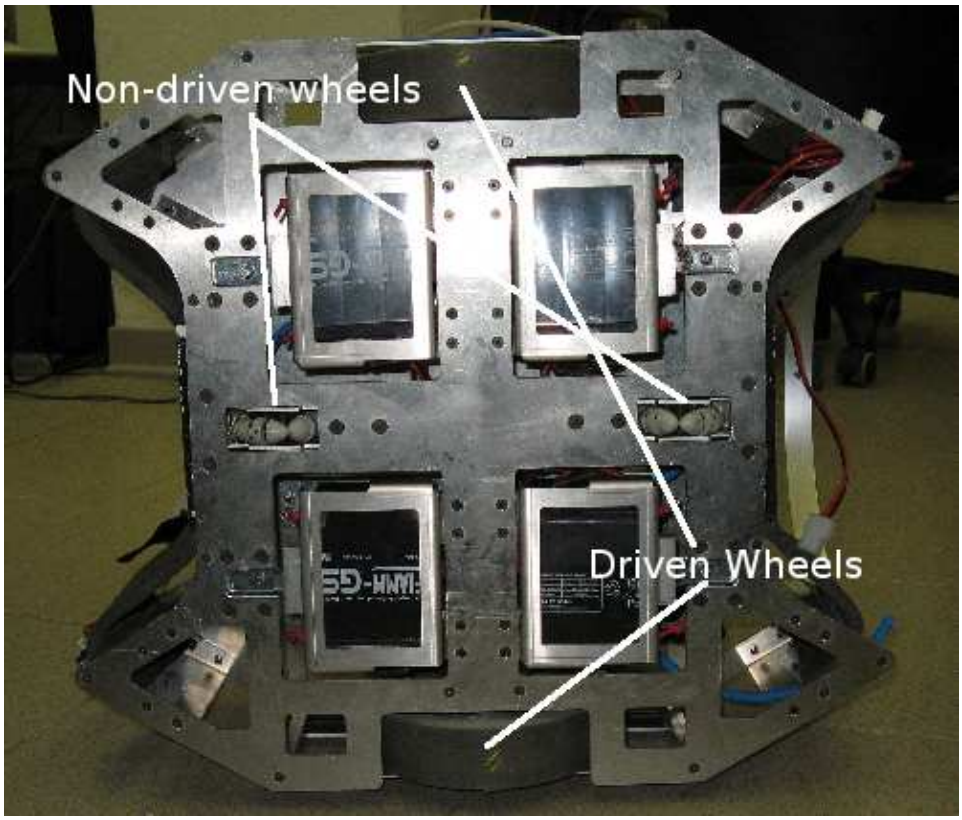


Figura 5.2: Vista dal basso del robot

stessa velocità angolare durante il moto. Questo può essere effettuato monitorando costantemente le velocità delle due ruote e uniformandole in caso di variazioni dovute a piccole differenze architetturali nei motori o differenti attriti legati alla non omogeneità della superficie. A tale scopo ogni motore è affiancato da un encoder di posizione collegato alla scheda di controllo AirBoard. Il moto rettilineo (sia avanti che indietro) è realizzato semplicemente dando la stessa velocità angolare alle due ruote, mentre la rotazione sul posto (sia oraria che antioraria) è ottenuta conferendo velocità angolari opposte ma di medesima entità alle due ruote. Grazie a questo fatto è sufficiente combinare i due movimenti sommando i loro contributi espressi in velocità angolare per ogni ruota, si otterrà in questo modo il movimento del veicolo richiesto.

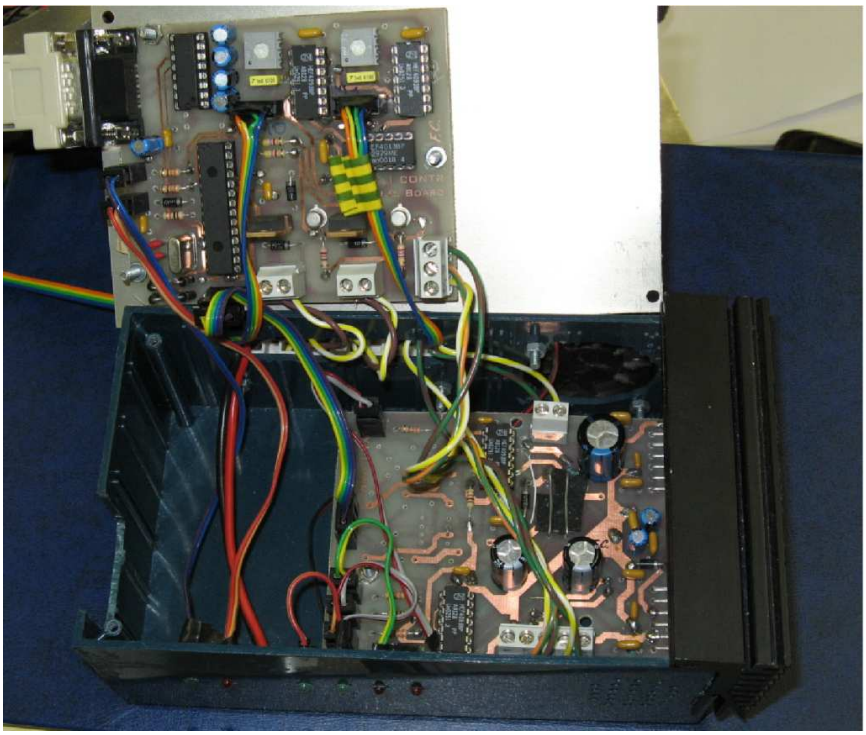


Figura 5.3: scheda AirBoard per il controllo del movimento del robot

5.1.2 AirBoard

La scheda di controllo AirBoard, rappresentata in figura 5.3, viene utilizzata per gestire il movimento del veicolo comandando i motori su richiesta del calcolatore.

Tale scheda comanda la rotazione dei motori attraverso l'impostazione di un setpoint in velocità ricevuto dal calcolatore tramite interfaccia seriale RS232. È necessario che il primo comando in velocità sia preceduto da un comando di attivazione dei motori, così come è auspicabile disattivare i motori tramite apposito comando prima di spegnere l'intero sistema.

L'Airboard contiene tra l'altro i valori di taratura e i parametri del controllore per gestire il corretto movimento dei motori secondo la filosofia "differential drive". Si rimanda al manuale della AirBoard per maggiori informazioni.

5.1.3 Telecamera

Si tratta di una telecamera "Fire-i Digital Camera", prodotta da UniBrain e dotata di interfaccia IEEE-1394a (FireWire) a 400Mbps. La risoluzione utilizzata è di 640x480 pixel a colori alla velocità di 15 fps.

La camera dispone delle funzioni di bilanciamento del bianco e luminosità automatiche o manuali con un'apertura di diaframma di F2.0 mentre l'ottica utilizzata per i test (tra le varie disponibili) dispone di messa a fuoco regolabile manualmente. Per le specifiche dettagliate della telecamera si rimanda al sito del produttore[2], mentre la telecamera è rappresentata in figura 5.12.

5.1.4 Sensori all'infrarosso

Si tratta di sensori di distanza prodotti da Sharp, modello GPD12, in grado di misurare distanze da 10 cm a 80 cm. Alla distanza massima rilevano campo libero in presenza di aperture superiori ai 6 cm.

L'uscita del sensore è di tipo analogico quantizzata secondo la curva riportata in figura 5.5. Per le specifiche dettagliate si rimanda in ogni caso ai datasheet presenti in appendice.

Sono stati utilizzati tre sensori collocati alla base del veicolo per poter rilevare la presenza di ostacoli nelle vicinanze e regolamentarne il movimento in maniera sicura e robusta. In particolare, un sensore è montato in posizione frontale centrale mentre gli altri due sono posizionati nei due angoli posteriori del robot (vedi figura 5.1).

5.1.5 Sensore ad ultrasuoni

Il sensore utilizzato è il modello LV-MaxSonar-EZ1 prodotto da MaxBotix[3] ed è in grado di misurare distanze da 15 cm a circa 6 m con la risoluzione di 1 pollice (circa 2,5 cm).

È da precisare che come ogni sensore ad ultrasuoni anch'esso presenta una zona cieca tra gli 0 cm e i 15 cm, pertanto oggetti in questo intervallo vengono segnalati a distanza costante di 15 cm. La zona di rilevazione è approssimabile ad un cono che si estende dal sensore verso l'area di rilevazione come riportato in figura 5.6.

I sensori ad ultrasuoni funzionano emettendo e ricevendo onde sonore ad alta frequenza (in genere sui 200kHz) impercettibili all'orecchio umano. Il sensore in esame è stato utilizzato in modalità "eco" pertanto esso emette

[2] http://www.unibrain.com/Products/VisionImg/tSpec_Fire_i_DC.htm
[3] www.maxbotix.com

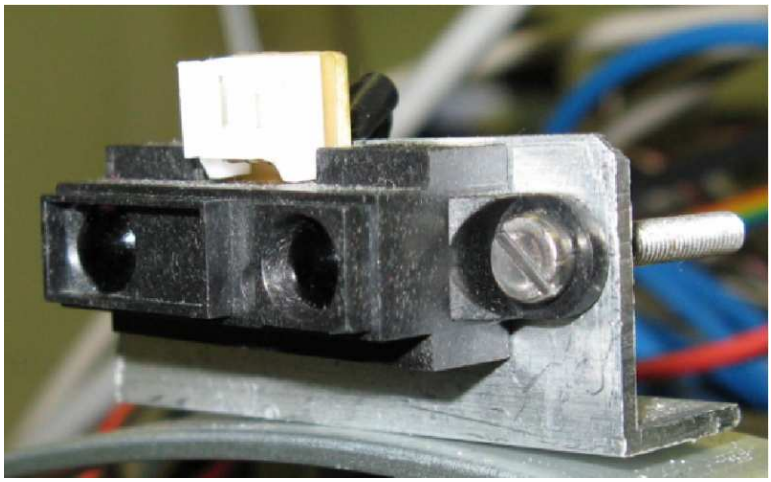


Figura 5.4: Immagine di un sensore ad infrarossi

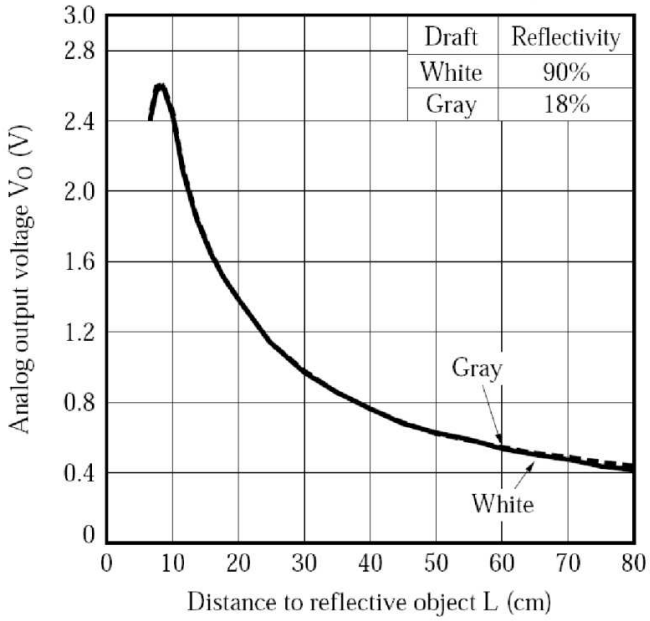


Figura 5.5: Curva di uscita dei sensori ad infrarosso in funzione della distanza dell'oggetto

l'onda sonora e quindi ne capta l'eco che rimbalza contro un oggetto. Calcolando il tempo intercorso tra emissione e ricezione e conoscendo la velocità del suono è facile risalire al valore di distanza dell'ostacolo. Essendo colloca-

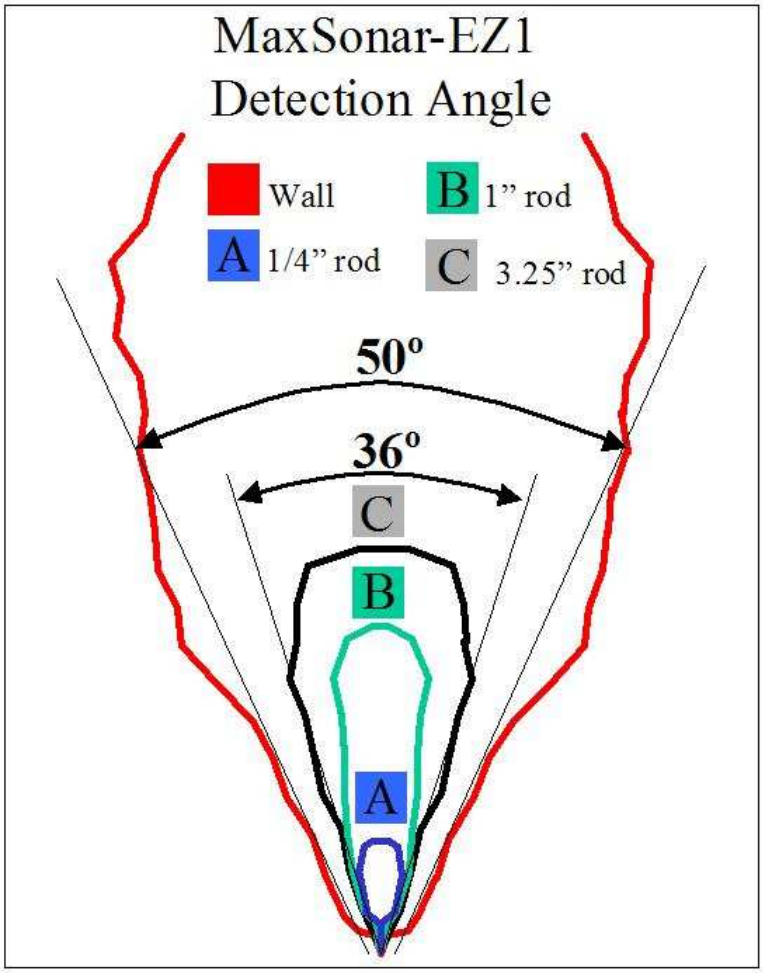


Figura 5.6: Angolo di rilevazione variabile del sensore per il target

to proprio al di sopra della telecamera del robot, il sensore misura approssimativamente la distanza dell'oggetto che in quel momento è inquadrato dalla telecamera, vedi figure 5.12 e 5.1.

Il sensore dispone di svariate uscite per la comunicazione del valore di distanza: modulazione pwm, valore analogico, seriale rs232. Nel caso specifico è stata utilizzata l'uscita analogica per la maggiore semplicità di lettura e di interfacciamento con la scheda di lettura valori dai sensori descritta nella sezione 5.2. La curva di uscita è lineare e misura circa 5000/512 mV per pollice misurato, ovvero circa 3.8 mV/cm. Per maggiori informazioni si rimanda al datasheet in appendice.

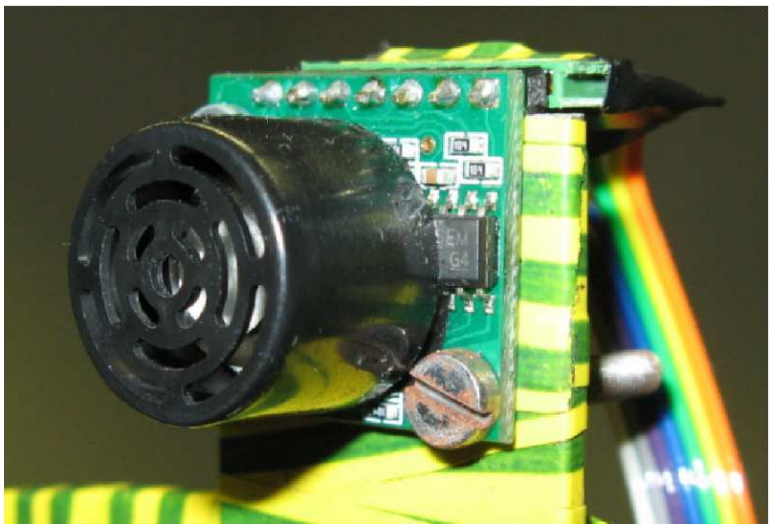


Figura 5.7: Immagine del sensore ad ultrasuoni montato sopra alla telecamera

5.1.6 Servomotori per movimento telecamera

Per il controllo di posizione della telecamere sono stati utilizzati due servomotori HI-TECH HS-422 (vedi figura 5.12) controllati in posizione attraverso un segnale PWM (Pulse Width Modulation o Modulazione ad ampiezza di impulso) come quello in figura 5.8. In questo tipo di modulazione digitale l'informazione è codificata sotto forma di durata nel tempo di ciascun impulso di un segnale. Esiste dunque un periodo entro il quale il segnale puó assumere valore alto soltanto per una frazione di tempo, espressa in percentuale, chiamata duty-cycle.

Ad esempio su un periodo di 2ms, un duty-cycle del 50% è realizzato mantenendo il segnale a livello logico alto per il primo millisecondo e poi basso fino al termine del secondo millisecondo. È implicito che i duty-cycle di 100% e 0% si realizzano rispettivamente mantenendo l'uscita costantemente alta e bassa.

I servomotori utilizzati sfruttano il duty-cycle come indicazione di posizione in gradi angolari. Il periodo utilizzato è di 20 ms con ampiezza di impulso da 0.9 ms (per 0° circa) a 2.1 ms (per 200° circa) centrata a 1.5 ms (per 90°). Per le specifiche dettagliate si veda l'appendice.

Come si può notare il periodo totale è molto più alto del massimo duty-cycle utilizzabile (che si ferma quasi al 10%) e serve soltanto a rinfrescare la posizione del servomotore. Questo periodo relativamente lungo è comunque utile perché permette al microcontrollore di effettuare anche altre operazioni

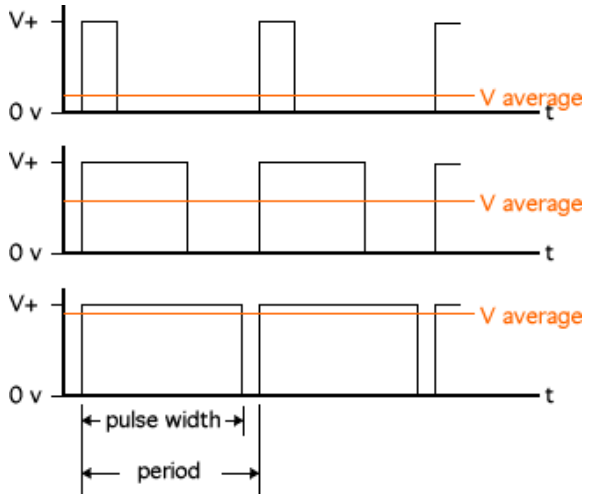


Figura 5.8: Schema di funzionamento modulazione PWM

senza perdere il posizionamento dei servomotori.

Per ruotare la telecamera sul piano orizzontale (angolo di pan) e sul piano verticale (angolo di tilt) vengono utilizzati due servomotori. Per esigenze strutturali l'angolo di pan varia da 0° a 180°, mentre l'angolo di tilt è più ridotto e varia da 45° (per evitare che il robot inquadri se stesso puntando verso il basso) a 130° (a causa dei vincoli meccanici della torretta che alloggia i due servomotori, vedi figura 5.12).

5.1.7 Calcolatore

Il calcolatore utilizzato è un "MacMini", vedi figura 5.9, prodotto da Apple dotato di processore "Intel Core Duo" a 1.83 Ghz e 512 MB di RAM su cui è stato installato il sistema operativo "Linux Gentoo 2007.0".

Per implementare gli algoritmi di visione sono state utilizzate le librerie open source di visione artificiale "OpenCV 1.0" alle quali è stata affiancata un'interfaccia grafica costruita tramite le librerie "Gtk+ 2.0".

In realtà le librerie OpenCV funzionano su ogni tipo di sistema Unix o Windows e infatti durante la fase di progettazione sono stati utilizzati anche dei notebook dotati di processore "Intel Pentium-M Centrino" e sistema operativo "Linux Kubuntu 7.04". Tuttavia per esigenze di velocità di elaborazione è stata scelta la macchina sopraindicata con la distribuzione Linux Gentoo, nota per l'elevato grado di ottimizzazione che offre. Grazie infatti alla possibilità di compilare il kernel di Linux ottimizzandolo per l'architettura del MacMini è stato possibile ottenere la media di 15 fps di

Figura 5.9: Calcolatore Mac Mini utilizzato

acquisizione a pieno regime nonostante i tempi di esecuzione degli algoritmi di tracciamento implementati.

5.2 Scheda di controllo servomotori e lettura sensori

La scheda di controllo dei servomotori rappresentata in figura 5.10 viene utilizzata per controllare, su comando del calcolatore, la posizione della telecamera. Essa effettua inoltre la lettura dei sensori di prossimità e distanza montati sul veicolo e ne invia le misurazioni al calcolatore.

La scheda è costituita da un microcontrollore PIC16F877A che implementa le funzionalità necessarie attraverso la configurazione delle proprie periferiche. In particolare, sono stati configurati:

- un'interfaccia di comunicazione RS232 (coadiuvata da un adattatore USB - RS232) per permettere alla scheda di dialogare con il calcolatore
- quattro canali di acquisizione analogica per la gestione tramite l'ADC integrato nel microcontrollore del valore letto dai tre sensori di prossimità e dal restante sensore di distanza
- due uscite digitali temporizzate tramite PWM (pulse width modulation) per controllare la posizione dei due servomotori adibiti al posizionamento e movimento della telecamera.

Ogni volta che è necessario variare la posizione della telecamere il microcontrollore riceve dal PC (tramite USB) un comando di posizione dei due

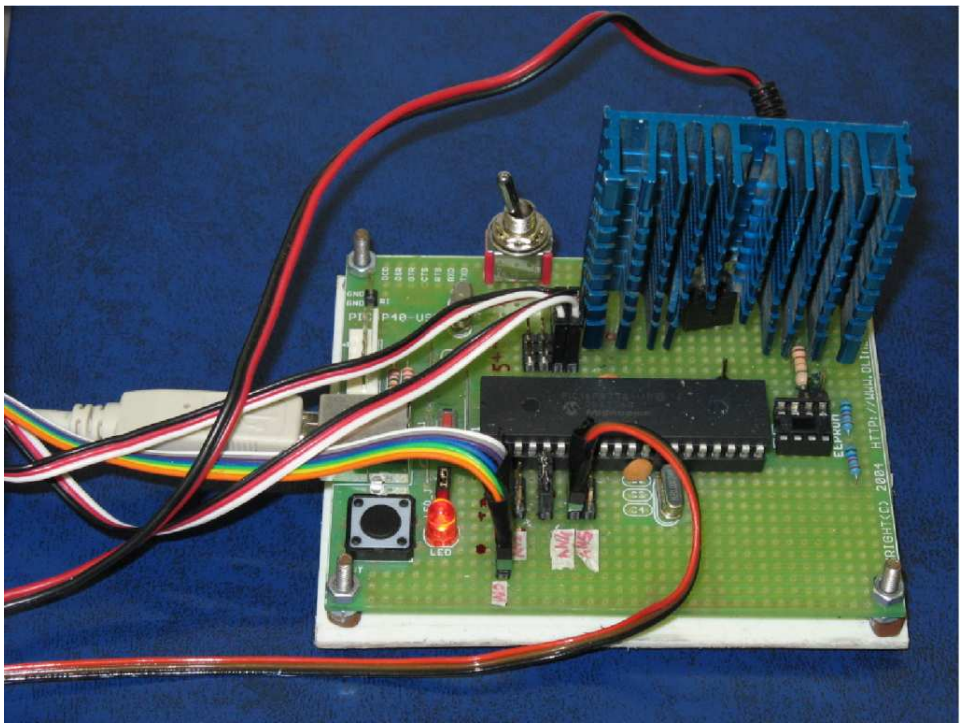


Figura 5.10: Scheda di controllo servomotori e lettura sensori

servomotori espresso in angolo di pan (orizzontale da 0° a 180°) e di tilt (verticale da 0° a 130°). I servomotori adibiti al controllo degli angoli sono rappresentati in figura 5.12.

Appena ricevuto il comando, il PIC, che gestisce la comunicazione seriale e la temporizzazione del periodo totale della PWM attraverso un paradigma ad interrupt, configura di conseguenza il duty cycle del segnale PWM e aspetta che siano terminati i 20 ms del periodo in corso.

Appena all'inizio del nuovo periodo il programma principale imposta a livello logico "alto" i valori di uscita dei pin di controllo dei servomotori per la durata corrispondente al comando precedentemente ricevuto realizzando di fatto il duty-cycle richiesto.

Una volta posizionata correttamente la telecamera viene poi effettuata la lettura dei valori di uscita dei sensori attraverso il convertitore analogico/digitale integrato nel microcontrollore ed i valori letti vengono comunicati al calcolatore tramite interfaccia seriale.

L'ADC integrato nel PIC dispone di 8 canali utilizzabili ciascuno per la lettura di un valore analogico di tensione (tipicamente da 0 V a 5 V) che

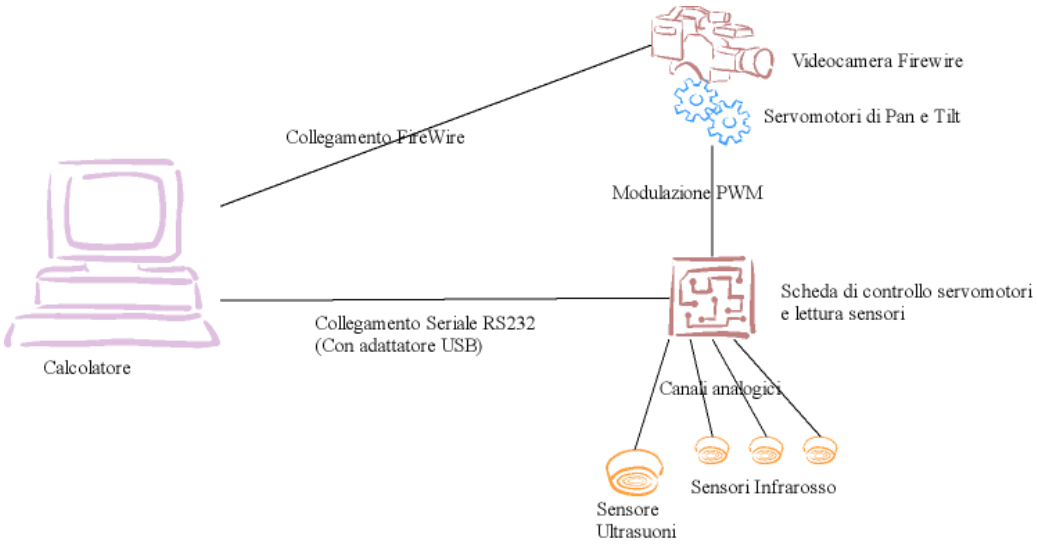


Figura 5.11: Schema del collegamento tra calcolatore, scheda di controllo servi e lettura sensori

verrà memorizzato in una variabile a 10 bit.

Per esigenze di velocità si è deciso di utilizzare soltanto gli 8 bit più significativi del valore campionato che, quindi, pur restando un valore a 10 bit avrà sempre nulli i 2 bit meno significativi. Tale operazione oltre a non danneggiare sensibilmente la misurazione ne realizza oltretutto un lieve filtro passa basso utile a mantenere più stabile il valore tra acquisizioni successive.

Quattro degli ingressi analogici del PIC sono collegati ai tre sensori di prossimità all'infrarosso e al sensore di distanza ad ultrasuoni. Il campo di funzionamento dei sensori IR è circa da 10 cm a 80 cm con un output dai 0.10 V ai 2.5 V rispettivamente, mentre il sensore di distanza ad ultrasuoni misura da circa 0.15 m a 6.00 m con valori di uscita pari a circa 0.010 V/inch (circa 0.004 V/cm) quando alimentato a 5 V.

Per una migliore robustezza delle misurazioni altri due ingressi analogici sono poi utilizzati per la lettura del riferimento massimo di tensione, pari alla tensione nominale di alimentazione (VSS = 5 V) e per il riferimento minimo di tensione pari al segnale di massa (GND). Questo si è reso necessario considerando che l'alimentazione comune di scheda, servomotori, e sensori può variare in funzione di fattori come la scarica delle batterie ed è quindi necessario che l'ADC conosca il reale valore massimo di alimentazione per una maggiore precisione nelle conversioni.

All'interno del programma eseguito dal PIC, per poter garantire una

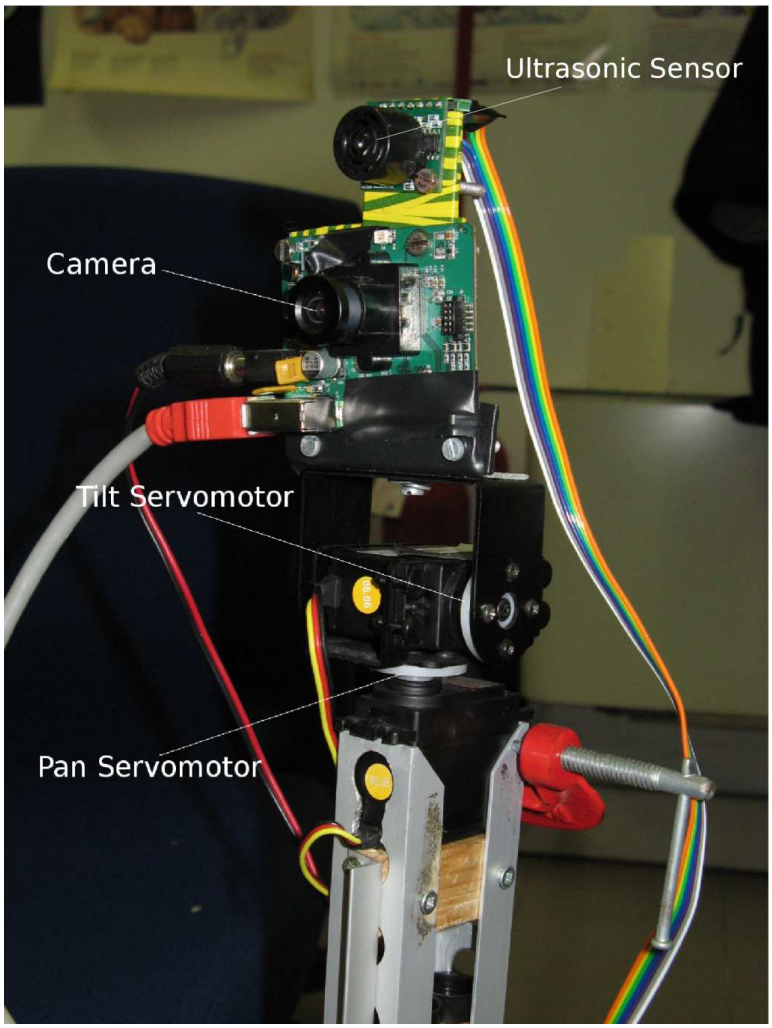


Figura 5.12: Torretta di controllo con telecamera e servomotori di pan e tilt

corretta modulazione PWM e a causa dei tempi minimi di attesa tra le acquisizioni, le misurazioni dei quattro sensori vengono eseguite alternando un sensore per ogni ciclo di esecuzione per poi inviare al calcolatore i valori letti al ciclo successivo.

Pertanto, essendo il ciclo del programma principale ripetuto all'infinito, soltanto ogni 4 cicli si avranno tutti e quattro i valori dei sensori e al quinto ciclo verrà inviato il messaggio al calcolatore. Tale scelta, oltre a mantenere rapido il tempo di esecuzione per ogni ciclo e permettere quindi una corretta temporizzazione dei servomotori, evita di inviare al calcolatore messaggi vicini con identici valori dei sensori poiché ogni messaggio è compilato con

quattro nuove misurazioni.

Da quanto riportato si può osservare che il canale di comunicazione da calcolatore a scheda viene utilizzato solo per il comando dei servomotori, mentre sul canale opposto (da scheda a calcolatore) viaggiano i valori di misurazione dei sensori del veicolo.

La periferica seriale a bordo del microcontrollore PIC16F877A è una RS232 standard, full-duplex, con un buffer di lettura da 1 byte. Nella situazione in oggetto essa è configurata a 115200 bps di velocità e gestita tramite interrupt sulla ricezione. In sostanza, non appena il buffer di ricezione sul PIC si riempie, un segnale di interrupt blocca la normale esecuzione del programma principale e impone al microcontrollore di memorizzare il comando di posizionamento servi ricevuto. Tale comando sarà poi processato nella relativa sezione del programma principale al termine della routine di interrupt a gestione della seriale.

5.3 Il movimento della telecamera

Al fine di poter mantenere con efficienza l'inquadratura sull'oggetto di cui effettuare il tracking è necessario poter ruotare la telecamera senza dover ruotare sempre l'intero veicolo.

Tra le varie soluzioni possibili si è deciso di utilizzare una coppia di servomotori da aeromodellismo, spesso utilizzati nell'ambito della robotica (vedi sezione 5.1.6). Il loro movimento è controllato da una scheda (sezione 5.2) che risulta collegata al calcolatore attraverso un'interfaccia seriale RS232.

5.3.1 Interfacciamento della scheda di controllo

Il funzionamento della scheda è semplice: i comandi vengono decisi dal calcolatore sulla base delle informazioni presenti nell'immagine e inviati sotto forma di stringhe di caratteri ASCII all'interfaccia seriale del microcontrollore presente sulla scheda. Poiché il calcolatore utilizzato dispone soltanto di un'interfaccia USB, è stata utilizzata una scheda dotata di convertitore USB-RS232 affinché la comunicazione potesse avvenire in maniera trasparente al microcontrollore.

La gestione della comunicazione seriale da parte del microcontrollore è effettuata secondo un paradigma ad interrupt, ovvero ogni qualvolta un carattere si presenta nel buffer di input, viene generato un segnale di interrupt, ed il chip interrompe l'esecuzione del programma principale per memorizzare il carattere ricevuto. Data l'alta velocità della comunicazione, che avviene a 115200 bit per secondo, una stringa di pochi caratteri, quale

è il comando di movimento dei servomotori, genera una serie di interrupt sequenziali che vengono di fatto gestiti uno dopo l'altro. Ciò è dovuto alla presenza, sul microcontrollore, di un buffer di ricezione non particolarmente ampio, equivalente ad 1 singolo carattere. Pertanto, per evitare di generare un errore di "overrun" dovuto al buffer pieno alla ricezione di ogni nuovo carattere, è necessario processare rapidamente i singoli caratteri. Il paradigma ad interrupt utilizzato è quindi stato scelto affinché la comunicazione sia esente da errori di questo tipo.

5.3.2 Comunicazione seriale e modulazione PWM

Uno dei problemi più evidenti riguarda la temporizzazione del controllo dei servomotori e l'esigenza che questa non influisca negativamente sulla possibilità di ricevere i comandi di posizionamento. Come regola fondamentale si è cercato di ridurre al massimo il codice presente nella routine di interrupt, eseguendovi le sole istruzioni fondamentali e delegando le operazioni più complesse al ciclo principale.

I servomotori utilizzati sono infatti comandati da un segnale modulato ad ampiezza di impulso (PWM), secondo una precisa temporizzazione proporzionale all'angolo di posizionamento. È necessario quindi generare un segnale ogni 10ms a livello di interrupt per poter sincronizzare correttamente la modulazione dei servomotori

Poiché la scheda deve sia generare il segnale PWM sia essere in grado di ricevere dal calcolatore i valori dei set point per i servomotori di posizionamento della telecamera, si è reso necessario decidere la priorità tra le due operazioni. A questo proposito, disponendo di un solo livello di interrupt, si è deciso di prediligere la ricezione dei comandi sul posizionamento effettivo, assegnando un controllo ad interrupt ai primi e lasciando al programma principale il secondo compito.

Questo comportamento, a prima vista atipico, è giustificato dal fatto che il movimento dei servomotori si effettua tramite comandi di posizione che vengono rapidamente aggiornati ad ogni fotogramma dell'immagine. Un piccolo ritardo nella modulazione porta ad errori di posizionamento stimati a circa 1° per comando, che risulta quindi trascurabile data la natura dinamica del posizionamento. D'altro canto un piccolo ritardo nella gestione della comunicazione seriale danneggia irrimediabilmente la correttezza dei dati ricevuti, perdendo completamente il nuovo valore di posizionamento e decrementando sensibilmente la reattività dell'applicazione. Ecco perché risulterebbe più dannoso garantire una esatta modulazione del segnale di

posizionamento, piuttosto che ricevere sempre correttamente il valore con cui generare tale segnale.

Si è così deciso di permettere alla gestione della comunicazione una breve interruzione del segnale PWM generato, gestendo dunque il controllo dei servomotori all'interno del programma principale. Va comunque considerato che per ridurre al massimo il ritardo introdotto, i comandi, inviati dal calcolatore, comprensivi dei set point per i due servomotori, hanno dimensioni totali di 4 caratteri ASCII (2 byte per la sincronizzazione del flusso di comandi, 2 byte per entrambi i valori di set point).

5.3.3 Funzionamento del firmware

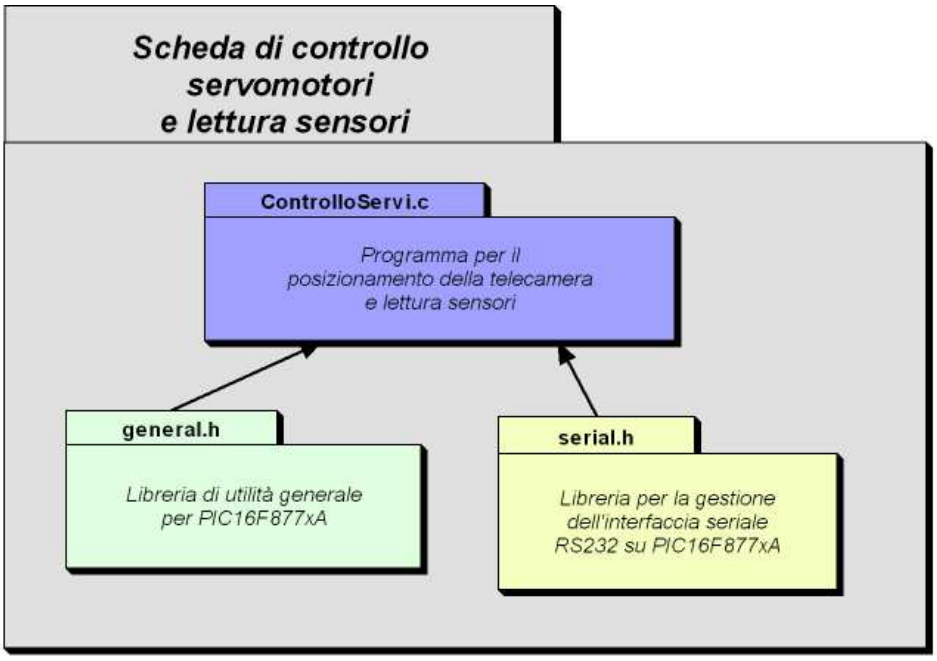


Figura 5.13: Programma per il movimento della telecamera

Come si può notare in figura 5.13, il firmware del microcontrollore si avvale di alcune semplici librerie sviluppate per modularizzare la gestione della comunicazione seriale ed altre funzioni di utilità generale.

Al di là di questo fatto il programma eseguito dal microcontrollore sulla scheda di controllo è diviso in due parti: il codice gestito a livello di interrupt ed il codice eseguito a ciclo infinito, chiamato anche programma principale o ciclo principale. (A tal proposito si veda la figura 5.14)

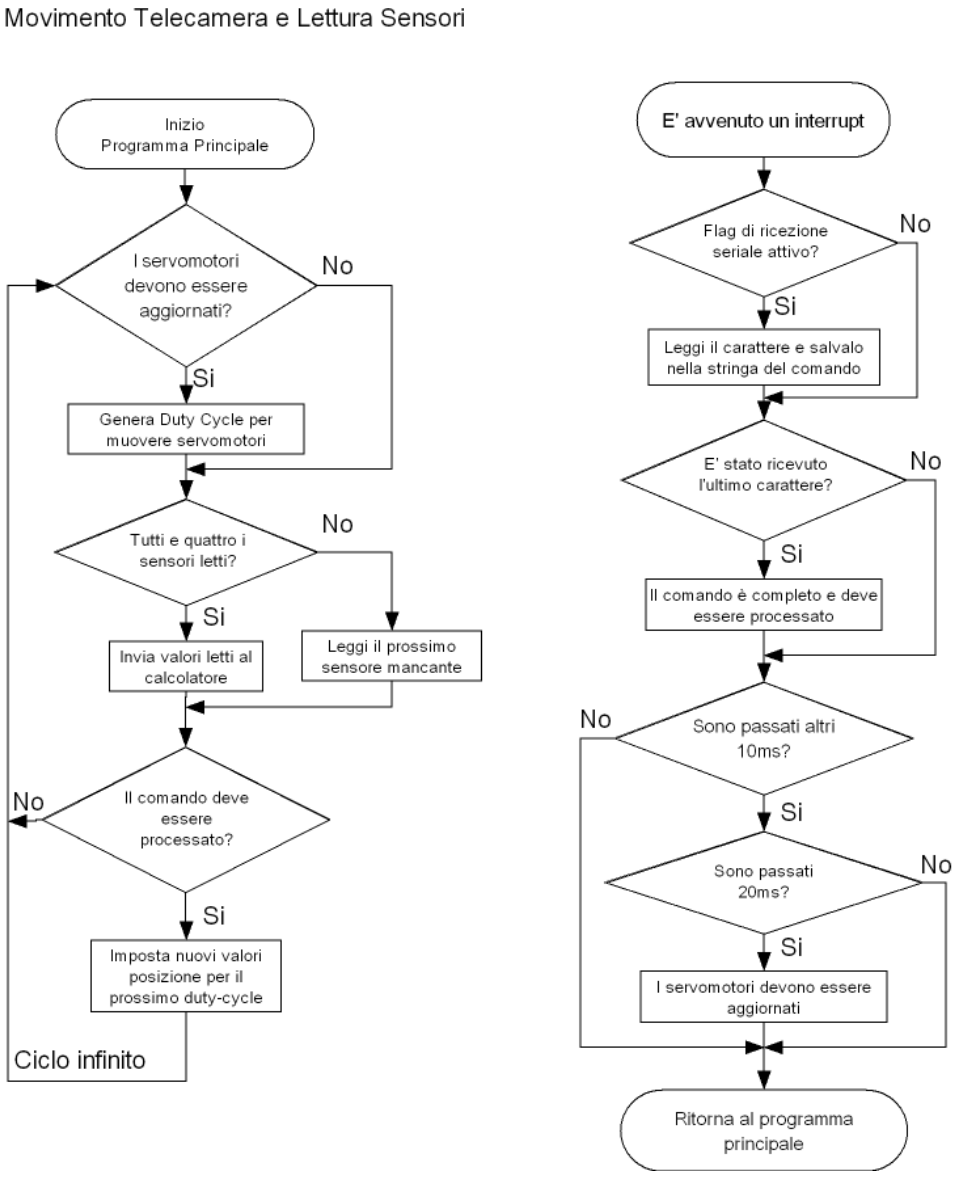


Figura 5.14: Diagramma di flusso del programma di movimento servomotori e lettura sensori

Per quanto riguarda l'interrupt viene generata un'interruzione sul ciclo principale ad ogni carattere ricevuto su interfaccia seriale, oppure ogni 10 millisecondi di tempo. In particolare, la routine di interrupt svuota il buffer di ricezione, di capacità un singolo byte, memorizzando il carattere letto in una stringa. I caratteri vengono memorizzati finché non si riceve un carattere

di terminazione (pari al valore “200” nella codifica ASCII), nel qual caso si segnala al programma principale la presenza di un nuovo comando completo da processare. In caso di errori nella comunicazione la stringa parzialmente ricevuta viene svuotata e si ricomincia da capo la ricezione.

Come seconda operazione a livello di interrupt, ogni 20ms (si considera la ripetizione di due timeout da 10ms ciascuno) viene impostata una variabile che segnala la scadenza del periodo della PWM e la necessità di rinfrescare la posizione dei servomotori generando un nuovo duty cycle.

Il ciclo principale si occupa invece innanzi tutto della generazione del duty cycle del segnale PWM, per posizionare i servomotori sulla base dell'ultimo valore ricevuto dal calcolatore. Una volta terminata la generazione di tale segnale (che può durare da uno a due millisecondi) si occupa della gestione dei sensori (vedi sezione 5.4.3 più avanti) e può poi processare l'ultimo comando valido ricevuto. In particolare si controlla se esiste un nuovo comando valido con nuovi valori di posizione. In caso affermativo si aggiornano le variabili predisposte al posizionamento dei motori, che saranno utilizzate nel prossimo ciclo durante la generazione del nuovo duty cicle.

5.4 Il movimento del veicolo

È chiaro che per poter muovere correttamente un robot in un ambiente è necessario poter disporre di informazioni su di esso e sugli eventuali ostacoli presenti.

I sensori di distanza utilizzati servono perciò a poter verificare l'assenza di ostacoli nei pressi del veicolo, e controllarne di conseguenza i movimenti in sicurezza.

5.4.1 Distanza dall'obiettivo e presenza di ostacoli

Sui quattro sensori a disposizione, i tre sensori ad infrarosso sono stati posizionati sul veicolo in posizione frontale e sui due angoli posteriori. Così facendo è possibile identificare ostacoli posti di fronte o sul retro del veicolo, consentendo inoltre di verificare le possibilità di rotazione grazie ai sensori posteriori in posizione angolare.

Il restante sensore ad ultrasuoni è stato invece posizionato proprio al di sopra della telecamera, per poter approssimativamente misurare la distanza dall'oggetto inquadrato, e quindi poter regolare il movimento di avvicinamento e inseguimento del veicolo all'obiettivo.

La lettura dei valori dai sensori è effettuata dalla scheda che comanda anche i servomotori (vedi sezione 5.2).

5.4.2 Assi di movimento

Dal momento che il veicolo sfrutta un sistema di movimento di tipo "differential drive" (vedi sezione 5.1.1), esso può muoversi sia lungo l'asse longitudinale sia ruotare in senso antiorario od orario. Il movimento finale viene così considerato dalla somma vettoriale dei due tipi di movimento ed è possibile in maniera semplice realizzare anche movimenti complessi. (Si pensi ad esempio al fatto che, per seguire un obiettivo in movimento lungo una traiettoria curvilinea, è sufficiente gestire la differente rotazione delle due ruote del veicolo, sommando i contributi traslatori con quelli rotazionali nel movimento finale).

Il movimento del robot è affidato alla scheda di controllo AirBoard (vedi sezione 5.1.2)

5.4.3 La lettura dei sensori

Il valore di tutti e quattro i sensori di distanza è dunque letto dal microcontrollore attraverso il convertitore analogico/digitale integrato e inviato al calcolatore ogni qual volta si hanno tutte le nuove misurazioni. Ogni quattro cicli del programma principale si effettua la lettura in sequenza di un sensore diverso per ciclo. Al quinto ciclo viene invece spedita al calcolatore la stringa contenente le quattro misurazioni. In pratica la sequenza che si ripete ogni cinque cicli è questa: lettura sensore ultrasuoni, lettura sensore infrarossi frontale, lettura sensore infrarossi sinistro, lettura sensore infrarossi destro, invio quattro misurazioni al calcolatore. Mantenendo ridotto il tempo di esecuzione di ogni ciclo, che in tal modo garantisce la buona temporizzazione dei servomotori, questo comportamento non ha ripercussioni negative sulla reattività del sistema.

Per quanto riguarda il segnale analogico proveniente dai sensori al microcontrollore, viene effettuata una conversione a 10 bit, di cui vengono però considerati soltanto gli 8 bit più significativi. Questo avviene per filtrare la misura e per snellire ancora una volta la comunicazione seriale: avendo valori ad 8 bit è possibile spedire le quattro misurazioni tramite codifica ASCII (un carattere per misurazione) generando stringhe lunghe soltanto 6 caratteri (includendo carattere iniziale e terminatore) o 48 bit.

L'invio delle misurazioni avviene a ciclo continuo ogni volta che si hanno a disposizione tutte e quattro le nuove misurazioni. Questo evita al calcolatore di dover espressamente richiedere le misurazioni e ritardare ulteriormente il programma principale, evitando ripercussioni sulla temporizzazione del segnale PWM.

Il calcolatore deve quindi leggere sul proprio canale di input seriale le misurazioni di distanza e decidere i movimenti conseguenti del veicolo. Questi vengono così inviati alla seconda scheda di controllo (AirBoard) adibita al movimento del veicolo.

5.4.4 Coordinazione veicolo-telecamera

Per poter coordinare i movimenti del veicolo con quelli della telecamera, si è deciso di rendere la telecamera più reattiva rispetto alle rotazioni del veicolo. Disponendo infatti di controllori separati per il comando dei due moduli, si ha un positivo effetto collaterale di coordinazione tra i due movimenti.

L'angolo di pan della telecamera viene aggiornato con un controllore parabolico quando l'oggetto esce dal centro dell'inquadratura. In sostanza viene considerata una zona neutra, nel centro dell'inquadratura, pari ad un area di 40 x 40 pixel, al cui interno deve essere mantenuto l'oggetto tracciato. Qualora esso ne esca, viene valutata la posizione in pixel del centro dell'oggetto da inseguire. Quando tale valore si avvicina ai bordi dell'immagine lungo l'asse x o y, si sposta la telecamera affinché l'oggetto torni nel centro dell'immagine inquadrata. Per quanto riguarda l'asse x, valori estremi di 0 o 640 pixel portano ad una correzione istantanea dell'angolo di pan pari a -6° e +6° rispettivamente. Parimenti se la posizione sull'asse y è di 0 o 480 pixel, l'angolo di tilt viene aggiornato istantamente con valori relativi di -6° o +6°. La legge che governa le correzioni del posizionamento dei servomotori in funzione della posizione del centro dell'oggetto nell'immagine è di tipo quadratico, da qui il nome parabolico.

Si osservi che tale parametro influenza particolarmente la reattività del sistema. I valori indicati rappresentano comunque il miglior compromesso, per la velocità dei servomotori utilizzati, tra capacità di mantenere l'obiettivo al centro dell'inquadratura ed assenza di sovraelongazioni nel posizionamento della telecamera.

La rotazione del veicolo è invece ottenuta impostando il setpoint dei due motori considerando due contributi sommati tra loro.

$$LeftMotor = -\frac{(RotationControl+(PanValue-90))}{4}$$

$$RightMotor = -\frac{(RotationControl+(PanValue-90))}{4}$$

Come si vede dalla formula, il primo contributo, "RotationControl", è l'output di un secondo controllore parabolico che, similmente a quello dedicato all'aggiornamento dell'angolo di pan, prende come ingresso il centro

dell'oggetto inquadrato e genera valori massimi di +10 e -10 a seconda della posizione di tale oggetto. (In questo caso l'area centrale neutra è di 60 pixel lungo l'asse x). Il secondo addendo è invece rappresentato dall'angolo compreso tra normale alla superficie della telecamera e direzione del veicolo. La somma finale è infine riscalata con un fattore di divisione negativo, sicché il valore finale è un contributo inferiore a zero.

Come si può notare dalla formula, la normale al veicolo, espressa in coordinate di pan, è un valore costante di 90°. In altre parole, il veicolo conosce l'angolo di pan correntemente assunto dalla telecamera, rispetto alla sua direzione, e ruota affinché la telecamera si posizioni ad un angolo di pan pari a 90°, annullando dunque la differenza tra direzione del veicolo e normale alla telecamera.

Ad esempio si supponga che la telecamera inquadri un oggetto al centro dell'immagine con valore di pan pari a 130°. Il veicolo comincerà a ruotare, in senso orario per effetto dell'angolo di pan maggiore di 90°, ed anche la camera, solidale al veicolo, tenderà a ruotare in senso orario. Tuttavia il controllore della camera controbilancierà la rotazione oraria muovendola in senso antiorario per mantenere l'oggetto al centro dell'inquadratura. Al termine dei due movimenti si configura la situazione in cui vi sono la telecamera posizionata a 90° di pan ed il veicolo diretto verso l'oggetto, esattamente al centro dell'inquadratura.

Capitolo 6

Prove sperimentali e valutazione

6.1 La fase di testing

La fase di testing è forse una delle più delicate. È il momento in cui il prototipo realizzato mostra punti di forza e debolezza. Si raccolgono i frutti del proprio lavoro considerando successi e insuccessi.

6.2 Tipologie di test

I test sull'applicazione sono stati condotti lungo tutta la fase di progettazione e realizzazione. Ogni modulo HW e SW è stato testato, per quanto possibile, singolarmente. In seguito, è stato testato il comportamento dell'insieme dei vari moduli e si sono eseguiti vari tipi di test cercando di condurre delle prove significative. Si è inoltre cercato di produrre delle situazioni di crescente complessità, per evidenziare soprattutto le criticità del prototipo.

I parametri sui quali è stato valutato il progetto discendono direttamente dalle specifiche assegnate:

- operare in un contesto real time e dinamico
- navigare in modo sicuro in presenza di ostacoli e persone
- l'utente deve poter selezionare un qualsiasi oggetto o la persona desiderata da inseguire in modo univoco

- tracking su target in movimento, camera in movimento e robot fermo
- tracking su target in movimento, camera in movimento e robot in movimento

6.3 Operare in un contesto real time dinamico

Operare in un contesto real-time significa confrontarsi con gli agenti dello stesso contesto e reagire in tempi ragionevolmente rapidi per poter interagire con questi sullo stesso piano. Nel caso in esame, gli agenti sono delle persone, abbiamo quindi preso come punto di riferimento le "prestazioni" di un essere umano e le abbiamo confrontate con quelle del robot.

6.3.1 Velocità di acquisizione ed elaborazione dei frame

L'occhio, e soprattutto il cervello umano, non è in grado di apprezzare frequenze di refresh superiori ai 25 fps (frame per secondo). Purtroppo non ci è possibile raggiungere questa velocità con la camera in dotazione poiché nella modalità di utilizzo, ovvero 640x480 a colori, essa non è in grado di superare i 15 fps. La frequenza massima concessa è quindi molto inferiore a quella ideale. L'ideale sarebbe campionare non solo a 25 fps, ma spingersi addirittura oltre, ottenendo un sistema estremamente reattivo con performance di eccellenza.

L'obiettivo è quindi di sfruttare a pieno le caratteristiche dell'hardware in dotazione. Oltretutto, il sistema è pensato per essere una parte di un sistema più complesso ed è quindi necessario prevedere l'aggiunta di altri moduli. Nel caso in cui la somma dei tempi necessari all'esecuzione di ogni modulo superasse una certa soglia, la richiesta operativa, real-time, non potrebbe essere soddisfatta.

A questo proposito, nell'interfaccia è stato introdotto un box in basso a sinistra che mostra la frequenza istantanea di acquisizione ed elaborazione delle immagini. Questo indicatore è stato costantemente monitorato durante tutta la fase di progettazione come indice di velocità dell'applicazione. La frequenza di elaborazione si è rivelata come una degli strumenti principali per la valutazione dell'impatto di nuovi algoritmi sull'applicazione. Inevitabilmente, all'introduzione di un nuovo strumento o algoritmo la frequenza calava rendendo indispensabili successive ottimizzazioni del codice per riottenere tempi di acquisizione accettabili. Nella configurazione finale, un'ulteriore contributo alla riduzione di questo parametro si ottiene disabilitando la visualizzazione delle immagini nell'interfaccia.

Alla fine del progetto, i risultati ottenuti sono stati positivi. Il sistema è in grado di sfruttare a pieno la velocità di acquisizione della camera e nel contempo di elaborare ogni frame disponibile. Questo primo risultato è particolarmente positivo soprattutto nell'ottica di integrare altri moduli nel progetto. Nonostante il carico computazionale relativamente elevato, che tipicamente accompagna i progetti di visione artificiale, al sistema rimangono quindi risorse disponibili per l'esecuzione di altri compiti.

6.3.2 Robustezza ai cambi di luce

In un contesto dinamico una delle variabili che influenzano maggiormente la visione è sicuramente l'intensità luminosa, che influisce su tutti gli algoritmi in uso: modifica i colori, modifica la nitidezza dell'immagine, genera ombre, ecc...

Essa non influisce invece sulle rilevazioni date dai sensori di distanza poiché questi sensori sono di tipo attivo e quindi sono fonti dello stesso segnale che catturano. Pertanto, nel caso in cui le condizioni di luce dovessero rendere inutilizzabile la camera, il robot sarebbe in grado di continuare ad evitare ostacoli, pur non rilevando più il target. È questa una peculiarità importante da tenere in considerazione.

Per quanto riguarda i test condotti all'interno del laboratorio in presenza di luce artificiale (neon), per verificare la resistenza ai cambi di luce è stata utilizzata dapprima una lampada a fluorescenza da 20 W in grado di produrre un flusso luminoso di 1230 lm[1] (corrispondente ad una normale lampadina ad incandescenza da 100 W), ma i risultati degli esperimenti si sono rilevati troppo qualitativi.

Per questo sono stati condotti test specifici utilizzando una sorgente di luce variabile. Con i mezzi a nostra disposizione è stato possibile valutare le prestazioni dell'algoritmo per un intervallo di intensità luminosa compreso tra circa 100 lx e circa 1200 lx[2].

L'intensità rilevata, in condizioni d'illuminazione normali all'interno del laboratorio, equivale al valore di partenza, ovvero tra gli 80 e i 150 lux. È possibile vedere il comportamento dell'applicazione a questo livello di luminosità nell'immagine 6.1.

[1] Il lumen, in sigla lm, è l'unità di misura del flusso luminoso. Equivale al flusso luminoso rilevabile in un angolo solido di 1 steradiante emesso da una sorgente isotropica con intensità luminosa di 1 candela. Ne discende che la stessa sorgente isotropica con intensità luminosa di 1 candela emette un flusso luminoso totale di 4π lumen.

[2] Il lux (simbolo lx) è l'unità di misura per l'illuminamento usata dal Sistema Internazionale. Un lux è pari a un lumen fratto un metro quadrato. Lo strumento di misura utilizzato ha risoluzione di 1 lx ed è in grado di rilevare fino a 40000 lx

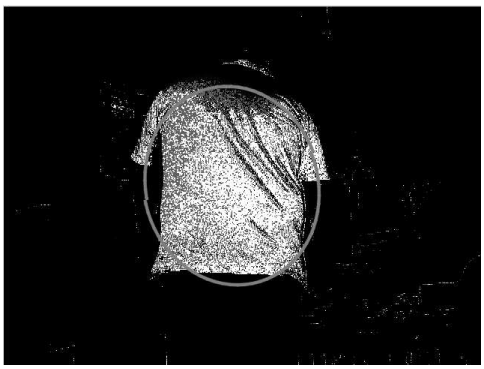


Figura 6.1: Il filtraggio colore in condizioni normali di illuminazione interna (100 lx)

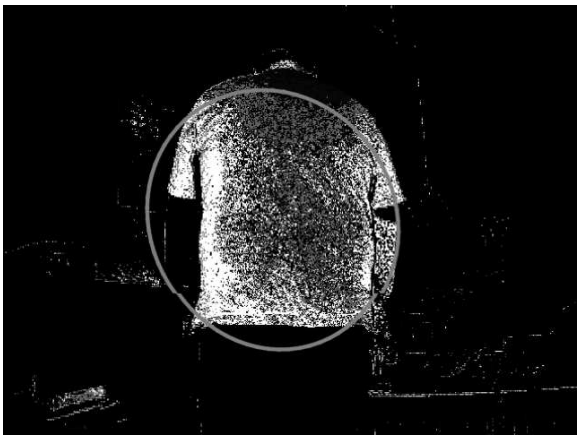


Figura 6.2: Il filtraggio colore con illuminazione a 350 lx

Come è possibile notare nella sequenza di immagini 6.1 6.2 6.3 6.4, la luce modifica l'aspetto dei colori causando un cambiamento nella zona evidenziata dalla selezione colore. Allo stesso tempo l'interazione tra l'algoritmo colore e le altre strategie rende stabile il riconoscimento, mantenendo l'applicazione robusta ai cambiamenti di luce. I risultati sono quindi positivi per cambi di luce relativamente limitati e graduali. In questi casi, il sistema è in grado di compensare il mutamento delle condizioni di ripresa tramite il bilanciamento del bianco automatico e l'aggiornamento del modello colore.

Il comportamento è diverso se la dinamica del cambiamento di luce è molto veloce, un ultimo test è stato condotto per mostrare l'effetto del passaggio da intensità di 100 lx a 1200 lx nell'arco di meno di un secondo. Per vedere l'effetto di questo cambiamento è sufficiente confrontare l'immagine

Figura 6.3: Il filtraggio colore con illuminazione a circa 900 lx

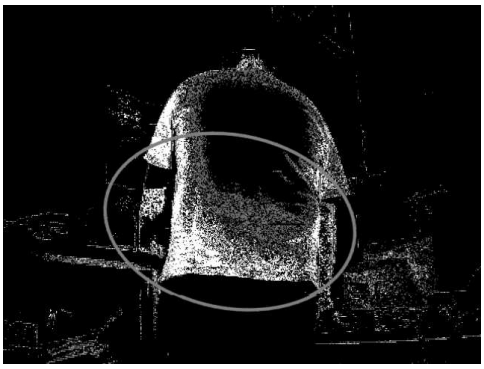


Figura 6.4: Il filtraggio colore con illuminazione a circa 1250 lx

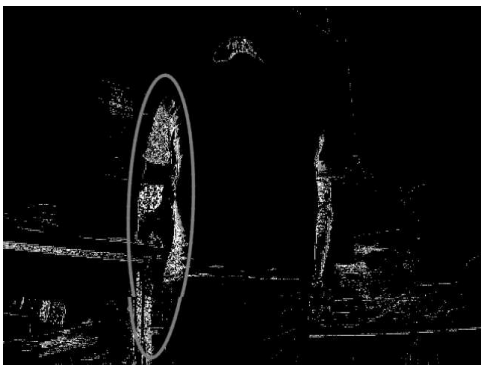


Figura 6.5: Passaggio da una illuminazione pari a 100 lx a una pari a 1200 lx in meno di un secondo

6.1 con l'immagine 6.5. È possibile notare come il repentino cambiamento di luminosità modifichi la scena in modo talmente radicale da impedire al modello di adattarsi.

Per ulteriori valutazioni sui cambiamenti di luce si rimanda comunque alla sezione 6.7.1.

6.3.3 Il movimento, rapidità

La camminata di un uomo supera raramente la velocità di 5/6 Km/h, il robot deve quindi essere in grado di restare al passo. I motori montati forniscono sufficiente potenza per far muovere il robot a velocità ben più sostenute di questa, risultando quindi sovradimensionati. Non volendo testare un robot da corsa, ci siamo limitati ad usare non più del 30% della coppia massima dei motori, che comunque viene raggiunta solo in casi molto limitati. I movimenti del robot sono inoltre proporzionali alla distanza dal target, quindi naturalmente interpretabili: veloci se l'obiettivo è lontano, lenti se è vicino. Nessun problema quindi per quanto riguarda la velocità operativa del robot.

6.4 Navigare in modo sicuro per sé e per le persone

Il compito del robot è seguire una e una sola persona, tutto ciò che è diverso dal suo target deve essere considerato un ostacolo da cui mantenersi ad una ragionevole distanza secondo una politica di massima prudenza.

I sensori di distanza forniscono contemporaneamente la misura dello spazio libero tra il robot e il primo ostacolo visibile anteriore e posteriore. Il sensore ad ultrasuoni fornisce invece la distanza tra la camera e l'oggetto tracciato.

6.4.1 Tipologie di ostacoli rilevati

Il vero problema riguarda le barriere architettoniche: il robot, poiché provvisto di piccole ruote e di una altezza dal suolo esigua, non è in grado di superare ostacoli anche di piccola entità. Il posizionamento dei sensori è stato pensato per evitare tutti gli ostacoli più alti di circa 15 cm mentre gli ostacoli più bassi non possono essere identificati. Questa è una forte limitazione, ma allo stadio attuale è risultata la soluzione più economica e rapida in tempi di sviluppo per dare comunque al robot un buon grado di autonomia. Grazie alla coppia dei motori è invece in grado di muoversi su piani inclinati in discesa e salita, a patto di muoversi su superfici abbastanza lisce.

6.4.2 Distanza dalla persona seguita

Un primo test riguarda camera in movimento e target immobile. Esso consiste nel posizionare il robot a circa quattro metri di distanza da un target fermo, inquadrare il target come oggetto da seguire e cominciare l'inseguimento. Il percorso è stato liberato da ogni ostacolo e nessun disturbo è stato introdotto durante il test, affinché il robot potesse portarsi ad una distanza nota dal soggetto rallentando progressivamente fino a fermarsi. Sono state condotte dieci ripetizioni dello stesso test con i seguenti risultati. (Il valore atteso è compreso tra 80 cm e 120 cm).

Test Numero	Distanza tra target e camera.
1	122
2	112
3	99
4	105
5	116
6	118
7	115
8	111
9	101
10	104
media	110.3

Il comportamento del robot in questo caso è risultato sempre quello desiderato: il robot parte alla massima velocità consentitagli per poi rallentare progressivamente in prossimità dell'obiettivo. La media delle ripetizioni è compresa tra le misure prefissate. Inoltre, come atteso, è spostata verso il valore superiore. L'interazione tra controllo del robot, della camera, sensori di distanza e sensore di visione permette di eseguire con affidabilità e sufficiente ripetibilità l'avvicinamento alla persona. Si considera quindi questo primo test operativo superato.

6.5 Selezione univoca di un oggetto o di una persona

Caratteristica fondamentale di un sistema di tracciamento è indubbiamente la capacità di identificare univocamente il soggetto di interesse (Vedi figura 6.6).

Figura 6.6: Due persone vestite con colori simili non vengono confuse

Si cercherà in questa sezione di evidenziare i casi in cui il modello da inseguire viene confuso con elementi simili nell'immagine (falsi positivi) e casi in cui il modello non viene riconosciuto pur essendo presente nell'immagine (falsi negativi).

Il problema della presenza di falsi negativi si presenta fin dalla fase di selezione del modello. Come già descritto nel capitolo 4 esistono due modalità di selezione dell'oggetto da inseguire: da parte dell'utente e automatica.

Nella selezione da parte dell'utente se la zona selezionata ha dimensioni inferiori ai 10 pixel il modello non viene creato, le informazioni relative ad una zona cosi ridotta vengono infatti considerate non sufficienti alla creazione di un modello significativo.

Esistono inoltre condizioni di selezione particolarmente sconsigliate: se la selezione inquadra zone in cui è presente solo nero intenso o bianco, l'istogramma colore risulta inconsistente rispetto alla creazione del modello. Una regione di colore nero porta ad un istogramma colore troppo ampio e poco localizzato mentre in una regione bianca si ottiene l'effetto opposto con un istogramma colore tendenzialmente privo di ogni colore. Per ulteriori approfondimenti si rimanda alla sezione 4.1.2.

Se, in aggiunta, l'area selezionata è priva di corner o caratteristiche univoche utili al template matching, il modello generato non può essere affidabile. In tali casi il target pur presente nella scena non può essere riconosciuto.

Alcuni di questi problemi sono risolti dalla selezione automatica, durante la quale la selezione del busto viene fatta in base all'identificazione del volto risultando molto più affidabile. In questa fase di set-up è però necessario rispettare alcune regole: la persona deve presentarsi frontalmente al robot e tutta la parte superiore del suo corpo deve essere inquadrata dalla telecamera. Inoltre, deve essere l'unica persona inquadrata dal robot o quantomeno la prima alla sua sinistra. Infatti l'applicazione, non potendo decidere in fase di set-up quale delle persone identificare, sceglie semplicemente la prima alla sua sinistra come mostrato in figura 6.7. In tali casi il target presente nella scena potrebbe arbitrariamente non essere scelto per la creazione del modello.

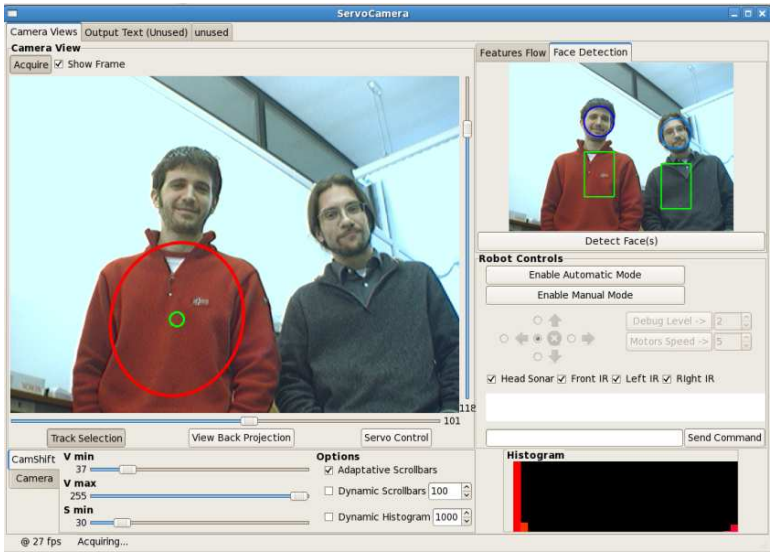


Figura 6.7: L'identificazione di più volti nella stessa immagine: solo il primo è selezionato.

Conclusa la fase di set-up ci occupiamo al caso in cui il sistema rilevi falsi positivi, ovvero in cui il modello risulti maggiormente rispondente ad un oggetto diverso dal target. Questa eventualità si verifica solo nel caso in cui l'aggiornamento del modello, a causa del cambiamento del target o delle condizioni della scena, non sia efficiente.

Una delle manifestazioni più classiche di questo comportamento è l'interazione faccia/mani. In algoritmi di tracciamento basati sul solo colore, ad esempio quello della pelle, volto e mani vengono spesso confusi e riuniti in un unico blob. Questo comportamento è per noi indesiderabile ed si è quindi deciso di ovviare a questo inconveniente.

Basandosi sui corner caratteristici della selezione e sul template matching, si può notare in figura 6.8 come volto e mani non vengano confusi. Il motivo di tale robustezza è dovuto soprattutto al filtraggio del frame di back-projection con la regione dei punti salienti così come descritto nel capitolo quattro, sezione 4.2.5.

Figura 6.8: Faccia e mani pur avendo lo stesso colore non si fondono un un unico blob colore grazie alla presenza di punti salienti sul volto che lo distinguono dalle mani

Un'ulteriore peculiarità del sistema è l'essere resistente alle parziali occlusioni; un esempio è mostrato nella figura 6.9.

In conclusione, sono state evidenziate in questa sezione alcune limitazioni del sistema che comunque non influiscono eccessivamente sulle sue possibilità di impiego.

6.6 Test su target e camera in movimento, robot stazionario

Dopo il primo test operativo descritto nella sezione 6.4.2, in cui venivano valutate le prestazioni del sistema a robot mobile e target fisso, affrontiamo ora il test in cui la posizione del robot è fissa, ma la camera e il target sono in movimento. Il presente test risulta più complicato del precedente, infatti se

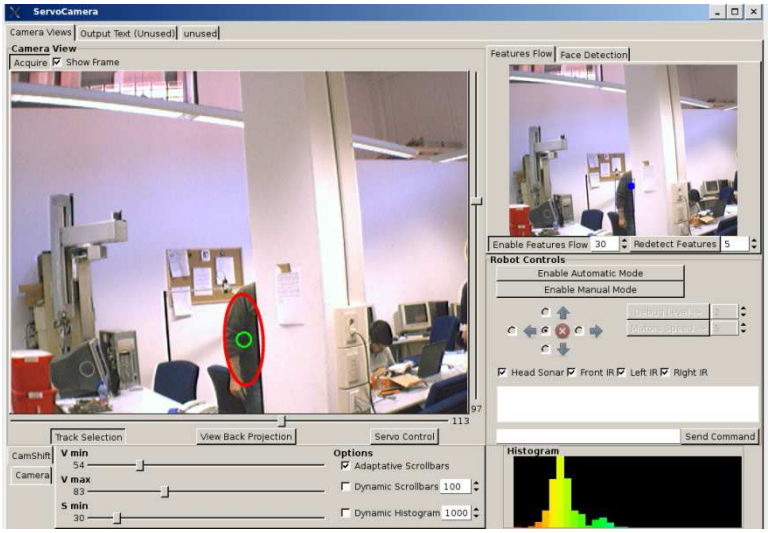


Figura 6.9: Il modello risulta resistente anche alle parziali occlusioni, una persona è parzialmente nascosta dietro una colonna, ma continua ad essere identificata

nel primo, con target immobile, il modello non subiva particolari variazioni, in questo caso verrà messa alla prova proprio la capacità di adattamento del modello ai cambiamenti.

Il test consiste nel posizionare il robot al centro del laboratorio, affrontare la fase di set-up e far seguire alla persona un percorso circolare prestabilito intorno al veicolo uscendo infine dal campo visivo della telecamera. Durante l'esperimento verranno monitorate le reazioni del sistema.

Il robot viene quindi posizionato al centro di una stanza a motori spenti, con la possibilità di muovere la telecamera lungo gli assi di pan e tilt, in grado quindi di inquadrare una zona di fronte e di fianco a sé. La fase di set-up è condotta tramite la procedura automatica ovvero tramite l'identificazione del volto e la selezione del busto come zona di riferimento per la costruzione del modello. Successivamente la persona inquadrata si volterà, si allontanerà dal robot seguendo un percorso in linea retta per circa 4 m e, conclusa questa parte del percorso, aggirerà il robot sfilando al suo fianco.

Il test è stato condotto in laboratorio con le persone presenti che rappresentano l'oggetto del tracciamento. Abbiamo chiesto a loro di muoversi in modo naturale e senza assecondare il robot. Otto test sono stati condotti con persone differentemente vestite e di diverso aspetto, la figura 6.10 mostra la traiettoria eseguita dalle persone, ove sono indicate alcune posizioni notevoli del percorso tramite etichette.

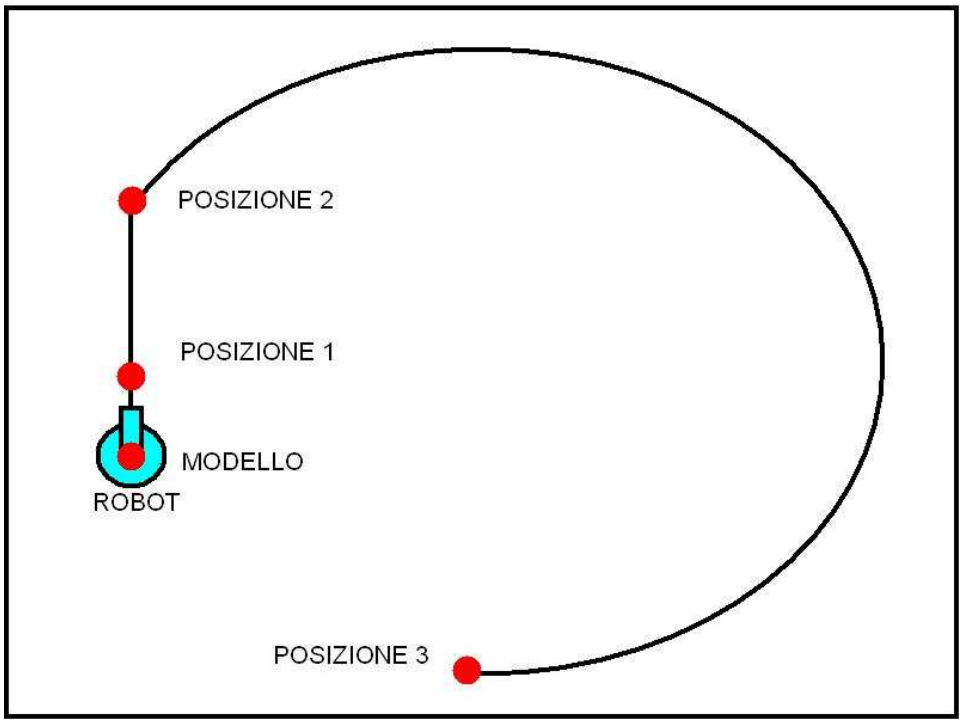


Figura 6.10: Schema della posizione del robot e traiettoria seguita dalla persona tracciata nel test

L'etichetta MODELLO corrisponde al punto di attivazione del robot e creazione automatica del modello.

L'etichetta POSIZIONE 1 corrisponde al momento in cui la persona si è voltata presentandosi di spalle al robot.

L'etichetta POSIZIONE 2 corrisponde al momento in cui la persona si è allontanata dal robot in linea retta.

L'etichetta POSIZIONE 3 corrisponde al momento in cui la persona dopo aver aggirato il robot è uscita dal suo campo visivo.

Il simbolo (Y) nella colonna "Modello" indica che il modello calcolato risulta valido mentre il simbolo (N) indica che il modello non è risultato valido. È stato anche rilevato se nelle successive posizioni il soggetto è stato ancora tracciato (Y) o è stato perso (N).

	Modello	Posizione 1	Posizione 2	Posizione 3
Soggetto 1	Y	Y	Y	Y
Soggetto 2	Y	Y	Y	Y
Soggetto 3	Y	N	N	N
Soggetto 4	Y	Y	Y	Y
Soggetto 5	Y	Y	Y	Y
Soggetto 6	Y	Y	Y	N
Soggetto 7	Y	Y	Y	Y
Soggetto 8	Y	Y	Y	Y

Figura 6.11: Risultati del test con robot e target in movimento

Come è possibile vedere dalla figura 6.11, la maggior parte dei test ha portato ai risultati sperati poiché le persone sono state seguite lungo tutto il loro percorso. Solo nel caso dei soggetti numero 3 e numero 6 il test non ha portato a buoni risultati, i motivi di questi insuccessi sono stati quindi analizzati.

In questi due casi il soggetto è stato perso in due momenti diversi. Nel primo caso il soggetto numero 3 vestiva in modo tale da apparire molto diverso se visto frontalmente o di spalle, causando dunque la perdita dell'inquadratura nel momento in cui la persona si è voltata. Il colore infatti, pur non essendo l'unico paramento su cui si basa il tracking, è sicuramente uno dei principali e, come osservato, ha una grande influenza sul funzionamento globale del sistema.

Comunque, nel momento in cui la persona si volta, anche gli altri parametri del modello subiscono un radicale aggiornamento. I corner caratteristici, ad esempio, cambiano velocemente e con loro l'aspetto del template differisce in maniera sensibile. Una possibile soluzione a questo comportamento potrebbe esser il più rapido aggiornamento del modello, in modo da seguire i cambiamenti repentini del soggetto da seguire. Per questo si potrebbe ricalcolare il modello non appena la persona si volta, aggiornandolo rapidamente per i primi secondi successivi al riconoscimento del volto. Però la maggiore adattabilità del modello che si ottiene porta anche ad una potenziale instabilità.

Nonostante i settaggi con cui sono stati condotti i test siano sembrati i migliori, è stata lasciata agli utenti e agli sviluppatori futuri la possibilità di modificare la velocità di adattamento dell'istogramma colore da interfaccia, senza dover modificare il codice, modificando semplicemente il relativo

controllo presente nell'interfaccia accanto alle funzionalità in questione.

Nel secondo caso, ovvero quello del soggetto numero 6, la persona viene persa durante la fase più complessa del tracking. Nel primo tratto rettilineo, dopo che la persona si è voltata, il soggetto non modifica radicalmente il suo aspetto. In prima approssimazione, egli si limita ad allontanarsi e l'effetto prospettico sulla distanza, relativamente breve, causa soltanto un lieve rimpicciolimento. Nel secondo tratto, ovvero quello in cui il soggetto 6 è stato perso, la persona oltre che allontanarsi ruota rispetto al robot. Si potrebbe perciò paragonare il primo tratto ad una traslazione e il secondo tratto ad una rototraslazione, operazione sicuramente molto più complessa. Tra l'altro, analizzando il comportamento del sistema, si è notato che il motivo della perdita del soggetto 6 non è dipeso direttamente dalla complessità del tratto, ma piuttosto dalla complessità dello sfondo. Infatti in questo caso il modello ha trovato una corrispondenza con un gruppo di scatole, presenti nello sfondo, di aspetto effettivamente diverso dalla persona, lasciando intuire che il problema è probabilmente inverso al caso precedente. Così se nel caso precedente il problema è stato attribuito alla scarsa adattabilità del modello, ora il modello si è adattato fin troppo velocemente e ha considerato una porzione dello sfondo come la meglio rispondente a quanto cercato.

Questi due casi dimostrano che il sistema, pur possedendo un buon grado di affidabilità, è comunque perfettibile. Ad ogni modo le regolazioni alle quali sono stati condotti i casi di test si sono rivelate un buon bilanciamento tra velocità di aggiornamento del modello dinamico e conservazione del modello originale. In conclusione, consideriamo soddisfacenti i risultati ottenuti in questo secondo test operativo.

6.7 Test su target in movimento e telecamera mobile

L'ultimo test condotto è quello più completo e che coinvolge l'intero sistema. È anche quello più complesso poiché il robot viene impiegato non solo per tracciare la persona in movimento muovendo la telecamera, ma anche seguendo attivamente il soggetto.

Il test è stato condotto creando il modello tramite la modalità automatica, anche in questo caso si parte quindi dal volto per identificare il busto. In seguito la persona si volta e percorre un tratto rettilineo fino a raggiungere l'uscita del laboratorio. Qui il robot deve affrontare il passaggio attraverso lo spazio ridotto della porta e un tratto in discesa lungo la pedana per l'accesso. Il tratto prosegue con una stretta svolta a destra e da qui il passaggio

attraverso una seconda porta che si apre sul cortile antistante il laboratorio. Nel cortile, la persona, e di seguito il robot, compiono una ampia svolta a sinistra per attraversare zone di ombra e luce diretta del sole e riguadagnare l'ingresso del laboratorio. (Vedi figura 6.12) Il percorso viene infine ripetuto in senso contrario in modo da tornare al punto di partenza.

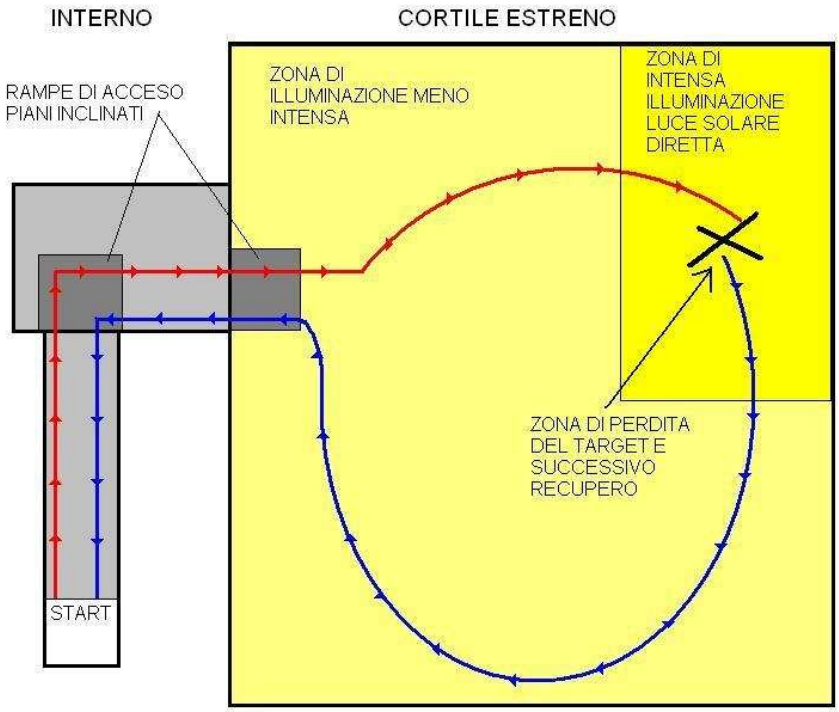


Figura 6.12: Schema della approssimativa traiettoria compiuta dal target e dal robot

Il comportamento del robot durante il test è stato soddisfacente, in quanto la fase di selezione del modello si è rivelata particolarmente efficace e con essa anche le successive fasi di aggiornamento hanno mantenuto il focus sul target in maniera positiva.

Il primo tratto rettilineo non ha rappresentato difficoltà maggiori di quelle affrontate nei test precedenti, ma il passaggio attraverso la porta e la discesa lungo il primo piano inclinato, comunque superati con successo, hanno impegnato il veicolo a causa dei piccoli ostacoli incontrati. In questo tratto l'intensità luminosa è compresa tra gli 80 lx e i 600 lx, ovvero una normale illuminazione ambientale per il nostro laboratorio.

La persona seguita ha usato l'accortezza di attendere qualche istante dopo il passaggio attraverso la porta, in modo da consentire l'avvicinamento

del robot e non uscire dal suo campo visivo trovandosi nascosta dietro al muro. Il secondo tratto, fino all'uscita nel cortile, si è rivelato tuttavia più complesso a causa degli spazi di azione ridotti.

Le vere complicazioni sono sorte passando dall'interno all'esterno del laboratorio allorché l'intenso cambio di luce ha imposto un rapido cambiamento del modello, infatti l'intensità luminosa è gradatamente aumentata superando il valore di 1000 lx fino a stabilizzarsi intorno ai 4000 lx. La variazione di luminosità in questo caso si è rivelata davvero intensa e molto superiore a quella testata in laboratorio con la lampada in grado di portare le condizioni fino a 1200 lx. Malgrado i timori, il sistema è stato comunque in grado di seguire il repentino cambiamento grazie all'interazione dei vari algoritmi implementati. Di fatto, questo è stato un risultato particolarmente positivo: la capacità di adattarsi ad un contesto così mutevole e di seguire una persona nonostante il suo aspetto fosse radicalmente diverso dal modello di partenza, ha incoraggiato a proseguire il test verso la zona del cortile colpita da luce diretta del sole.

Le condizioni di illuminazione, muovendosi verso tale zona, sono così ulteriormente mutate, causando questa volta la perdita del target una volta irraggiato dalla luce solare diretta. Anche se il passaggio tra le due zone diversamente illuminate fosse stato meno rapido, non sarebbe stato comunque possibile aggiornare il modello alla nuova condizione, dato che l'eccessiva luminosità, come già descritto nella sezione 6.5, non permette di valutare correttamente l'istogramma colore dell'obiettivo. La luce solare diretta, infatti, supera facilmente i valori rilevabili tramite il sensore utilizzato, in grado di misurare valori fino a 40000 lx, e può arrivare anche a 100.000 lx in una giornata soleggiata (ben di più dei 100 lx rilevati in laboratorio). A questo punto si è dovuto ricorrere ai comandi manuali per riportare il robot in una zona di illuminazione meno intensa e l'esperimento è proseguito senza intoppi fino alla sua conclusione.

6.7.1 Commento al test finale

Come accennato introducendo questo capitolo, i test servono a mostrare i punti di forza di quello che si è creato, ma anche a scoprirne le debolezze. Il robot ha mostrato un comportamento eccellente finché non si è cercato di "superare le colonne d'Ercole"...

L'aspetto più importante è quindi cercare di comprendere come mai il target sia stato perso nella zona di massima insolazione. Controllando le immagini catturate dal robot e compiendo altri piccoli test sul campo nella stessa zona critica, è stato verificato che la luce diretta del sole modifica pro-

fondamente le immagini. Qualora la telecamera inquadri oggetti illuminati da una forte luce solare, essa risulta "accecata" poiché tutti i colori virano verso il bianco, mentre il contributo di ombre e luci rende indistinta la figura. In sostanza è lo stesso effetto che si potrebbe ottenere in una ripresa o fotografia sovraesposta, dove perciò la quantità di luce immagazzinata è eccessiva.

Supponendo che il bilanciamento automatico della telecamera non fosse in grado di compensare un tale aumento di luminosità, è stato tentato l'utilizzo delle funzioni di luminosità e bilanciamento del bianco manuali inserite nell'interfaccia. Purtroppo neppure questo accorgimento ha portato ai risultati sperati, non riuscendo ad ottenere un'immagine esposta correttamente.

Dunque, se nel primo cambiamento di luminosità il sistema è riuscito a condurre un tracciamento efficace, il risultato è dovuto non solo alla bontà dell'algoritmo adottato, ma anche ad una seconda condizione: le immagini catturate all'interno e fuori dell'edificio, dove la luce è intensa, hanno valori di luminosità entro i limiti gestibili dalla telecamera. Nel caso di luce diretta del sole invece le immagini risultano difficilmente utilizzabili per il riconoscimento di una persona e talvolta incomprensibili allo stesso occhio umano.

Il test finale è stato condotto con il robot in complete condizioni operative, ovvero con veicolo, camera e target in movimento. Coordinare il movimento di camera e robot, compensando i beccheggi tipici del veicolo e i cambi di direzione del target, è un operazione complessa che dovrebbe prevedere l'aggiunta di capacità predittive. I comportamenti del robot appaiono naturali e immediatamente intuibili, ma si sente la mancanza di un modulo di navigazione e autolocalizzazione che sfrutti strategie comportamentali di alto livello e un sistema di sensoristica più completo.

Ad ogni modo per quanto riguarda le considerazioni sugli sviluppi futuri e valutazioni più approfondite sull'intero sistema, si rimanda al capitolo 7.

Capitolo 7

Direzioni future di ricerca e conclusioni

"Ettore: È inutile che continui a girarci intorno, scrivi questa tesi, laureati e se vuoi proprio continuarla, fallo dopo che l'hai finita!
Simone:...va bene, mi hai convinto..."

AiRLab: dialoghi in laboratorio.

7.1 Visione artificiale

Sistemi di visione artificiale intelligenti di vario tipo sono ormai presenti nella realtà comune, dove, ad esempio, camere di videosorveglianza controllano le vie delle nostre città, i luoghi dove lavoriamo o le strade su cui viaggiamo. Gli strumenti dotati di visione, ora semplici spettatori, stanno però tramutandosi sempre di più in agenti attivi, in grado di interagire nel contesto naturale.

In questa ottica il presente progetto si inserisce nell'area della visione artificiale, collegando problemi di tracking e riconoscimento alla robotica mobile ed alle sue problematiche.

7.1.1 Visione e inseguimento di obiettivi

Al fine di tracciare ed inseguire oggetti deformabili in movimento tramite sensori di visione, negli ultimi vent'anni sono state proposte varie soluzioni. Algoritmi stabili ed efficienti per l'identificazione di caratteristiche di basso livello sono ormai implementati e presenti in tutte le librerie di visione, mentre algoritmi ad alto livello, per il riconoscimento di particolari pattern in

contesti specifici, sono in continua evoluzione ed oggetto di studio in questo momento. Nonostante esistano sistemi stabili, benché ancora relativamente onerosi, per il tracciamento del volto, il problema del riconoscimento delle persone è tuttora abbastanza complesso. Un aspetto infatti non trascurabile e non sempre semplice da realizzare, per il riconoscimento e l'interpretazione delle azioni di una persona nel suo ambiente, è garantire un certo grado di integrazione tra il sistema ed il contesto naturale.

7.1.2 Assistenza personale

Esiste un crescente interesse per il potenziale di robot domestici che si prendano cura, per esempio, di anziani e bambini. Realizzare partner per assistenza personale, in grado di lavorare in modo naturale ed efficace sia in casa che in ambienti pubblici, è obiettivo di molte aziende e laboratori di ricerca. Tale coesistenza richiede capacità estremamente sofisticate: l'efficacia di tali robot dipende prevalentemente dalla loro capacità di identificare l'aspetto dei diversi individui. Per accompagnare le persone, devono essere inoltre capaci di riconoscerle e seguirle una volta identificate. È infatti assai diverso "riconoscere" una persona, distinguendola dal contesto inanimato o animale, dalla sua "identificazione", che implica la distinzione tra una particolare persona o un'altra.

7.2 Obiettivo della ricerca

L'obiettivo di questa ricerca è consistito quindi nell'attrezzare un robot in grado di tracciare oggetti complessi e deformabili in un contesto dinamico, con particolare interesse al tracciamento di persone.

Prima di tutto si è deciso di identificare l'individuo grazie ad un sensore visivo e ad un sistema di elaborazione dell'immagine ad alta velocità, riconoscendo istantaneamente il colore dei vestiti della persona. Per questo si è scelta, tra le funzioni messe a disposizione da OpenCV, la funzione che implementa l'algoritmo Camshift [8] come la più adatta alle nostre esigenze di tracciamento. La buona affidabilità è pertanto ottenuta attraverso la costruzione dell'istogramma colore.

Per rinforzare e caratterizzare in modo univoco l'oggetto da inseguire, sono state utilizzate due tipologie di feature: i corner caratteristici ed il template matching. Nel caso dei corner, l'identificazione del movimento è stata inoltre applicata utilizzando procedure per l'identificazione del flusso ottico, combinando il filtraggio dei corner caratteristici con la rappresentazione piramidale dell'immagine, e utilizzando l'algoritmo di Lucas-Kanade [53].

La creazione del modello da seguire avviene partendo dal riconoscimento del volto dell'individuo, realizzato tramite l'utilizzo delle funzioni di face detection.

L'abilità nell'inseguimento di un individuo in movimento richiede delle capacità che vanno ben oltre il riconoscimento facciale, è infatti necessario coordinare i movimenti di camera e veicolo con l'elaborazione delle informazioni visive e dei sensori di prossimità. Quando il soggetto controllato si muove in avanti, anche il robot farà altrettanto; quando la persona si ferma, il robot vi si avvicinerà e si fermerà ad una distanza "di sicurezza".

Ciò è realizzato calcolando costantemente la posizione e la distanza dal soggetto e regolando la velocità del veicolo per mantenere una distanza costante. Non appena il soggetto si muove, il sensore ad ultrasuoni integrato rileva la presenza di ostacoli sulla strada del robot, che elaborerà queste informazioni mantenendo il contatto visivo con la persona.

7.3 Il lavoro realizzato

Al fine di raggiungere lo scopo, è stata realizzata una camera motorizzata, indipendente dal veicolo mobile che la ospita, in grado di articolare il proprio movimento su due gradi di libertà.

Come spesso accade in robotica, la prima fase di lavoro ha richiesto conoscenze nel campo dell'elettronica e meccanica di base. Per la scheda di controllo si è scelto un microcontrollore con interfaccia seriale (affiancato da un convertitore seriale/usb): un sistema semplice ed economico in grado di gestire contemporaneamente ingressi analogici dei sensori, pilotaggio motori e interfaccia di comunicazione.

Dopo la prima fase di messa a punto dei componenti hardware, il lavoro è passato alla programmazione, in prima istanza del microcontrollore e, una volta terminato il primo passo, del programma di tracking sul calcolatore. È stata predisposta un'interfaccia per l'interazione con l'utente, con un paradigma di gestione ad eventi, attraverso la quale è possibile non solo attivare o disattivare le varie funzionalità implementate, ma anche accedere alle loro regolazioni e settaggi.

La base di partenza è stata il riconoscimento del colore, a cui si sono aggiunte altre funzionalità: tracciamento di corner notevoli, tracking di template e riconoscimento facciale.

7.3.1 Problematiche incontrate

La prima fase della realizzazione non ha comportato particolari difficoltà data l'esperienza, acquisita nel corso di progetti precedenti, nell'utilizzo di microcontrollori PIC e servomotori con comunicazione seriale. Problemi più seri sono stati risolti durante la realizzazione dell'algoritmo di tracking.

Il solo filtraggio colore, benché molto veloce e affidabile, non è in grado di tracciare oggetti in modo univoco, a meno che non si tratti di oggetti di tonalità costante e perfettamente diversa dallo sfondo. Ovviamente non è possibile imporre questa condizione in un contesto naturale, oltretutto dinamico nel caso di telecamere in movimento, pertanto si è combinato il filtraggio colore con l'utilizzo di caratteristiche notevoli.

Per non compromettere l'efficacia della soluzione, in un'ottica di utilizzo generale, non sono state fatte assunzioni di nessun genere sullo sfondo dell'area inquadrata. D'altra parte le assunzioni fatte hanno riguardato le condizioni di ripresa e la presenza di volti nell'immagine. Infatti nella fase di set-up automatico si è assunto che la persona si presenti frontalmente al robot, mostrandosi in una inquadratura che ne comprenda quantomento il mezzobusto.

Inerentemente alle condizioni di ripresa, si è cercato di rendere il sistema il più possibile robusto ai cambi di luce e di mantenere un alto frame rate, in modo da ottenere spostamenti relativi tra frame successivi di entità ridotta, garantendo al sistema una certa accuratezza.

Risolto il problema di identificazione e tracciamento si è passati al controllo dei movimenti di camera e robot, in modo da integrare le informazioni provenienti dai sensori di prossimità con quelle generate dal sistema di computer vision.

7.4 Valutazioni sulla realizzazione

Diversi test sono stati condotti per testare l'effettiva efficacia del sistema. I punti chiave evidenziati sono prevalentemente positivi:

- Il frame rate risulta abbastanza elevato da consentire una buona reattività del sistema a repentini cambi di direzione dell'oggetto da inseguire.

- La navigazione del robot, grazie alle presenza dei sensori di prossimità e distanza, è al contempo prudente e sicura per le persone e per il robot stesso.

- Le informazioni ottenute dai sensori di distanza, in particolare dal sonar solidale alla telecamera, si integrano bene nel contesto comportamentale del robot e ne permettono una buona autonomia.

- Il tracking è affidabile anche nel caso di oggetti deformabili, riadattandosi nel caso di variazioni di area.

- Il sistema è robusto qualora siano presenti più oggetti simili: grazie al calcolo dei punti salienti sull'oggetto di interesse, è in grado di isolarlo dagli altri.

- La presenza dei punti salienti garantisce, assieme ad una più robusta rilevazione del blob colore, un buon tracking anche nel caso di parziali occlusioni dell'obiettivo.

- L'inseguimento è robusto anche nel caso di variazioni nell'illuminazione, in particolare tra i 100 lx ed i 1200 lx. Si riscontra un certo degrado di prestazioni soltanto in situazioni di illuminazione "estrema", ovvero tra i 1200 lx ed i 4000 lx. Soprattutto nel caso in cui si superino i 4000 lx il tracking diviene poco affidabile e piuttosto erratico. I repentini cambi di luce causano infatti una certa degenerazione nel calcolo dell'istogramma colore.

- Fatta eccezione per i casi assai sfavorevoli è altresì vero che quanto più lento e graduale è il cambio di luminosità, tanto più il sistema è in grado di adattarvisi.

7.5 Prospettive future

Proseguendo il discorso, lo studio potrà essere proseguito con le finalità di rendere ulteriormente affidabile il tracking nelle situazioni critiche.

In particolare un'idea potrebbe considerare la possibilità di rendere il sistema di inseguimento resistente alle forti luminosità, allontanandosi in tal caso dal concetto di blob colore e pesando maggiormente il flusso ottico, una volta filtrate le ombre presenti nella scena.

Una seconda ipotesi di miglioramento si potrebbe ottenere con l'irrobustimento del sistema alle occlusioni totali attraverso la creazione di un modello statico, da ricercare nell'ambiente nel caso di perdita del target. Ciò permetterebbe inoltre di ridurre i falsi positivi, condizione particolarmente indesiderata in questo caso.

È inoltre possibile aggiungere un predittore per il tracciamento e l'inseguimento dell'obiettivo, che potrebbe affiancarsi ad una più affidabile navigazione, generata da uno specifico modulo.

Infine algoritmi di intelligenza artificiale o sistemi comportamentali (MrBrian [7]) possono essere integrati nel sistema, nell'ottica costante di migliorare il sistema e renderlo sempre più autonomo.

Proprio per questo motivo, tra l'altro, una funzionalità da integrare quanto prima è la possibilità di registrare le configurazioni, impostate tramite l'interfaccia, in un file di descrizione XML. Tale configurazione potrebbe poi essere utilizzata da un esecuzione automatica del programma, quindi senza interfaccia grafica e senza la necessità dell'interazione con l'utente.

Lo studio intrapreso ha dunque permesso di raggiungere risultati compatibili con gli obiettivi prefissati di tracking e inseguimento di persone. Soddisfatti da quanto realizzato, possiamo affermare di aver compiuto un buon passo verso il più ampio obiettivo di un instancabile assistente personale robotico.

⋆⋆⋆

Bibliografia

[1] Kai O. Arras and Sjur J. Vestli. Hybrid, high-precision localisation for the mail distributing mobile robot system MOPS. In *Proc. IEEE International Conference on Robotics and Automation (ICRA'98)*, Leuven, Belgium, 1998.

[2] B. Schunk B. Horn. Determining optical flow. *Artificial Intelligence*, pages 225–239, 1991.

[3] P. Horn B. Klaus. Robot vision. *The MIT Press, Cambridge, Massachusetts*, 1986.

[4] R. Barnes. *Motion Time Study: Design Measurement of Work*. 1980.

[5] Ami Berler and Solomon Eyal Shimony. Bayes networks for sonar sensor fusion. In *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 14–21, 1997.

[6] Hans-Joachim Bohme, Torsten Wilhelm, Jurgen Key, Carsten Schauer, Christof Schroter, Horst-Michael Gros, and Torsten Hempel. An approach to multi-modal human–machine interaction for intelligent service robots. *Robotics and Autonomous Systems*, 2003.

[7] A. Bonarini, M. Matteucci, and M. Restelli. A novel model to rule behavior interaction. *Proceedings of the 8th Conference on Intelligent Autonomous Systems (IAS-8)*, pages 199–206, 2004.

[8] Gary R. Bradski. Computer vision face tracking for use in a perceptual user interface. *Intel Technology Journal*, 1(Q2):15, 1998.

[9] Daniele Calisi, Luca Iocchi, and Riccardo Leone. Person following through appearance models and stereo vision using a mobile robot. In *VISAPP (Workshop on on Robot Vision)*, pages 46–56, 2007.

[10] C.Richards, C. Smith, and N. Papanikolopoulos. Detection and tracking of traffic objects in ivhs vision sensing modalities. *In Proc. Fifth Annual Meeting of ITS America*, 1995.

[11] Cucchiara, C.Grana, G.Tardini, and R.Vezzani. Probabilistic people tracking for occlusion handling. In *In Proc. of 17th Int. Conf. on Pattern Recognition*, 2004.

[12] D.Beymer and K. Konolige. Tracking people from a mobile platform. In *International Joint Conferences on Articial Intelligence*, 2001.

[13] Andrew T. Duchowski. *Eye Tracking Methodology: Theory and Practice*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[14] Harris C. G. e Stephens M. A combined edge and corner detector. *In 4th Alvey Vision Conference,*, pages 189–192, 1988.

[15] Shi J. e Tomasi C. Good features to track. *CVPR*, pages 593–600, 1994.

[16] J. Fritsch, M. Kleinehagenbrock, S. Lang, G. Fink, and G. Sagerer. Audiovisual person tracking with a mobile robot, 2004.

[17] M. Mordonini. G. Adorni, S. Cagnoni. Cellular-automata based optical flow computation for just-in-time applications. *10th International Conference on Image Analysis and Processing (ICIAP99), Venice, Italy*, pages 612–617, 1999.

[18] S. Cagnoni e M. Mordonini. G. Adorni, F. Bergenti. License-plate recognition for restricted-access area control systems. *Multimedia Video-Based Surveillance Systems: Requirements, Issues and Solutions. Kluwer*, 2000.

[19] G.Chivil, F.Mezzaro, A.Sgorbissa, and R.Zaccaria. Follow the leader behaviour through optical fow minimization. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2004.

[20] O. Gigliotta, M. Carretti, S. Shokur, and S. Nolfi. Towards a person-follower robot. *CNR e Università di palermo*, 2003.

[21] Rachel Gockley, Jodi Forlizzi, and Reid Simmons. Natural person-following behavior for social robots. pages 17–24, 2007.

[22] Alessandro Gregori. *Un sistema di visione stereo per il tracciamenteo di persona da parte di robot mobili.* PhD thesis, università sefli studi di parma, 2005.

[23] D. Grest and R. Koch. Realtime multi-camera person tracking for immersive environments. *Multimedia Information Processing*, 2004.

[24] G. Lawitzky. H. Endres, W. Feiten. Field test of a navigation system: Autonomous cleaning in supermarkets. *In Proc. of the 19998 IEEE International Conference en Robotics & Automation (ICRA '98), 1998.*, 1998.

[25] M. Herman e R. Chellappa. H. Liu, T. Hong. A general motion model and spatiotemporal filters for computing optical flow. *International Journal of Computer Vision*, pages 141–172, 1997.

[26] N. Hirai and H. Mizoguchi. Visual tracking of human back and shoulder for person following robot. *Advanced Intelligent Mechatronics, 2003.*, 2003.

[27] H. I. Christensen H.Sidenbladh, D. Kragik. A person following behaviour of a mobile robot. In *Proceedings of IEEE Internation Conference on Robotics and Automation*, 1999.

[28] D.P. Huttenlocher, G.A. Klanderman, and W.J. Rucklidge. Comparing images using the hausdorff distance. *PAMI*, 15(9):850–863, September 1993.

[29] L. S. Davis. I.Haritaoglu, D. Harwood. Real-time surveillance of people and their activities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):1, 2000.

[30] Ioannis Iossifidis, Carsten Bruckhoff, Christoph Theis, Claudia Grote, Christian Faubel, and Gregor Schöner. CORA: An anthropomorphic robot assistant for human environment.

[31] L. Kitchen and A. Rosenfeld. Gray level corner detection. *Pattern recognition Letters*, pages 95–102, 1982.

[32] Kwon, Yoon, Park, and Kak. person tracking with a mobile robot using two uncalibrated indipendently moving cameras. In *In Proceedings of IEEE International Conference on Robotics, Automation (ICRA)*, 2005.

[33] Megaitalia. Ofro, catalogo on line dei prodotti, 2007.

[34] W. Whittaker. Montemerlo M., S. Thrun. Conditional particle ters for simultaneous mobile robot localization and people-tracking. *1*, 1:695 701, May2002.

[35] M.Piaggio, P.Fornaro, A.Piombo, L.Sanna, and R.Zaccaria. An optical flow person following behaviour. In *In Proceedings of the IEEE ISIC/CIRNISAS Joint Conference*, 1998.

[36] P. Ferrari M.Tarokh. Robotic person following using fuzzy control and image segmentation. *Journal of Robotic Systems*, 20:1, 2003.

[37] Buxton D. W. Murray B. F. Experiments in the machine interpretation of visual motion. *Experiments in the Machine Interpretation of Visual Motion, MIT Press.*, 1990.

[38] H.Hu N.Bellotto. Multisensor integration for human-robot interaction. *The IEEE Journal of Intelligent Cybernetic Systems*, 1(1):1, July 2005.

[39] P. Nesi. Variational approach to optical flow estimation managing discontinuities. pages 419–439, 1993.

[40] G. Neva. *Analisi del movimento tramite sensori ottici omnidirezionali.* PhD thesis, Universit‘ degli Studi di Parma - Facolta’ di Ingegneria, 2001.

[41] Faugeras O. Three-dimensional computer vision: a geometric viewpoint. *Cambridge, MA: MIT Press*, 1993.

[42] Akihisa Ohya and Takumi Munekata. Intelligent escort robot moving together with human -interaction in accompanying behavior-. *FIRA Robot Congress*, 1:1, 2002.

[43] M. J. Jones P. Viola. Rapid object detection using a boosted cascade of simple features. *IEEE CVPR*, 2001.

[44] Luciann Panait, Azhaar Rafiq, Ahmed Mohamed, Francisco MOra, and R. Marrell. Robotic assistant for laparoscopy. *Journal of Laparoendoscopic and Advanced Surgical techniquues*, 2006.

[45] J.K. Aggarwal Q.Cai, A. Mitchie. Tracking human motion in an indoor environment. *In 2nd International Conference on Image Processing*, 1995.

[46] J. Maydt R. Lienhart. An extended set of haar-like features for rapid object detection. *IEEE ICIP*, pages 900–903, 2002.

[47] Bouthemy Ricquebourg. Real-time tracking of moving persons by exploring spatio-temporal image slices. In *1*, volume 22, pages 797–808, August 2000.

[48] Nicholas Roy, Gregory Baltus, Dieter Fox, Francine Gemperle, Jennifer Goetz, Tad Hirsch, Dimitris Margaritis, Michael Montemerlo, Joelle Pineau, Jamieson Schulte, and Sebastian Thrun. Towards personal service robots for the elderly. In *Workshop on Interactive Robots and Entertainment (WIRE 2000*, 2000.

[49] W.J. Rucklidge, J.J. Noh, and D.P. Huttenlocher. Tracking non-rigid objects in complex scenes. In *Cornell*, 1992.

[50] Schulz, Burgard, Fox, and Cremers. People tracking with a mobile robot using semple based joint probabilistic data association. *International Journal of Robotics Research IJRR*, 2003.

[51] Reid Simmons, R. Goodwin, K. Haigh, S. Koenig, and J. Sullivan. A layered architecture for office delivery robots. In *First International Conference on Autonomous Agents*, pages 235 – 242, February 1997.

[52] Brady J. M. Smith S. M. Susan - a new approach to low level image processing. *Int. Journal of Computer Vision*, 23(1):45–78, 1997.

[53] Lucas B. Kanade T. An iterative image registration technique with an application to stereo vision. *In DARPA Image Understanding Workshop*, pages 121–130, 1981.

[54] Sebastian Thrun, M. Beetz, Maren Bennewitz, Wolfram Burgard, A.B. Creemers, Frank Dellaert, Dieter Fox, Dirk Hahnel, Chuck Rosenberg, Nicholas Roy, Jamieson Schulte, and Dirk Schulz. Probabilistic algorithms and the interactive museum tour-guide robot minerva. *International Journal of Robotics Research*, 19(11):972–999, November 2000.

[55] Teruko Yata, Akihisa Ohya, and Shin'ichi Yuta. Fusion of omni-directional sonar and omni-directional vision for environment recognition of mobile robots. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation*, pages 3926–3931, April 2000.

[56] Takashi Yoshimi, Manabu Nishiyama, Takafumi Sonoura, Hideichi Nakamoto, Seiji Tokura, Hirokazu Sato, Fumio Ozaki, NobutoMatsuhira Corporate, and Toshiba Corporation. Development of a person fol-

lowing robot with vision based target detection. 2006. iroshi Mizoguchi Dept. of Mechanical Engineering Tokyo University of Science.

Appendice A

Listato del codice sorgente

Nel seguito vengono riportato il codice sorgente dei vari file che compongono il sistema.

A.1 Codice firmware per il movimento della telecamera

A.1.1 ControlloServi.c

Listing A.1: pic/ControlloServi.c

```
/* ControlloServi.c */
//Codice per PIC16F877A per controllo servomotori e lettura sensori

// #INCLUDE e #DEFINE
#define DEBUG_PRG       1       //<==!!!== FLAG PER IL DEBUG
#define MHZ             *1000L

#define XTAL_FREQ       20MHZ
#define BASETEMPI       2       //Postscaler Software per Timer0 (moltiplica per 10ms)
#define MAX_LEN         11      //Dimensione massima comando letto da seriale
#define TEMPO_GRADO     6       //Microsecondi da attendere per muoversi di un grado

#define LED             RE1
#define BUT             RE2

//Delay iniziali per pan e tilt
#define DELAY_PAN       138
#define DELAY_TILT      136

//Delay per invio valore sensori
#define DELAY_INVIO 1

//Angoli min e max per pan e tilt (gradi)
#define MAX_PAN         180
#define MIN_PAN         0
#define MAX_TILT        130
#define MIN_TILT        0

//Fattori di conversione da angoli a us di attesa
#define DEG2TIME_PAN    2.535
```

```
#define DEG2TIME_TILT   2.5556

#include <htc.h>
#include <stdio.h>
#include <string.h>

#include "general.h"            //Funzioni generali
#include "serial.h"             //Gestione RS232

// MACRO
#define DelayUs {asm("nop"); asm("nop"); asm("nop"); asm("nop");}       //Ritarda di un microsec. circa

// VARIABILI GLOBALI
char baseTempi;         //Postscaler per Timer0
bit angoloRicevuto;     //Flag per la lettura dell'angolo di inclinazione
bit comandoRicevuto;    //Flag per la lettura del comando da seriale
bit aggiornaServi;      //Flag per l'aggiornamento dei servi

int tempoPan;           //Angolo posizione del servomotore pan
int tempoTilt;          //Angolo posizione del servomotore tilt
int angoloTilty;        //Angolo inclinazione robot
char angoloh;           //Byte letti da I2C
char angolol;

unsigned char comando[MAX_LEN];         //Stringa per il comando letto da seriale

short int lunghezza;    //lunghezza comando letto da seriale nell'interrupt
short int numSensore;   //Sensore di cui effettuare la misurazione

int inviaSensori;       //Contatore per inviare sensori

// ROUTINE DI SERVIZIO INTERRUPT
//Invia un carattere dall'interrupt
#pragma interrupt_level 1
void i_putch(unsigned char c)
{
        TXREG=c;                //Mette nel buffer di trasmissione il carattere da spedire
        while(!TRMT)
                continue;       //Non fa niente finchï¿½ non ï¿½ spedito
}

//Gestione dei flag di interrupt
#pragma interrupt_level 1
void interrupt isr(void)
{
        char c=0;       //variabile locale per la lettura di un carattere
        int i=0;        //variabile per il ritardo dei servi

        if(RCIF)        //Interrupt se ha ricevuto qualcosa dalla seriale
        {
                if(OERR || FERR)        //In caso di errori
                {
                        CREN=0;         //Spegne continuous receive
                        i_putch('E');   //Echo per debug
                        i_putch('\r');
                        i_putch('\n');
                        c=RCREG;        //toglie un carattere da RCREG
                        c=RCREG;        //toglie un secondo carattere da RCREG
                        lunghezza=0;    //Azzera la lunghezza della stringa letta
                        CREN=1;         //Riabilita la receive
                }
                else
                {
                        c=RCREG;                //legge il carattere corretto
                        comando[lunghezza++]=c; //salva il carattere nella stringa
                        if(c==200)              //se il carattere ï¿½ il terminatore
                        {
```

```
98                                  comandoRicevuto=1;      //ï¿½ stato ricevuto un comando da
                                        processare
99                          }
100                 }
101                 RCIF=0;         //riabbassa il flag dell'interrupt seriale
102         }
103
104         if(T0IF)        //Interrupt ogni 10ms per la PWM dei servomotori
105         {
106                 if (numSensore<4) numSensore++;
107
108                 if ((--baseTempi)==0)
109                 {
110                                 aggiornaServi=1;        //Setta flag per l'aggiornamento dei servi
111                                 baseTempi=BASETEMPI;    //Ogni BASETEMPI overflow di TMR0 (20ms
                                        con BASETEMPI a 2)
112                 }
113
114                 TMR0=60;        //Ogni 10ms
115                 T0IF=0;         //Abbassa il flag dell'interrupt TMR0
116         }
117 }
118
119 // PROGRAMMA PRINCIPALE
120 void main(void)
121 {
122         //VARIABILI LOCALI
123         int angoloPan;  //Valori letti da seriale per i motori
124         int angoloTilt;
125         int i;
126
127         //Valori dei sensori di distanza
128         char distanzaIRFront, distanzaIRLeft, distanzaIRRight;  //3 Infrarossi (80cm max)
129         char distanzaUS;                                        //1 Ultrasuoni (6mt max)
130
131         CONFIG_FUSES(HS & WDTDIS);              //Configurazione bit del PIC: High speed ON,
                WatchDog OFF
132
133         //CONFIGURAZIONE PORTE PIC
134         //Porta A: ingressi analogici (RA0, RA1, RA2, RA3, RA5) e RA4 uscita digitale
135         TRISA=0b00101111;       //Porta A tutti ingressi da bit 0 a 5 (analogici tranne il 4), gli
                altri non sono usati
136         //RA0, RA1, RA5 sono gli ingressi dei sensori (vedi define all'inizio del file)
137
138         //Porta B: servomotori (RB1, RB2 + RB4, RB5), interrupt esterno (RB0), ISCP (RB3,RB6,RB7)
139         TRISB=0b11000001;       //RB3 OUT, RB6 IN, RB7 IN)
140         RB2=RB1=0;              //Servomotori spenti
141         RB4=RB5=0;
142
143         //Porta D: non utilizzata: 8 ingressi gestibili liberamente (con PSPMODE della porta E
                disattivo)
144
145         //CONFIGURAZIONE SERIALE
146         usartInit(115200,20000000,2);   //Inizializza seriale, interrupt ricezione ON, interrupt
                invio OFF
147
148         //PROGRAMMAZIONE TIMERO //Serve alla PWM dei servomotori
149         T0CS=0;                 //Sorgente del clock timer0 interna (Fosc/4)
150         PSA=0;                  //Prescaler attivo
151         PS0=1; PS1=1; PS2=1;    //Prescaler a 1:256
152         TMR0=0;
153         baseTempi=1;            //per farlo scattare subito
154
155         //Per far partire l'ADC si selezione il canale su ADCON0, si accende ADON, si attiva ADGO
                e si aspetta
156         //che ADGO torni basso, una volta fatto si ha il valore su ADRESH:ADRESL in 10 bit
157
158         //Porta E: Bottone e Led + RE0 ingresso analogico eventuale
159         TRISE=0b00000101;       //PSPMode (bit 4) disattivo, IN bottone su RE2, OUT led RE1, IN
```

```
            sensore RE0;

        //CONFIGURAZIONE ADC
        ADON=0;
        ADGO=0;

        ADCON1=0b01001011;      //Configurazione Digitale dei piedini 1 e 2 della porta E (e analogici sulla porta A e RE0)
                                //Formato ADRESH:ADRESL = XXXXXXXX:XXuuuuuu con Fosc/64, V+ e V- da piedini 5 e 4
        LED=1;                  //LED acceso

        TMR0IE=1;               //Abilita interrupt sul TIMER0
        SSPIE=0;                //Disabilita interrupt per I2C
        ADIE=0;                 //Disabilit interrupt per ADC
        RCIE=1;                 //Abilita interrupt seriale

        PEIE=1;                 //Abilita interrupt periferici (Seriale)
        GIE=1;                  //Abilita interrupt globali

        //Inizializzazione variabili globali
        comandoRicevuto=0;
        angoloRicevuto=0;
        tempoPan=(int)(float)(90*DEG2TIME_PAN);         //Valore iniziale a centro scala
        tempoTilt=(int)(float)(90*DEG2TIME_TILT);       //Valore iniziale a centro scala

        comando[0]='\0';comando[1]='\0';
        comando[2]='\0';comando[3]='\0';
        comando[4]='\0';comando[5]='\0';
        comando[6]='\0';comando[7]='\0';
        comando[8]='\0';comando[9]='\0';

        angoloPan=angoloTilt=0;
        angoloTilty=angolol=angoloh=0;
        distanzaIRFront=distanzaIRLeft=distanzaIRRight=distanzaUS=0;

        lunghezza=0; numSensore=0; inviaSensori=0;

        PRINT_ASSERT(DEBUG_PRG,"\r\nConfigurazione iniziale terminata\r\n");

        while(1) //loop del programma principale
        {
                //AGGIORNAMENTO POSIZIONE SERVI
                if (aggiornaServi)      //E' scattato l'interrupt poco tempo fa'
                {
                        TMR0IE=0;
                        RB1=1;
                        for(i=tempoPan+DELAY_PAN; i>0; i--)  //Attende un delay iniziale e poi ritarda di tempoPan
                        {
                                DelayUs;        //Dovrebbe ritardare un microsecondo
                        }
                        RB1=0;
                        TMR0IE=1;

                        TMR0IE=0;
                        RB2=1;
                        for(i=tempoTilt+DELAY_TILT; i>0; i--) //Attende un delay iniziale e poi ritarda di TempoTilt
                        {
                                DelayUs;        //Dovrebbe ritardare un microsecondo
                        }
                        RB2=0;
                        TMR0IE=1;
                        aggiornaServi = 0;
                }

                //CONTROLLO SENSORI E ACQUISIZIONE ADC
                if (numSensore==0) //UltraSuoni
```

```
224         {
225                         ADCON0=0b10000001;      //Fosc/64, (canale 000—AN0—RA0), Stop
226
227                         ADGO=1;                 //Avvia acquisizione
228                         while (ADGO)            //Attendi che sia terminata
229                         {
230                                 continue;
231                         }
232                         distanzaUS = ADRESH;    //Legge gli 8 bit piï¿½ significativi
233             }
234         else if (numSensore==1)         //Infrarossi Front
235         {
236                         ADCON0=0b10001001;      //Fosc/64, (canale 001—AN1—RA1), Stop.
237                         ADGO=1;                 //Avvia acquisizione
238                         while (ADGO)            //Attendi che sia terminata
239                         {
240                                 continue;
241                         }
242                         distanzaIRFront = ADRESH;       //Legge gli 8 bit piï¿½ signficativi
243             }
244         else if (numSensore==2)         //Infrarossi Left
245         {
246                         ADCON0=0b10100001;      //Fosc/64, (canale 100—AN4—RA5), Stop.
247                         ADGO=1;                 //Avvia acquisizione
248                         while (ADGO)            //Attendi che sia terminata
249                         {
250                                 continue;
251                         }
252                         distanzaIRLeft = ADRESH;        //Legge gli 8 bit piï¿½ signficativi
253             }
254         else if (numSensore==3)         //Infrarossi Right
255                 {
256                         ADCON0=0b10101001;      //Fosc/64, (canale 101—AN5—RE0), Stop.
257                         ADGO=1;                 //Avvia acquisizione
258                         while (ADGO)            //Attendi che sia terminata
259                         {
260                                 continue;
261                         }
262                         distanzaIRRight = ADRESH;       //Legge gli 8 bit piï¿½ signficativi
263                 }
264                 else if (numSensore==4)
265                 {
266                         if(inviaSensori==DELAY_INVIO)
267                         {
268                                 printf("R%c%c%c%c.\r\n",distanzaUS,distanzaIRFront,distanzaIRLeft,
                                        distanzaIRRight);
269                                 inviaSensori=0;
270                         }
271                         else inviaSensori++;
272
273                         numSensore=0;
274                 }
275
276                 //GESTIONE INPUT
277                 if(BUT==0)      //Bottone premuto
278                 {
279                         //Stampa un po' di roba per debug
280                         //printf("\r\nPAN:%d\tTILT:%d",tempoPan+DELAY_PAN,tempoTilt+DELAY_TILT);
281
282                         printf("\r\nIRL:%d\tIRF:%d\tIRL:%d\tUS:%d",distanzaIRLeft,distanzaIRFront,
                                distanzaIRRight,distanzaUS);
283                         printf("\r\nPEIE:%d,lunghezza:%d,comando:%s",PEIE,lunghezza,comando);
284                         printf("\r\nSSPIE:%d,SSPIF:%d",SSPIE,SSPIF);
285                 }
286
287                 if(comandoRicevuto) //Se ha appena letto qualcosa da seriale
288                 {
289                         if(comando[0]=='S' && comando[3]==200)  //Messaggio comando servomotori in
```

```
                                angoli (espressi con due caratteri, "Spt.")
290                         {
291                                 //Leggi valori angoli inviati
292                                 angoloPan=comando[1];
293                                 angoloTilt=comando[2];
294
295                                 //Controlla valori fuori scala pan e tilt
296                                 if(angoloPan>MAX_PAN) angoloPan=MAX_PAN;
297                                 else if(angoloPan<MIN_PAN) angoloPan=MIN_PAN;
298                                 if(angoloTilt>MAX_TILT) angoloTilt=MAX_TILT;
299                                 else if(angoloTilt<MIN_TILT) angoloTilt=MIN_TILT;
300
301                                 //Configura comandi per i servomotori
302                                 tempoPan=(int)(float)(angoloPan*DEG2TIME_PAN);
303                                 tempoTilt=(int)(float)(angoloTilt*DEG2TIME_TILT);
304
305                         }//end comando angoli
306
307                         comandoRicevuto=0;      //Riabbassa flag comando ricevuto
308                         lunghezza=0;            //Azzera contenuto del comando
309                 }//end comandoRicevuto
310         }//end while(1)
311 }//end main
```

A.1.2 general.h

Listing A.2: pic/general.h

```
1  /* General Library for PIC16F87xA(For Now) */
2  /* by Matteo Merlin - last update 09/01/2006 */
3
4  /* HEADER SECTION */
5
6  //#define MHZ 1000000
7  //#define KHZ 1000
8
9  /* Spegne e riaccende (o viceversa) un bit */
10 #define TOGGLEBIT(x) x=!x; x=!x;
11
12 /* Restituisce il valore assoluto di un numero */
13 #define ABS(x) (x)<0 ? -(x) : (x)
14
15 /* Rinomina la macro di configurazione dei fuses */
16 #define CONFIG_FUSES(x) __CONFIG(x)
17
18 /* Costruisce un blocco ASSERT: se y è vero allora x */
19 #define ASSERT(y,x)     if(y) {x}
20
21 /* Blocco ASSERT con printf: se y è vero allora stampa la stringa x */
22 #define PRINT_ASSERT(y,x) if(y) {printf(x);}
23
24 unsigned int crc(unsigned char data);   //Calcola il crc (int) di un byte in ingresso
25
26 /* IMPLEMENTAZIONE */
27 //Calcola il crc (int) di un byte in ingresso
28 unsigned int crc(unsigned char data)
29 {
30         // Utilizza l'algoritmo a 16bit CCITT (X^16 + X^12 + X^5 + 1)
31         // (thanks to Shane Tolmie: www.microchipc.com)
32         static unsigned int crc;
33
34         crc = (unsigned char)(crc >> 8) | (crc << 8);
35         crc ^= data;
36         crc ^= (unsigned char)(crc & 0xff) >> 4;
```

```
37          crc ^= (crc << 8) << 4;
38          crc ^= ((crc & 0xff) << 4) << 1;
39
40          return crc;
41  }
```

A.1.3 serial.h

Listing A.3: pic/serial.h

```
1   /* RS232 Library for PIC16F87xA (For Now) */
2   /* by Matteo Merlin - last update 26/03/2007 */
3
4   /* Header File */
5
6   /* Se non definita altrove, definisce il timeout di comunicazione */
7   #ifndef COMM_DELAY
8           #define COMM_DELAY 10000 //in cicli di attesa
9   #endif
10
11  #ifndef ABS || #ifndef TOGGLEBIT
12          #include <general.h>
13  #endif
14
15
16  /* Imposta i pin USART per i PIC16F87xA */
17  #if defined(_16F873A) || defined(_16F876A) || defined(_16F874A) || defined(_16F877A)
18          #define RX_PIN TRISC7
19          #define TX_PIN TRISC6
20  #else
21          #error Supportati solamente PIC16F87xA
22  #endif
23
24
25  int usartInit(long baud, long clock, char interr);                    // Abilita la USART
        asincrona sul PIC
26  void putch(unsigned char c);                                // Spedisce un carattere
27  unsigned char getch();                                      // Riceve un carattere (timeout: non
        bloccante)
28  unsigned char getche();                                     // Legge un carattere e lo rispedisce
        indietro
29  //unsigned char kbhit();                                            //???
30  void putstr(const char *stringa);                           // Spedisce una stringa di caratteri
31  int getstr(char* buffer, int maxlen, char endchar);         // Riceve una stringa di caratteri (
        timeout: non bloccante)
32  bit commErr(void);                                          // Gestisce e segnala la presenza di
        errori nella trasmissione
33
34
35  /* IMPLEMENTAZIONE */
36  #define DEBUG 0
37
38  /* Abilita la USART asincrona sul PIC */
39  int usartInit(long baud, long clock, char interr)
40  {
41          float error=0;          //errore percentuale di trasmissione
42          float errortmp=0;
43
44          long temp=0;
45
46          RX_PIN = 1; TX_PIN = 1; //Abilita i pin per la USART
47          BRGH = (baud>=9600);    //Alta velocitï¿½ per baud>=9600
48
```

```
49          if (BRGH)
50          {
51                  SPBRG = ((clock/(16*baud)))-1;  //valore per trasmissione
52
53                  temp = (((SPBRG+1)*16)*baud);   //denominatore
54                  error = (100*(float)(clock-temp))/(float)temp; //Errore di trasmissione (arr
                        difetto)
55
56                  temp = (((SPBRG+2)*16)*baud);   //Errore con arr eccesso
57                  errortmp = (100*(float)(clock-temp))/(float)temp;
58          }
59          else
60          {
61                  SPBRG = (clock/(64*baud))-1;    //valore per trasmissione
62
63                  temp = (((SPBRG+1)*64)*baud);   //denominatore
64                  error = (100*(float)(clock-temp))/(float)temp; //Errore di trasmissione (arr
                        difetto)
65
66                  temp = (((SPBRG+2)*64)*baud);                   //Errore con arr eccesso
67                  errortmp = (100*(float)(clock-temp))/(float)temp;
68          }
69
70          if((ABS(errortmp)) < (ABS(error))) //Se con eccesso ï¿½ meglio
71          {
72                  SPBRG++;        //arrotonda in eccesso
73                  error=errortmp;                 //usa l'errore con eccesso
74          }
75
76          if (error>4) return (int)error; // Errore di trasmissione troppo elevato (Cambiare
                BaudRate o Clock)
77
78          SYNC = 0;               //modo asincrono
79          SPEN = 1;               //Abilita RX_PIN e TX_PIN
80          TX9 = 0; RX9 = 0;       //Trasmissione e ricezione a 8bit
81
82          CREN=1; TXEN=1;         //Abilita ricezione e trasmissione
83
84          if(interr==0)   //Configurazione Interrupt
85          {       RCIE = 0; TXIE = 0;     }       //Disabilita Ricezione e Trasmissione
86          else if(interr==1)
87          {       RCIE = 1; TXIE = 1;     }       //Abilita Ricezione e Trasmissione
88          else if(interr==2)
89          {       RCIE = 1; TXIE = 0;     }       //Abilita Ricezione e disabilita Trasmissione
90          else if(interr==3)
91          {       RCIE = 0; TXIE = 1;     }       //Disabilita Ricezione e abilita Trasmissione
92
93          return (int)error;
94  }
95
96  /* Spedisce un carattere */
97  void putch(unsigned char c)
98  {
99          TXREG=c;                        //Mette nel buffer di trasmissione il carattere da spedire
100         while(!TRMT)
101                 continue;               //Non fa niente finchï¿½ non ï¿½ spedito
102         //while(!(TRMT && TXIF))        //Possibile alternativa (non testata!!)
103 }
104
105 /* Riceve un carattere (no timeout: bloccante) */
106 /*unsigned char getch()                 //legge un carattere da seriale
107 {
108         while(!RCIF)                    //aspetta finchï¿½ non ï¿½ avvenuta la ricezione
109                 continue;
110
111         if(commErr())                   //se ci sono errori
112                 togglebit(CREN);        //resetta errori
113
```

```
114         return RCREG;                   //restituisci il carattere letto
115 }*/
116
117 /* Riceve un carattere (timeout: non bloccante) */
118 unsigned char getch()
119 {
120         long int z;
121         for(z=0; z<=COMM_DELAY; z++)  //Ciclo di lettura
122         {
123                 if(RCIF)        //aspetta finchï¿½ non ï¿½ avvenuta la ricezione o esaurito il
                        timeout
124                 {
125                         if(commErr())                   //se ci sono errori
126                                 TOGGLEBIT(CREN);        //resetta errori
127
128                         return RCREG;           //restituisciil carattere letto
129                 }
130         }
131         return 0;                       //restituisci 0
132 }
133
134
135 /* Legge un carattere e lo rispedisce indietro */
136 unsigned char getche()
137 {
138         unsigned char c;
139         putch(c = getch());     //spedisce il carattere letto
140         return c;
141 }
142
143 /* Spedisce una stringa di caratteri */
144 void putstr(const char *stringa)
145 {
146         do
147         {
148                 putch(*stringa); //Spedisci il prossimo carattere di *stringa
149                 stringa++;
150         }
151         while((*stringa)!=0); //Finchï¿½ la stringa non ï¿½ stata spedita tutta
152 }
153
154 /* Riceve una stringa di caratteri (fino al terminatore) (con timeout: non bloccante) */
155 int getstr(char* buffer, int maxlen, char endchar)
156 {
157         int l=0;                        //indica la lunghezza della stringa comando
158         char c=0;                       //indica il carattere appena letto
159
160         do
161         {
162                 if(c=getch())           //Se la lettura di c da esito positivo
163                         buffer[l++]=c;  //Salva il carattere letto
164         }
165         while((l<maxlen) && (c!=endchar));      //resta nel ciclo finchï¿½ legge caratteri, finche
                non ha
166                                                                                 //esaurito lo
                                                                                        spazio o non
                                                                                        ha incontrato
                                                                                        il carattere
                                                                                        di END
167         return l;                       //ritorna la lunghezza della stringa letta
168 }
169
170 /* Gestisce e segnala la presenza di errori nella trasmissione */
171 bit commErr(void)
172 {
173         if(!OERR && !FERR)                      //Se non ci sono errori
174         {
175                 return 0;
```

```
176         }
177         else
178         {
179                 #if DEBUG
180                                 if (OERR) putstr("OVERRUN ERROR\n\r");  //Errore di OVERRUN
181                                 if (FERR) putstr("FRAME ERROR\n\r");    //Errore di FRAME
182                 #endif
183
184                 return 1;
185         }
186 }
```

A.2 Codice dell'interfaccia per l'utilizzo del robot

A.2.1 main.c

Listing A.4: servocamera/src/main.c

```
1  /* main.c */
2  //Questo file \'e stato generato da Glade, ed \'e il main che costruisce e avvia la semplice
       interfaccia con i vari componenti.
3
4  /*
5   * Initial main.c file generated by Glade. Edit as required.
6   * Glade will not overwrite this file.
7   */
8
9  #ifdef HAVE_CONFIG_H
10 #  include <config.h>
11 #endif
12
13 //Librerie per Gtk+
14 #include <gtk/gtk.h>
15
16 //Librerie dell'interfaccia
17 #include "interface.h"
18 #include "support.h"
19
20 //Librerie di controllo telecamera
21 #include "servocamera.h"    //Core dell'applicazione di controllo servocamera
22
23 int main (int argc, char *argv[])
24 {
25   GtkWidget *MainWindow;
26
27 #ifdef ENABLE_NLS
28   bindtextdomain (GETTEXT_PACKAGE, PACKAGE_LOCALE_DIR);
29   bind_textdomain_codeset (GETTEXT_PACKAGE, "UTF-8");
30   textdomain (GETTEXT_PACKAGE);
31 #endif
32
33   gtk_set_locale ();
34   gtk_init (&argc, &argv);
35
36   add_pixmap_directory (PACKAGE_DATA_DIR "/" PACKAGE "/pixmaps");
37
38   /*
39    * The following code was added by Glade to create one of each component
40    * (except popup menus), just so that you see something after building
41    * the project. Delete any components that you don't want shown initially.
42    */
43   MainWindow = create_MainWindow();
```

```
44    gtk_widget_show (MainWindow);
45
46    int error;
47    if ((error=initSerialComm())<0)       //Setta la seriale e segnala eventuale errore
48    {
49        if (error==-1) fprintf(stderr,"Could not initialize serial for servo motors...\n");
50        else if (error==-2) fprintf(stderr,"Could not initialize serial for robot motors...\n");
51        else if (error==-3) fprintf(stderr,"Could not initialize serial for both servo and motors
              ...\n");
52    }
53
54    move(fd_serial_servo,90,90);  //Posiziona servomotori in posizione centrale
55
56    // Apre ohci e assigna l'handle (Il numero \'e la porta)
57    // da usare per le feature della firewire via libdc1394
58    fwHandle = dc1394_create_handle(0);
59    if (fwHandle==NULL)
60          fprintf( stderr, "Unable to aquire a raw1394 handle\n\n");
61    else
62    {
63          // Get the camera nodes and describe them as we find them
64          numNodes = raw1394_get_nodecount(fwHandle);
65          fwNodes = dc1394_get_camera_nodes(fwHandle,&numCameras,1);
66          fflush(stdout);
67          if (numCameras<1)
68          {
69              fprintf( stderr, "no cameras found on hub 1\n");
70              dc1394_destroy_handle(fwHandle);
71          }
72    }
73
74    if (setCaptureDevice(argc,argv)==-1)       //Setta la periferica di acquisizione e segnala l'
          eventuale errore
75    {
76        fprintf(stderr,"Could not initialize capturing...\n");
77        return -1;
78    }
79
80    blobDetection(INIT);      //In "blobdetect.h":    Inizializza una volta per tutte le variabili
          di blobDetection
81    faceDetection(INIT);      //In "facedetect.h":    Inizializza una volta per tutte le variabili
          di FaceDetection
82    templateMatching(INIT);   //In "template.h":      Inizializza una volta per tutte le variabili
          di templateMatching
83    flowFeatures(INIT);       //In "flowfeat.h":      Inizializza una volta per tutte le variabili
          di FlowDetection
84
85    getAllWidgetMain(MainWindow);     //In "servocamera.h":   Ottiene i puntatori a tutti i Widget
          dell'interfaccia
86    initServoCamera();                //In "servocamera.h":   Operazioni preliminari per servocamera
87
88    //Qualche automatismo
89  //    gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(captureButton), TRUE);  //Abilita l'avvio
        automatico dell'acquisizione video
90  //    gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(flowButton), TRUE);     //Abilita l'avvio
        automatico di corner e template
91  //    gtk_button_click(GTK_BUTTON(faceButton), TRUE);                        //Abilita il
        rilevamento automatico di facce
92
93    //!gtk_main() si limita a mostrare e a gestire i vari widget dell'interfaccia
94    //!E' la pressione del tasto "acquire" (descritta nel file callbacks.c) che avvia l'algoritmo
          centrale.
95    //!Una volta avviato il main, l'esecuzione di questo codice si ferma qui, per ripartire solo
          alla alla chiusura dell'applicazione.
96
97    gtk_main ();  //Avvia interfaccia e rimani fermo finchÃ© non viene chiusa la finestra principale
98
99    //Da qui si riparte non appena la finestra principale \'e stata chiusa
100
```

```
101   if(fd_serial_motors>0)    //Se il descrittore per il robot \'e aperto
102   {
103     //Ferma il robot, si mandano tre comandi identici per sicurezza (qualora si perdessero i primi
            due)
104     send(fd_serial_motors,"b2\r");
105     send(fd_serial_motors,"b2\r");
106     send(fd_serial_motors,"b2\r");
107   }
108
109   if (capture)
110     cvReleaseCapture(&capture);
111
112   if (fwHandle)
113   {
114     dc1394_release_camera(fwHandle,&camera);
115     dc1394_destroy_handle(fwHandle);
116   }
117   printf("Exiting...\n");
118
119   return 0;
120 }
```

A.2.2 servocamera.h

Listing A.5: servocamera/src/servocamera.h

```
1  /* servocamera.h */
2  // File principale per l'interfacciamento dell'interfaccia con i vari algoritmi
3
4  //Librerie di OpenCV
5  #include <cv.h>
6  #include <highgui.h>
7
8  //Librerie generiche
9  #include <time.h>                    //Funzioni per il calcolo dei tick
10 #include <libdc1394/dc1394_control.h> //Per le libdc1394 (firewire)
11
12 //Define della dimensione della finestra per mostrare l'immagine
13 #define         NORMAL_SIZE     1
14 #define         HALF_SIZE       0.5
15
16 //Define dei colori pi\'u usati in BGR (per opencv RGB = BGR)
17 #define         RED             CV_RGB(0,0,255)
18 #define         BLUE            CV_RGB(255,0,0)
19 #define         GREEN           CV_RGB(0,255,0)
20 #define         WHITE           CV_RGB(255,255,255)
21 #define         BLACK           CV_RGB(0,0,0)
22 #define         GRAY50          CV_RGB(127,127,127)
23 #define         GRAY25          CV_RGB(192,192,192)
24
25 //Define per l'apertura della seriale
26 #define         SERVO           1
27 #define         MOTORS          2
28
29 //Numero di cicli (frame) tra un comando al robot e l'altro
30 #define         ROBOT_MAX_DELAY     1
31 //Numero di cicli (frame) tra un interrogazione ai sensori e l'altra
32 #define         SENSOR_MAX_DELAY    1
33
34 //Altri file dell'applicazione
35 #include "blobdetect.h"              //Contiene funzioni e variabili per il blobTracking
36 #include "facedetect.h"              //Contiene funzioni e variabili per il faceDetection
37 #include "flowfeat.h"                //Contiene funzioni per il flusso delle features
38 #include "servodriver.h"             //Contiene le funzioni di movimento dei motori
```

```
39 #include "trackcontrol.h"             //Contiene le funzioni per i modelli di controllo
40 #include "template.h"                //Contiene le funzioni per la rilevazione di template
41
42 /*********************************/
43 /* Variabili condivise tra i file */
44 /*********************************/
45     //Variabili dagli altri file negli #include
46     //In blobdetect.h
47         extern IplImage *blobFrame;     //Frame contenente il blob tracciato
48         extern CvBox2D track_box;       //Box di tracking
49         extern int startingFrames;
50
51     //In facedetect.h
52         extern IplImage *faceFrame;     //Frame contenente la/le faccia/e trovate
53
54     //In flowfeat.h
55         extern IplImage *featFrame;     //Frame contenente le features trovate
56         extern int count;               //Contatore delle feature
57
58     //In callbacks.c
59         extern CvRect selection;         //Selezione dell'immagine principale
60         extern int selecting;            //Indica se si sta effettuando una selezione (callbacks.c
               )
61         extern int vmin, vmax, smin;     //Estremi filtraggio backproject
62
63 /*****************************/
64 /* Variabili di questo file */
65 /*****************************/
66     int fd_serial_servo = 0;            //File descriptor per la seriale
67     int fd_serial_motors = 0;           //File descriptor per i motori del robot
68
69     CvCapture *capture = 0;             //Stream di cattura video (Da Camera o da File)
70     IplImage *frame = 0;                //Variabile contenente il frame acquisito
71     IplImage *mainFrame =0;             //copia di frame usata per la vista principale
72     IplImage *halfFrame = 0;            //Copia del frame a grandezza dimezzata
73
74     double mainloop_newtime=0;          //Valori tick per cronometrare il mainloop_newtime
75     double mainloop_oldtime=0;          //Valori tick per cronometrare il mainloop_oldtime
76
77     double framegrabber_newtime=0;      //Valori tick per misurare l'fps
78     double framegrabber_oldtime=0;      //Valori tick per misurare l'fps
79
80     double cpuFreq = 0;                 //Frequenza della cpu (per misurare fps)
81     char fps[20];                       //Stringa contenente il valore dei fps
82     float fpsval=0;                     //Valore dei fps trovati
83
84     int pan=90,tilt=90;
85
86     int HeadSonar=0, FrontIR=0, LeftIR=0, RightIR=0;    //Valori dei sensori sul robot
87
88     //Per la firewire
89     dc1394_cameracapture camera;        //Device di cattura Firewire
90     int numNodes; int numCameras;       //Nodi e camere (?)
91     raw1394handle_t fwHandle;           //Handle della camera trovata
92     nodeid_t * fwNodes;                 //Puntatore al nodo trovat o(?)
93     dc1394_feature_set fwFeatures;      //Elenco delle caratteristiche della camera trovata
94
95     /*Widget dell'interfaccia da utilizzare*/
96     GtkWidget *mainImage;               //Immagine principale della camera
97     GtkWidget *panScroll;               //Scrollbar per il controllo pan
98     GtkWidget *tiltScroll;              //Scrollbar per il controllo tilt
99     GtkWidget *fpsLabel;                //Etichetta per gli fpsLabel
100    GtkWidget *statusbar;               //Barra di stato
101
102    GtkWidget *histImage;               //Immagine dell'istogramma
103    GtkWidget *trackButt;               //Bottone per il tracking
104    GtkWidget *backprojButt;            //Bottone per il back projection
105    GtkWidget *toggleServo;             //Bottone per il controllo dei servomotori
106    GtkWidget *vminScroll;              //Scrollbar per valore vmin
```

```
107     GtkWidget *vmaxScroll;              //Scrollbar per valore vmin
108     GtkWidget *sminScroll;              //Scrollbar per valore vmin
109
110     GtkWidget *faceImage;               //Immagine per la faccia
111     GtkWidget *faceButton;               //Bottone per le facce
112
113     GtkWidget *captureButton;           //Bottone per l'acquisizione
114
115     GtkWidget *redetectButton;          //Bottone per il riavvio del feature detection
116     GtkWidget *flowButton;              //Bottone per il flusso delle feature
117     GtkWidget *flowImage;               //IMmagine per il flusso delle features
118
119     GtkWidget *autoCalibCheck;          //Check per l'autocalibrazione scrollbars iniziale
120     GtkWidget *dynScrollCheck;          //Check per l'autocalibrazione scrollbars continua
121     GtkWidget *dynHistCheck;            //Check per l'autocalibrazione dell'istogramma
122     GtkWidget *dynHistSpin;             //Valore di aggiornamento dinamico istogramma
123     GtkWidget *dynScrollSpin;           //Valore di aggiornamento dinamico barre
124
125     GtkWidget *stepFeaturesSpin;         //Valore di features minimo prima del ricalcolo
126     GtkWidget *maxFeaturesSpin;         //Valore di features massimo da calcolare
127
128     GtkWidget *manualWhiteBalToggle;    //Abilita bilanciamento del bianco manuale/automatico
129     GtkWidget *manualBrightnessToggle;  //Abilita bilanciamento brightness manuale/automatico
130     GtkWidget *blueWBScroll;            //Valore Blue per WB manuale
131     GtkWidget *redWBScroll;             //Valore Red per WB manuale
132     GtkWidget *brightnessScroll;        //Valore brightness
133
134     GtkWidget *showImageCheck;          //Abilita visualizzazione frame principale
135
136     //Pulsanti del robot.
137     GtkWidget *robotAutoEnable;         //Movimento automatico del robot
138     GtkWidget *robotManualEnable;       //Movimento manuale del robot
139     GtkWidget *robotForwardRadio;       //Muovi avanti
140     GtkWidget *robotBackwardRadio;      //Muovi indietro
141     GtkWidget *robotLeftRadio;          //Ruota antiorario
142     GtkWidget *robotRightRadio;         //Ruota orario
143     GtkWidget *robotHaltRadio;          //Ferma
144     GtkWidget *robotDebugToggle;        //Abilita debug
145     GtkWidget *robotSpeedToggle;        //Abilita velocit\'a
146     GtkWidget *robotSpeedSpin;          //Selettore velocit\'a
147     GtkWidget *robotDebugSpin;          //Selettore livello debug
148     GtkWidget *robotManualControlBox;   //Box controllo manuale
149     GtkWidget *robotCommandEntry;       //Testo comando manuale
150     GtkWidget *robotCommandButton;      //Invia comando
151     GtkWidget *robotOutputText;         //Testo output dal robot
152
153 /***********************************/
154 /* Funzioni generali di questo file */
155 /***********************************/
156 void initServoCamera(void)
157 {
158     cpuFreq = cvGetTickFrequency()*1000000;
159     halfFrame = cvCreateImage( cvSize((int)(frame->width/2),(int)(frame->height/2)), 8, 3 );
160 }
161
162 //Inizializza tutti i widget cercandoli una volta per tutte
163 void getAllWidgetMain(GtkWidget *MainWindow)
164 {
165     mainImage = lookup_widget(GTK_WIDGET(MainWindow), "mainImage");
166     fpsLabel = lookup_widget(GTK_WIDGET(MainWindow), "fpsLabel");
167     statusbar = lookup_widget(GTK_WIDGET(MainWindow), "statusbar");
168     trackButt = lookup_widget(GTK_WIDGET(MainWindow), "trackButt");
169     histImage = lookup_widget(GTK_WIDGET(MainWindow), "histImage");
170     backprojButt = lookup_widget(GTK_WIDGET(MainWindow), "backprojButt");
171     vminScroll = lookup_widget(GTK_WIDGET(MainWindow),"vminScroll");
172     vmaxScroll = lookup_widget(GTK_WIDGET(MainWindow),"vmaxScroll");
173     sminScroll = lookup_widget(GTK_WIDGET(MainWindow),"sminScroll");
174     panScroll = lookup_widget(GTK_WIDGET(MainWindow),"panScroll");
175     tiltScroll = lookup_widget(GTK_WIDGET(MainWindow),"tiltScroll");
```

```
176     faceButton = lookup_widget(GTK_WIDGET(MainWindow),"faceButton");
177     faceImage = lookup_widget(GTK_WIDGET(MainWindow),"faceImage");
178     toggleServo = lookup_widget(GTK_WIDGET(MainWindow),"toggleServo");
179     flowButton = lookup_widget(GTK_WIDGET(MainWindow),"flowButton");
180     captureButton = lookup_widget(GTK_WIDGET(MainWindow),"captureButton");
181     redetectButton = lookup_widget(GTK_WIDGET(MainWindow),"redetectButton");
182     flowImage = lookup_widget(GTK_WIDGET(MainWindow),"flowImage");
183     autoCalibCheck = lookup_widget(GTK_WIDGET(MainWindow),"autoCalibCheck");
184     dynScrollCheck = lookup_widget(GTK_WIDGET(MainWindow),"dynScrollCheck");
185     dynHistCheck = lookup_widget(GTK_WIDGET(MainWindow),"dynHistCheck");
186     dynScrollSpin = lookup_widget(GTK_WIDGET(MainWindow),"dynScrollSpin");
187     dynHistSpin = lookup_widget(GTK_WIDGET(MainWindow),"dynHistSpin");
188     stepFeaturesSpin = lookup_widget(GTK_WIDGET(MainWindow),"stepFeaturesSpin");
189     maxFeaturesSpin = lookup_widget(GTK_WIDGET(MainWindow),"maxFeaturesSpin");
190 
191     robotAutoEnable = lookup_widget(GTK_WIDGET(MainWindow),"robotAutoEnable");
192     robotManualEnable = lookup_widget(GTK_WIDGET(MainWindow),"robotManualEnable");
193     robotForwardRadio = lookup_widget(GTK_WIDGET(MainWindow),"robotForwardRadio");
194     robotBackwardRadio = lookup_widget(GTK_WIDGET(MainWindow),"robotBackwardRadio");
195     robotLeftRadio = lookup_widget(GTK_WIDGET(MainWindow),"robotLeftRadio");
196     robotRightRadio = lookup_widget(GTK_WIDGET(MainWindow),"robotRightRadio");
197     robotDebugToggle = lookup_widget(GTK_WIDGET(MainWindow),"robotDebugToggle");
198     robotSpeedToggle = lookup_widget(GTK_WIDGET(MainWindow),"robotSpeedToggle");
199     robotSpeedSpin = lookup_widget(GTK_WIDGET(MainWindow),"robotSpeedSpin");
200     robotDebugSpin = lookup_widget(GTK_WIDGET(MainWindow),"robotDebugSpin");
201     robotHaltRadio = lookup_widget(GTK_WIDGET(MainWindow),"robotHaltRadio");
202     robotManualControlBox = lookup_widget(GTK_WIDGET(MainWindow),"robotManualControlBox");
203     robotOutputText = lookup_widget(GTK_WIDGET(MainWindow),"robotOutputText");
204 
205     manualWhiteBalToggle = lookup_widget(GTK_WIDGET(MainWindow),"manualWhiteBalToggle");
206     manualBrightnessToggle = lookup_widget(GTK_WIDGET(MainWindow),"manualBrightnessToggle");
207     blueWBScroll = lookup_widget(GTK_WIDGET(MainWindow),"blueWBScroll");
208     redWBScroll = lookup_widget(GTK_WIDGET(MainWindow),"redWBScroll");
209     brightnessScroll = lookup_widget(GTK_WIDGET(MainWindow),"brightnessScroll");
210     showImageCheck = lookup_widget(GTK_WIDGET(MainWindow),"showImageCheck");
211 }
212 
213 //Configura la periferica di input e lo stream di cattura video
214 int setCaptureDevice(int argc, char *argv[])
215 {
216     //Configura input dalla camera oppure da un file video
217     if( argc == 1 || (argc == 2 && strlen(argv[1]) == 1 && isdigit(argv[1][0])))
218         capture = cvCaptureFromCAM( argc == 2 ? argv[1][0] - '0' : 0 );
219     else if( argc == 2 )
220         capture = cvCaptureFromAVI( argv[1] );
221 
222     if( !capture ) return -1;     //Restituisce -1 se fallisce, 0 se tutto ok
223     else
224     {
225         frame = cvQueryFrame(capture);
226 
227         //Crea immagine principale
228         mainFrame = cvCreateImage( cvGetSize(frame), 8, 3 );
229 
230         return 0;
231     }
232 }
233 
234 //Crea descrittori seriale per servi e robot
235 int initSerialComm(void)
236 {
237     int error=0;
238 
239     fd_serial_servo = initSerialCam(SERVO);     //Inizializza seriale per la scheda controllo
        servi
240     if (fd_serial_servo<0)                      //Gestione errore di apertura porta seriale,
241         error = -1;
242 
243     fd_serial_motors = initSerialCam(MOTORS); //Inizializza seriale per la scheda di controllo
```

```
        motori
    if (fd_serial_motors<0)                    //Gestione erroe di apertura della porta
        error -= 2;

    return error;
}

//Conversione da IplImage a PixBuf
GdkPixbuf* iplImage_to_gdkPixbuf(IplImage *source, float scale)
{
    GdkPixbuf *destination;

    if(scale != 1)
    {
        cvResize(source, halfFrame, CV_INTER_LINEAR );

        destination = gdk_pixbuf_new_from_data((guchar*)halfFrame->imageData, GDK_COLORSPACE_RGB,
            FALSE, 8, halfFrame->width, halfFrame->height, halfFrame->widthStep, NULL, NULL);
    }
    else
    {
        destination = gdk_pixbuf_new_from_data((guchar*)source->imageData, GDK_COLORSPACE_RGB,
            FALSE, 8, source->width, source->height, source->widthStep, NULL, NULL);
    }
    return destination;
}

//Mostra una IplImage in un Widget Immagine
gint showGtkImage(IplImage *cvImage, GtkWidget *gtkImage, float scale)
{
    GdkPixbuf *pixbuff;
    pixbuff = iplImage_to_gdkPixbuf(cvImage, scale);
    gtk_image_set_from_pixbuf(GTK_IMAGE(gtkImage),pixbuff);
    return 0;
}

//Imposta i valori pan e tilt per centrare la selezione inquadrata
void servoControl(void)
{
        //Controllo del blob colore
        float x_in,x_out,y_in,y_out;

        x_in= track_box.center.x;
        y_in= track_box.center.y;

        //Controllore parabolico delle x
        control x_par_ctrl;
        costruisci_controllore(&x_par_ctrl,0,640,-6,6,320,40);
        x_out=solver2(&x_par_ctrl,x_in);

        //Controllore parabolico delle y
        control y_par_ctrl;
        costruisci_controllore(&y_par_ctrl,0,480,-6,6,240,40);
        y_out=solver2(&y_par_ctrl,y_in);

        //Legge il valore corrente delle scrollbar
        pan = gtk_range_get_value(GTK_RANGE(panScroll));
        tilt = gtk_range_get_value(GTK_RANGE(tiltScroll));

        int oldPan = pan; int oldTilt = tilt;

        //Aggiorna valori pan e tilt per centrare il tracking
        pan+=(int)x_out; tilt-=(int)y_out;

        gtk_range_set_value (GTK_RANGE(panScroll),(int)(pan));
        gtk_range_set_value (GTK_RANGE(tiltScroll),(int)(tilt));

        //Legge il valore corrente delle scrollbar
        pan = gtk_range_get_value(GTK_RANGE(panScroll));
```

```
310         tilt = gtk_range_get_value(GTK_RANGE(tiltScroll));
311
312         if ((oldPan != pan) || (oldTilt != tilt))
313             move(fd_serial_servo,pan,tilt);
314 }
315
316 //Linearizza il valore letto dai sensori infrarossi e restituisce i cm (circa)
317 int linearInfrared(float voltSensor)
318 {
319     float distanzaIR;
320
321     //Linearizza la misura su tre rette
322     if (voltSensor >= 2.4)
323         distanzaIR = 0;
324     else if (voltSensor < 2.4 && voltSensor >= 1.4)
325         distanzaIR = (3.4-voltSensor)*10.0;
326     else if (voltSensor < 1.4 && voltSensor >= 0.6)
327         distanzaIR = (2.0-voltSensor)*37.5;
328     else if (voltSensor < 0.6 && voltSensor >= 0.4)
329         distanzaIR = ( 1.0-voltSensor)*150.0;
330     else if (voltSensor < 0.4)
331         distanzaIR = 99;
332
333     return (int)distanzaIR;
334 }
335
336 //Controlla i movimenti del robot per raggiungere l'obbiettivo
337 void robotControl()
338 {
339         int motor1, motor2, newM1Value, newM2Value;         //Valori dei motori (e correzioni)
340         int distUS, distIRL, distIRF, distIRR;              //Valori dei sensori di posizione
341         static float x_in,x_out,y_in,y_out,pan_in,pan_out;  //Valori usati dai controllori (x,y,
                pan)
342
343         //Legge il valore corrente delle scrollbar
344         int pan=gtk_range_get_value(GTK_RANGE(panScroll));
345         int tilt=gtk_range_get_value(GTK_RANGE(tiltScroll));
346
347         x_in= track_box.center.x; y_in= track_box.center.y; //Posizione del blob tracciato
348         pan_in = pan;                                       //Valore corrente del pan
349
350         //Controllore parabolico per le x
351         static control x_par_ctrl;
352         costruisci_controllore(&x_par_ctrl,0,640,-10,10,320,60);
353         x_out=solver2(&x_par_ctrl,x_in);
354
355         pan_out = (pan_in-90)/4.0;    //Nuovo valore di pan (per riallineare camera-robot)
356
357         //Rotazione dell'inquadramento + riallineamento camera-robot
358         newM1Value = -x_out/4.0 - pan_out;
359         newM2Value = -x_out/4.0 - pan_out;
360
361         //Gestione dei sensori
362         if(FrontIR>30 && RightIR>30 && LeftIR>30)   //Nessun ostacolo
363         {
364            if(HeadSonar>120 && FrontIR>50) //Oggetto lontano e nessun ostacolo davanti
365            {
366                 //Avvicinati
367                 newM1Value += HeadSonar/300.0*25.0;
368                 newM2Value -= HeadSonar/300.0*25.0;
369            }
370            if(HeadSonar<85 && RightIR>50 && LeftIR>50) //Oggetto vicino (e nessun ostacolo dietro)
371            {
372                 //Allontanati
373                 newM1Value -= (65-HeadSonar)/3.0;
374                 newM2Value += (65-HeadSonar)/3.0;
375            }
376         }
377
```

```
378         motor1 = newM1Value; motor2 = newM2Value;   //Variabili di appoggio per comodit\'a di
                debug
379
380         //Invia comando ai motori del robot
381         char comando[12]="";
382         sprintf(comando,"m2%+04d%+04d\r",motor1,motor2);
383         send(fd_serial_motors,comando);
384 }
385
386 //Ottiene valori dei sensori sul robot
387 void getSensorValues(void)
388 {
389     struct timespec delay;          //per la delay (?)
390     unsigned char US=0,IRL=0,IRF=0,IRR=0;   //Valori char per i sensori
391
392     //Leggi risposta ricevuta
393     char risposta[16]="";
394     read(fd_serial_servo, risposta, 16);
395
396     //   Cerca il formato corretto e legge i sensori
397     for(i=0; i<11; i++)
398     {
399         if(risposta[i]=='R')
400         {
401             US=risposta[i+1];
402             IRF=risposta[i+2];
403             IRL=risposta[i+3];
404             IRR=risposta[i+4];
405
406             //Calcola valori in centimetri (circa) per ogni sensore
407             HeadSonar = (float)US*4.0/2*2.54;
408             FrontIR = linearInfrared((float)IRF*4.0/1024*5.0);
409             LeftIR = linearInfrared((float)IRL*4.0/1024*5.0);
410             RightIR = linearInfrared((float)IRR*4.0/1024*5.0);
411
412             fprintf(stderr,"US%d IRF%d IRL%d IRR%d\n",HeadSonar,FrontIR,LeftIR,RightIR);
413             i=12; //Esci subito dal ciclo
414         }
415     }
416 }
417
418 //Loop principale del programma
419 void mainLoop(GtkWidget *MainWindow)
420 {
421     //Inizializza interrogazione sensori e movimento robot
422     static int robotCommandDelay = 0;
423     static int sensorValuesDelay = 0;
424
425     if(frame) //Se l'acquisizione del nuovo frame \'e positiva
426     {
427         cvCopy( frame, mainFrame, 0 );  //Crea e inizializza mainFrame;
428
429         //Se il tracciamento delle features nel flusso ottico \'e attivo e non si sta selezionando
430         if (!selecting && (selection.width > 0 && selection.height > 0) && (GTK_TOGGLE_BUTTON (
                flowButton))->active)
431         {
432             if (startingFrames == FRAMES_BEFORE_FEATURES)   //Se sono passati ESATTAMENTE i frames
                    prima di calcolare del features
433             {
434                 templateMatching(FIRST_INSTANCE);           //Aggiungi il template matching
435                 flowFeatures(FIRST_INSTANCE);               //aggiungi anche le features;
436                 startingFrames++;                           //Incrementa il contatore dei frames
                        iniziali
437             }
438             else if (startingFrames > FRAMES_BEFORE_FEATURES)    //Se sono passati i primi
                    FRAMES_BEFORE_FEATURES di adattamento
439             {
440                 flowFeatures(RUN);                                  //Aggiorna il tracking delle
                        features
```

```
441                 templateMatching(RUN);                                  //Aggiorna il tracking dei
                        template
442                 if ((GTK_TOGGLE_BUTTON(showImageCheck))->active)    //Se la visualizzazione \'e
                        attiva, visualizza le features
443                     showGtkImage(featFrame, flowImage, HALF_SIZE);
444             }
445             else startingFrames++;  //Incrementa il contatore dei frames iniziali
446         }
447
448         //Se la selezione \'e completata
449         if(!selecting && (selection.width > 0 && selection.height > 0) && (GTK_TOGGLE_BUTTON (
                trackButt))->active)
450         {
451             gtk_statusbar_push (GTK_STATUSBAR(statusbar), 0, "Tracking blob..."); //Scrivi nella
                    status bar
452
453             blobDetection(RUN);                                     //Avvia il tracking su blobFrame
454             showGtkImage(histFrame, histImage, NORMAL_SIZE);    //Mostra immagine istogramma
455
456             if((GTK_TOGGLE_BUTTON(backprojButt))->active)           //Se il bottone di back
                    projection \'e attivo
457                 cvCvtColor( backproject, mainFrame, CV_GRAY2RGB );  //Copia il backprojection su
                        mainFrame
458             else                                                    //altrimenti
459                 cvCopy(blobFrame, mainFrame,0);                     //Copia blobFrame su mainFrame
460         }
461
462         if (sensorValuesDelay==SENSOR_MAX_DELAY)    //Se \'e il momento di interrogare i sensori
463         {
464             getSensorValues();      //Ottieni misurazioni
465             sensorValuesDelay=0;    //Riazzera contatore
466         }
467         else sensorValuesDelay++;   //Altrimenti Incrementa contatore
468
469         if ((track_box.size.width >0 && track_box.size.height > 0)) //Se sta tracciando qualcosa
470         {
471             if ((GTK_TOGGLE_BUTTON (toggleServo))->active)  //Se i servi sono attivi
472                 servoControl();                             //Controlla i servi
473
474             if (robotCommandDelay==ROBOT_MAX_DELAY)                 //Se \'e il momento di muovere
                    il robot
475             {
476                 if ((GTK_TOGGLE_BUTTON (robotAutoEnable))->active)  //Se il robot \'e in
                        configurazione automatica
477                    robotControl();                                  //Muovi robot
478
479                 robotCommandDelay=0;    //Riazzera contatore
480             }
481             else robotCommandDelay++;   //Altrimenti Incrementa contatore
482         }
483         else //Se NON sta tracciando
484             if ((GTK_TOGGLE_BUTTON (robotAutoEnable))->active)  //Robot in configurazione manuale
485             {
486                 send(fd_serial_motors,"m2+000+000\r");          //Ferma il robot
487                 robotCommandDelay = 0;                          //Azzera contatore
488             }
489
490         //Durante la selezione di un immagine inverte i colori
491         if(selecting && selection.width > 0 && selection.height > 0)
492         {
493             cvSetImageROI( mainFrame, selection );
494             cvXorS( mainFrame, cvScalarAll(255), mainFrame, 0 );
495             cvResetImageROI(mainFrame);
496         }
497
498         //Mostra l'immagine elaborata se visualizzazione attiva
499         if ((GTK_TOGGLE_BUTTON(showImageCheck))->active)
500             showGtkImage(mainFrame, mainImage, NORMAL_SIZE);     //Visualizza mainFrame
501
```

```
502         gtk_statusbar_push (GTK_STATUSBAR(statusbar), 0, "Acquiring...");   //Aggiorna status bar
503     }
504     else    //Se il nuovo frame non esiste
505     {
506         gtk_statusbar_push (GTK_STATUSBAR(statusbar), 0, "No more frames to acquire.");
507     }
508 }
509
510 //Ottiene un nuovo frame e esegue calcolo fps
511 void frameGrabber(GtkWidget *MainWindow)
512 {
513     IplImage *tempFrame;
514
515     if((tempFrame = cvQueryFrame(capture))) //Se l'acquisizione del nuovo frame \'e positiva
516     {
517         cvCvtColor(tempFrame, frame, CV_BGR2RGB);     //Converte immagine sorgente da BGR a RGB
518
519         //Calcolo dei frame e visualizzazione su fpsLabel;
520         framegrabber_newtime = cvGetTickCount();
521         fpsval = cpuFreq/(framegrabber_newtime-framegrabber_oldtime);
522         framegrabber_oldtime = framegrabber_newtime;
523
524         sprintf(fps,"@%3.0f fps",fpsval);
525         gtk_label_set_text(GTK_LABEL(fpsLabel), fps);
526     }
527 }
```

A.2.3 callbacks.c

Listing A.6: servocamera/src/callbacks.c

```
1  /* callbacks.c */
2  //Questo file, generato inizialmente da Glade GUI, gestisce gli eventi generati dai componenti
       dell'interfaccia
3
4  #ifdef HAVE_CONFIG_H
5  #  include <config.h>
6  #endif
7
8  //Librerie di OpenCV
9  #include <cv.h>
10 #include <highgui.h>
11
12 //Librerie di Gtk+
13 #include <gtk/gtk.h>
14 #include <gdk/gdkkeysyms.h>            //Tabella Simboli dei tasti per l'utilizzo da parte di Gtk
       +
15
16 //Librerie dell'interfaccia grafica (generate da Glade)
17 #include "callbacks.h"
18 #include "interface.h"
19 #include "support.h"
20
21 //Altre Librerie
22 #include <libdc1394/dc1394_control.h>   //Controlli dc1394 (FireWire)
23
24 //Define della dimensione della finestra
25 #define  NORMAL_SIZE    1
26 #define  HALF_SIZE      0.5
27
28 //Define dei modi di funzionamento dei vari algoritmi
29 #define     INIT               0   //Inizializzazione algoritmo
30 #define     FIRST_INSTANCE     1   //Prima passata dell'algoritmo (ove necessario distinguere)
31 #define     RUN                2   //Algoritmo a regime
```

```
32
33  /*****************************************/
34  /* Variabili e funzioni condivise tra i file */
35  /*****************************************/
36      //In servocamera.h
37      //Funzioni
38          extern void mainLoop(GtkWidget *MainWindow);
39          extern void frameGrabber(GtkWidget *MainWindow);
40          extern gint showGtkImage(IplImage *cvImage, GtkWidget *gtkImage, float scale);
41          extern void filterSVScrollbars();
42
43      //Variabili
44          extern dc1394_cameracapture camera;
45          extern raw1394handle_t fwHandle;
46          extern nodeid_t * fwNodes;
47          extern dc1394_feature_set fwFeatures;
48
49          extern int fd_serial_servo;
50          extern int fd_serial_motors;
51
52          extern GtkWidget *trackButt;
53          extern GtkWidget *flowButton;
54          extern GtkWidget *histImage;
55          extern GtkWidget *tiltScroll;
56          extern GtkWidget *panScroll;
57          extern GtkWidget *vminScroll;
58          extern GtkWidget *vmaxScroll;
59          extern GtkWidget *sminScroll;
60          extern GtkWidget *faceImage;
61          extern GtkWidget *autoCalibCheck;
62          extern GtkWidget *mainImage;
63          extern GtkWidget *dynHistSpin;
64          extern GtkWidget *dynScrollSpin;
65          extern GtkWidget *statusbar;
66          extern IplImage *mainFrame;
67          extern GtkWidget *stepFeaturesSpin;
68          extern GtkWidget *maxFeaturesSpin;
69          extern GtkWidget *redWBScroll;
70          extern GtkWidget *blueWBScroll;
71          extern GtkWidget *brightnessScroll;
72
73          //Pulsanti del robot.
74          extern GtkWidget *robotAutoEnable;
75          extern GtkWidget *robotManualEnable;
76          extern GtkWidget *robotForwardRadio;
77          extern GtkWidget *robotBackwardRadio;
78          extern GtkWidget *robotLeftRadio;
79          extern GtkWidget *robotRightRadio;
80          extern GtkWidget *robotDebugToggle;
81          extern GtkWidget *robotSpeedToggle;
82          extern GtkWidget *robotSpeedSpin;
83          extern GtkWidget *robotDebugSpin;
84          extern GtkWidget *robotHaltRadio;
85          extern GtkWidget *robotManualControlBox;
86          extern GtkWidget *robotCommandEntry;
87          extern GtkWidget *robotCommandButton;
88
89      //in blobdetect.h
90          extern IplImage *histFrame;
91          extern IplImage *hsv;
92          extern int vmin, vmax, smin;
93          extern int dynHistFactor, dynScrollFactor;
94          extern CvBox2D track_box;       //Box di tracking
95          extern CvRect trackWindow;
96
97      //in facedetect.h
98          extern void faceDetection(int status);
99          extern IplImage *faceFrame;
100         extern CvRect *facce[20];
```

```
101         extern int STEP_CORNERS, MAX_CORNERS;
102
103 /***************************/
104 /* Variabili di questo file */
105 /***************************/
106     gint id_mainLoop = 0;       //id del mainLoop (serve per terminarlo)
107     gint id_framegrabber = 0;   //id del framegrabber (serve per terminarlo)
108     gint id_trackfollow = 0;    //id del movimento dei servomotori
109     CvPoint origin;             //origine della selezione
110     gint x,y;                   //Valori usati per la posizione del puntatore del mouse
111     CvRect selection;           //Area selezionata
112     int selecting = 0;          //Flag selezione in corso
113
114     //Valori indicanti la pressione dei tasti del mouse (non implementati)
115     int rightbutton_pressed = 0;
116     int leftbutt_pressed = 0;
117     int midbutt_pressed = 0;
118
119 void    //Pressione pulsante di rilevazione volti
120 on_FaceButton_clicked                   (GtkButton       *button,
121                                         gpointer         user_data)
122 {
123     faceDetection(RUN);
124     showGtkImage(faceFrame, faceImage, HALF_SIZE);
125
126     if( selection.width > 0 && selection.height > 0 )
127     {
128         selecting = 1;
129         gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(trackButt),TRUE); //Premi il pulsante di
              tracking
130         blobDetection(FIRST_INSTANCE);  //Prima istanza di BlobDetection
131         showGtkImage(histFrame, histImage, NORMAL_SIZE);
132         selecting = 0;
133     }
134 }
135
136 void    //Pressione pulsante di acquisizione video
137 on_captureButton_toggled                (GtkToggleButton *togglebutton,
138                                         gpointer         user_data)
139 {
140     //Imposta la visualizzazione dell'immagine principale in idle;
141     if ((GTK_TOGGLE_BUTTON (togglebutton))->active)     //Se il bottone non \'e attivo (il main
          non \'e gi\'a in esecuzione)
142     {
143         gtk_statusbar_push (GTK_STATUSBAR(statusbar), 0, "Acquisition started.");
144         id_framegrabber = gtk_idle_add((GtkFunction)frameGrabber, NULL);   //Avvia il frame
              grabber in modalit\'a idle a 15 fps circa
145         id_mainLoop = gtk_idle_add((GtkFunction)mainLoop, NULL);               //Avvia il
              mainLoop in modalit\'a idle e ne conserva l'id
146     }
147     else                    //Se il main \'e gi\'a in esecuzione
148     {
149         gtk_statusbar_push (GTK_STATUSBAR(statusbar), 0, "Acquisition stopped.");
150         gtk_idle_remove(id_mainLoop);           //Spegne il mainLoop
151         gtk_idle_remove(id_framegrabber);     //Spegne il framegrabber
152     }
153 }
154
155
156 gboolean    //Click del mouse sull'area da selezionare
157 on_eventImage_button_press_event        (GtkWidget       *widget,
158                                         GdkEventButton  *event,
159                                         gpointer         user_data)
160 {
161     gtk_widget_get_pointer(widget, &x, &y);
162     fprintf(stderr,"eventImage button pressed at %d,%d\n",x,y);
163     if (!mainFrame) return FALSE;
164     else
165     {
```

```
166         leftbutt_pressed = 1;
167
168         if( mainFrame->origin )
169             y = mainFrame->height - y;
170
171         origin = cvPoint(x,y);
172         selection = cvRect(x,y,0,0);
173         selecting = 1;
174
175         return TRUE;
176     }
177 }
178
179 gboolean    //Rilascio del mouse sull'area da selezionare
180 on_eventImage_button_release_event      (GtkWidget       *widget,
181                                          GdkEventButton  *event,
182                                          gpointer         user_data)
183 {
184     gtk_widget_get_pointer(widget, &x, &y);
185     if (!mainFrame) return FALSE;   //Se non si sta acquisendo non gestire l'evento
186     else
187     {
188         fprintf(stderr,"eventImage button released at %d,%d\n",x,y);
189         if (leftbutt_pressed)    //Pressione pulsante sinistro
190         {
191             if( mainFrame->origin )
192                 y = mainFrame->height - y;
193
194             if( selection.width > 0 && selection.height > 0 )   //Se esiste una selezione
195             {
196                 gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(trackButt),TRUE); //Premi il
                        pulsante di tracking
197                 gtk_statusbar_push (GTK_STATUSBAR(statusbar), 0, "Selection Done");
198
199                 blobDetection(FIRST_INSTANCE);  //Prima istanza di BlobDetection
200
201                 showGtkImage(histFrame, histImage, NORMAL_SIZE);
202                 selecting = 0;
203             }
204             leftbutt_pressed = 0;
205         }
206     return TRUE;
207     }
208 }
209
210
211 gboolean    //Al movimento del mouse sull'immagine principale
212 on_eventImage_motion_notify_event       (GtkWidget       *widget,
213                                          GdkEventMotion  *event,
214                                          gpointer         user_data)
215 {
216     gtk_widget_get_pointer(widget, &x, &y);
217     if (!mainFrame) return FALSE;   //Se non c'\'e alcuna immagine
218     else
219     {
220         if( mainFrame->origin )
221             y = mainFrame->height - y;
222
223         if( selecting)
224         {
225             selection.x = MIN(x,origin.x);
226             selection.y = MIN(y,origin.y);
227             selection.width = selection.x + CV_IABS(x - origin.x);
228             selection.height = selection.y + CV_IABS(y - origin.y);
229
230             selection.x = MAX( selection.x, 0 );
231             selection.y = MAX( selection.y, 0 );
232             selection.width = MIN( selection.width, mainFrame->width );
233             selection.height = MIN( selection.height, mainFrame->height );
```

```
234             selection.width -= selection.x;
235             selection.height -= selection.y;
236         }
237         return TRUE;
238     }
239 }
240 
241 void    //Al rilascio del bottone di tracking
242 on_trackButt_released                   (GtkButton       *button,
243                                         gpointer         user_data)
244 {
245     selection.x=0;
246     selection.y=0;
247     selection.width=0;
248     selection.height=0;
249 }
250 
251 void    //Al movimento della barra per Vmin
252 on_vminScroll_value_changed             (GtkRange        *range,
253                                         gpointer         user_data)
254 {
255     vmin = gtk_range_get_value(GTK_RANGE(vminScroll));
256 }
257 
258 
259 void    //Al movimento della barra per Vmax
260 on_vmaxScroll_value_changed             (GtkRange        *range,
261                                         gpointer         user_data)
262 {
263     vmax = gtk_range_get_value(GTK_RANGE(vmaxScroll));
264 }
265 
266 
267 void    //Al movimento della barra per Smin
268 on_sminScroll_value_changed             (GtkRange        *range,
269                                         gpointer         user_data)
270 {
271     smin = gtk_range_get_value(GTK_RANGE(sminScroll));
272 }
273 
274 
275 gboolean    //Al movimento della barra dell'angolo Pan
276 on_panScroll_change_value               (GtkRange        *range,
277                                         GtkScrollType    scroll,
278                                         gdouble          value,
279                                         gpointer         user_data)
280 {
281     int pan=gtk_range_get_value(GTK_RANGE(panScroll));
282     int tilt=gtk_range_get_value(GTK_RANGE(tiltScroll));
283     move(fd_serial_servo,pan,tilt);
284     return FALSE;
285 }
286 
287 
288 gboolean    //Al movimento della barra dell'angolo Tilt
289 on_tiltScroll_change_value              (GtkRange        *range,
290                                         GtkScrollType    scroll,
291                                         gdouble          value,
292                                         gpointer         user_data)
293 {
294     int pan=gtk_range_get_value(GTK_RANGE(panScroll));
295     int tilt=gtk_range_get_value(GTK_RANGE(tiltScroll));
296     move(fd_serial_servo,pan,tilt);
297     return FALSE;
298 }
299 
300 
301 
302 void    //Alla pressione del pulsante di Features e Template
```

```
303 on_FlowButton_toggled                  (GtkToggleButton *togglebutton,
304                                         gpointer         user_data)
305 {
306     if ((GTK_TOGGLE_BUTTON (togglebutton))->active)     //Se il bottone non era attivo
307     {
308         if ((trackWindow.width >0 && trackWindow.height > 0)) //Se sta gi\'a tracciando
309         {
310             templateMatching(FIRST_INSTANCE);
311             flowFeatures(FIRST_INSTANCE);
312         }
313     }
314     else                    //Se il main \'e gi\'a in esecuzione
315     {
316     }
317 }
318
319
320 void    //Alla pressione del pulsante di nuova rilevazione corner e template
321 on_redetectButton_clicked                  (GtkButton       *button,
322                                         gpointer         user_data)
323 {
324     if ((trackWindow.width >0 && trackWindow.height > 0))
325     {
326         fprintf(stderr,"Redetecting...\n");
327         templateMatching(FIRST_INSTANCE);
328         flowFeatures(FIRST_INSTANCE);
329     }
330 }
331
332
333 void    //Alla modifica del fattore di adattamento dinamico delle barre
334 on_dynScrollSpin_value_changed         (GtkSpinButton   *spinbutton,
335                                         gpointer         user_data)
336 {
337     dynScrollFactor = gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(dynScrollSpin));
338     fprintf(stderr,"Scroll Factor %d\n",dynScrollFactor);
339 }
340
341
342 void    //Alla modifica del fattore di adattamento dinamico dell'istogramma
343 on_dynHistSpin_value_changed           (GtkSpinButton   *spinbutton,
344                                         gpointer         user_data)
345 {
346     dynHistFactor = gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(dynHistSpin));
347     fprintf(stderr,"Hist Factor %d\n",dynHistFactor);
348 }
349
350
351 void    //Alla modifica del valore massimo delle feature utilizzate
352 on_maxFeaturesSpin_value_changed       (GtkSpinButton   *spinbutton,
353                                         gpointer         user_data)
354 {
355     MAX_CORNERS = gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(maxFeaturesSpin));
356 }
357
358 void    //Alla modifica del valore massimo di feature per passo
359 on_stepFeaturesSpin_value_changed      (GtkSpinButton   *spinbutton,
360                                         gpointer         user_data)
361 {
362     STEP_CORNERS = gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(stepFeaturesSpin));
363 }
364
365 void    //Alla pressione del tasto di abilitazione movimento automatico robot
366 on_robotAutoEnable_toggled             (GtkToggleButton *togglebutton,
367                                         gpointer         user_data)
368 {
369     if ((GTK_TOGGLE_BUTTON (togglebutton))->active)     //Attivazione
370     {
371         //Disabilita pulsante manuale
```

```
372         gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(robotManualEnable),FALSE);
373         gtk_widget_set_sensitive(robotManualEnable, FALSE);
374         //Disabilita controlli manuali
375         gtk_widget_set_sensitive(robotManualControlBox, FALSE);
376 
377         int debugLevel = gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(robotDebugSpin));
378         char command[5];
379 
380         sprintf(command,"d%d\r",debugLevel);
381 
382         send(fd_serial_motors, command);  //Invia debug mode
383         send(fd_serial_motors,"m2+000+000\r");
384         send(fd_serial_motors,"a2\r");  //Invia abilitazione motori
385     }
386     else //Disattivazione
387     {
388         gtk_widget_set_sensitive(robotManualEnable,TRUE);        //Riabilita modalit\‘a manuale
389         gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(robotAutoEnable),FALSE);     //Disabilita
                il pulsante automatic mode
390         gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(robotManualEnable),FALSE);   //Disabilita
                il pulsante manual mode
391 
392         send(fd_serial_motors,"b2\r");  //Blocca i due motori
393     }
394 }
395 
396 
397 void    //Alla pressione del pulsante di movimento manuale robot
398 on_robotManualEnable_toggled           (GtkToggleButton *togglebutton,
399                                         gpointer         user_data)
400 {
401     if ((GTK_TOGGLE_BUTTON (togglebutton))->active)     //Attivazione
402     {
403         //Disabilita il pulsante automatic mode
404         gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(robotAutoEnable),FALSE);
405         gtk_widget_set_sensitive(robotAutoEnable, FALSE);
406         //Abilita controlli manuali
407         gtk_widget_set_sensitive(robotManualControlBox, TRUE);
408         gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(robotHaltRadio),TRUE);
409 
410         int debugLevel = gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(robotDebugSpin));
411         char command[5];
412 
413         sprintf(command,"d%d\r",debugLevel);
414 
415         send(fd_serial_motors, command);  //Invia debug mode
416         send(fd_serial_motors,"m2+000+000\r");
417         send(fd_serial_motors,"a2\r");  //Invia abilitazione motori
418     }
419     else //Disattivazione
420     {
421         gtk_widget_set_sensitive(robotAutoEnable,TRUE);  //Riabilita modalit\‘a automatica
422         gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(robotAutoEnable),FALSE); //Disabilita il
                pulsante automatic mode
423         gtk_toggle_button_set_active(GTK_TOGGLE_BUTTON(robotManualEnable),FALSE); //Disabilita il
                pulsante manual mode
424         //Disabilita controlli manuali
425         gtk_widget_set_sensitive(robotManualControlBox, FALSE);
426 
427         send(fd_serial_motors,"b2\r");  //Blocca i due motori
428     }
429 }
430 
431 
432 void    //NON UTILIZZATO, PER SVILUPPI FUTURI
433 on_robotCommandButton_pressed          (GtkButton       *button,
434                                         gpointer         user_data)
435 {
436 }
```

```
437
438 void    //Alla pressione del pulsante muovi avanti robot
439 on_robotForwardRadio_toggled           (GtkToggleButton *togglebutton,
440                                         gpointer         user_data)
441 {
442         int robotSpeed = gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(robotSpeedSpin));
443         char command[12];
444
445         sprintf(command,"m2+%03d-%03d\r",robotSpeed, robotSpeed);
446         send(fd_serial_motors, command);  //Invia debug mode
447 }
448
449
450 void    //Alla pressione del pulsante ruota antiorario robot
451 on_robotLeftRadio_toggled              (GtkToggleButton *togglebutton,
452                                         gpointer         user_data)
453 {
454         int robotSpeed = gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(robotSpeedSpin));
455         char command[12];
456
457         sprintf(command,"m2+%03d+%03d\r",robotSpeed, robotSpeed);
458         send(fd_serial_motors, command);  //Invia debug mode
459 }
460
461
462 void //Alla pressione del pulsante ferma robot
463 on_robotHaltRadio_toggled              (GtkToggleButton *togglebutton,
464                                         gpointer         user_data)
465 {
466         send(fd_serial_motors,"m2+000-000\r");
467 }
468
469
470 void    //Alla pressione del pulsante ruota orario robot
471 on_robotRightRadio_toggled             (GtkToggleButton *togglebutton,
472                                         gpointer         user_data)
473 {
474         int robotSpeed = gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(robotSpeedSpin));
475         char command[12];
476
477         sprintf(command,"m2-%03d-%03d\r",robotSpeed, robotSpeed);
478         send(fd_serial_motors, command);  //Invia debug mode
479 }
480
481
482 void    //Alla pressione del pulsante muovi indietro robot
483 on_robotBackwardRadio_toggled          (GtkToggleButton *togglebutton,
484                                         gpointer         user_data)
485 {
486         int robotSpeed = gtk_spin_button_get_value_as_int(GTK_SPIN_BUTTON(robotSpeedSpin));
487         char command[12];
488
489         sprintf(command,"m2-%03d+%03d\r",robotSpeed, robotSpeed);
490         send(fd_serial_motors, command);  //Invia debug mode
491 }
492
493
494 gboolean    //Alla pressione di un tasto nella casella di comando manuale al robot
495 on_robotCommandEntry_key_press_event   (GtkWidget       *widget,
496                                         GdkEventKey     *event,
497                                         gpointer         user_data)
498 {
499         gchar *entry;
500         char comando[15];
501
502         if (event->keyval == GDK_Return)
503         {
504             entry = g_strdup(gtk_entry_get_text((GTK_ENTRY(widget))));
505             sprintf(comando,"%s\r",entry);
```

```
506             send(fd_serial_motors,comando);
507         }
508         return FALSE;
509 }
510
511
512 void    //Controllo bilanciamento del bianco manuale
513 on_blueWBScroll_value_changed          (GtkRange        *range,
514                                         gpointer         user_data)
515 {
516     int redWB = gtk_range_get_value(GTK_RANGE(redWBScroll));
517     int blueWB = gtk_range_get_value(GTK_RANGE(blueWBScroll));
518     dc1394_set_white_balance(fwHandle, camera.node, redWB, blueWB);
519 }
520
521
522 void    //Controllo bilanciamento del bianco manuale
523 on_redWBScroll_value_changed           (GtkRange        *range,
524                                         gpointer         user_data)
525 {
526     int redWB = gtk_range_get_value(GTK_RANGE(redWBScroll));
527     int blueWB = gtk_range_get_value(GTK_RANGE(blueWBScroll));
528     dc1394_set_white_balance(fwHandle, camera.node, redWB, blueWB);
529 }
530
531
532 void    //Controllo bilanciamento del bianco manuale
533 on_brightnessScroll_value_changed      (GtkRange        *range,
534                                         gpointer         user_data)
535 {
536     int brightness = gtk_range_get_value(GTK_RANGE(brightnessScroll));
537     dc1394_set_brightness(fwHandle, camera.node, brightness);
538 }
539
540
541 void    //Controllo bilanciamento del bianco manuale
542 on_manualWhiteBalToggle_toggled        (GtkToggleButton *togglebutton,
543                                         gpointer         user_data)
544 {
545     int redWB = gtk_range_get_value(GTK_RANGE(redWBScroll));
546     int blueWB = gtk_range_get_value(GTK_RANGE(blueWBScroll));
547
548     if ((GTK_TOGGLE_BUTTON (togglebutton))->active)     //Attivazione
549     {
550         dc1394_auto_on_off(fwHandle, camera.node, FEATURE_WHITE_BALANCE,0); //Bilanciamento bianco
                manuale
551         dc1394_set_white_balance(fwHandle, camera.node, redWB, blueWB);
552     }
553     else    //Disattivazione
554     {
555         dc1394_auto_on_off(fwHandle, camera.node, FEATURE_WHITE_BALANCE,1); //Bilanciamento bianco
                automatico
556     }
557 }
558
559
560 void    //Controllo bilanciamento del bianco manuale
561 on_manualBrightnessToggle_toggled      (GtkToggleButton *togglebutton,
562                                         gpointer         user_data)
563 {
564     int brightness = gtk_range_get_value(GTK_RANGE(brightnessScroll));
565     if ((GTK_TOGGLE_BUTTON(togglebutton))->active)      //Attivazione
566     {
567         dc1394_auto_on_off(fwHandle, camera.node, FEATURE_BRIGHTNESS,0);    //Brightness Manuale
568         dc1394_set_brightness(fwHandle, camera.node, brightness);
569     }
570     else        //Disattivazione
571     {
572         dc1394_auto_on_off(fwHandle, camera.node, FEATURE_BRIGHTNESS,1);    //Brightness
```

```
            Automatico
573     }
574
575 }
```

A.2.4 blobdetect.h

Listing A.7: servocamera/src/blobdetect.h

```
1  /* blobdetect.h */
2  // Algoritmo di rilevazione del blob colore nell'immagine
3
4  //Librerie di OpenCV
5  #include <cv.h>
6  #include <highgui.h>
7
8  //Modi di funzionamento degli algoritmi
9  #define         INIT            0               //Inizalizzazione variabili e strutture
10 #define         FIRST_INSTANCE  1               //Prima passata
11 #define         RUN             2               //Esecuzione a regime
12
13 #define         NUMCOLORS       32              //Grandezza dell'istogramma hue
14
15 //Parametri di adattativit\'a dinamica
16 #define         DYN_HIST_FACTOR     1000.f        //Fattore di adattamento dinamico histogramma
17 #define         DYN_SCROLL_FACTOR   100.f         //Fattore di adattamento dinamico scrollbar
18
19 #define         FRAMES_BEFORE_FEATURES  7         //Numero di frame da processare prima di usare
       con le features
20
21 /*********************************/
22 /* Variabili condivise tra i file */
23 /*********************************/
24     //in callbacks.c
25         extern CvRect selection;
26
27     //in servocamera.h
28         extern IplImage *frame;
29         extern GtkWidget *vminScroll;
30         extern GtkWidget *vmaxScroll;
31         extern GtkWidget *sminScroll;
32         extern GtkWidget *autoCalibCheck;
33         extern GtkWidget *dynScrollCheck;
34         extern GtkWidget *dynHistCheck;
35         extern GtkWidget *flowButton;
36
37     //in flowfeat.h
38         extern CvPoint2D32f* points[2];
39         extern int count;
40         extern CvPoint2D32f featCenter;     //Centro delle features
41         extern CvRect featRect;             //Rettangolo contenente tutte le features;
42         extern CvBox2D featBox;             //Box contenente tutte le features
43         extern int currentFeatures;         //Numero di features attualmente considerate
44
45     //In template.h
46         extern CvRect rettangolo;
47
48 /***************************/
49 /* Variabili di questo file */
50 /***************************/
51     IplImage *blobFrame=0, *histFrame = 0;                                  //Frame da
           visualizzare
52     IplImage *hsv = 0, *hue = 0, *saturation = 0, *value = 0;               //Immagini HSV
            e singoli canali
```

```
53      IplImage *mask = 0, *backproject = 0;
54      IplImage *colormask = 0;                                                    //Filtro per i
            colori da non tracciare
55
56      CvHistogram *histHue = 0, *histHueNew = 0, *histSat = 0, *histVal = 0;      //Istogrammi
            per HSV
57
58      int vmin=10, vmax=256, smin=30;                 //Valori delle scrollbar
59      int dynHistFactor=1000, dynScrollFactor=100;    //Fattori aggiornamento dinamico scrollbar e
            istogramma
60      int startingFrames = 0;                         //Contatore per i frames iniziali
61
62      CvRect trackWindow;
63      CvConnectedComp track_comp;
64      CvBox2D track_box;
65
66      //Variabili per l'istogramma HUE
67      float hueRanges_arr[] = {0,180};                  //Range valori di hue
68      float* hueRanges = hueRanges_arr;
69      int hueDims = NUMCOLORS;                          //numero di barre (colori) utilizzati dall
            'istogramma hue
70      int maxHueIndex;                                 //indice dello hue massimo nell'istogramma
71      float maxHueValue;                                //Intensit\'a del colore massimo
72      CvScalar maxColor;                                //Colore RGB domninante nell'istogramma
            hue
73      int binWidth;                                     //Larghezza colonne dell'istogramma
74      double binHeight, binHeightNew;                                 //Altezza delle colonne
            dell'istogramma
75
76      float maxSatValue, maxValValue, minSatValue, minValValue;
77
78      int i;                                            //Contatore generico per i cicli
79
80      //Variabili per gli istogrammi SATURATION e VALUE
81  //      float satvalRanges_arr[] = {0, 256};                        //Range valori di
        saturation e value
82  //      float* satvalRanges = satvalRanges_arr;
83  //      int satvalDims = 128;                                       //numero di barre per l'
        istogramma saturation e value
84  //      int maxSatIndex, maxValIndex, minSatIndex, minValIndex;     //indici saturazione e
        value max e min negli istogrammi
85
86  //Funzioni
87  CvScalar hsv2rgb( float hue )                               //Conversione colori da hue (0 -
        180) a RGB
88  {
89          int rgb[3], p, sector;
90          static const int sector_data[][3]=
91          {{0,2,1}, {1,2,0}, {1,0,2}, {2,0,1}, {2,1,0}, {0,1,2}};
92          hue *= 0.033333333333333333333333333333333f;
93          sector = cvFloor(hue);
94          p = cvRound(255*(hue - sector));
95          p ^= sector & 1 ? 255 : 0;
96
97          rgb[sector_data[sector][0]] = 255;
98          rgb[sector_data[sector][1]] = 0;
99          rgb[sector_data[sector][2]] = p;
100
101         return cvScalar(rgb[0], rgb[1], rgb[2],0);
102 }
103
104 //Aggiorna i valori delle barre VMIN VMAX e SMIN
105 void updateScrollbarsValues(CvRect area, int factor)
106 {
107     //Inizializza i valori minimi e massimi per la ricerca
108     minSatValue = 256;
109     maxValValue = 0;
110     minValValue = 256;
111     CvScalar color;
```

```
112
113     //Ricerca valori massimi e minimi saturazione e value nella selezione
114     cvSetImageROI(hsv, area);
115     int i;
116     for(i=0; i<area.height; i++)
117     {
118         int j;
119         for(j=0; j<area.width; j++)
120         {
121             color=cvGet2D(hsv,i,j);         //Legge il colore HSV del pixel
122             if (minSatValue > color.val[1]) minSatValue = color.val[1];    //Aggiorna la
                    saturazione minima
123             if (minValValue > color.val[2]) minValValue = color.val[2];    //Aggiorna il valore
                    minimo
124             if (maxValValue < color.val[2]) maxValValue = color.val[2];    //Aggiorna il valore
                    massimo
125         }
126     }
127     cvResetImageROI(hsv);
128
129     if (minSatValue<30) minSatValue = 30;
130     if (minValValue<10) minValValue = 10;
131
132     //Setta le barre ai valori trovati
133     gtk_range_set_value (GTK_RANGE(vminScroll), (int)(vmin+((minValValue-vmin)/factor)));
134     gtk_range_set_value (GTK_RANGE(vmaxScroll), (int)(vmax+((maxValValue-vmax)/factor)));
135     gtk_range_set_value (GTK_RANGE(sminScroll), (int)(smin+((minSatValue-smin)/factor)));
136 }
137
138 void blobDetection(int status)
139 {
140     if(status==INIT) //Inizializzazione,crea le variabili una volta sola
141     {
142         startingFrames = 0;
143
144         blobFrame = cvCreateImage( cvGetSize(frame), 8, 3 );
145         blobFrame->origin = frame->origin;
146
147         hsv = cvCreateImage(cvGetSize(frame), 8, 3);
148         hue = cvCreateImage(cvGetSize(frame), 8, 1);
149
150         mask = cvCreateImage(cvGetSize(frame), 8, 1);
151
152         backproject = cvCreateImage( cvGetSize(frame), 8, 1);
153
154         colormask = cvCreateImage(cvGetSize(frame), 8, 1);
155
156         histHue = cvCreateHist(1, &hueDims, CV_HIST_ARRAY, &hueRanges, 1 );
157         histHueNew = cvCreateHist(1, &hueDims, CV_HIST_ARRAY, &hueRanges, 1);
158
159         histFrame = cvCreateImage( cvSize(320,100), 8, 3 );
160         cvZero( histFrame );
161     }
162     else    //Se le variabili sono gi\'a state inizializzate/Create
163     {
164         //I valori vmin,vmax,smin sono aggiornati dalle funzioni di callback delle rispettive
                scrollbar
165         cvCopy(frame, blobFrame, 0);
166         cvCvtColor( blobFrame, hsv, CV_RGB2HSV );
167
168         if (status==FIRST_INSTANCE)     //Solo alla prima passata
169         {
170             startingFrames = 0;         //Azzera il contatore dei frame iniziali
171
172             //Se l'auto calibrazione iniziale \'e attiva
173             if((GTK_TOGGLE_BUTTON(autoCalibCheck))->active)
174             {
175                 updateScrollbarsValues(selection,1);
176             }
```

```
177
178             cvInRangeS(hsv, cvScalar(0,smin,MIN(vmin,vmax),0),cvScalar(180,255,MAX(vmin,vmax),0),
                    mask); //Sogliatura su hsv con valori di massima ampiezza
179             cvSplit(hsv, hue, 0, 0, 0 ); //Crea canale separato hue
180
181             //Legge colori presenti nella selezione per creare l'istogramma
182             //Setta ROI su mask e canale singolo hue
183             cvSetImageROI(hue, selection);
184             cvSetImageROI(mask, selection);
185
186             //Calcola istogrammi H
187             cvCalcHist( &hue, histHue, 0, mask );
188             cvGetMinMaxHistValue(histHue, 0, &maxHueValue, 0, &maxHueIndex);
189
190             //Riscala l'istogramma sul massimo trovato
191             cvConvertScale( histHue->bins, histHue->bins, maxHueValue ? 255. / maxHueValue : 0., 0
                    );
192
193             //Toglie ROI
194             cvResetImageROI( hue );
195             cvResetImageROI( mask );
196
197             trackWindow = selection;   //Traccia la selezione
198
199             //leggi colore Massimo
200             maxColor = hsv2rgb((maxHueIndex)*180.f/hueDims);
201
202             //Disegna l'istogramma HUE su histFrame
203             cvZero(histFrame);  //Azzera immagine dell'istogramma colore
204             binWidth = histFrame->width / hueDims;
205             for( i = 0; i < hueDims; i++ )
206             {
207                 binHeight =  cvQueryHistValue_1D(histHue, i)*histFrame->height/255;
208                 CvScalar color = hsv2rgb(i*180.f/hueDims);       //Trova il colore in RGB da HSV
209
210                 //Disegna barra colore nell'istogramma
211                 cvRectangle( histFrame, cvPoint(i*binWidth,histFrame->height),cvPoint((i+1)*
                        binWidth,histFrame->height - (int)binHeight),color, -1, 8, 0 );
212             }
213
214             status=RUN;
215         }
216         if (status==RUN)    //Ad ogni passata dopo la prima
217         {
218             //Se l'auto calibrazione continua \'e attiva
219             if((GTK_TOGGLE_BUTTON(dynScrollCheck))->active)
220             {
221                 updateScrollbarsValues(track_comp.rect, dynScrollFactor);
222             }
223
224             cvInRangeS( hsv, cvScalar(0,smin,MIN(vmin,vmax),0),cvScalar(180,255,MAX(vmin,vmax),0),
                    mask ); //Sogliatura su hsv con valori dalle barre
225             cvSplit( hsv, hue, 0, 0, 0 );   //Preleva i canali dall'hsv
226
227             cvCalcBackProject( &hue, backproject, histHue ); //Calcola il brackproject
228             cvAnd( backproject, mask, backproject, 0 );     //e lo filtra con mask
229
230             //Se vi sono delle feature e il tracking delle feature \'e attivo, disegna un ellisse
                    nel loro centro
231             if (featCenter.x>0 && featCenter.y>0 && (GTK_TOGGLE_BUTTON(flowButton))->active &&
                    startingFrames>FRAMES_BEFORE_FEATURES)
232             {
233                 cvSet( colormask, CV_RGB(64,64,64), NULL);
234                 cvRectangle(colormask, cvPoint(featRect.x,featRect.y), cvPoint(featRect.x+featRect
                        .width,featRect.y+featRect.height), WHITE, -1, 8, 0);
235
236                 cvAnd(backproject, colormask, backproject, 0);
237
238                 //Disegna rettangolo del template ritrovato sulla finestra a colori
```

```
239                 cvRectangle(blobFrame, cvPoint(rettangolo.x,rettangolo.y), cvPoint((rettangolo.x+
                        rettangolo.width),(rettangolo.y+rettangolo.height)), BLUE, 1, 8, 0);
240             }
241
242             //Aggiorna il tracking della selezione con Camshift
243             cvCamShift( backproject, trackWindow, cvTermCriteria( CV_TERMCRIT_EPS |
                    CV_TERMCRIT_ITER, 1, 1  ), &track_comp, &track_box );
244
245             trackWindow = track_comp.rect; //Imposta la finestra di tracking al blob trovato da
                    Camshift
246
247             track_box.angle = -track_box.angle; //Corregge angolo del blob di da disegnare
248
249             //Disegna ellisse di tracking sia sulla finestra a colori che sul backproject
250             cvCircle(blobFrame, cvPointFrom32f(track_box.center), 10, GREEN,2, 8, 0);
251             cvEllipseBox( blobFrame, track_box, RED, 3, CV_AA, 0 );
252             cvEllipseBox( backproject, track_box, GRAY50, 3, CV_AA, 0 );
253
254             //Se l'istogramma \'e dinamico
255             if((GTK_TOGGLE_BUTTON(dynHistCheck))->active)
256             {
257                 //Filtra nuovamente la maschera
258                 cvInRangeS( hsv, cvScalar(0,smin,MIN(vmin,vmax),0),cvScalar(180,255,MAX(vmin,vmax)
                        ,0), mask ); //Sogliatura su hsv con valori dalle barre
259                 cvSplit( hsv, hue, 0, 0, 0 );   //Preleva i canali dall'hsv
260
261                 //Setta la ROI come l'area tracciata
262                 cvSetImageROI( hue, track_comp.rect);
263                 cvSetImageROI( mask, track_comp.rect);
264
265                 //Calcola il nuovo istogramma
266                 cvCalcHist( &hue, histHueNew, 0, mask );
267                 cvGetMinMaxHistValue(histHueNew, 0, &maxHueValue, 0, &maxHueIndex);
268                 //Riscala con il massimo valore
269                 cvConvertScale( histHueNew->bins, histHueNew->bins, maxHueValue ? 255. /
                        maxHueValue : 0., 0 );
270
271                 cvResetImageROI( hue );
272                 cvResetImageROI( mask );
273
274                 cvZero(histFrame);  //Azzera immagine dell'istogramma colore
275                 binWidth = histFrame->width / hueDims;
276                                 //
277                 for(i = 0; i < hueDims; i++ )   //Costruisce l'istogramma aggiornandone i valori
                        rispetto al nuovo istogramma
278                 {
279                     binHeightNew = cvQueryHistValue_1D(histHueNew, i);
280                     binHeight = cvQueryHistValue_1D(histHue, i);
281
282                     cvSetReal1D(histHue->bins, i, (double)(binHeight+(double)((binHeightNew-
                            binHeight)/dynHistFactor)));
283                 }
284
285                 //Riscala il vecchio istogramma ora modificato
286                 cvGetMinMaxHistValue( histHue, 0, &maxHueValue, 0, &maxHueIndex );
287                 cvConvertScale( histHue->bins, histHue->bins, maxHueValue ? 255. / maxHueValue :
                        0., 0 );
288
289                 maxColor = hsv2rgb(maxHueIndex*180.f/hueDims);                          //Trova il
                         colore massimo
290
291                 //Disegna il nuovo istogramma;
292                 for(i = 0; i< hueDims; i++)
293                 {
294                     binHeight = cvQueryHistValue_1D(histHue, i)*histFrame->height/255;
295                     CvScalar color = hsv2rgb(i*180.f/hueDims);
296                     //Disegna barra colore nell'istogramma
297                     cvRectangle( histFrame, cvPoint(i*binWidth,histFrame->height),cvPoint((i+1)*
                            binWidth,histFrame->height - (int)binHeight),color, -1, 8, 0 );
```

```
298                 }
299             }
300         }
301     }
302 }
```

A.2.5 flowfeat.h

Listing A.8: servocamera/src/flowfeat.h

```
1  /* flowfeat.h */
2  //Codice per il rilevamento dei punti salienti/features.
3
4  //Librerie per OpenCV
5  #include <cv.h>
6  #include <highgui.h>
7
8  //Librerie matematiche e altro
9  #include <math.h>
10 #define     PI      3.141592
11
12 //Modi di funzionamento degli algoritmi
13 #define         INIT            0               //Inizializzazione variabili e strutture
14 #define         FIRST_INSTANCE  1               //Prima passata
15 #define         RUN             2               //Esecuzione a regime
16
17 /*********************************/
18 /* Variabili condivise tra i file */
19 /*********************************/
20     /* In servocamera.h */
21         extern IplImage *frame;
22
23     /* In blobdetect.h */
24         extern CvBox2D track_box;
25         extern CvScalar maxColor;
26         extern CvRect trackWindow;
27         extern int maxHueIndex;
28
29     /* In template.h */
30         extern int ricalcola_t;
31
32 /*************************/
33 /* Variabili di questo file */
34 /*************************/
35
36     //Parametri per rilevazione punti salienti
37     double QUALITY = 0.01;      //Valore di qualit\'a
38     double MIN_DISTANCE = 20;   //Distanza minima tra i corner trovati
39     int BLOCK_SIZE = 3;         //Dimensione blocchi
40     int USE_HARRIS = 1;         //Flag di utilizzo Harris
41     double HARRIS_K = 0.04;     //Valore K di Harris
42
43     //Parametri per il numero di features
44     int MAX_CORNERS = 30;       //Massimo numero di corner da ricercare in totale
45     int STEP_CORNERS = 5;       //Massimo numero di corner da ricerca ad ogni passo
46
47     //Variabili di flowfeat
48     IplImage *featFrame = 0, *grey = 0, *prev_grey = 0, *pyramid = 0, *prev_pyramid = 0, *
           swap_temp;
49     CvPoint2D32f* points[2] = {0,0}, *swap_points;
50     CvPoint2D32f* newCorners = {0};
51
52     CvPoint2D32f* flux = {0};     //questo per le feature \'e una somo novit\'a \'e un vettore che
            contiene in x il flusso rispetto a x e in y quello rispetto a y
```

```
53
54      char* valid = 0;
55      int count = 0;
56      int need_to_init = 0;
57      int night_mode = 0;
58      int flags = 0;
59      int add_remove_pt = 0;
60      CvPoint pt;
61      CvPoint2D32f featCenter;    //Centro delle features
62      CvRect featRect;            //Rettangolo contenente tutte le features;
63      CvBox2D featBox;            //Box per il disegno dell'ellisse sulle features
64
65      IplImage *eig = 0;
66      IplImage *temp = 0;
67
68      int i, k, c, h, j;         //Contatori vari
69
70      //Per l'aggiornamento delle features
71      // CvPoint2D32f remainingFeatures[MIN_FEATURE]; //vettore contenete le ultime MIN_FEATURE dell
            'algoritmo
72      int remainingFeaturesNum=0;                 //contatore feature rimanenti;
73      CvScalar featureRGB;                        //colore della singola features
74      CvScalar featureHSV;                        //colore della singola features
75      CvScalar dominant_color_hsv;                //colore della singola features
76      IplImage *featFrameHSV;                     //immagine features in versione hsv
77      int minFeatures=15, maxFeatures=200;
78      int stepFeatures=5;                         //Numero di feature da calcolare ad ogni frame
79      int currentFeatures=0;                      //Numero di features attualmente considerate
80      int ricalcola_f=1;
81      int win_size = 10;                          //distanza tra le feature calcolate in modo fisso
82
83  void flowFeatures(int init_status)
84  {
85      if (init_status == INIT)
86      {
87          //Alloca i vari buffer di flowfeature
88          featFrame = cvCreateImage( cvGetSize(frame), 8, 3 );
89          featFrame->origin = frame->origin;
90          grey = cvCreateImage( cvGetSize(frame), 8, 1 );
91          prev_grey = cvCreateImage( cvGetSize(frame), 8, 1 );
92          pyramid = cvCreateImage( cvGetSize(frame), 8, 1 );
93          prev_pyramid = cvCreateImage( cvGetSize(frame), 8, 1 );
94          points[0] = (CvPoint2D32f*)cvAlloc((MAX_CORNERS)*sizeof(points[0][0]));         //Array di
                punti salienti vecchi
95          points[1] = (CvPoint2D32f*)cvAlloc((MAX_CORNERS)*sizeof(points[0][0]));         //Array di
                punti salienti nuovo
96          valid = (char*)cvAlloc(MAX_CORNERS);                                            //Array di
                flag per validit\'a punti salienti
97          newCorners = (CvPoint2D32f*)cvAlloc((STEP_CORNERS )*sizeof(points[0][0]));      //Array di
                corner nuovi ad ogni frame
98          flux=(CvPoint2D32f*)cvAlloc((MAX_CORNERS )*sizeof(points[0][0]));               //Array
                che contiene il flusso
99
100         flags = 0;
101
102         eig = cvCreateImage( cvGetSize(frame), 32, 1 );
103         temp = cvCreateImage( cvGetSize(frame), 32, 1 );
104         featFrameHSV = cvCreateImage( cvGetSize(frame), 8, 3 );
105
106         currentFeatures = count = 0;
107     }
108     else //if (!INIT)
109     {
110         if(frame)   //Se esiste un frame
111         {
112             cvCopy( frame, featFrame, 0 );  //Copia per i corner
113         }
114
115         cvCvtColor( featFrame, grey, CV_RGB2GRAY );             //Crea frame gray in scala di
```

```
            grigi
116
117         if ((init_status == FIRST_INSTANCE)||(ricalcola_f==1))
118         {
119             if (ricalcola_f==1)
120             {
121                 ricalcola_f=0;
122
123                 //Setta le ROI per la ricerca all'interno dell'area tracciata
124                 cvSetImageROI(grey, trackWindow);
125                 cvSetImageROI(eig, trackWindow);
126                 cvSetImageROI(temp, trackWindow);
127
128                 count = STEP_CORNERS;
129                 //Trova i nuovi punti salienti nell'area tracciata
130                 cvGoodFeaturesToTrack( grey, eig, temp,points[1] , &count, QUALITY, MIN_DISTANCE,
                        0, BLOCK_SIZE, USE_HARRIS, HARRIS_K );
131                 //cvGoodFeaturesToTrack( grey, eig, temp,points[1], &count, QUALITY, MIN_DISTANCE,
                         0, 3, 0, 0.04 );
132
133                 //Resetta le ROI
134                 cvResetImageROI(grey);
135                 cvResetImageROI(eig);
136                 cvResetImageROI(temp);
137
138                 //Per ogni punto trovato
139                 for( i = 0; i < count; i++ )
140                 {
141                     points[1][i].x=points[1][i].x+trackWindow.x;
142                     points[1][i].y=points[1][i].y+trackWindow.y;
143                 }
144                 currentFeatures = count;   //I punti considerati sono aumentati*/
145             }
146         }//(first instance)
147         else if (init_status == RUN)
148         {
149             //Se sta tracciando qualcosa
150             if(trackWindow.height>win_size && trackWindow.width>win_size)   //Blob pi\'u grande di
                     win_size
151             {
152                 if(currentFeatures > 0)     //Se ha gi\'a delle features
153                 {
154                     //Calcola lo spostamento dei punti trovati
155                     cvCalcOpticalFlowPyrLK( prev_grey, grey, prev_pyramid, pyramid, points[0],
                            points[1], currentFeatures, cvSize(win_size,win_size), 3, valid, 0,
                            cvTermCriteria(CV_TERMCRIT_ITER|CV_TERMCRIT_EPS,10,1), flags );
156                     flags |= CV_LKFLOW_PYR_A_READY;
157
158                     //Azzera baricentro dei punti trovati prima del ricalcolo
159                     featCenter.x = featCenter.y = 0;
160
161                     //Azzera rettangolo circoscritto prima del ricalcolo
162                     featRect.x = featRect.y = 1000; featRect.width = featRect.height = 0;
163
164                     //Azzera box delle features
165                     featBox.center.x = featBox.center.y = 0;
166                     featBox.size.width = featBox.size.height = 0;
167                     featBox.angle = 90;
168
169                     //Calcola estremi della regione tracciata
170                     int minX,minY,maxX,maxY;
171                     minX = trackWindow.x; minY = trackWindow.y;
172                     maxX = trackWindow.x+trackWindow.width; maxY = trackWindow.y+trackWindow.
                            height;
173
174                     //calcolo ora il flusso di ogni singola features
175                     for( i =0; i < currentFeatures; i++ )
176                     {
177                         flux[i].x=(points[0][i].x)-(points[1][i].x);
```

```
178                     flux[i].y=(points[0][i].y)-(points[1][i].y);
179                 }
180 
181                 //Per ogni punto trovato
182                 //k ovvero i punti rimasti dopo il flusso
183                 for( i = k = 0; i < currentFeatures; i++ )
184                 {
185                     //Butta via i punti salienti fuori dalla regione trackata
186                     if ((points[1][i].x<minX)||(points[1][i].x>maxX)||(points[1][i].y<minY)||(
                            points[1][i].y>maxY))    continue;
187 
188                     //Aggiorna origine del rettangolo che circoscrive i vari punti
189                     if(featRect.x>points[1][i].x) featRect.x = points[1][i].x;  //aggiorna il
                            minimo x
190                     if(featRect.y>points[1][i].y) featRect.y = points[1][i].y;  //aggiorna il
                            minimo y
191 
192                     //Calcola il totale delle x e y dei punti per il calcolo del baricentro
193                     featCenter.x+=points[1][i].x;
194                     featCenter.y+=points[1][i].y;
195 
196                     //Ricompatta l'array dei nuovi punti validi
197                     points[1][k++] = points[1][i];
198 
199                     //Disegna un cerchio del colore dominante nell'area tracciata
200                     cvCircle(featFrame, cvPointFrom32f(points[1][i]), 5, maxColor, -1, 8,0);
201                     cvCircle(backproject, cvPointFrom32f(points[1][i]), 5, WHITE, -1, 8,0);
202                     cvLine(backproject, cvPointFrom32f(points[1][i]), cvPointFrom32f(points
                            [1][i+1]),WHITE,1,8,0);
203                 } //for dei vari punti
204 
205                 //Aggiorna larghezza e lunghezza del rettangolo delle features
206                 for (i = 0; i<k; i++)
207                 {
208                     if((featRect.x+featRect.width) < points[1][i].x) featRect.width = (points
                            [1][i].x - featRect.x); //aggiorna larghezza
209                     if((featRect.y+featRect.height) < points[1][i].y) featRect.height = (
                            points[1][i].y - featRect.y); //aggiorna altezza
210                 }
211                 currentFeatures = k;  //aggiorna il valore di punti trovati/rimasti dopo il
                        flusso
212 
213                 //Costruisci il featBox per il disegno dell'ellisse
214                 featBox.center.x = featRect.x+(featRect.width/2);
215                 featBox.center.y = featRect.y+(featRect.height/2);
216 
217                 //Correggi altezza larghezza per visualizzare correttamente l'ellisse
218                 if (featRect.width > featRect.height)
219                 {
220                     featBox.size.width = featRect.height;
221                     featBox.size.height = featRect.width;
222                 }
223                 else
224                 {
225                     featBox.size.width = featRect.width;
226                     featBox.size.height = featRect.height;
227                 }
228 
229                 if (currentFeatures>0)    //Se ha trovato dei punti/ovvero ne ha ancora dopo
                        il flusso
230                 {
231                     featCenter.x/=currentFeatures;   //Calcola la x del baricentro dei punti
232                     featCenter.y/=currentFeatures;   //Calcola la y del baricentro dei punti
233 
234                     //Disegna rettangolo
235                     cvRectangle(featFrame, cvPoint(featRect.x,featRect.y), cvPoint(featRect.x+
                            featRect.width,featRect.y+featRect.height), GREEN, 1, 8, 0);
236                     cvCircle(featFrame, cvPoint(featCenter.x,featCenter.y), 10, GREEN, -1,
                            8,0);//Disegna un cerchio vuoto per il baricentro
```

```
237                         cvCircle(featFrame, cvPoint(featBox.center.x,featBox.center.y), 10, BLUE,
                                -1, 8, 0); //Disegna un cerchio per il centro geometrico
238                     } //if (currentFeatures)
239                 } //fine di currentFeatures>0
240
241                 if(currentFeatures<(MAX_CORNERS-STEP_CORNERS))   //Se deve cercare delle altre
                        features;
242                 {
243                     count = STEP_CORNERS;
244
245                     //Setta le ROI per la ricerca all'interno dell'area tracciata
246                     cvSetImageROI(grey, trackWindow);
247                     cvSetImageROI(eig, trackWindow);
248                     cvSetImageROI(temp, trackWindow);
249
250                     //Trova i nuovi punti salienti nell'area tracciata
251                     cvGoodFeaturesToTrack( grey, eig, temp, newCorners, &count, QUALITY,
                            MIN_DISTANCE, 0, BLOCK_SIZE, USE_HARRIS, HARRIS_K);
252
253                     //Toglie le ROI
254                     cvResetImageROI(grey);
255                     cvResetImageROI(eig);
256                     cvResetImageROI(temp);
257
258                     j = 0; //Per tenere conto delle feature nuove
259                     for( i = 0; i < count; i++ )    //Per ogni punto nuovo i
260                     {
261                         int equals = 0;             //Resetta flag di uguaglianza
262                         for( k = 0; k < currentFeatures; k++ ) //Per ogni feature k gi\'a presente
263                         {
264                             //Se le x sono uguali
265                             if ((points[1][k].x==newCorners[i].x+trackWindow.x))
266                             {
267                                 //Se le y sono uguali
268                                 if ((points[1][k].y==newCorners[i].x+trackWindow.y))
269                                 {
270                                     equals=1;   //Imposta flag per evitare di aggiungere il nuovo
                                            punto i
271                                     break;      //Inutile controllare i rimanenti punti
272                                 }
273                             }
274                         }
275
276                         //Se il nuovo punto i non esiste gi\'a aggiungilo in posizione
                            currentFeatures+j
277                         if (!equals)
278                         {
279                             points[1][currentFeatures+j].x=newCorners[i].x+trackWindow.x;
280                             points[1][currentFeatures+j].y=newCorners[i].y+trackWindow.y;
281                             j++;
282                         }
283                      }
284                      currentFeatures += j;   //I punti considerati sono aumentati di j
285
286                 }   //(ricerca nuove features)*/
287             }//Se sta tracciando
288         } //else if status==RUN
289
290         //Swappa i vari array di punti e di servizio (vecchio = nuovo, per avere disponibile il
                nuovo)
291         //(Lo fa sia per la FIRST_INSTANCE che per la RUN
292         CV_SWAP( prev_grey, grey, swap_temp );
293         CV_SWAP( prev_pyramid, pyramid, swap_temp );
294         CV_SWAP( points[0], points[1], swap_points );
295     } //else |= INIT
296 }//main
```

A.2.6 template.h

Listing A.9: servocamera/src/template.h

```
/* template.h */
// Algoritmo per il rilevamento dei template

//Librerie per OpenCV
#include <cv.h>
#include <highgui.h>

//Librerie matematiche e altro
#include <math.h>

//Modi di funzionamento degli algoritmi
#define         INIT            0               //Inizalizzazione variabili e strutture
#define         FIRST_INSTANCE  1               //Prima passata
#define         RUN             2               //Esecuzione a regime

//Parametri dei template
#define         MIN_FEATURE     10              //Numero di features da tenere
#define         FEAT_OFFSET     20              //offfset tra un a feature e l'altra nella
    template

#define         SEARCH_WIN      100             //ampiezza per la ricerca della template

#define         TEMP_WIDTH      60              //larghezza della template
#define         TEMP_HEIGHT     60              //altezza della template

/*********************************/
/* Variabili condivise tra i file */
/*********************************/
    /* In servocamera.h */
        extern IplImage *frame;

    /* In blobdetect.h */
        extern CvBox2D track_box;
        extern CvScalar maxColor;
        extern CvRect trackWindow;
        extern int maxHueIndex;

     /* In flowfeat.h */
        extern int ricalcola_f;
/*************************/
/* Variabili di questo file */
/*************************/

    //variabli per la template
    IplImage *templ_img = 0;                    //immagine colori in cui ricercare il template
    IplImage *templ_img_grey=0;                 //immagine bianco e nero in cui ricercare il
        template
    IplImage *templ_grey=0;                     //template da ricercare in scala di grigio
    IplImage *templ_match=0;                    //immagine risultante dall'algoritmo di match
        template release image
    IplImage *controllo=0;                      //immagine che mostra l'attuale zona inquadrata
        per il debug

    CvRect templ_rect;                                      //rettangolo da cui selezionare la
        template
    int templ_x=0,templ_y=0,templ_width=0,templ_height=0;   //origine x e y della template e sue
        dimensioni

    CvRect search_win_rect;                     //rettangolo che inquadra la zona di ricerca
    int match_width=0,match_height=0;           //dimensione dell'immagine della zona di ricerca
    int match_x=0,match_y=0;                    //origine della zona di ricerca

```

```
57      CvRect rettangolo;                          //rettangolo che inquadra la zona che corrisponde
            alla template dopo la ricerca
58      int rettangolo_x=0,rettangolo_y=0,rettangolo_width=0,rettangolo_height=0; //origine x e y
            della template e sue dimensioni
59
60      CvScalar similitudine,similitudine_max;     //luminosit\'a del singolo pixel di match template
             usata per la ricerca del migliore
61      CvScalar similitudine_min;                  //percentulale minima di similitudine tra il
            modello della template e la template attuale
                                                        //in pratica il livello minim o per il
            ricalcolo della template
62
63      int best_x, best_y;                 //coordinate x,y in cui si trove il migliore match
64      int h,j;                            //contatori per cicli for
65
66      int template_flux_x=0,template_flux_y=0;
67      int blob_center_x=0, blob_center_y=0;
68      int template_center_x=0, template_center_y=0;
69
70      double frame_number=0;              //numero di frame di permanenza della template
71      int ricalcola_t=1;                  //flag di ricalcolo template
72
73  void templateMatching(int init_status)
74  {
75      if (init_status == INIT)
76      {
77          similitudine_min.val[0]=0.5;    //Minimo valore di ricalcolo della template
78
79          //Valori iniziali del template iniziale (templ) e del template ritrovato (rettangolo)
80          templ_x=rettangolo_x=280; templ_y=rettangolo_y=200;
81          templ_width=rettangolo_width=80; templ_height=rettangolo_height=80;
82
83          rettangolo=templ_rect=cvRect(templ_x,templ_y,templ_width,templ_height);
84
85          templ_img = cvCreateImage( cvGetSize(frame), 8, 3 );      //creo l'imagine in cui
                ricercare la template a colori
86          templ_img_grey = cvCreateImage( cvGetSize(frame),8, 1 );  //la copia in b/n
87          templ_grey= cvCreateImage( cvGetSize(frame),8, 1 );       //la template stessa in seguito
                ne usero' solo la roi di dimensioni ridotte
88
89          controllo= cvCreateImage( cvGetSize(frame),8, 1 );        //una immagine di controllo per
                il debug
90
91          //calcolo le dimensioni dell'immagine risultato e la creo
92          match_width=((cvGetSize(frame)).width)-templ_width+1;
93          match_height=((cvGetSize(frame)).height)-templ_height+1;
94          templ_match=cvCreateImage(cvSize(match_width,match_height), 32, 1); //32bit, monocanale
95      }
96      else //if (!INIT)
97      {
98          if(frame)
99          {
100             cvCopy( frame, templ_img, 0);   //Copia per i template
101         }
102
103         cvCvtColor( templ_img, templ_img_grey, CV_RGB2GRAY);    //Crea l'immagine in cui fare la
                ricerca in scala di grigi
104
105         if ((init_status == FIRST_INSTANCE) || (ricalcola_t==1))
106         {
107             if (ricalcola_t==1);
108             {
109                 ricalcola_t=0;
110
111                 //Setta i parametri per la finestratura in dipendenza dal blob inquadrato
112                 templ_x=rettangolo_x=trackWindow.x; templ_y=rettangolo_y=trackWindow.y;
113                 templ_width=rettangolo_width=trackWindow.width; templ_height=rettangolo_height=
                        trackWindow.height;
114
```

```
115                 //Sistema la finestra della template per farla stare nella finestra di ricerca
116                 if(templ_width>(cvGetSize(frame)).width-SEARCH_WIN)
117                     templ_width=rettangolo_width=(cvGetSize(frame)).width-SEARCH_WIN;
118
119                 if(templ_height>(cvGetSize(frame)).height-SEARCH_WIN)
120                     templ_height=rettangolo_height=(cvGetSize(frame)).height-SEARCH_WIN;
121
122                 if(templ_x<SEARCH_WIN/2)
123                     templ_x=rettangolo_x=SEARCH_WIN/2;
124
125                 if(templ_y<SEARCH_WIN/2)
126                     templ_y=rettangolo_y=SEARCH_WIN/2;
127
128                 if(templ_x>(frame->width)-rettangolo_width-(SEARCH_WIN/2))
129                     templ_x=rettangolo_x=frame->width-rettangolo_width-(SEARCH_WIN/2);
130
131                 if(templ_y>(frame->height)-rettangolo_height-(SEARCH_WIN/2))
132                     templ_y=rettangolo_y=frame->height-rettangolo_height-(SEARCH_WIN/2);
133
134                 //rettangolo per template in ingresso (templ_rect) e uscita (rettangolo)
135                 rettangolo=templ_rect=cvRect(templ_x,templ_y,templ_width,templ_height);
136
137                 cvResetImageROI(templ_grey);    //resetto la roi che  rappresenta la template
138
139                 //riempio l'immagine da cui poi restrarro la template
140                 cvCopy( templ_img_grey,  templ_grey, 0 );   //Copia per la ricerca template
141                 cvSetImageROI(templ_grey,templ_rect );      //la setto con i bnuovi valori
142             }
143         }//(first instance)
144         else if (init_status == RUN)
145         {
146             //Posiziona la finestra di ricerca
147             if(rettangolo_width>(cvGetSize(frame)).width-SEARCH_WIN) rettangolo_width=(cvGetSize(
                    frame)).width-SEARCH_WIN;
148             if(rettangolo_height>(cvGetSize(frame)).height-SEARCH_WIN) rettangolo_height=(
                    cvGetSize(frame)).height-SEARCH_WIN;
149             if(rettangolo_x<SEARCH_WIN/2) rettangolo_x=SEARCH_WIN/2;
150             if(rettangolo_y<SEARCH_WIN/2) rettangolo_y=SEARCH_WIN/2;
151             if(rettangolo_x>(frame->width)-rettangolo_width-(SEARCH_WIN/2)) rettangolo_x=frame->
                    width-rettangolo_width-(SEARCH_WIN/2);
152             if(rettangolo_y>(frame->height)-rettangolo_height-(SEARCH_WIN/2)) rettangolo_y=frame->
                    height-rettangolo_height-(SEARCH_WIN/2);
153
154             //disegno la zona da cui estraggo la template, \'e il rettangolo di default (poco
                    utile)
155             cvRectangle(featFrame, cvPoint(templ_rect.x,templ_rect.y), cvPoint((templ_rect.x+
                    templ_rect.width),(templ_rect.y+templ_rect.height)), RED, 1, 8, 0);
156
157             //resetto la roi dell'immagine in cui ricercare
158             cvResetImageROI(templ_img_grey);
159
160             match_x=rettangolo_x-(SEARCH_WIN/2); match_y=rettangolo_y-(SEARCH_WIN/2);
161             match_width=rettangolo_width+SEARCH_WIN; match_height=rettangolo_height+SEARCH_WIN;
162
163             //calcolo la zona di ricerca come un rettangolo calcolato sulla attuale posizione
                    della template maggiorato della search windows
164             search_win_rect=cvRect(match_x,match_y,match_width,match_height);
165             //la setto con i nuovi valori
166             cvSetImageROI(templ_img_grey, search_win_rect);
167
168             //ora in base alle dimensioni del template e alle dimensioni dell'immagine devo
                    calcolare le dimensioni della roi di match
169             match_width=rettangolo_width+SEARCH_WIN-rettangolo_width+1;
170             match_height=rettangolo_height+SEARCH_WIN-rettangolo_height+1;
171
172             cvReleaseImage(&templ_match);               //dealloco la match prima di riallocarla
                    con le nuove misure corrette
173
174             //ecco che la alloco con le corrette misure
```

```
175             templ_match=cvCreateImage(cvSize(match_width,match_height), 32, 1);
176
177             //faccio il matching della template
178             //la template \'e quella selezionata nella init e rimarrr\'a sempre uguale fini a una
                    nuova init
179             cvMatchTemplate (templ_img_grey, templ_grey, templ_match, CV_TM_CCOEFF_NORMED);
180             //metodi alternativi      CV_TM_CCOEFF_NORMED     CV_TM_CCORR_NORMED      CV_TM_CCOEFF
                       CV_TM_SQDIFF   CV_TM_CCORR
181             //CV_TM_CCOEFF_NORMED sembra il migliore
182
183             //rilascio la roi dell'immagine sorgente quella nella quale eseguo la ricerca in modo
                    tale da averla pronta alla possima iterazione
184             cvResetImageROI(templ_img_grey);
185
186             //Utilizzo i valori della templ_match (matching Table) per capire dove \'e finita la
                    template\
187             best_x=rettangolo.x; best_y=rettangolo.y;
188
189             //la luminosit\'a del singolo pixel corrisponde alla similitudine tra la template e la
                     porzione attuale dell'immagine sorgente
190             similitudine.val[0]=0;
191             similitudine_max=cvGet2D(templ_match,(match_width/2),(match_height/2));
192
193             int start_y=rettangolo.y-(SEARCH_WIN/2); int stop_y=rettangolo.y+(SEARCH_WIN/2);
194             int start_x=rettangolo.x-(SEARCH_WIN/2); int stop_x=rettangolo.x+(SEARCH_WIN/2);
195
196             //Ricerca del massimo nella matching table
197             for(h = start_y; h <stop_y ; h++ )
198             {
199                 for( j =start_x; j <stop_x ; j++ )
200                 {
201                      similitudine=cvGet2D(templ_match,(h-(rettangolo.y-(SEARCH_WIN/2))),(j-(
                         rettangolo.x-(SEARCH_WIN/2))) );
202                      // get the (i,j) pixel value
203                      if (similitudine.val[0] >similitudine_max.val[0])
204                      {
205                         similitudine_max.val[0]=similitudine.val[0];
206                         best_x=j; best_y=h;
207                         frame_number++;
208                      }
209                 }
210             }
211
212            //finita la ricerca aggiorno i valori del rettangolo di massimo matching e li pongo
                   uguali ai best trovati stando attento alle dimensioni massime dell immagine
213             if(best_x>(frame->width)-rettangolo_width-(SEARCH_WIN/2)) best_x=frame->width-
                    rettangolo_width-(SEARCH_WIN/2);
214             if(best_x<SEARCH_WIN/2) best_x=SEARCH_WIN/2;
215             rettangolo_x=best_x;
216
217             if(best_y>(frame->height)-rettangolo_height-(SEARCH_WIN/2)) best_y=frame->height-
                    rettangolo_height-(SEARCH_WIN/2);
218             if(best_y<SEARCH_WIN/2) best_y=SEARCH_WIN/2;
219             rettangolo_y=best_y;
220
221             //preparo anche le due altre dimensioni del rettangolo oltre a x e a y setto anche
                    width e height
222             rettangolo_width=templ_width; rettangolo_height=templ_height;
223
224             //aggiorno il rettangolo
225             rettangolo=cvRect(rettangolo_x,rettangolo_y,rettangolo_width,rettangolo_height);
226
227             //visulaizzo il rettangolo
228             cvRectangle(featFrame, cvPoint(rettangolo.x,rettangolo.y), cvPoint((rettangolo.x+
                    rettangolo.width),(rettangolo.y+rettangolo.height)), BLUE, 1, 8, 0);
229
230             //questa \'e l'immagine di controllo si tratta di quello che la ricerca di template in
                     movimento sta inquadrando in questo momento
231             cvResetImageROI(controllo);
```

```
232             cvCopy(templ_img_grey,  controllo, 0 );  //Fanne una copia per creare l'immagine di
                    debug
233
234             // rettangolo: ha la posizione del template
235             // points[0]: ha i punti prima del flusso
236             // points[1]: ha i punti dopo il flusso
237             // flux: ha il flusso di ogli feature
238
239             blob_center_x = trackWindow.x+(trackWindow.width/2); blob_center_y = trackWindow.y+(
                    trackWindow.height/2);
240             template_center_x=rettangolo_x+(rettangolo_width/2); template_center_y=rettangolo_y+(
                    rettangolo_height/2);
241
242             if(similitudine_max.val[0]<similitudine_min.val[0]) //Se la template \'e poco
                    affidabile
243             {
244                 //Ricalcola template
245                 ricalcola_t=1; ricalcola_f=0;
246                 frame_number=0;
247                 fprintf(stderr,"Ricalcolo TEMPLATE\n");
248             }
249             if /* I centri sono autoinclusi uno nell'altro */ ((blob_center_x<rettangolo_x)||(
                    blob_center_x>rettangolo_x+rettangolo_width)||(blob_center_y<rettangolo_y)||(
                    blob_center_y>rettangolo_y+rettangolo_height)||(template_center_x<trackWindow.x)
                    ||(template_center_x>trackWindow.x+trackWindow.width)||(template_center_y<
                    trackWindow.y)||(template_center_y>trackWindow.y+trackWindow.height))
250             {
251                 fprintf(stderr,"Risultati divergenti:");
252                 if(frame_number<100)    //se la cifra di merito per la template \'e bassa
253                 {
254                     //Ricalcola template
255                     ricalcola_t=1; ricalcola_f=0;
256                     frame_number=0;
257                     fprintf(stderr,"Aggiorno TEMPLATE\n");
258                 }
259                 else
260                 {
261                     //Ricalcola features
262                     trackWindow = rettangolo;
263                     ricalcola_t=0; ricalcola_f=1;
264                     fprintf(stderr,"Ricalcolo FEATURES\n");
265                 }
266             }
267         }//else if status==RUN
268     } //else != INIT
269 }//main
```

A.2.7 facedetect.h

Listing A.10: servocamera/src/facedetect.h

```
1  /* facedetect.h */
2  // Codice per la rilevazione di volti (o altri modelli) nell'immagine
3
4  //Librerie di OpenCV
5  #include <cv.h>
6  #include <highgui.h>
7
8  //Modi di funzionamento degli algoritmi
9  #define          INIT           0         //Inizializzazione variabili e strutture
10 #define          RUN            2         //Esecuzione a regime
11
12 //Modello dell'oggetto da rilevare
13 #define          DET_ALG          "frontalface_alt2.xml"    //Volto frontale
```

```
14 //#define         DET_ALG         "upperbody.xml"           //Corpo superiore
15 //#define         DET_ALG         "lowerbody.xml"           //Corpo inferiore
16
17 //Librerie matematiche e altro
18 #define     SEN45   0.7071
19
20 /*********************************/
21 /* Variabili condivise tra i file */
22 /*********************************/
23     /* In servocamera.h */
24         extern IplImage *frame;
25         extern GtkWidget *faceImage;
26
27     /* In blobdetect.h */
28         extern CvBox2D track_box;
29
30     /* In callbacks.c */
31         extern int selecting;       //Flag selezione in corso
32
33 /*************************/
34 /* Variabili di questo file */
35 /*************************/
36     static CvMemStorage* objectStorage = 0;             //Variabile per la memorizzazione degli
           oggetti
37     static CvHaarClassifierCascade* detectCascade = 0;  //Tipo di classificatore
38
39     CvRect elements[20];            //Array di oggetti rilevati
40
41     //Inizializza immagini
42     IplImage *grayImg = 0;          //Immagine grigia (temporanea)
43     IplImage *faceFrame = 0;        //Immagine per la face detection
44     IplImage *smallSearchImg = 0;   //Immagine pi\'u piccola usata per la ricerca
45     double scale = 1.2;             //Fattore di scala per il rimpicciolimento del frame di
           ricerca
46
47 //Funzione per la face detection
48 int detectObjects(IplImage* image, CvRect regions[])
49 {
50     static CvScalar colors[] =  //Colori con cui evidenziare diverse objects
51     {
52         {{0,0,255}},
53         {{0,128,255}},
54         {{0,255,255}},
55         {{0,255,0}},
56         {{255,128,0}},
57         {{255,255,0}},
58         {{255,0,0}},
59         {{255,0,255}}
60     };
61
62     cvCvtColor( image, grayImg, CV_RGB2GRAY );              //Crea immagine grigia
63     cvResize( grayImg, smallSearchImg, CV_INTER_LINEAR );   //Ridimensiona immagine grigia
64     cvEqualizeHist( smallSearchImg, smallSearchImg );       //Normalizza istogramma B/W dell'
           immagine piccola
65     cvClearMemStorage( objectStorage );                     //Cancella immagazzinamento di memoria
            per le objects
66
67     if (detectCascade)        //Se ha inizializzato un algoritmo di rilevamento
68     {
69         double t = (double)cvGetTickCount();    //Legge i tick iniziali
70
71         CvSeq* objects = cvHaarDetectObjects( smallSearchImg, detectCascade, objectStorage, 1.1,
               3, 0, cvSize(30, 30) ); //Ricerca nell'immagine piccola in bianco e nero
72
73         t = (double)cvGetTickCount() - t;       //Legge i tick passati
74         printf( "%d objects found in %gms\n",objects->total, t/((double)cvGetTickFrequency()
               *1000.) );  //Stampa tempo di rilevamento
75
76         int i;
```

```
77          for( i = 0; i < (objects ? objects->total : 0); i++ )   //Per ogni oggetto trovato
78          {
79              CvRect* r = (CvRect*)cvGetSeqElem( objects, i );    //Legge l'iesimo elemento come
                    Rect
80
81              //Riaggiusta centro e raggio dell'elemento trovato (toglie scalatura)
82              CvPoint center; int radius;
83              center.x = cvRound((r->x + r->width*0.5)*scale);
84              center.y = cvRound((r->y + r->height*0.5)*scale);
85              radius = cvRound((r->width + r->height)*0.25*scale);
86
87              cvCircle( image, center, radius, colors[i%8], 3, 8, 0 );    //Disegna un cerchio sulla
                     faccia trovata
88
89              int size = cvRound(radius*SEN45);   //Calcola semilato del quadrato inscritto nel
                    cerchio.
90
91              //Costruisce il quadrato inscritto nel cerchio della faccia e lo salva in elements (
                    array di Rect)
92              CvRect faceRect;
93              faceRect.x = cvRound(center.x-size);
94              faceRect.y = cvRound(center.y-size);
95              faceRect.height = faceRect.width = cvRound(size*2);
96
97              //Trova il rettangolo del busto basandosi sulla faccia
98              CvRect trunkRect;
99              trunkRect.x = cvRound(r->x*scale);
100             trunkRect.y = cvRound((r->y+(r->height*1.5))*scale);
101             trunkRect.width = cvRound(r->width*scale);
102             trunkRect.height = cvRound((r->height*1.5)*scale);
103
104             //Risistema l'origine del rettangolo se \'e fuori dall'immagine
105             if (trunkRect.x < 0) trunkRect.x=0;
106             if (trunkRect.y < 0) trunkRect.y=0;
107
108             //Risistema le dimensioni del rettangolo se sono fuori dall'immagine
109             if (trunkRect.x+trunkRect.width > image->width) trunkRect.width = (image->width-
                    trunkRect.x);
110             if (trunkRect.y+trunkRect.height > image->height) trunkRect.height = (image->height-
                    trunkRect.y);
111
112             cvRectangle(image,cvPoint(trunkRect.x, trunkRect.y), cvPoint(trunkRect.x+trunkRect.
                    width, trunkRect.y+trunkRect.height), GREEN, 3, 8, 0);
113
114             elements[i] = trunkRect;    //Seleziona busto
115 //             elements[i] = faceRect;     //Seleziona faccia
116         }
117         return objects->total;                                          //Restituisce il numero di
                 oggetti rilevati
118     }
119 }
120
121 void faceDetection(int status)
122 {
123     if (status==INIT)
124     {
125         //operazioni preliminari per la face detection
126         faceFrame = cvCreateImage( cvGetSize(frame), 8, 3);             //Crea la faceFrame grande
                quanto il frame (640x480 in genere)
127         faceFrame->origin = frame->origin;                                      //Imposta l'
                origine
128
129         objectStorage = cvCreateMemStorage(0);                                  //Crea lo spazio
                in memoria per allocare gli oggetti
130         detectCascade = (CvHaarClassifierCascade*)cvLoad( DET_ALG, 0, 0, 0 );   //Legge l'
                algoritmo di object detection in DET_ALG
131
132         grayImg = cvCreateImage( cvSize(frame->width,frame->height), 8, 1 );    //Crea Immagine
                grigia grande quanto il frame
```

```
133         smallSearchImg = cvCreateImage( cvSize( cvRound (frame->width/scale),cvRound (frame->
                height/scale)), 8, 1 ); //Immagine piccola scalata
134     }
135     else if (status==RUN)
136     {
137         cvCopy(frame, faceFrame, 0);
138 
139         if(detectObjects(faceFrame, elements))  //Se trova almeno un'oggetto nel faceFrame
140         {
141             selection=elements[0];              //Seleziona il primo oggetto
142         }
143         else                                    //altrimenti
144             selection.width=selection.height=0; //Azzera selezione
145      }
146 }
```

A.2.8 servodriver.h

Listing A.11: servocamera/src/servodriver.h

```
1  /* servodriver.h */
2  // Driver basso livello per la comunicazione seriale coi servomotori
3  
4  //Librerie varie
5  #include <sys/types.h>
6  #include <sys/stat.h>
7  #include <time.h>
8  #include <fcntl.h>
9  #include <termios.h>
10 #include <unistd.h>         //Include per il g++, al posto di stdio.h
11 #include <errno.h>
12 
13 //Separa canali di comunicazione con servomotori (SERVO) e scheda AirBoard (MOTORS)
14 #define SERVO           1
15 #define MOTORS          2
16 
17 #define BAUDRATE        B115200             //Baudrate
18 #define SERVODEVICE     "/dev/ttyUSB0"      //Device per i servomotori
19 #define MOTORDEVICE     "/dev/ttyUSB1"      //Device per i motori del robot
20 #define STR_LEN         9                   //Lunghezza comandi per i servomotori ("PxxxTxxx." con
       xxx angoli in gradi)
21 
22 struct termios oldtio,newtio;               //Strutture per la configurazione della seriale
23 
24 int initSerialCam(int device);              //Inizializza la seriale/USB sulla MODEMDEVICE
25 void closeSerialCam();                      //Chiudi la seriale/USB
26 int move(int fd, int pan, int tilt);        //Manda il comando ai servomotori
27 char receiveEcho(int fd);                   //Riceve primo carattere risposta (Non utilizzato)
28 int send(int fd, char *comando);           //Funzione di invio comando generica
29 int receive(int fd, char *risposta);        //Funzione di ricezione comando generica
30 
31 //inizializza seriale/usb, restituisce il file descriptor
32 int initSerialCam(int device)
33 {
34     int fd;     //Filedescriptor per la seriale/usb
35 
36     if (device == SERVO)    //Inizializza device per Servomotori
37     {
38         fd = open(SERVODEVICE, O_RDWR | O_NOCTTY );         //Apre la seriale/usb
39         if (fd <0) {perror(SERVODEVICE); return(-1); }      //Controllo errori se non pu\'o essere
               aperta
40 
41         tcgetattr(fd,&oldtio);          //Salva le impostazioni della seriale/usb correnti
42         cfsetispeed(&newtio, BAUDRATE);     //Aggiorna baudrate di input
```

```
43          cfsetospeed(&newtio, BAUDRATE);     //Aggiorna baudrate di output
44          newtio.c_cflag |= (CLOCAL | CREAD); //Aggiorna altre impostazioni della seriale/usb
45  //          newtio.c_cc[VMIN]=0;
46          tcsetattr(fd,TCSANOW,&newtio);  //Attiva nuova configurazione
47
48          return fd;                      //Restituisce file descriptor attivo
49      }
50      else if (device == MOTORS)  //Inizializza device per motori del robot
51      {
52          fd = open(MOTORDEVICE, O_RDWR | O_NOCTTY );     //Apre la seriale/usb
53          if (fd <0) {perror(MOTORDEVICE); return(-1); }  //Controllo errori se non pu\'o essere
                aperta
54
55          tcgetattr(fd,&oldtio);          //Salva le impostazioni della seriale/usb correnti
56          cfsetispeed(&newtio, BAUDRATE);     //Aggiorna baudrate di input
57          cfsetospeed(&newtio, BAUDRATE);     //Aggiorna baudrate di output
58          newtio.c_cflag |= (CLOCAL | CREAD); //Aggiorna altre impostazioni della seriale/usb
59  //          newtio.c_cc[VMIN]=0;
60          tcsetattr(fd,TCSANOW,&newtio);  //Attiva nuova configurazione
61
62          return fd;                      //Restituisce file descriptor attivo
63          }
64  }
65
66  //Chiudi porta seriale;
67  void closeSerialCam(int fd)
68  {
69      tcsetattr(fd,TCSANOW,&oldtio);      //Ripristina vecchia configurazione della seriale;
70  }
71
72  //Muove i motori (Non bloccante)
73  int move(int fd, int pan, int tilt)
74  {
75      int size_comm;              //lunghezza comando inviato;
76      char comando[STR_LEN];      //stringhe per comando ai servomotori
77      struct timespec delay;      //Struttura per il delay tra un invio e l'altro
78
79      sprintf(comando,"S%c%c%c",pan,tilt,200);    //Costruisce comando da inviare
80
81      delay.tv_sec=0;     //Configura i secondi di ritardo
82      delay.tv_nsec=1;    //Configura i nanosecondi di ritardo
83
84      size_comm=write(fd,comando,strlen(comando));    //Spedisce il comando
85      return size_comm;   //Restituisce i caratteri effettivamente inviati
86  }
87
88  //Invia comando generico
89  int send(int fd, char *comando)
90  {
91      int size_comm;      //lunghezza comando inviato
92
93      size_comm = write(fd,comando,strlen(comando));  //Invia comando e salva i caratteri
            effettivamente inviati
94
95      if (size_comm<=0)   //Se non ha inviato alcun carattere
96      {
97          fputs("Cannot write to serial\n",stderr);   //Segnala errore se non ha inviato nulla
98      }
99      return size_comm;   //Restituisci i caratteri effettivamente inviati}
100 }
101
102 //Ricevi comando generico
103 int receive(int fd, char *risposta)
104 {
105     int lunghezza;
106     lunghezza = read(fd,risposta,strlen(risposta));     //Leggi risposta
107     risposta[lunghezza]=0;                              //Termina la risposta con uno 0 (per
            chiudere la stringa)
108     return lunghezza;                                   //Restituisci il numero di caratteri
```

```
        effettivamente inviati
}

//Riceve l'echo del primo carattere della risposta
char receiveEcho(int fd)
{
    int size_resp;
    char risposta[255];

    size_resp = read(fd,risposta,255);
    risposta[size_resp]=0;
    return risposta[0];
}
```

A.2.9 trackcontrol.h

Listing A.12: servocamera/src/trackcontrol.h

```
/* trackcontrol.h */
// Funzioni di utilit\'a per definizione, costruzione, utilizzo, dei vari controllori per i
    servomotori e motori robot

//Librerie generali
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <assert.h>
#include <math.h>
#include <float.h>
#include <limits.h>
#include <time.h>
#include <ctype.h>

//Struttura del controllore
typedef struct t_control
{
    float input_min;
    float input_max;
    float output_min;
    float output_max;
    float center;
    float amplitude;
} control;

//Funzione di costruzione controllore
void costruisci_controllore(control* struttura_controllore, float input_min, float input_max,
    float output_min,float output_max, float center,float amplitude)
{
    struttura_controllore->input_min=input_min;
    struttura_controllore->input_max=input_max;
    struttura_controllore->output_min=output_min;
    struttura_controllore->output_max=output_max;
    struttura_controllore->center=center;
    struttura_controllore->amplitude=amplitude;
}

float y_retta_per_2_punti_data_x(float x, float x_a, float y_a, float x_b,float y_b)
{
    float y;
    y=(((y_b-y_a)/(x_b-x_a))*(x-x_a))+y_a;
    return y;
}

float polinomio_lagendre(float p, float x_a, float y_a, float x_b,float y_b,float x_c,float y_c)
```

```
45  {
46      float lagendre;
47      lagendre=(y_a*(p-x_b)*(p-x_c)/(x_a-x_b)/(x_a-x_c))+(y_b*(p-x_a)*(p-x_c)/(x_b-x_a)/(x_b-x_c))+(
            y_c+(p-x_a)*(p-x_b)/(x_c-x_a)/(x_c-x_b));
48      return lagendre;
49  }
50
51  float y_parabola_per_3_punti_data_x(float x, float x_a, float y_a, float x_b,float y_b,float x_c,
        float y_c)
52  {
53      float y,p,a,b,c;
54      c=polinomio_lagendre(0,x_a,y_a,x_b,y_b,x_c,y_c);
55      b=((polinomio_lagendre(1,x_a,y_a,x_b,y_b,x_c,y_c))-(polinomio_lagendre(-1,x_a,y_a,x_b,y_b,x_c,
            y_c)))/2;
56      a=((polinomio_lagendre(1,x_a,y_a,x_b,y_b,x_c,y_c))+(polinomio_lagendre(-1,x_a,y_a,x_b,y_b,x_c,
            y_c))-2*c)/2;
57      y=(a*(x*x))+(b*x)+c;
58      return y;
59  }
60
61  float solver2(control* ctrl,float input)
62  {
63      float output=0;
64      float inizio_tratto_orizzontale=ctrl->center-(0.5*ctrl->amplitude);
65      float fine_tratto_orizzontale=ctrl->center+(0.5*ctrl->amplitude);
66      if ((input>=ctrl->input_min)&&(input<inizio_tratto_orizzontale))
67      {
68          output=-(y_parabola_per_3_punti_data_x(input, ctrl->input_min, -(ctrl->output_min), ctrl->
                input_max, ctrl->output_max, inizio_tratto_orizzontale, 0));
69      }
70      else if ((input>=inizio_tratto_orizzontale)&&(input<fine_tratto_orizzontale))
71      {
72          output=0;
73      }
74      else if ((input>=inizio_tratto_orizzontale)&&(input<=ctrl->input_max))
75      {
76          output=y_parabola_per_3_punti_data_x(input, ctrl->input_min, -(ctrl->output_min), ctrl->
                input_max, ctrl->output_max, inizio_tratto_orizzontale, 0);
77      }
78      return output;
79  }
80
81  float solver1(control* ctrl ,float input )
82  {
83      float output=0;
84      float inizio_tratto_orizzontale=ctrl->center-(0.5*ctrl->amplitude);
85      float fine_tratto_orizzontale=ctrl->center+(0.5*ctrl->amplitude);
86
87      //primo tratto crescente del controllore
88      if ((input>=ctrl->input_min)&&(input<inizio_tratto_orizzontale))
89      {
90          output=y_retta_per_2_punti_data_x(input,ctrl->input_min ,ctrl->output_min,
                inizio_tratto_orizzontale,0);
91      }
92      else if ((input>=inizio_tratto_orizzontale)&&(input<fine_tratto_orizzontale))
93      {
94          output=0;
95      }
96      else if ((input>=inizio_tratto_orizzontale)&&(input<=ctrl->input_max))
97      {
98          output=y_retta_per_2_punti_data_x(input,fine_tratto_orizzontale,0,ctrl->input_max,ctrl->
                output_max);
99      }
100
101     return output;
102 }
103
104 #ifdef TEST
105 int main(void )
```

```
106 {
107     float x_in,x_out,y_in,y_out;
108     control x_par_ctrl;
109     costruisci_controllore(&x_par_ctrl,0,640,-15,15,320,50);
110 
111     for(x_in=0; x_in<=640; x_in++)
112     {
113         //test per il controllore PARABOLICO delle x
114         x_out=solver2(&x_par_ctrl,x_in);
115         printf( "\n X:\t IN:%f\t OUT:%f",x_in,x_out );
116     }
117 
118     control y_par_ctrl;
119     costruisci_controllore(&y_par_ctrl,0,480,-10,10,240,50);
120 
121     for(y_in=0; y_in<=480; y_in++)
122     {
123         y_out=solver2(&y_par_ctrl,y_in);
124         printf( "\n Y:\t IN:%f\t OUT:%f",y_in,y_out );
125     }
126     return 0;
127 }
128 #endif
```

Appendice B

Datasheet di riferimento

Eventuali Datasheet di riferimento.

B.1 Specifiche della telecamera

telecamera 1/2

telecamera 2/2

B.2 Sensore ad ultrasuoni

ultrasuoni 1/2

ultrasuoni 2/2

B.3 Sensori ad infrarosso

infrarosso 1/4

infrarosso 2/4

infrarosso 3/4

infrarosso 4/4

B.4 Specifiche servomotori

servomotori 1/1

servomotori bianca per riempire il foglio