UNIVERSITÀ DEGLI STUDI DI PADOVA
FACOLTÀ DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
CORSO DI LAUREA SPECIALISTICA IN INGEGNERIA ELETTRONICA
CERTIFICAZIONE E QUALITÀ DEI SISTEMI ELETTRONICI

Sviluppo di algoritmi per la correzione real time degli errori *distraction*, *occlusion* e *fusion* in stereofotogrammetria

RELATORE: Ch. mo Prof. Cobelli Claudio
CORRELATORI: Ing. Crovato Digo
Sawacha Zimi, PhD

STUDENTE: Foletto Federico

ANNO ACCADEMICO 2009/2010

Indice

A tutti coloro che hanno contribuito alla mia educazione.

L'autore

To all those who contributed to my education.

Author

Summary

This project was developed at BTS company together with the Laboratory of Movement Bioengineering of the department of Information Engineering of University of Padua. BTS is a company leader in the biomedical sector in 4 continents for production of optoelectronic systems. The purpose of this project is to implement an algorithm which improves identification of marker by stereophotogrammetry in case distraction, occlusion and fusion occure. The algorithm should be able to efficiently recognize markers in real time, and enhance the process of image segmentation during the acquisition of mage. This is important also to emprove both the tracking and data association functions.

The first part of the final project was focused on analyzing for available markers identification in the literature. Whereas the second part was dedicated to the development an algorithm that allowed to recognise markers from stereophotogrammetric images in real time.

The abovementioned algorithm should be able to minimize the computational time. So far it was be exploited data compression to perform the data compression on board of the cameras, thus gaining useful cycles to be use with alternative algorithms inside of a simple blob analysis with thresholding.

This procedure was developed in Matlab environment and it was applied to identify 9400 markers out of 310 images acquired in critical conditions in terms of markers identification. The experimental part of the project consisted in acquiring images where distraction, occlusion and fusion

of markers may occur and this tesi developed algorithm on these images. In order to manage both the images and the algorithms was developed with a graphical user interface witch allows a simple analysis of results.

The proposed algorithm was able to distinguish first of all blob with compatible area with marker, and to discard. The others then blobs were divided into small area blobs and medium-sized area blob: small area blobs are free information so are recognised as marker without further analysis, medium-sized area blob are parameterized to obtain form factors, area factors, main axis and profile. Finally some specific parameters are determined which allow ther algorithm to associate a blob to a marker.

Introduzione

L'attività di tesi è stata svolta presso l'azienda BTS s.p.a. in collaborazione con il Laboratorio di Bioingegneria del Movimento del Dipartimento di Ingegneria dell'Informazione dell'Università degli Studi di Padova. BTS è un'azienda leader nel settore biomedico in 4 continenti per la produzione di sistemi optoelettronici. Lo scopo del progetto è implementare un algoritmo in grado di consentire la corretta identificazione dei *marker* tramite stereofotogrammetria in particolari condizioni di occlusione, fusione o falsi *marker* (distraction, occlusion e fusion). Il suddetto algoritmo deve essere in grado di effettuare il corretto riconoscimento dei *marker* in real time per permettere di migliorare il processo di segmentazione dell'immagine durante l'acquisizione dell'immagine stessa. Questo al fine anche di favorire le funzioni di *tracking* e data association successive.

La prima parte del lavoro di tesi è stata dedicata all'analisi delle tecniche per l'identificazione dei *marker* presenti in letteratura. Mentre la seconda parte è stata dedicata allo sviluppo di un algoritmo che permettesse di riconoscere i *marker* nelle immagini stereofotogrammetriche in condizioni di real time. Il suddetto algoritmo doveva essere inoltre in grado di minimizzare le operazioni di calcolo. A tal fine si è sfruttata la compressione dei dati a bordo delle telecamere guadagnando così cicli utili per l'uso di algoritmi alternativi rispetto alla semplice blob analysis con sogliatura.

Tale procedura è stata quindi sviluppata in ambiente MatLab elaborando 310 immagini campione con circa 9400 *marker* che rappresentano le condizioni di identificazione critiche. Al fine di ottenere dei dati sperimen-

tali si sono ricreate in laboratorio le condizioni più critiche di acquisizione e successivamente sono state scelte immagini campione sulle quali testare l'algoritmo sviluppato. La gestione delle immagini e degli algoritmi è stata semplificata dallo sviluppo di un'interfaccia grafica che permettesse anche una più semplice analisi dei risultati.

L'algoritmo proposto ha permesso di distinguere inizialmente i blob con area compatibile con quella di un *marker*, e di fare in modo che i restanti venissero scartati. Successivamente i blob vengono a loro volta distinti in blob di piccole aree e medio-grandi aree: i blob con piccole aree essendo privi di informazione vengono riconosciuti come *marker* senza ulteriori analisi, i blob con dimensioni sufficienti vengono parametrizzati, ricavando così fattori di forma, fattori di area, asse principale e profilo. Dai parametri ricavati infine è possibile dedurre se il blob è un *marker*.

Capitolo 1

Azienda e prodotti BTS

1.1 Presentazione aziendale

BTS S.p.A. nasce nel 1986 in seguito ad uno spin-off del Centro di Bioingegneria della Fondazione Don Gnocchi e del Politecnico di Milano con lo scopo di sfruttare industrialmente le innovazioni metodologiche per l'analisi del movimento sviluppate dai ricercatori del Centro di Bioingegneria.

Nel 2000 nasce a Padova un'azienda chiamata eMotion, il primo spin-off dell'Università di Padova. Anch'essa si occupa di innovazioni tecnologiche per l'analisi del movimento.

Rispettivamente nel 1999 e nel 2000 BTS ed eMotion vengono acquistate da TC Sistema S.p.A. . Nel 2004 la società viene acquistata da 12 dipendenti con la partecipazione di 3 imprenditori esterni dando origine all'attuale BTS Bioengineering, composta da 28 dipendenti (di cui 21 ingegneri, 11 dei quali dedicati alla ricerca e sviluppo). L'azienda ha sede amministrativa a Milano, mentre a Padova si trova il centro per lo sviluppo dei suoi prodotti.

Oggi BTS fornisce 340 clienti in 40 paesi del Mondo e 10.000 pazienti all'anno vengono assistiti nel loro percorso clinico da sistemi BTS. BTS si è affermata come uno dei leader mondiali delle tecnologie per l'analisi del movimento e mira a diventare il riferimento industriale per le tecnologie

e le applicazioni dell'analisi del movimento nel mondo clinico.

Figura 1.1: Logo di BTS Bioengineering

1.1.1 Prodotti

BTS produce strumentazioni scientifiche innovative per l'analisi del movimento. I suoi sistemi supportano il lavoro di medici, ingegneri e ricercatori che hanno la necessità di ottenere informazioni oggettive ed accurate, utili per i loro scopi.

Le soluzioni BTS si possono raccogliere in cinque gruppi:

- Sistemi integrati per l'analisi clinica del movimento e della postura (SMART-D, SMART-Performance, SMART-Analyzer, SMART-Clinic, SMART-Performance, VIXTA, Sway, Digivec);
- Sistemi di analisi del movimento markerless (BTS NIRVANA);
- Sistemi per l'acquisizione e l'elaborazione di segnali elettromiografici (FreeEMG, PocketEMG, EMG Analyser, EMGenius, Myolab);
- Sistemi optoelettronici per la misura e la ventilazione polmonare (OEP System);
- Sistemi per il controllo degli stati vegetativi (Dream).

Per attuare un costante sviluppo di nuove tecnologie, l'azienda si avvale della collaborazione dei più importanti centri di competenza a livello internazionale e favorisce i contatti e le relazioni tra gli specialisti e gli operatori del settore dell'analisi del movimento. Inoltre BTS è impegnata nel contribuire attivamente al lavoro dei clienti offrendo loro soluzioni, consulenza, tecnologie, idee e servizi, puntando ad un rapporto duraturo basato sulla qualità e sul senso di responsabilità.

1.1.2 Settore di applicazione

Le applicazioni dei prodotti BTS sono molteplici, ma dobbiamo senz'altro sottolineare la grande influenza in settori come ortopedia, riabilitazione, neurologia, pneumologia, medicina sportiva, terapia intensiva, monitoraggio degli stati vegetativi, ergonomia e ricerca. Conseguentemente si possono trovare prodotti BTS in ospedali, centri di cura e cliniche riabilitative dove si ottengono le migliori cure per le disfunzioni del movimento e la reale efficacia dei percorsi di riabilitazione.

Istituti scientifici e universitari usano prodotti BTS per compiere ricerche nei campi della biomeccanica, biometria, diagnostica e scienze motorie. Tale tecnologia è stata anche impiegata nei più importanti programmi spaziali e impiegati da NASA, ESA, CNRS, nei programmi di volo parabolico e per una serie di esperimenti dell'International Space Station Program. Inoltre i sistemi BTS sono stati istallati sulle stazioni Spacelab e MIR.

Numerose organizzazioni di medicina sportiva usano prodotti BTS con l'obiettivo di massimizzare le prestazioni degli atleti, prevenire infortuni e migliorare l'efficacia della riabilitazione post-traumatica. Sottolineiamo l'uso di tali apparecchiature da parte del A.C. Milan, per le valutazioni di routine dei loro giocatori, e nella riabilitazione di alcuni componenti della squadra nazionale olimpica italiana.

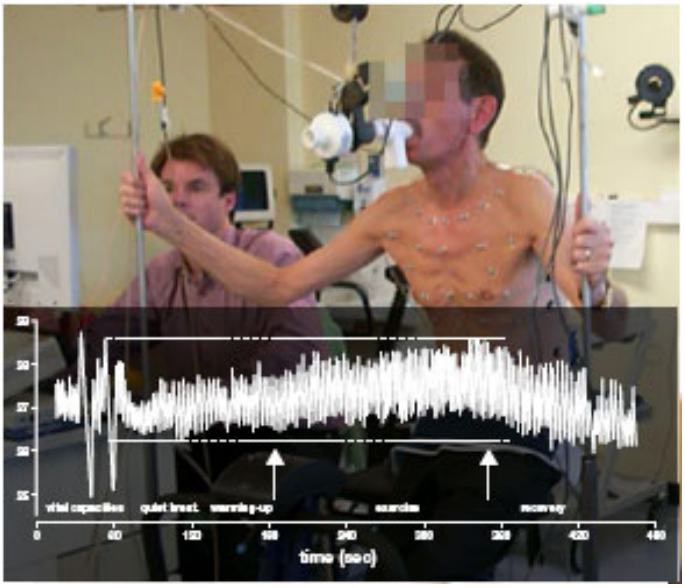

Figura 1.2: Esempio di un Sistema OEP

1.2 Sistema di acquisizione SMART

Tramite i sistemi optoelettronici otteniamo le coordinate tridimensionali di *marker* apposti sul corpo del soggetto, nello studio di analisi del movimento i *marker* vengono posti su punti di repere anatomici. I *marker* possono essere di due categorie: *marker* passivi (piccole sfere riflettenti) e *marker* attivi (piccoli emettitori luminosi). Il sistema SMART usa *marker* passivi sferici o semisferici con diametri che vanno dai 3 mm ai 20 mm. La sfericità dei *marker* garantisce la miglior riflessione dei raggi infrarossi.

I *marker* sono identificati dal sistema stereofotogrammetrico riflettendo la luce infrarossa emessa dagli emettitori montati nei pressi delle telecamere. Il sistema rilevando le coordinate tridimensionali dei *marker* permette di ricavare le traiettorie, le grandezze angolari, le velocità ed l'accelerazione, la cinematica del movimento del segmento corporeo sul quale sono stati posti i *marker*.

I sistemi optoelettronici, come lo SMART, sono composti da una parte hardware (quali telecamere, hub, schede PCI) e una software (driver, soft-

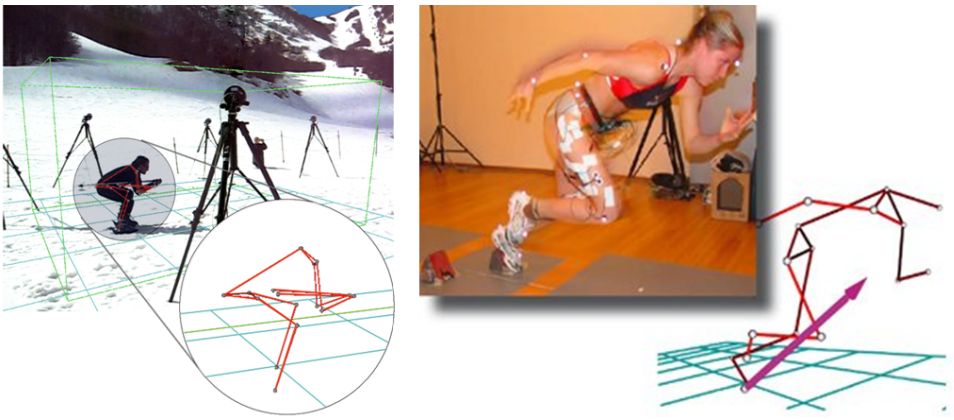

Figura 1.3: Esempi di acquisizioni con sistemi BTS

Figura 1.4: A sinistra un *marker* BTS, a destra un *marker* classico

ware di acquizione e analisi dati). Uno dei sistemi optoelettronici forniti da BTS è lo SMART-D, tale sistema è composto da

- Telecamere con emettitore e filtro IR
- Workstation per acquisizione
- SMART capture
- SMART tracker
- SMART analyzer

Con esso possono essere integrati elementi quali pedane di forza, pedane di pressione, accelerometri, sonde EMG e segnali video a colori. Uno

Figura 1.5: Sistema di acquisizione Smart-D sulla sinistra e un sistema EMG wireless sulla destra

dei vantaggi di un sistema di questo tipo è la flessibilità; infatti possono variare di volta in volta la posizione delle telecamere, le condizioni di acquisizione e di luminosità in ambiente controllato, all'aperto, in palestra, in piscina o in ambienti spaziali.

Un altro punto di forza del sistema SMART è la possibilità di essere montato e smontato facimente favorendone il trasporto.

1.2.1 Hardware di acquisizione SMART

Il sistema SMART-D, dal punto di vista hardware, è costituito dalle seguenti parti:

- **Videocamere**: le videocamere usate dalla BTS hanno tipicamente risoluzione video pari a 800x600 e una frequenza di lavoro che può andare da 60 a 250 Hz. Tali videocamere sono provviste di filtri IR in modo da filtrare la restante componente luminosa. Comunemente le videocamere usano tecnologia CCD o CMOS:

  – Sensori CCD (Charge Coupled Device) sono composti da una matrice di consensatori Mos. Nei Mos viene accumulata una carica proporzionalmente al numero di fotoni incidenti, successivamente la carica accumulata viene portata ad un circuito in let-

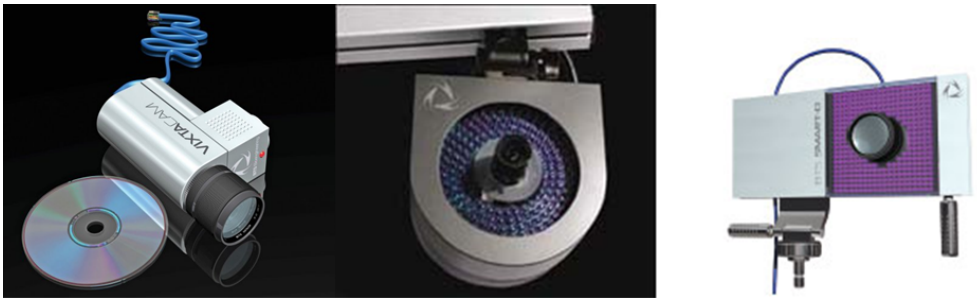

Figura 1.6: Alcune telecamere BTS

tura e convertita con un segnale analogico. Selezionato il pixel di interesse il segnale acquisito verrà portato ad un convertitore analogico digitale per la conversione in una stringa di bit.

– Sensori CMOS (Complementary Metal Oxide Semiconductor) hanno una struttura simile ai sensori CCD, ma si discostano da questi in quanto ciascun pixel è collegato ad un convertitore analogico-digitale, che permette di fornire direttamente in uscita un segnale di tipo digitale. Questo tipo di dispositivi hanno il vantaggio di richiedere una circuiteria esterna molto più ridotta ma lo svantaggio ridurre l'area sensibile del sensore.

Le videocamere BTS usano la tecnologia CCD poichè presenta un rapporto segnale / rumore superiore ai sensori CMOS e maggiore sensitività per ridurre il tempo di shutter sotto i $100\mu s$. I sensori CMOS obbligano infatti a ricorrere ad uno shutter non uniforme (rolling shutter) a meno di ridurre il numero di pixel del sensore utilizzabili, che si traduce in un aumento della distorsione dell'immagine per gli oggetti in movimento. Il segnale in uscita non sarà a colori ma a 256 livelli di grigio.

- **Illuminatori** : posteriormente al piano focale di ogni telecamera viene montato un illuminatore infrarossi in modo che la radiazione emessa non disturbi l'acquisizione della videocamera su cui è montata. Tali illuminatori emettono impulsi di luce infrarossa di

lunghezza d'onda $880nm$, tipicamente intorno a μs, ad alta potenza controllati digitalmente. L'illuminazione artificiale può essere modificata per compensare l'illuminazione infrarossa già presente nell'ambiente, tipo l'illuminazione solare.

- **Lenti**: le diverse videocamere permettono di montare dei set di lenti C-mount compatibili da 4.5mm a 8mm che permettono di lavorare con volumi e in ambienti diversi. Tipicamente viene fornito un set di sei lenti e zoom da 6-12mm e 25mm. La messa a fuoco ed il diaframma vengono regolati manualmente.

- **Workstation**: se i primi sistemi SMART prevedevano l'uso di più hub, attualmente tutto è integrato nella workstation. L'integrazione in un solo workstation garantisce solidità e affidabilità del sistema, vengono infatti eliminati molti cavi di interconnessione usando semplici cavi GIGABIT Ethernet e alimentazione. La workstation prevede l'acquisizione video fino 16 videocamere con definizione di 1.4 Mp e frequenza massima di 500Hz , sincronizzabili con con un massimo di 80 canali analogici [1] per il collegamento con: elettromiografi, piattaforme di forza, treadmill sensorizzati e cicloergometri, dispositivi aptici o qualsivoglia dispositivo interfacciabile. La workstation si interfaccia all'utente come un personal computer provvisto di scheda di acquisizione e conversione analogico-digitale dei segnali acquisiti e del software per la gestione del sistema. SMART-D usa un architettura Intel XEON® e PCI-X a banda larga.

1.2.2 Capture

Questo software permette la gestione completa e real time della calibrazione e dell'acquisizione dei dati cinematici e dinamici. All'avvio del programma vengono eseguiti una serie di diagnostiche di sistema affinchè tutte le telecamere, elettromiografi o sistemi analogici siano correttamente

[1] Frequenza di acquisizione massima per canale 1kHz, risoluzione A/D 16bit.

impostati e collegati all'hub o workstation. Una volta avviato si possono visualizzare in real time sia i segnali analogici acquisiti che le immagini video delle singole telecamere. Tale visione permette di gestire la sensibilità delle telecamere che acquisiscono immagini IR a 256 livelli di grigio.

Siccome prima di ogni serie di acquisizioni, o comunque obbligatoriamente ogni volta vengono cambiati parametri di acquisizione (spostate videocamere, cambiate ottiche, tolte o aggiunte videocamere), è indispensabile eseguire la procedura di calibrazione.

BTS ha sviluppato un metodo di calibrazione flessibile, veloce e preciso: THOR. THOR sfrutta una terna di riferimento destrorsa dove uno dei suoi assi è estraibile, e usa i parametri di linearizzazione delle telecamere [2] e posizione delle piattaforme di forza rispetto al sistema di riferimento del laboratorio.

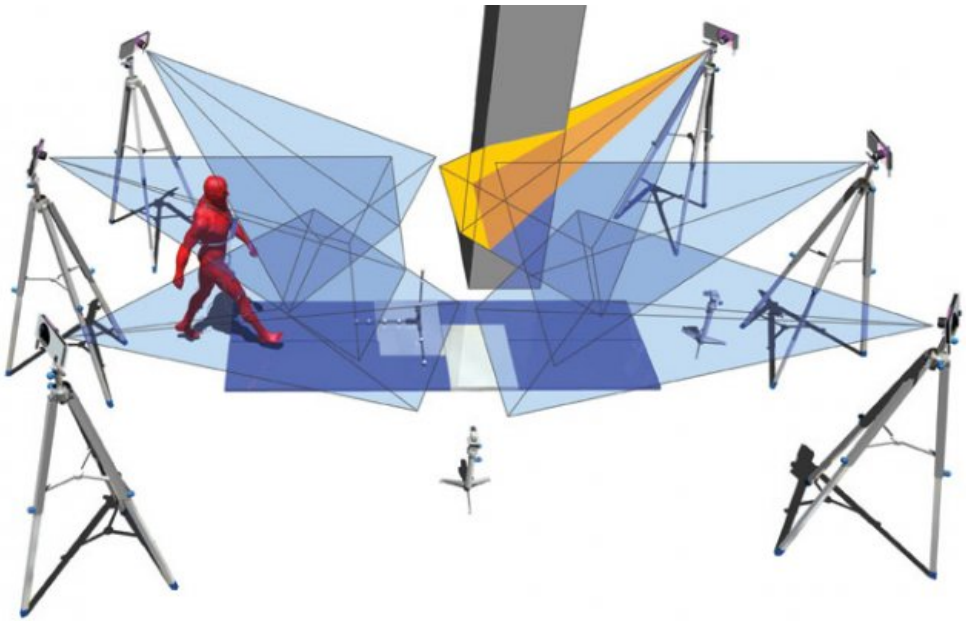

Figura 1.7: Configurazione tipica di un sistema di stereofotogrammetrico

La procedura di calibrazione si divide in quattro passi fasi:

[2]Tali dati sono matrici particolarmente importanti che permettono di gestire la deformazione ottica introdotta da diverse lenti nelle immagini.

- Sequenza Axes: si fissa la posizione della terna di riferimento del laboratorio;

- Eventuale Force Plate Position: in cui si fissa la posizione delle piattaforme di forza e pressione rispetto la terna di riferimento;

- Sequenza Wand: si definisce lo spazio fisico dove si effettuano le misurazioni di interesse, questo viene definito come volume di lavoro;

- Valutazione dei risultati della calibrazione: si visualizza il risultato numerico delle singole telecamere e complessivo del algoritmo di calibrazione; se i parametri statistici dell'errore di ri-proiezione sul piano immagine di ciascuna telecamera risultano elevati è consigliabile ripetere la calibrazione e valutare eventuali cause di tale errore.

Dopo la calibrazione il sistema è pronto per acquisire dati, e tutto ciò al di fuori del volume di calibrazione sarà considerato fonte di errore perciò non considerato.

Le singole acquisizioni vengono salvate in file con estensione TDF, in ogni file sono presenti la totalità dei dati cinematici e dinamici.

1.2.3 Tracker

È un ambiente grafico che permette la ricostruzione tridimensionale dei dati utilizzando i dati bidimensionali acquisiti dalle videocamere e quelli provenienti dalla calibrazione. Ad ogni *marker* viene assegnata una traiettoria (*tracking*) e successivamente è possibile assegnare alla traiettoria un nome (labelling).

Il *tracking* ricostruisce la traiettoria del *marker* collegando la posizione del *marker* in due frame successivi; tale processo si basa su algoritmi che usano stimatori ricorsivi dello stato dei sistemi dinamici (come il filtro di Kalman). L'operazione di labelling rappresenta la seconda fase di elaborazione dei dati e tale operazione viene eseguita manualmente per proto-

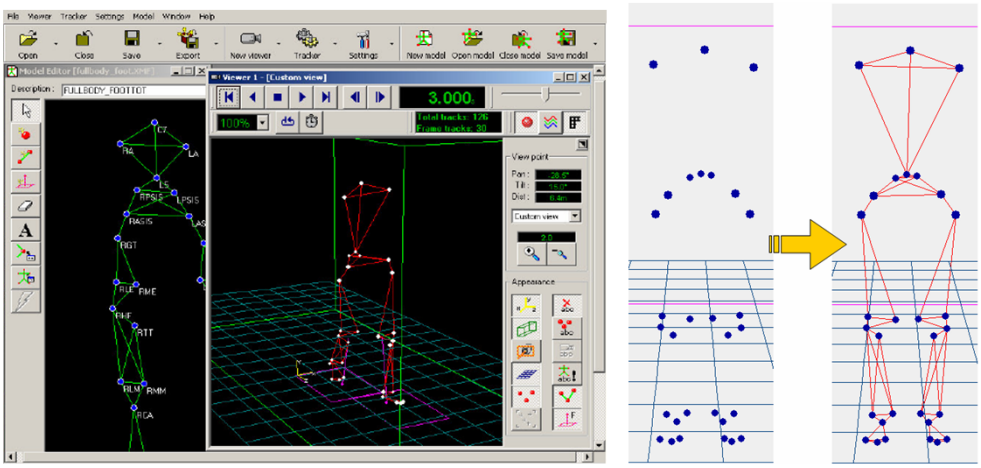

Figura 1.8: Videata di SMART tracker con un esempio di labelling sulla destra

colli complessi, mentre per protocolli più semplici può essere eseguita da qualche applet in modo automatico.

Può accadere che alcuni *marker* scompaiono o non vengono rilevati dalle videocamere, questo fenomeno è una delle maggiori cause di errori nella ricostruzione delle traiettorie. Risolvendo a monte il problema dell'individuazione di *marker* fusi o distorti si risolvono conseguentemente problemi di *tracking* e *labelling*.

1.2.4 Analyzer

Questo software che consente di eseguire un'analisi biomeccanica e l'elaborazione dei dati di cinematica e dinamici. Si sottolineano alcune operazioni usate di frequente nello SMART analyzer, tutte operazioni che avvengono in ambiente grafico intuitivo:

- creazione di protocolli di analisi;
- l'importazione o esportazione dei dati acquisiti, in particolare si sottolinea la facile elaborazione dei file acquisiti anche in Matlab usando delle toolbox fornite da BTS;

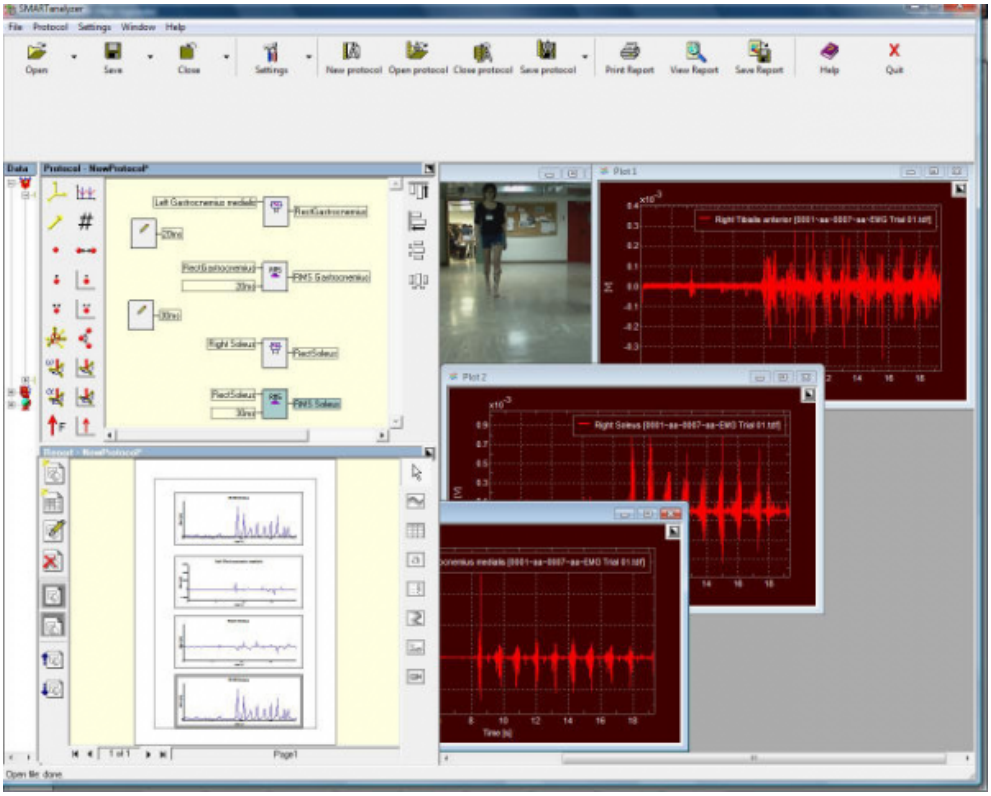

Figura 1.9: Una videata di Smart analyzer

- visualizzazione 3D dell'acquisizione con il sistema di riferimento della calibrazione;

- calcolo di tutti i parametri necessari all'utente ;

- creazione di report clinici in cui vengono riassunti i dati dell'acquisizione tramite grafici, tabelle o testo, tale funzione semplifica il lavoro in ambiente clinico per la consegna del referto al medico o paziente.

L'analisi risulta essere oltre che semplice anche veloce sfruttando le librerie IPP della Intel per l'elaborazione numerica dei segnali.

1.3 Un'azienda che cresce

La continua evoluzione dei sistemi stereofotogrammetrici avviene in diverse forme. Negli ultimi anni BTS ha puntato sulla compattezza, precisione ed affidabilità del proprio sistema. La compattezza è stata migliorata eliminando cablaggi nel sistema, tipo l'uso dell'alimentazione Power Over Ethernet ed eliminando gli hub che vengono integrati nella workstation. La precisione del sistema è stata migliorata aumentando le prestazioni delle singole telecamere, sia in definizione che frequenza, aumentando le capacità di calcolo del sistema. L'affidabilità del sistema è migliorata sia attraverso algoritmi come quelli presentati in questa tesi che attraverso upgrate che possono risolvere alcuni bug nel software o aggiungere nuove funzioni.

Attualmente in ambito biomedico solo i sistemi stereofotogrammetrici garantiscono quell'affidabilità e precisione necessaria per la cura delle persone. BTS si sta impegnando nella ricerca di nuove tecnologie ma anche nel miglioramento dell'attuale stereofotogrammetria infrarossa, permettendo di formire ai suoi clienti prodotti sempre aggiornati e al livello di eccellenza.

Capitolo 2

Identificazione delle traiettorie di marker tramite stereofotogrammetria

In questo capitolo viene illustrata l'architettura di sistemi stereofotogrammetrici infrarossi a marcatori passivi mettendo in evidenza le trasformazioni ed elaborazioni logiche a cui sono sottoposti i dati. Un sistema optoelettronico multi camera [1] prevede una struttura di acquisizione hardware ed una infrastruttura software di elaborazione. Come abbiamo già visto nel capitolo precedente, anche i sistemi BTS mantengono una struttura di acquisizione composta di telecamere, illuminatori, workstation (dove sono presenti schede di acquisizione ed elaborazioni dei segnali) e un software, in tal caso lo SMART capture, che gestisce l'acquisizione dei dati.

2.1 Acquisizione dell'immagine

L'acquisizione fisica del segnale luminoso si basa su tecnologia CCD, che è già stato descritto nel paragrafo 1.2.1. Aumentando il numero di pixel per unità di area del sensore, è possibile migliorare la risoluzione spaziale del sistema, ad esempio se consideriamo un volume di misura cubico di 5

metri, dove il *marker* ha diametro $1,5cm$ e le telecamere una risoluzione di 1000x1000 pixel, teoricamente di area di diametro sul piano dell'immagine risulterà di 3 pixel. Ma uno studio sulla forma del *marker* di 3 pixel diventa poco significativo perché 3 pixel è un set di informazioni molto povero. In realtà non si hanno a disposizione solo 3 pixel, ma 5 o 6 poiché qualche pixel attiguo sarà parzialmente illuminato dal *marker*. Questi pixel di contorno saranno poi utili per ricavarsi la posizione esatta e il profilo del *marker*.

L'immagine acquisita è rappresentata da una matrice dove ogni valore corrisponde all'intensità rilevata da un pixel in posizione (x, y). Per ricostruire le posizioni e le traiettorie 3D sarà necessario costruire un modello matematico del sistema, per costruirlo devono essere note la lunghezza focale, le coordinate del punto principale, i coefficienti di distorsione e la posizione del sistema di riferimento della telecamera rispetto a quello assoluto. Le coordinate dei *marker* si ricavano tramite rototraslazioni del sistema di riferimento a cui sono riferiti i dati a disposizione. Inizialmente si ha a disposizione l'immagine, questa verrà trasformata dal sistema di riferimento del sensore al sistema di riferimento del piano immagine, e a sua volta dal sistema di riferimento del piano immagine ci si riporterà al sistema di riferimento della telecamera.

La calibrazione del sistema definisce il sistema di riferimento assoluto a cui ogni dato di ogni telecamera viene riportato. Ogni telecamera avrà il proprio sistema di riferimento. Al momento della calibrazione il sistema sarà in grado di riportare le matrici rappresentanti l'immagine dal sistema di riferimento della telecamera a quello del sistema assoluto. Queste operazioni vengono eseguite tramite dei semplici prodotti tra matrici, cioè per riportare i dati dalla telecamera al sistema di riferimento assoluto per ogni immagine viene eseguito un prodotto con la matrice definita al momento della calibrazione e del setup del sistema.

Riassumendo, qualunque sia la fonte del segnale luminoso acquisito dalle videocamere, questo viene trasdotto in segnale elettrico che viene

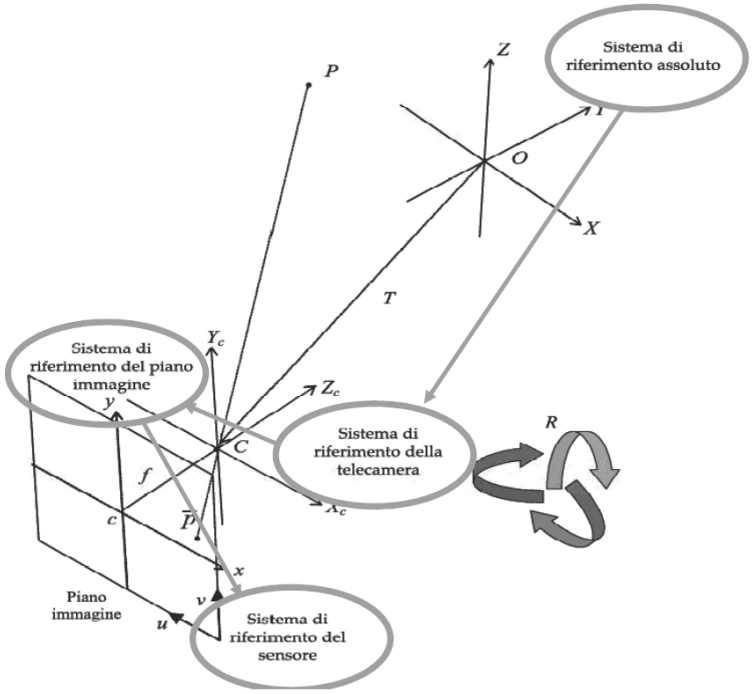

Figura 2.1: Definizione dei sistemi di riferimento e proiezione nel piano immagine di un punto O

inviato ad una scheda di acquisizione presente in un elaboratore, l'elaboratore ricompone l'immagine bidimensionale del frame acquisito e lo relaziona un sistema di riferimento assoluto. Prima di estrapolare dall'immagine le coordinate 3D del *marker* si dovranno trattare le immagini per individuare marcatori e blob.

2.2 Pre-processing: sogliatura

Lo scopo principale delle tecniche di pre-elaborazione non è quello di mantenere quanta più informazione possibile nella descrizione del blob, quanto quello di individuare il minor numero possibile di informazioni necessarie per una corretta classificazione del blob.

Una delle tecniche più comuni ed efficaci per mettere in evidenza in un'immagine un oggetto rispetto allo sfondo è senz'altro quello della so-

gliatura . La sogliatura prevede l'annullamento dei pixel con intensità minore (in alcuni casi maggiore) di un valore di soglia prefissato. Questa tecnica è spesso usata per riportare immagini a livelli di grigio ad immagini binarie, cioè viene posto 1 dove l'intensità del pixel supera il valore di soglia, 0 dove l'intensità del pixel è sotto il valore di soglia.

La sogliatura delle immagini nella sterefotogrammetria ha due funzioni: la prima quella di mettere in evidenza possibili blob, la seconda quella di eliminare dati non utili ai fini della blob analysis. Si vedrà in seguito come la sogliatura può essere ripetuta prima per la compressione di dati, poi per l'individuazione di blob.

2.2.1 Il livello di sogliatura

La sogliatura può essere di due tipi: a soglia globale o a soglia adattiva. La sogliatura globale prevede che l'intera immagine venga filtrata con il medesimo valore di soglia. Può capitare che siano presenti in un'immagine dei riflessi molto ampi, in queste zone perciò sarebbe necessario un valore di sogliatura molto più alto per individuare i *marker*. Se prendessimo il valore più alto di soglia andremmo a perdere delle informazioni nelle zone dove basterebbe un livello di soglia basso. In questi casi entra in gioco la sogliatura adattiva: la sogliatura adattiva prevede la variazione del livello di sogliatura a seconda del livello medio di sfondo rilevato in un'area dell'immagine. Quest'ultima tecnica sarebbe utile in applicazioni IR ma va scartata perché introduce complessità di calcolo che non viene ripagata poi in termini di prestazioni globali. Si preferisce infatti alzare il livello di sogliatura globale e perdere qualche informazione sulla dimensione del *marker*.

Il livello di sogliatura viene individuato attraverso un algoritmo di minimizzazione dell'errore. Se si osserva l'istogramma 2.2-a della frequenza delle tonalità di grigio in un'immagine si distingueranno due gaussiane: la prima molto ampia che rappresenta lo sfondo, la seconda, traslata rispetto alla prima che indica le tonalità dei *marker*. Siccome il problema

consiste nel decidere se un pixel appartiene o meno allo sfondo o ad un *marker*, il problema viene trattato come un test statistico delle ipotesi, dove ogni distribuzione viene approssimata con una gaussiana. L'obbiettivo è minimizzare la probabilità di riconoscere un pixel come sfondo quando invece appartiene ad un *marker* e viceversa. Se poniamo un livello di sogliatura C, tutti pixel con valore inferiore a C sono classificati come sfondo e quelli con valore superiore come appartenenti all'oggetto. In un'immagine dove definiamo la densità dello sfondo come $p_2(z)$ e la densità dell'oggetto come $p_1(z)$, la probabilità di classificare in modo errato un punto dello sfondo sarà l'integrale della sua densità nel intervallo definito come sfondo:

$$E_1(C) = \int_C^{255} p_2(z)dz \tag{2.1}$$

e la probabilità di classificare in modo errato un punto dell'oggetto come sfondo

$$E_2(C) = \int_0^C p_1(z)dz \tag{2.2}$$

Da cui si calcola la probabilità di errore totale

$$E(C) = P_1E_2(C) + P_2E_1(C) \tag{2.3}$$

Per trovare il valore di C che renda minimo l'errore, si differenzia e si pone uguale a 0:

$$P_1p_1(C) = P_2p_2(C) \tag{2.4}$$

Risolvendo rispetto a C si ricava la soluzione,che è $P_1 = P_2$ cioè il punto di intersezione delle due densità. Graficamente la soluzione è intuitiva, infatti si trova il punto di minimo in cui si intersecano le due curve. In figura 2.2-b si osserva il livello di sogliatura globale ottima.

Nelle immagini reali le densità dello sfondo e dei marcatori si comportano differentemente. Come si può osservare in figura 2.3 e 2.4 la densità della sfondo e dei mecatori sono agli estremi della scala di tonalità di grigio. La distribuzione dello sfondo può variare sensibilmente a seconda dei disturbi presenti.

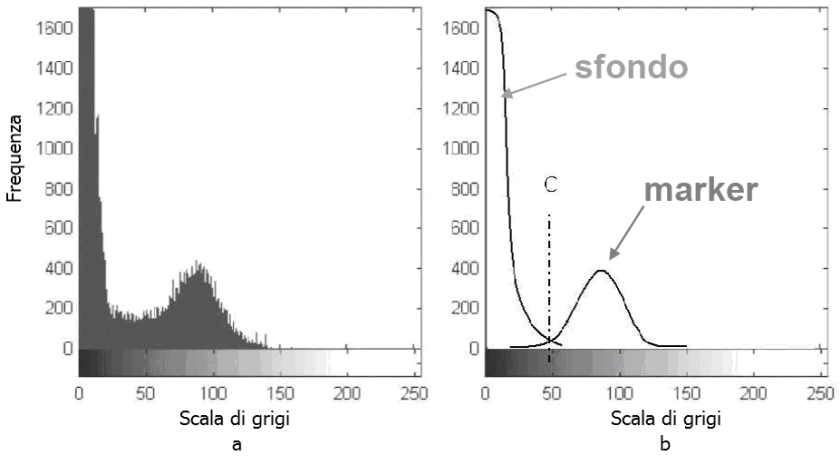

Figura 2.2: A sinistra l'istogramma della frequenza delle tonalità di grigio, a destra la densità ideale dello sfondo e marcatori

La sogliatura è un parametro che può essere deciso dall'utenza, preimpostato dalla casa produttrice o calcolato in fase di calibrazione di volta in volta. Ad esempio nei prodotti BTS è preimpostato nel prodotto però l'utente ha la possibilità di cambiarla di una percentuale ristretta.

2.2.2 Sogliatura e IR

La visione nelle frequenze dell'alto infrarosso ci permette una buona acquisizione dell'immagine proprio perché l'ambiente circostante non riflette ed emette tale componente. In assenza di *marker* l'immagine dovrebbe risultare completamente nera. Come si è visto in realtà non è cosi, anche in ambienti controllati vi possono essere superfici lucide che riflettono l'infrarosso dando luogo a blob estranei all'acquisizione. In figura 2.6 si può osservare il profilo di un riflesso, a differenza di un profilo di un *marker*, o più *marker* come in figura 2.5, si osserva che i riflessi non hanno picchi di gradiente, se non in rari casi. Da questo si può dedurre che una sogliatura ad alti livelli, tipo 200 su 255, potrebbe eliminare anche quei blob parassiti che altrimenti si dovrebbero elaborare. Questa affermazione è

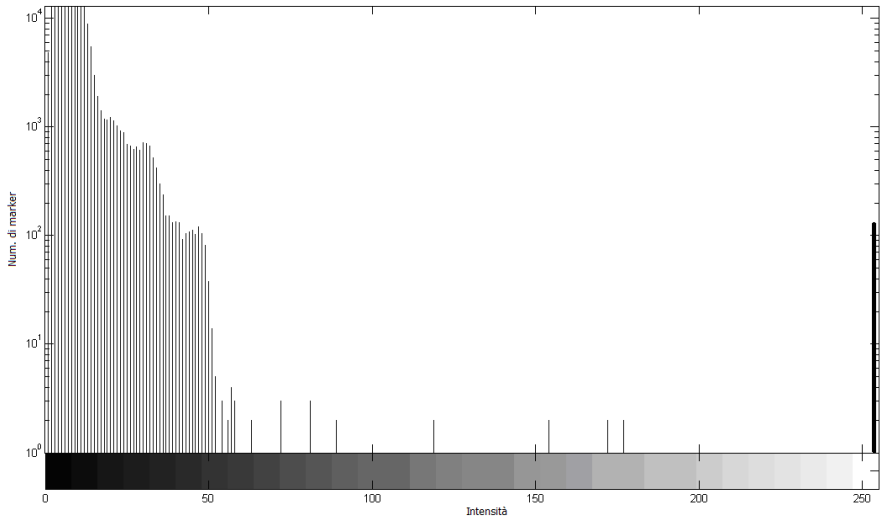

Figura 2.3: Istogramma della frequenza delle tonalità di grigio di un'immagine reale priva di riflessi

esatta ma alzando eccessivamente la sogliatura si ha una perdita eccessiva di informazione, infatti vengono eliminati *marker* di ridotta riflessività e si perdono informazioni sul contorno di *marker* e blob.

2.2.3 Quantizzazione

I primi sistemi stereofogrammetrici non elaboravano immagini a livelli di grigio ma operavano su immagini binarie, dove il valore 1 corrispondeva ad un pixel con intensità oltre il valore di soglia e 0 altrimenti. Con l'introduzione di nuovi protocolli nello studio dell'analisi del movimento, l'uso per la produzione cinematografica e sportiva di sistemi stereofotogrammetrici nasce l'esigenza di aumentare l'affidabilità e risoluzione dei dati acquisiti. Le immagini binarie risultano povere di informazioni per rispondere a queste richieste, si decide perciò di elaborare le immagini a 256 livelli di grigio. Se con le immagini binarie l'analisi della forma era vincolata alla sola posizione dei pixel, in quella a livelli di grigio è possibile valutare la forma di un oggetto anche valutando l'intensità dei pixel.

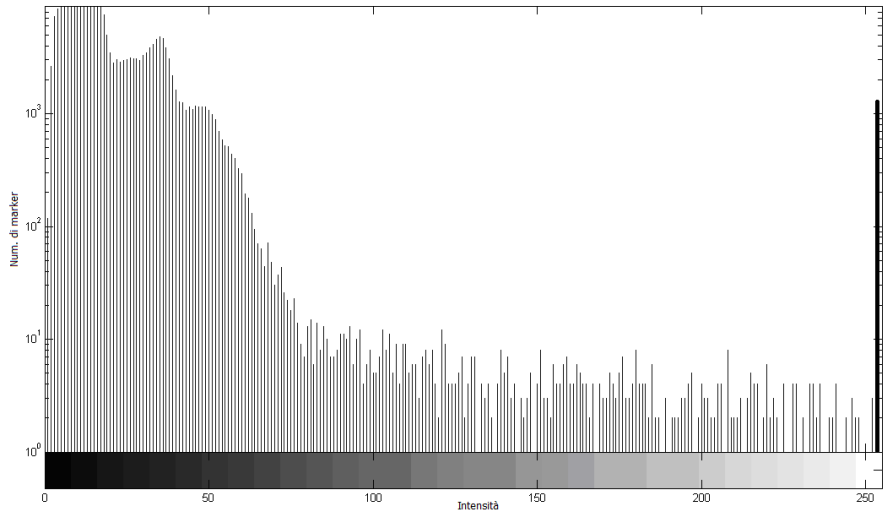

Figura 2.4: Istogramma della frequenza delle tonalità di grigio di un'immagine reale con riflessi

In figura 2.7 si osserva il riconoscimento della forma del *marker* con tecnologia BTS a livelli di grigio (a sinistra) e con sistemi a 1 bit, in grado di distinguere solo il bianco e il nero.

2.3 Compressione lossless

L'esigenza di elaborare i dati in *real time* è un grosso limite che esclude molte delle migliori soluzioni proposte dalla comunità scientifica. Se si ha però la possibilità di comprimere i dati, allora si presenta la possibilità di risparmiare operazioni di elaborazione sul processo di riconoscimento dei blob.

Se si prende in considerazione l'architettura Intel, che è appunto quella usata da BTS, si osserva che [1]:

- Operazione di addizione, sottrazione e comparazione impiegano 4 cicli di clock

[1]Dati indicativi ricavati su processori Intel e AMD degli ultimi 5 anni.

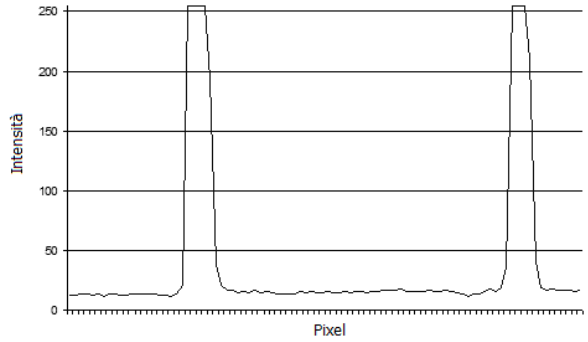

Figura 2.5: Profilo di un immagine dove si mettono in evidenza due marcatori e il livello di saturazione della tonalità di grigio

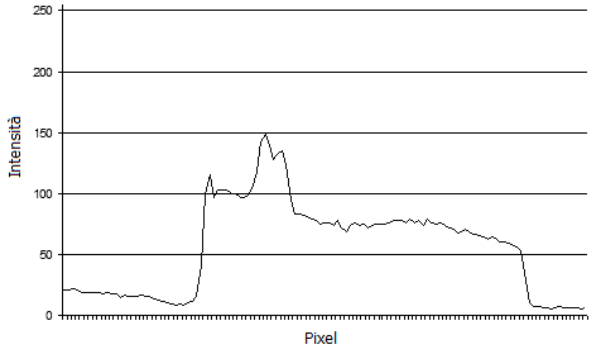

Figura 2.6: Profilo di un immagine dove si mette in evidenza un blob che non rappresenta un marcatore

- Operazione di valore assoluto impiegano 12 cicli di clock
- Operazione di moltiplicazione impiegano 16 cicli di clock
- Operazione di divisione impiegano 40 cicli di clock
- Operazioni trigonometriche impiegano 320 cicli di clock
- Operazione di potenza impiegano 400 cicli di clock

ragion per cui operazioni trigonometriche e di potenza sono considerate proibitive in applicazioni *real time,* mentre operazioni di prodotto e

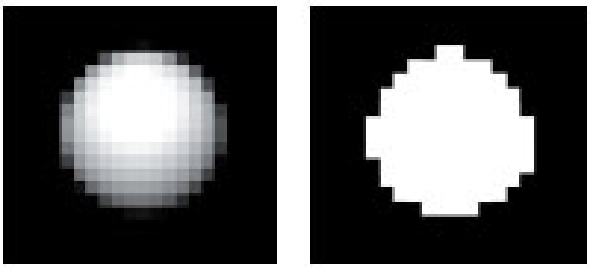

Figura 2.7: A sinistra un blob in cui è mantenuta l'informazione sull'intensità, a destra un blob con quantizzazione binaria

divisioni vanno limitate. Similmente si deve considerare che la cache del processore mantiene una quantità limitata di memoria, perciò va sfruttata al meglio. In ingresso non memorizzeremo più l'intera immagine ma una riga dell'immagine, che va processata velocemente per lasciar spazio alla successiva. Processati i dati, si dovrà tenere in memoria solo l' essenziale per l'elaborazione.

Per gestire l'elaborazione in *real time* si dovrà agire su due fronti: compressione dei dati e scelta del algoritmo di riconoscimento di forma, dove sarà integrato l'algoritmo di segmentazione e classificazione.

Per ridurre la mole di dati da analizzare sarà necessario effettuare qualche tipo di compressione. La compressione non dovrà però degradare le immagini usate per l'individuazione dei *marker*. Usando l'infrarosso in ogni immagine si nota una grande componente di sfondo, che in una scala di 256 livelli di grigio [2] sta tra i valori 0 e 15 comprensivo di rumore; se si ha la possibilità di effettuare una sogliatura preliminare a bordo camera, questi valori di sfondo privi di informazione vengono eliminati. Risulta che mediamente solo il $0,01\%$ [3] dell'area dell'immagine possiede informazione utile che deve essere trasportata ed elaborata alla workstation. Con quest'ultima informazione si deduce che il $99,9\%$ dell'immagine sarà com-

[2] Lo 0 corrisponderà fondo soglia, il nero, e il 255 il valore di saturazione, il bianco.

[3] Valore calcolato da un immagine campione con risoluzione 640x480 con 52 *marker* e 4 blob.

posto da un'intensità di sfondo, cioè il $99,9\%$ avranno il medesimo valore che si ripete.

Utilizzando una compressione lossless, cioè senza perdita, basata sulla compressione di elementi uguali riusciamo a ridurre il tempo di elaborazione trascurando l'analisi di quelle informazioni, tipo lo sfondo, che sarebbero scartate. Un esempio di questo tipo di compressioni sono la Run length encoding, Codifica di Huffman e codifica Lempel Ziv Welch. Qui di seguito viene fatta una presentazione dei tre algoritmi.

2.3.1 Run length encoding

L'algoritmo Run-length encoding (RLE) è storicamente il primo algoritmo di compressione per le immagini. Inizialmente è utilizzato nei fax, attualmente è usato anche in algoritmi integrati in protocolli di rete (tipo IBM SNA o trasmissione di filmati AVI) o per la compressione di immagini bmp. Questo algoritmo parte dalle ipotesi che i dati siano composti di un insieme di elementi finiti e vi siano lunghe sequenze dove lo stesso elemento viene ripetuto.

La forza di questa tecnica sta nella velocità di compressione - decompressione . Per esempio supponiamo di avere un'immagine dove la prima riga è formata da duecento pixel neri, l'algoritmo memorizzerà il valore del primo pixel seguito dal valore duecento: in questo modo avremo rappresentato duecento elementi con due elementi, il tutto senza elaborare dati o mantenere in memoria nulla. Similmente, la decompressione sarà una semplice ripetizione dell'elemento per quelle volte che è ripetuto.

Il primo grosso vantaggio di questa tecnica è che elaborando i dati in *real time* si può cominciare l'elaborazione della prima riga dell'immagine senza attendere che la seconda sia completa. In più la decompressione non è necessaria, infatti i dati compressi con RLE sono talmente manifesti che si possono segmentare risparmiando diversi cicli di confronto. Questa tecnica è molto vantaggiosa per i simboli che si ripetono consecutivamente, perde la sua efficienza per simboli anche se frequenti ma non consecutivi.

In immagini parzialmente sogliate si hanno ampie aree dove si ripeteranno i medesimi valori, la codifica RLE risulterà perciò molto efficiente nella stereofotogrammetria infrarossa.

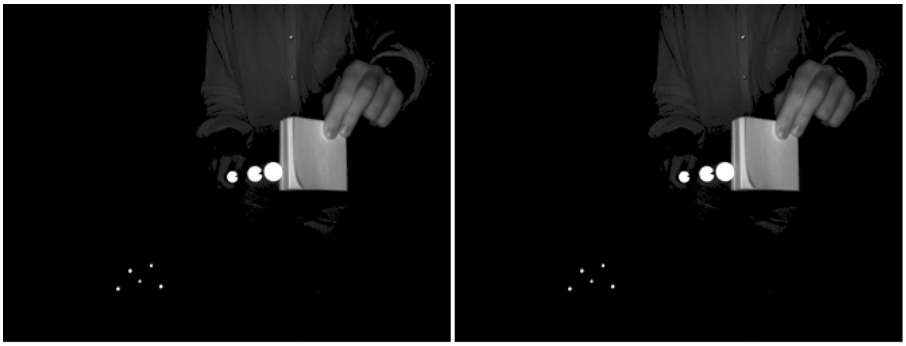

Figura 2.8: A sinistra un'immagine non compressa, a destra un'immagine compressa con RLE dove è applicata una sogliatura di 20

2.3.2 Codifica di Huffman

Questa tecnica è stata sviluppata nel 1952 da David A. Huffman. La codifica Huffman esprime il carattere più frequente nella maniera più breve possibile e crea una rappresentazione unica di ciascun prefisso a seconda della sua ripetizione.

La parte centrale dell'algoritmo riguarda la costruzione di un albero binario, costruito in base al numero di occorrenze di ogni carattere da codificare. La struttura finale dell'albero avrà i caratteri maggiormente ripetuti posizionati vicini alla radice dell'albero mentre gli altri caratteri saranno via via sempre più distanti in base alla loro frequenza.

Lo svantaggio di questa tecnica è la complessità di compressione e decompressione poichè richiede delle operazioni di confronto, costruzione e memorizzazione dell'albero non immediate. Sicuramente la codifica di Huffman offre il vantaggio della migliore compressione e la possibilità di decidere a priori il dizionario per la compressione - decompressione, poichè la cadenza degli elementi che compongono le immagini sono stati-

sticamente uniformi (come già detto il $99,9\%$ dei valori sono di sfondo, $0,04\%$ di valori sono saturati e i restanti sono valori intermedi utili per definire il contorno dei marcatori).

2.3.3 Codifica Lempel Ziv Welch

Mentre la codifica di Huffman usa codici (prefissi) a lunghezza variabile per far sì che a caratteri più frequenti corrispondano codici più corti, il metodo LZW usa codici a lunghezza fissa per codificare sequenze di caratteri di lunghezza variabile. In questo metodo è interessante il fatto che le sequenze codificate non siano casuali, ma corrispondano a sequenze che compaiono con una certa frequenza. Se una sequenza lunga tende a ripetersi, il metodo se ne accorge e in modo dinamico l'aggiunge a un dizionario di conversione. Tale dizionario, a differenza di quello creato dal metodo di Huffman, non deve essere memorizzato e spedito con la codifica, ma è ricostruibile dalla codifica stessa.

Come per la codifica di Huffman, questa soluzione è scartata per la complessità di calcolo nella codifica - decodifica dei dati, o generalmente questa codifica ha prestazioni peggiori di quella di Huffman. Anche volendo sfruttarla non si trarrebbe beneficio neppure dall'uso del dizionario integrato nella codifica data l'omogeneità dei dati acquisiti e dalla possibilità di costruire a priori un dizionario.

Come per la codifica di Huffman esistono diverse varianti di questo metodo. Tale tecnica è utilizzata nella codifica delle immagini in formato GIF e nei file TIFF.

2.4 Segmentazione

La segmentazione è quell'insieme di processi che viene svolto al fine di riconoscere, parametrizzare e classificare i blob nell'immagine. Un gruppo di pixel attigui organizzati in una struttura sono chiamati blob. Nell'im-

magine 2.9-a si possono distinguere i pixel rossi da quelli dello sfondo, tale gruppo viene chiamato blob e dalla posizione di tali pixel viene definita una forma.

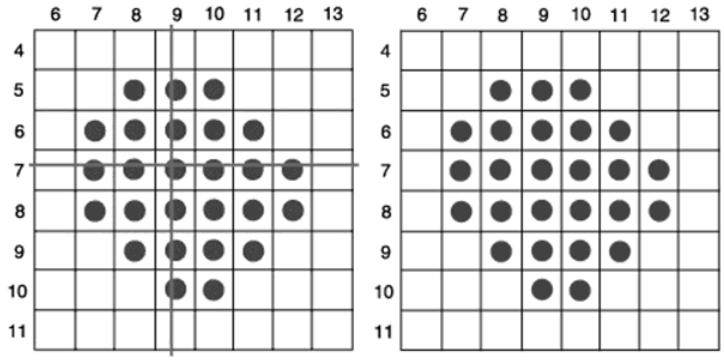

Figura 2.9: Esempio di blob e il suo baricentro ricavato sulla destra

Prima di individuare forme in un'immagine in alcuni casi, specie in applicazioni mediche dove è richiesta un'elevata immunità dai disturbi, si eseguono alcune operazioni di filtraggio o pulizia dell'immagine. Un esempio sono gli operatori morfologici per le immagini, tipo le operazioni di *dilation* ed *erosion*. L'operazione di *dilation* è simile alla convoluzione lineare ma con l'operazione di massimo al posto della somma della convoluzione e l'addizione al posto dei prodotti. Con questa operazione l'immagine di output tende ad essere più luminosa di quella di input, e i dettagli scuri (ad es. punti neri in uno sfondo bianco) sono o ridotti o eliminati, dipende dalla dimensione dell'elemento strutturante. L'immagine risultante risulterà con un'area soprasoglia maggiore di quella in ingresso.

L'operazione di *erosion* è l'operazione inversa della precedente, questa deriva dalla correlazione lineare, con l'operazione di minimo al posto della somma della correlazione e la sottrazione al posto dei prodotti. Con questa operazione l'immagine di output tende ad essere più scura di quella di input, e i dettagli chiari (ad es. punti bianchi in uno sfondo nero) sono o

ridotti o eliminati. L'immagine risultante risulterà con un'area soprasoglia minore di quella in ingresso.

La combinazione delle operazioni di *dilation* ed *erosion* permettono di filtrare l'immagine da rumore, oppure da hot pixel nel CCD [4]. Queste operazioni possono causare fusioni non volute tra due blob vicini.

In applicazioni *real time*[2] queste operazioni sono poco usate perché aumentano la complessità computazionale, infatti per aumentare l'immunità ai disturbi solitamente si agisce o sulla sogliatura o sull'algoritmo di riconoscimento di forma.

L'operazione di segmentazione continua con l'individuazione di ogni singolo blob, ricavandone una parametrizzazione che lo rappresenti in modo da effettuare la successiva classificazione dei blob.

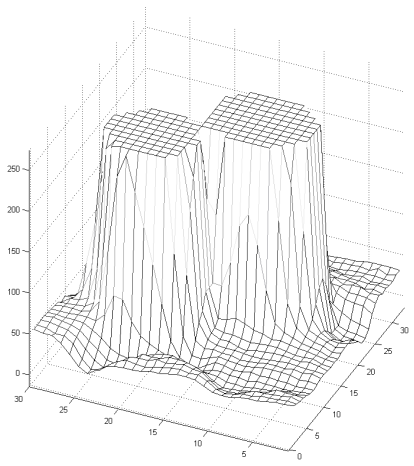

Figura 2.10: Blob fusi rappresentati per mettere in evidenza l'intensità dei pixel

Il blob è un oggetto dotato di dimensione, ciò significa che per rappresentare la sua posizione e spostamento ne dobbiamo prendere un riferimento. La posizione di ogni blob sarà rappresentato dal suo baricentro pesato dei pixel sopra la soglia, e le coordinate verranno calcolate:

[4]Hot pixel sono pixel non più funzionanti a causa di una corrente di buio costante che genera in uscita valori diversi da quelli misurati. Sono detti anche pixel morti.

$$m_x = \frac{1}{N}\sum_{i=1}^{N} x_i$$
$$m_y = \frac{1}{N}\sum_{i=1}^{N} y_i \tag{2.5}$$

Questa però non è l'unica tecnica per ricavare una rappresentazione puntiforme del *marker*, si può calcolare un baricentro partendo dalla sola forma o proprietà geometriche del *marker*, oppure dal picco di intensità più alto o scegliere arbitrariamente un estremo del suo bordo. Il cicle fitting prevede che a partire dal blob si ricostruisca il cerchio, che rappresenterebbe la proiezione del *marker* sul piano immagine, e da questo si ricava il baricentro. La scelta cade a favore del baricentro pesato dei pixel perché è possibile eliminare difetti introdotte da piccole deformazioni o rumore nell'immagine.

La prima misura e la prima verifica che vengono eseguite sul blob sono quelle inerenti all'area. Il sistema è impostato per rilevare *marker* di una dimensione prestabilita in un volume, questo significa che è possibile prevedere una dimensione massima e minima dell'area di un *marker*. Se il blob supera l'area massima che potrebbe avere un marcatore questo viene subito scartato. Questo tipo di test è molto efficiente sia per eliminare le riflessioni, sia per eliminare blob dovuti all'acquisizione del segnale infrarosso emesso da un emettitore puntato su una telecamera.

Un'altra parametrizzazione utile per la descrizione del blob è il suo asse principale, che può essere facilmente individuato partendo dalle medie pesate di ogni riga dove sfruttando delle approssimazioni passerà una sola retta; il coefficiente angolare di tale retta verrà indicato come inclinazione del blob o asse principale. Noto l'asse principale del blob è possibile calcolare le proiezioni del *marker* sui suoi assi: si può osservare in figura 2.11 la proiezione di due *marker* fusi sui suoi assi; la prima proiezione mette in evidenza i due *marker*, il più piccolo sulla sinistra, il più grande spo-

stato sulla destra, la seconda proiezione evidenzia solo una distribuzione poiché i due marcatori risultano sovrapposti.

BTS impega le più avanzate metodiche di riconoscimento offerte dalla teoria della visione artificiale. I passi per il riconoscimento del *marker* comprendono in sequenza:

- algoritmo di segmentazione edge-based sull'immagine a livelli di grigio
- contorni del blob modellati tramite profilo a rampa per ridurre l'impatto del *blurring* dovuto al campionamento
- *smoothing* tramite gaussiana e applicazio ne di soglia *optimal thresholding*
- *edge linking* e *template matching* basato su ellisse

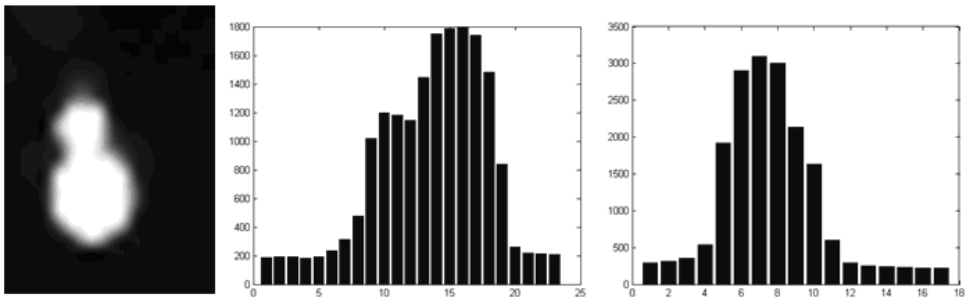

Figura 2.11: Proiezioni sui due assi principali di due *marker* di dimensione diversa

Con i vincoli di elaborazione in *real time* la scelta dell'algoritmo di segmentazione è piuttosto limitata ad un'elaborazione riga per riga dove il blob soprasoglia viene ricostruito a partire da un insieme di righe connesse. Questa tecnica consiste in un ciclo sulle righe e sulla riga, scorrendo ogni riga l'algoritmo seleziona il primo elemento appartenente al blob, e continua fino a trovare l'ultimo elemento appartenente al medesimo blob.

Selezionati questi elementi vengono memorizzati e l'algoritmo continuerà a scorrere la riga alla ricerca del successivo blob.

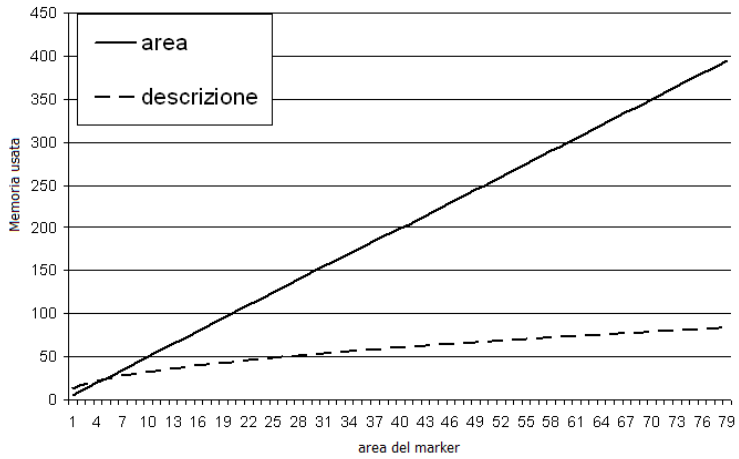

Figura 2.12: Uso di memoria per memorizzare tutti i pixel appartenenti ad un blob o solo una loro rappresentazione

Nonostante la scelta più semplice sarebbe quella di memorizzare il valore di ogni singolo pixel per ogni blob rilevato, questa tecnica potrebbe risultare parecchio costosa nell'uso di memoria. Ad ogni elemento individuato dovremmo misurare e salvare le posizioni e l'intensità di tutti i pixel della riga selezionata del blob, se invece scegliamo di elaborare a priori alcuni dati quali il peso della riga (somma delle intensità) e la sua media pesata, si ha la possibilità di salvare meno dati mantenendo le informazioni utili per la connessione con successivi blob e il finale riconoscimento di forme.

Per la successiva elaborazione del *marker* sarà necessario il calcolo di medie pesate, coordinate dei pixel di contorno e di altri parametri. Perciò al momento dell'acquisizione dei dati, vengono subito calcolati i parametri rappresentativi e memorizzati. In questo modo lo spazio necessario per la memorizzazione dei dati avrà un andamento logaritmico e non lineare, infatti dipenderà dal numero di righe e non dall'area. Questa tecnica risulta molto vantaggiosa per grandi blob o nell'uso di telecamere ad altissima

definizione. In figura 2.12 si osserva come all'aumento dell'area cambi sensibilmente l'uso della memoria tra la memorizzazione di ogni singolo pixel e i dati rappresentativi del blob.

L'algoritmo per il riconoscimento di forma può inserirsi o alla fine dell'elaborazione di ogni singola riga o alla fine dell'elaborazione di ogni immagine. Si devono considerare poi casistiche di *distraction*, *occlusion* e *fusion* di blob. Infatti quegli errori tipici del riconoscimento di forma, nativamente sono problemi anche nell'algoritmo di segmentazione. Come si osserva dalle immagini, scorrendo riga per riga dall'alto può capitare che due blob distinti si fondano in uno, se tale casistica non fosse gestita in alcuni casi verrebbero rilevati due blob incompleti. Nell'immagine 2.13 è evidenziato un blob dove si deve gestire la fusione, la riga rossa indica la prima fila di pixel che appartiene ad entrambi i blob e per cui avverrà la fusione.

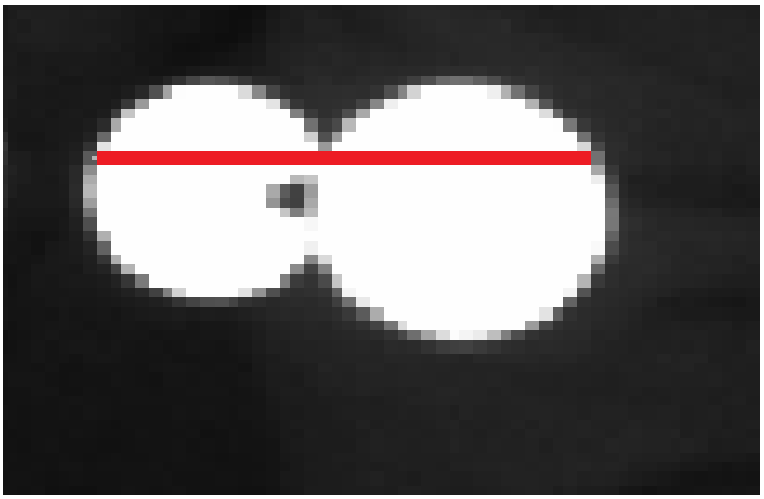

Figura 2.13: Fusione tra due marker

La gestione delle occlusioni-fusioni dovrà avvenire parallelamente alla segmentazione, perciò alla fine dell'elaborazione di ogni singola riga entrerà in gioco l'algoritmo che controllerà le fusioni tra blob. Appurato che due blob si fondono entrerà in azione un'operazione di unione dei

parametri raccolti di entrambi i blob, in particolare cambieranno le medie calcolate, l'area, le dimensioni, gli estremi e i pesi.

2.5 Teniche di riconoscimento di forme

Nell'analisi delle immagini un problema ricorrente è quello di stabilire se all'interno di una data immagine è presente o meno un template prestabilito, nel nostro caso la proiezione di un *marker* nel piano immagine. Questo problema ha trovato diverse soluzioni nel tempo, ad esempio l'uso di maschere [3] per la ricerca del *marker* nell'immagine, fattori di forma [4], cross-correlazione [5], momenti geometrici [6], trasformate di Fourier [7], tecniche di scheletrizzazione [8], dove ciascun punto dello scheletro è definito in termini di distanza dai punti più vicini del contorno, e quelle di *thinning* [9], che trasformano un template in una serie di archi digitali semplici giacenti approssimativamente lungo gli assi mediani.

2.5.1 Template matching

Se i *marker* fossero definiti da forma e dimensione predefinita, e sfruttassimo dati binari (abbastanza comune in sistemi stereofotogrammetrici IR) l'individuazione dei *marker* potrebbe essere eseguita con una semplice tecnica di confronto. Se cercassimo i minimi della differenza binaria totale tra un *marker* campione $u(m,n)$, che chiameremo template, e l'immagine originale $v(m,n)$, basterebbe calcolare la funzione:

$$\gamma_{vu}(p,q) = \sum_m \sum_n [v(m,n) \oplus u(m-p, n-q)] \tag{2.6}$$

dove $\oplus$ è l'operazione booleana di OR esclusivo. La funzione $\gamma_{vu}(p,q)$ ci fornirà il numero di pixel dell'immagine che non coincidono con quelli del template posizionato alle coordinate (p,q). Il che significa che i minimi questa funzione può fornire la presenza di template uguali alla maschera fornita.

Questa tecnica è inapplicabile nei sistemi stereofotogrammetrici, oltre per la complessità computazionale, ma perchè i *marker* variano forma, dimensione ed inclinazione, ragion per cui servirebbe un set ampissimo di template per individuare il *marker* e aumenterebbe a dismisura la complessità computazionale.

2.5.2 Cross-correlazione

Anche se possiamo prevedere la forma e un margine di dimensione dei *marker*, non è realistico aspettarsi di trovare una copia esatta di un *marker* campione $u(m,n)$, che chiameremo template o kerner. Diventa perciò importante rilevare una misura quantitativa della somiglianza di un blob $v(m,n)$ con il template. Per rilevare un *marker* sarà perciò necessario traslare il template in tutte le possibili posizioni e rotazioni. Si può definire la misura di dissomiglianza come

$$\begin{aligned}\sigma^2(p,q) = \sum_m \sum_n [v(m,n) - u(m-p,n-q)]^2 = \\ \sum_m \sum_n |v(m,n)|^2 + \sum_m \sum_n |u(m,n)|^2 \\ -2 \sum_m \sum_n v(m,n) u(m-p,n-q)\end{aligned} \tag{2.7}$$

per cui si troverà un *marker* quando $\sigma^2(p,q)$ raggiunge il valore minimo, il minimo si ricaverà massimizzando il terzo membro dell'equazione, che chiameremo cross-correlazione

$$c_{vu} = \sum_m \sum_n v(m,n) u(m-p,n-q) \tag{2.8}$$

Applicando la disuguaglianza di Cauchy-Schwarz si ha che

$$\begin{aligned}|c_{vu}| = \left| \sum_m \sum_n v(m,n) u(m-p,n-q) \right| \leq \\ \leq \sqrt{\sum_m \sum_n |v(m,n)|^2} \sqrt{\sum_m \sum_n |u(m,n)|^2}\end{aligned} \tag{2.9}$$

perciò l'uguaglianza è verificata se e solo se

$$v(m,n) = \alpha u(m-p, n-q) \tag{2.10}$$

dove α è una costante arbitraria che può essere posta ad 1 per comodità. La cross-correlazione c_{vu} raggiungerà il suo massimo quando la posizione traslata del template coincide con la regione di immagine osservata, in altri termini si può scrivere:

$$c_{vu}(m,n) = \sum_m \sum_n |v(m,n)|^2 > 0 \tag{2.11}$$

e il massimo ha luogo quando in una particolare posizione dell'immagine esiste una copia perfetta del template. Con questa tecnica i picchi della funzione cross-correlazione all'interno di un'immagine forniscono le posizioni dei *marker*.

Come già detto, i *marker* per essere individuati necessitano una rotazione e cambiamento di scala del template, e se la rotazione sarà di un angolo $0 \leq \vartheta < 2\pi$, il cambiamento di scala prevede una scelta da parte dell'utente. Si ottiene una corrispondente formulazione

$$v(m,n) = \alpha u(\frac{m-p}{\xi}, \frac{n-q}{\eta}, \vartheta) \tag{2.12}$$

dove (p,q) sono le coordinate della traslazione, ϑ l'angolo di rotazione dell'immagine rispetto al template, (ξ,η) i fattori di scala nelle due dimensioni dell'immagine. L'ultima modellizzazione prevede l'analisi dell'immagine mutando il template variando i cinque parametri (p,q,ξ,η,ϑ), cosa che diventa impraticabile sia dal punto di vista computazionale che per uso di memoria in *real time*.

La funzione di cross-correlazione $c_{vu}(p,q)$ può essere calcolata, oltre come già visto, anche come trasformata inversa di Fourier della funzione

$$C_{vu}(\omega_1,\omega_2) = V(\omega_1,\omega_2)U*(\omega_1,\omega_2) \tag{2.13}$$

dove U e V sono le trasformate di Fourier di u e v e $U*$ è il complesso coniugato di U. Il calcolo diretto della cross-correlazione è conveniente quando la superficie del pattern è piccola rispetto a quella dell'immagine sotto

esame, in caso contrario è preferibile ricorrere a una trasformata rapida di Fourier (Fast Fourier Transform, FFT). Anche in questo caso l'operazione risulta inefficiente dal punto di vista computazionale ed inapplicabile in applicazioni in tempo reale.

2.5.3 Descrittori di Fourier

I coefficienti della trasformata discreta di Fourier forniscono un'ulteriore relazione tra un'immagine e un set di parametri campione. Anche se solitamente si usa applicare la trasformata di Fourier discreta direttamente alla funzione intensità $f(x, y)$, talvolta si determinano i descrittori di Fourier del contorno del *marker*.

Dato un insieme di punti (x_i, y_i), con $i = 0, 1, ..., N$ e con $(x_0, y_0) = (x_N, y_N)$, che costituisce il campionamento in verso orario di una curva chiusa semplice di lunghezza L. Questo metodo [10] [11] è basato sulla lunghezza d'arco l_i tra (x_0, y_0) e (x_i, y_i) e l'angolo $\beta(l_i)$ tra la tangente alla curva in (x_0, y_0) e la tangente in (x_i, y_i). L'angolo $\beta(l_i)$ viene poi normalizzato :

$$\alpha(t) = \frac{\beta L t}{2\pi} \quad 0 \leq t < 2\pi \tag{2.14}$$

in questo modo tale quantità diventa invariante per traslazione, rotazione e cambiamento di scala. I descrittori complessi di Fourier sono definiti come:

$$c_v = \frac{1}{2\pi} \int_0^{2\pi} \alpha(t) e^{-ivt} dt \tag{2.15}$$

2.5.4 Momenti geometrici

L'idea di usare i momenti per il riconoscimento di forme in immagini deriva dallo sviluppo dei momenti invarianti di Hu [6] [12]. In funzioni a due dimensioni continue i momenti si calcolano come

$$M_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} x^p y^q f(x, y) dx dy \tag{2.16}$$

in un'immagine con scala di grigi, dove abbiamo quantità discrete e la funzione $I(x, y)$ rappresenterà l'intensità del pixel

$$M_{ij} = \sum_x \sum_y x^i y^j I(x, y) \tag{2.17}$$

Si definiscono i momenti centrali, che sono momenti invarianti per traslazione:

$$\mu_{pq} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} (x - \bar{x})^p (y - \bar{y})^q f(x, y) dx dy \tag{2.18}$$

dove le coordinate del centro di massa possono essere definite come

$$\begin{aligned} \bar{x} &= \frac{M_{10}}{M_{00}} \\ \bar{y} &= \frac{M_{01}}{M_{00}} \end{aligned} \tag{2.19}$$

in particolare in un'immagine, dove abbiamo quantità discrete, i momenti centrali si calcolano come

$$\mu_{pq} = \sum_x \sum_y (x - \bar{x})^p (y - \bar{y})^q f(x, y) \tag{2.20}$$

e se vengono calcolate si ottengono le seguenti formulazioni (momenti centrali fino al terzo ordine)

$$\begin{aligned} \mu_{00} &= M_{00} \\ \mu_{01} &= 0 \\ \mu_{10} &= 0 \\ \mu_{11} &= M_{11} - \bar{x} M_{01} = M_{11} - \bar{y} M_{10} \\ \mu_{20} &= M_{20} - \bar{x} M_{10} \\ \mu_{02} &= M_{02} - \bar{y} M_{01} \\ \mu_{21} &= M_{21} - 2\bar{x} M_{11} - \bar{y} M_{20} + 2\bar{x}^2 M_{01} \\ \mu_{12} &= M_{12} - 2\bar{y} M_{11} - \bar{x} M_{02} + 2\bar{y}^2 M_{10} \\ \mu_{30} &= M_{30} - 3\bar{x} M_{20} + 2\bar{x}^2 M_{10} \\ \mu_{03} &= M_{03} - 3\bar{y} M_{02} + 2\bar{y}^2 M_{01} \end{aligned} \tag{2.21}$$

Conseguentemente possiamo ricavarci momenti invarianti per scala:

$$\eta_{ij} = \frac{\mu_{ij}}{\mu_{00}^{\left(1+\frac{i+j}{2}\right)}} \tag{2.22}$$

Riassumendo è possibile calcolare momenti invarianti per traslazione, scala e rotazione; tali momenti sono anche detti momenti invarianti di Hu [13]:

$$\begin{aligned}
I_1 &= \eta_{20} + \eta_{02} \\
I_2 &= (\eta_{20} - \eta_{02})^2 + (2\eta_{11})^2 \\
I_3 &= (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \\
I_4 &= (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \\
I_5 &= (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + \\
&\quad (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] \\
I_6 &= (\eta_{20} - \eta_{02})[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \\
I_7 &= (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2] + \\
&\quad (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2]
\end{aligned} \tag{2.23}$$

dove il momento I_1 è detto di inerzia del centroide dell'immagine, il momento I_7 permette di riconoscere con sicurezza l'immagine.

L'informazione sull'orientamento dell'immagine può essere ottenuto con i momenti di secondo ordine, calcolando la matrice di covarianza:

$$\begin{aligned}
\mu'_{20} &= \frac{\mu_{20}}{\mu_{00}} = \frac{M_{20}}{M_{00}} - \bar{x}^2 \\
\mu'_{02} &= \frac{\mu_{02}}{\mu_{00}} = \frac{M_{02}}{M_{00}} - \bar{y}^2 \\
\mu'_{11} &= \frac{\mu_{11}}{\mu_{00}} = \frac{M_{11}}{M_{00}} - \bar{x}\bar{y}
\end{aligned} \tag{2.24}$$

dove la matrice covarianza dell'immagine $I(x, y)$ si calcolerà come

$$\mathrm{cov}[I(x, y)] = \begin{bmatrix} \mu'_{20} & \mu'_{11} \\ \mu'_{11} & \mu'_{02} \end{bmatrix} \tag{2.25}$$

L'autovettore di questa matrice corrisponde all'asse di maggior e minor intensità, l'orientazione può perciò essere estratta dall'angolo dell'autovettore associato con l'autovalore più grande; l'angolo si calcola con la seguente formula:

$$\Theta = \frac{1}{2}\arctan\left(\frac{2\mu'_{11}}{\mu'_{20} - \mu'_{02}}\right) \tag{2.26}$$

Gli autovalori della matrice covarianza sono calcolati da

$$\lambda_i = \frac{\mu'_{20} + \mu'_{02}}{2} \pm \frac{\sqrt{4\mu'^2_{11} + (\mu'_{20} - \mu'_{02})^2}}{2} \tag{2.27}$$

Può essere calcolata anche l'eccentricità

$$ec = \sqrt{1 - \frac{\lambda_2}{\lambda_1}} \tag{2.28}$$

Questo metodo è facilmente implementabile via hardware [14] e se ben progettato permette di risparmiare cicli di elaborazione. L'elaborazione hardware anche se più conveniente è meno flessibile ad aggiornamenti del sistema e più costosa.

I momenti geometrici sono molto sensibili al rumore, esistono però altri momenti detti di Zernike e Tchebichef che invece presenatano una alta immunità al rumore, ma aumentano esponenzialmente la complessità di calcolo e uso di memoria. Un momento di Zernike [15] di ordine p con ripetizione q è definito come segue:

$$\begin{aligned} Z_{pq}(x,y) &= \frac{p+1}{\pi}\sum_X\sum_Y f(x,y)W_{pq}(r,\vartheta) \\ W_{pq}(r,\vartheta) &= R_{pq}(t)e^{iq\vartheta} \\ R_{p,\pm q}(t) &= \sum_{k=q,p-|k|=-\infty}^{p} B_{pkq}r^k \\ B_{pkq} &= \frac{(-1)^{(p-k)/2}((p+k)/2)!}{((p-k)/2)!((k-q)/2)!((k+q)/2)!} \end{aligned} \tag{2.29}$$

2.5.5 Fattori di forma

Altro metodo per il riconoscimento di forma è tramite i fattori di forma che si posso ricavare dal blob. I fattori di forma sono parametrizzazioni che si ricavano da proprietà geometriche del template obbiettivo.

I marcatori che vogliamo individuare sono sferici, il che significa che le loro proiezioni nel piano immagine appariranno rotondi o al più elissoidi per effetto di distorsioni ottiche. Nel caso più semplice il marcatore apparirà tondo, perciò sarà caratterizzato da un'altezza e larghezza uguali, comunque sia il verso di orientazione. Ragion per cui si può definire come primo il fattore di forma il rapporto tra larghezza ed altezza. Posto f_f il fattore di forma si può scrivere

$$f_f = \frac{w}{h} \tag{2.30}$$

dove h è l'altezza e w la larghezza del marcatore. Per semplificare scriviamo che se $w > h$ allora

$$f_f = \frac{h}{w} \tag{2.31}$$

Da cui si ricava la formulazione definitiva di f_f, che assumerà valori nell'intervallo finito da 0 a 1:

$$\begin{cases} f_f = \frac{w}{h} & h \geq w \\ f_f = \frac{h}{w} & h < w \end{cases} \iff 0 < f_f \leq 1 \tag{2.32}$$

Un secondo parametro che possiamo considerare è il rapporto tra l'area del blob e l'area iscritta nel rettangolo con altezza e larghezza del blob considerato. Idealmente in un blob di area unitaria con $w = 1$ e $h = 1$ l'area del blob è

$$\pi \left(\frac{w}{2}\right)^2 \cong 0,78 \tag{2.33}$$

il che significa che il rapporto tra l'area del blob e l'area del rettangolo (in quest'ultimo caso unitaria) deve avere un valore intorno a $0,78$. Si definisce perciò il fattore di area f_a, a partire dall'area del blob a_b e l'area

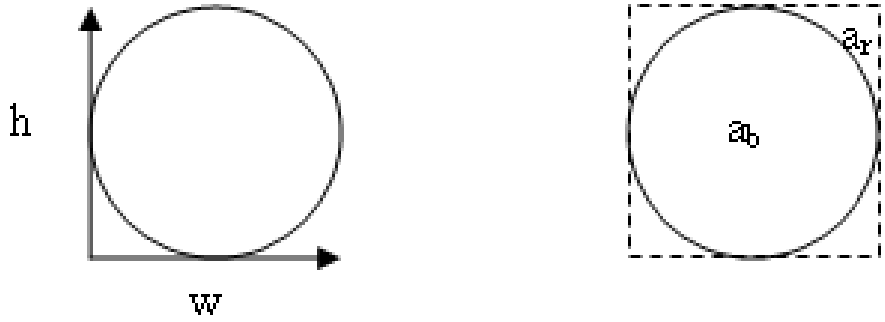

Figura 2.14: parametrizzazione di un blob

del rettangolo a_r di dimensioni hxw.

$$f_a = \frac{a_b}{a_r} \Longleftrightarrow 0 < f_a \leq 1 \tag{2.34}$$

Si sottolinea che un *marker* con f_a maggiore di $0,78$ dovrà comunque essere considerato *marker*. Infatti per piccole aree o bassa definizione delle videocamere viene persa informazione sulla forma, ad esempio un pixel rappresentante un *marker* avrà $f_a = 1$, similmente 4 pixel come si può osservare in figura 2.15.

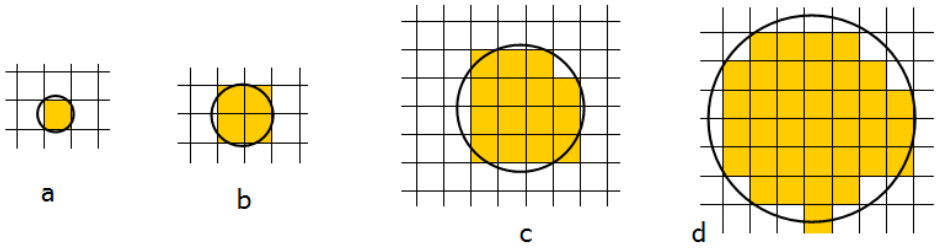

Figura 2.15: Si osserva come varia la rappresentazione discreta del blob sul piano immagine al variare della dimensione

Se uniamo entrambe le caratteristiche, si ha a disposizione una semplice parametrizzazione dei blob per il riconoscimento di forma. Questa tecnica permette l'individuazione veloce con un semplice confronto dei *marker* senza difetti permettendo di focalizzare l'attenzione su quelli sicuramente con difetti. L'obiezione che può essere mossa contro questa

strategia è quella della possibilità di falsi positivi: possono comparire casistiche dove un blob appare come un marcatore ma non lo è, in questi casi si osserva l'errore che ne consegue. Se un blob, che non è un marcatore, ma appare come tale verrà conseguentemente considerato nel processo di *tracking*, se questo errore appare in un singolo frame, il processo di *tracking* presumibilmente non verrà alterato, nel caso in cui questi finti marcatori compaiono in più frame il tracker ne costruirà una traiettoria, tale traiettoria verrà però scartata al momento del *labelling* da parte dell'operatore. Se il falso *marker* è in posizioni compromettente nel protocollo usato, compare in più frame e il *labelling* avviene in maniera automatica, allora si può cadere in errore. In generale però sono molto più comuni e problematici errori di falso negativo, cioè scompare o non viene rilevato il marcatore, piuttosto di quelli di falso positivo.

2.5.6 Trasformata di Hough

Questa tecnica [16] è basata sulla validazione delle ipotesi, cioè definita la curva che si intende cercare nell'immagine, per ogni pixel si calcolano i parametri di tutte le curve che potrebbero passare per quel punto e si incrementano le celle di uno spazio n-dimensionale (dove n numero dei parametri) che corrispondono alle varie curve. In questo modo si ottiene una funzione di accumulazione per ogni pixel definita nello spazio dei parametri. Elaborati tutti i pixel, i massimi della funzione di accumulazione rappresenteranno le curve individuate, cioè i punti nello spazio dei parametri che hanno accumulato il maggior numero di voti che rappresentano le curve con probabilità elevata di essere presenti nell'immagine. L'obbiettivo finale è far si che il centro della nostra figura sia la zona più votata nello spazio parametri, dove quest'ultimo avrà la stessa dimensione dell'immagine in cui sono presenti gli oggetti.

Il principio non si discosta molto dal *template matching*, solo che non viene più studiato il blob ma i bordi del blob. lo studio dei soli bordi può rappresentare un vantaggio per una minor complessità della segmentazio-

ne e memorizzazione di dati parametrici dell'immagine, ma implica una perdità di dati: ad esempio l'intensità dei singoli pixel.

Come per la cross-correlazione questop metodo ha una elevata complessità di elaborazione che la rende proibitiva per applicazioni *real time*. Si deve però sottolineare come sia uno dei pochi metodi che tramite il voter permette di rilevare figure parzialmente occluse; altra caratteristica interessante della trasformata di Hough è quella di non risentire del rumore presente in una immagine.

2.6 Classificazione

Riconosciuta una forma sarà necessario classificarla, nella stereofotogrammetria IR saranno *marker* o blob. Se in sistemi OCR (tecniche per il riconoscimento dei caratteri) sono molto usati sistemi di reti neurali, MLP (multilayer perceptron), GAL (Gaining Algorithm Learning), K-NN (K-Nearest Neighbour), e fuzzy K-NN (fuzzy K-Nearest Neighbour) nel nostro caso il set di kerner o template da individuare è molto limitato.

Dovendo gestire *distraction*, *occlusion* e *fusion* si deve comprendere come si presenteranno nel piano immagine i marcatori, in tal modo si costruirà una tabella decisionale su quali criteri applicare nei casi che si presenteranno. Il caso più semplice sarà quello di un *marker* tondo e pulito, in tal caso non si fa altro che mantenere i dati acquisiti e memorizzarli. Il secondo caso sarà quello di due *marker* fusi, ma in tal caso si deve decidere quando si tratta di un *marker* deforme e quando di due *marker*. Due marcatori appoggiati possono essere distinti dal fattore di forma intorno $0,5$. In figura 2.16 si può vedere la rappresentazione delle casistiche a cui si è fatto riferimento.

La gestione di tali anomalie dovrà essere flessibile anche al tipo di dati in ingresso. Se consideriamo due tipi di aquisizioni: la prima con un setup ad hoc (ambiente controllato, ottiche regolate e nessuna distorsione rilevabile), la seconda con un setup inperfetto (presenza di distorsioni). Usando

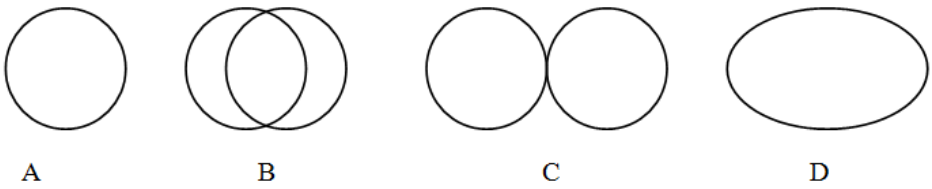

Figura 2.16: Alcune forme da classificare come marker

il primo setup in ingresso rileveremo marcatori tondi, marcatori fusi, marcatori occlusi e blob estranei: i marcatori tondi vengono individuati per primi, si esegue poi un secondo algoritmo per individuare i marcatori fusi, in seguito vengono individuati possibili marcatori occlusi e i restanti blob verranno scartati. Usando il secondo setup rileveremo meno i marcatori ideali poichè in presenza di distorsione o sfuocamento i marcatori potrebbero allungarsi in qualche direzione, aumentare l'area rilevata e fondersi, aumentando il numero e complessità di tutte le casistiche secondarie già citate. In questo secondo caso è consigliabile rilassare i parametri che definiscono i *marker* per non acquisire troppi falsi negativi che andrebbero a logorare le operazioni di *tracking*.

La soluzione ottimale nel caso di elaborazioni *real time* sarà quella di integrare la classificazione con il riconoscimento forme, che a sua volta integrato nell'algoritmo di segmentazione. In questo modo individuato un blob viene subito processato e classificato, eliminando dalla memoria dati superflui.

2.7 *tracking*, data association e High-level processing

Ricostruite le posizioni dei marcatori sull'immagine, è possibile ricostruire la posizione del punto nello spazio noto il setup di calibrazione del sistema. Per la ricostruzione 3D di un marcatore sarà necessario che questo sia

presente in almeno 2 immagini, provenienti da due videocamere distinte nel medesimo istante di campionamento. In figura 2.17 si può osservare la geometria epipolare per due telecamere, dati tre punti in uno spazio a tre dimensioni passerà un solo piano, detto epipolare, da una singola immagine non è possibile ricavare la posizione del *marker* poiché non viene individuata la profondità, ma viene individuata una retta su cui giace il *marker*; la seconda immagine permette di ricavare una seconda retta che intersecandosi con quest'ultima definirà la profondità e la posizione del marcatore. Il marcatore dovrebbe perciò essere definito come il punto di intersezione delle due rette. Nella realtà, sia per limiti di risoluzione delle videocamere sia per il rumore sovrapposto, tali rette risultano sghembe, come in figura 2.18 , perciò la posizione del marcatore è definita dalla media tra i punti di minima distanza tra le due rette.

Ricostruite le posizioni 3D di ciascun marcatore in ogni frame sarà necessario calcolare le traiettorie, iniziare l'operazione di *tracking*. Questo problema molto complesso è stato risolto con due soluzioni basate su considerazioni distinte: regolarità della traiettoria e informazioni a priori sul moto del soggetto. Algoritmi di *tracking* basati sulla regolarità della traiettoria sono basati sull'uso stimatori ricorsivi dello stato (filtro di Kalman). Un filtro di Kalman è strutturato in due passi:

- predizione: l'algoritmo calcola la posizione attesa del marcatore, in base alle precedenti posizioni, al modello dinamico usato ed alla varianza della predizione (complessità dovuta al calcolo congiunto della traiettorie di più marcatori);

- aggiornamento: l'algoritmo aggiorna lo stato del marcatore;

Il filtro di Kalman va tarato, in particolare secondo l'errore che commettono le telecamere in risoluzione e frequenza di acquisizione delle immagini. In presenza di urti questi filtri danno logicamente dei problemi. Algoritmi di *tracking* basati sulle informazioni a priori su forma e tipo

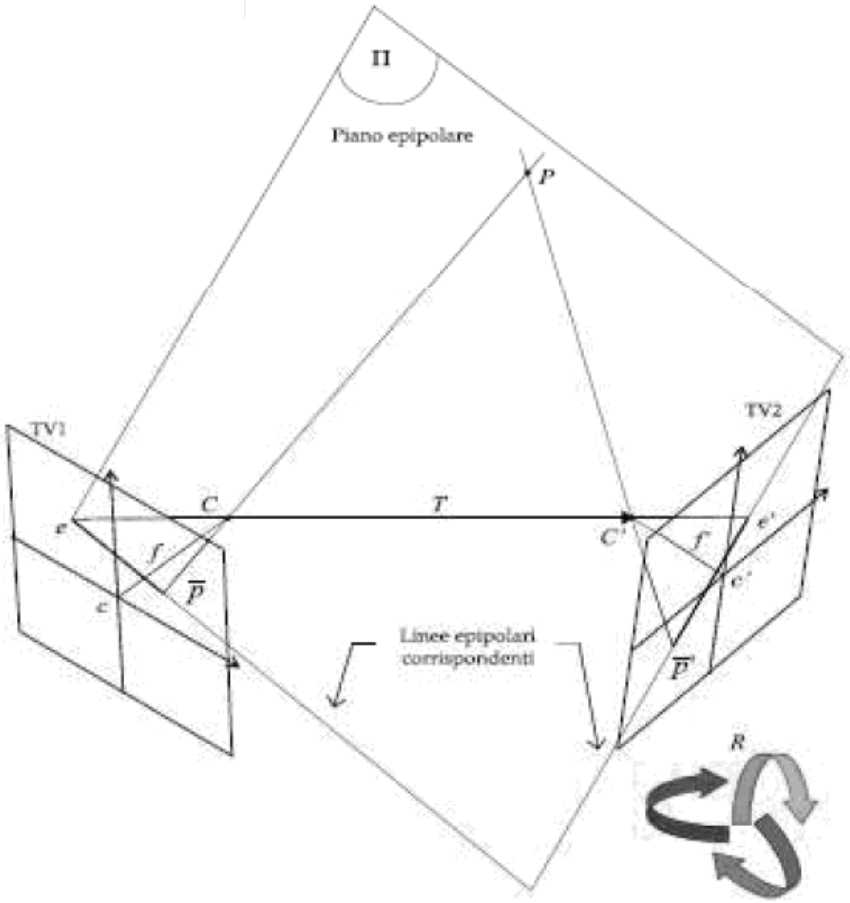

Figura 2.17: Geometria epipolare per due telecamere

di moto risultano di complessa scrittura ed elaborazione; questi si basano sul concetto statistico di forma, densità di probabilità congiunta della posizione relativa dei marcatori e della loro velocità.

Buoni risultati di *tracking* si sono ottenuti mediando le due soluzioni, vengono posti dei vincoli al moto dei marcatori e si sfrutta un filtro di Kalman per la predizione ed aggiornamento degli stati del *marker*. In figura 2.19 il diagramma di flusso di un filtro di Kalman: nella parte sinistra il modello stima una posizione del *marker*, nella parte destra viene aggiornato lo stato, ma non con le coordinate 3D del marcatore ma le informazioni delle singole telecamere, in tal modo è possibile prevedere e

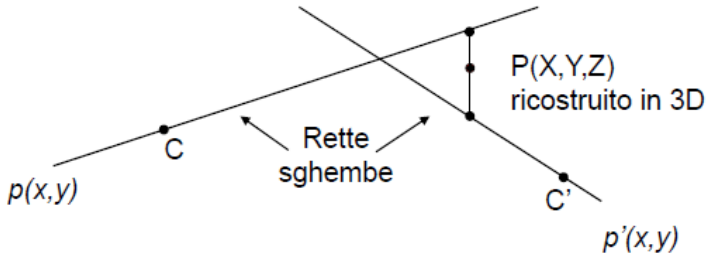

Figura 2.18: Ricostruzione 3D tramite triangolazione

gestire occlusioni; il grande vantaggio di quest'ultimo modello è che basta una singola telecamera per ricavare la posizione di un marcatore.

2.8 Fonti di errore

I sistemi stereofotogrammetrici sono soggetti ad errori intrinseci e estrinseci al sistema stesso. Si descriveranno di seguito le principali tipologie di errore che si possono riscontrare:

- **Incertezza del sistema di visione**: questi errori vengono introdotti da tutti i componenti del sistema, in particolare dalle telecamere. L'incertezza della posizione del *marker* dipende fortemente dalla risoluzione della videocamera, pertanto una alta risoluzione permette di definire correttamente forma, area e posizione del *marker*. A questa categoria appartengono errori sistematici, causati da inaccuratezze nel modello di misura o nella calibrazione, i quali possono generare distorsione o cattiva stima dei parametri, ed errori casuali quali rumore elettrico, flickering ed errori di campionamento; gli errori sistematici o casuali sono anche detti a bassa o alta frequenza;

- **Incertezza dovuta alla mancanza di corrispondenza tra punto effettivo e atteso nel piazzamento del *marker*** : questo tipo di errore è insito nell'oggetto della misura, in particolare il corpo umano poichè

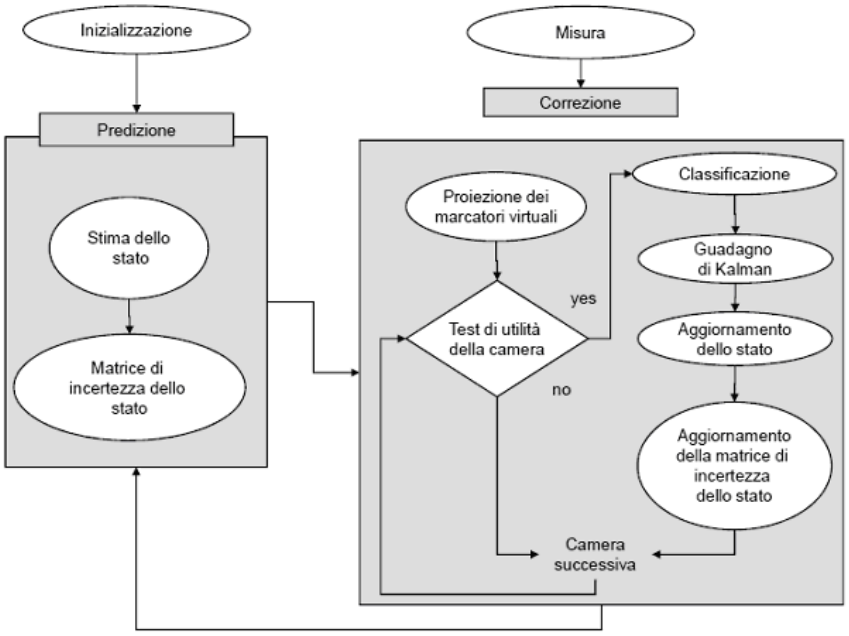

Figura 2.19: Procedura della stima del movimento basato su filtro di Kalman e modelli biomeccanici

è costituito da organi rigidi e tessuti molli. In questo processo gli errori sono sostanzialmente di tre tipi: errore nel posizionamento del *marker* , errore dovuto al movimento della pelle (*skin effect*), errore dovuto alle forze di inerzia agenti sul *marker* ;

- **Incertezza dovuta al calcolo della posizione delle terne rispetto ai segmenti corporei**: questo tipo di errore è influenzato direttamente dagli errori precedenti, infatti se i dati usati contengono errori nel processo di calcolo tali errori potrebbero amplificarsi a discapito del risultato finale;

Mentre l'incertezza dovuta alla mancanza di corrispondenza tra punto effettivo e atteso nel piazzamento del *marker* dipende sostanzialmente dal protoccolo, dal soggetto di studio, dalla competenza del personale e parametri che variano di volta in volta, l'incertezza del sistema stereofotogrammetrico rimane costante sempre (salvo acquisizioni in ambienti par-

ticolarmente ostili). Con questo si può affermare che il metodo principale per migliorare la qualità e l'affidabilità dei nostri risultati è migliorare il sistema stereofotogrammetrico.

2.9 Misura dell'errore

In letteratura [17] [18] esistono diverse proposte riguardo a test di accuratezza per sistemi stereofotogrammetrici, ma si deve ancora giungere ad una standardizzazione delle procedure di verifica. Non si è giunti ancora ad un modello accettato all'unanimità dalla comunità internazionale poichè ogni test proposto mette in evidenza un aspetto della misura piuttosto che altri.

Ogni test si basa su elementi di geometria nota in movimento. Dall'acquisizione del sistema stereofotogrammetrico si ricostruisce questa geometria, si confronta con le misure reali e se ne ricava conseguentemente il grado di accuratezza. Questi test solitamente si riferiscono alla misura della distanza tra due marcatori, come ad esempio nel *Japanese performance test* o MAL test, oppure nella misura dello spostamento di marcatori (come Test di gravità proposto da Capozzo). Vediamo alcuni test :

- Test del pendolo: un pendolo rigido, composto da una barra di $0.5m$ cui sono applicati due *marker* sferici alla distanza di $0.2m$ e $0.4m$ dall'asse di rotazione, viene fatto oscillare nei piani XZ o YZ. Dai dati acquisiti viene calcolata la distanza tra i due *marker* e la varianza che questo valore assume. Il confronto tra la distanza nota e quella stimata fornisce informazioni sull'accuratezza delle misure effettuate dal sistema e permette di stimare la componente casuale di errore associata al processo di misura. Questo test è stato proposto nel 1993 durante il progetto CAMARC (Cappozzo, Benedetti et al.) ed è rappresentato in figura 2.20-a;

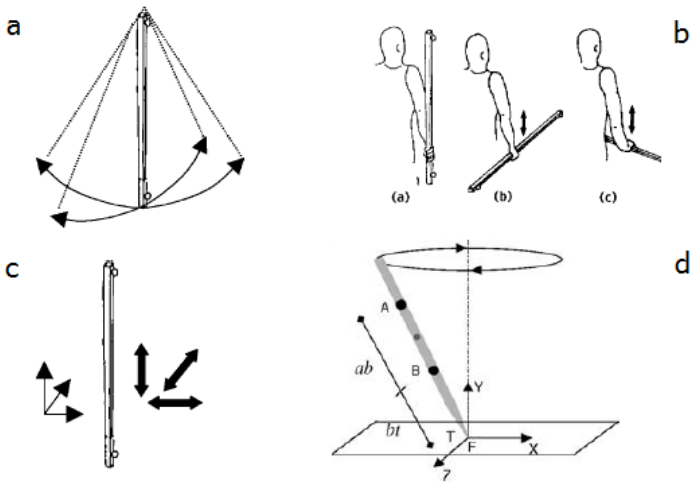

Figura 2.20: Schematizzazione dei test di accuratezza

- Full volume test: una barra rigida con due *marker* alle sue estremità viene mossa parallelamente alla direzione dei tre assi del laboratorio spaziando tutto il volume di misura. Il calcolo della distanza tra i due punti utilizzando i dati acquisiti anche in questo caso fornisce informazioni sull'accuratezza delle misure effettuate dal sistema. Anche questo test è stato proposto durante il progetto CAMARC (Cappozzo, Benedetti et al.) ed è rappresentato in figura 2.20-c;

- Test di gravità: Un corpo con un *marker* viene fatto cadere da un'altezza nota, in tal modo si verifica l'accuratezza dell'accelerazione misurata. Anche questo test è stato proposto da Ehara nel 1995 CAMARC (Cappozzo, Benedetti et al.);

- *Japanese performance test*: una barra rigida di lunghezza nota con due *marker* alle sue estremità, viene tenuta in mano in direzione verticale mentre la persona cammina seguendo un percorso predefinito. Anche questo test è stato proposto da Ehara nel 1995 ed è rappresentato in figura 2.20-b;

- Mal test (Movement Analysis Laboratory): su una barra rigida sono

montati due *marker* sferici A e B, un punto target T coincide con il perno dell'asta, supposto anche posizione nota. Si eseguono test statici e test dinamici facendo ruotare la barra e si ricava l'errore di misura del target. Questo test è stato proposto da Cappozzo e Della Croce nel 2000 ed è rappresentato in figura 2.20-d;

Nonostante le prestazioni del sistema nominalmente, in termini di accuratezza e precisione, risultino elevate, in realtà dipendono da una varietà di fattori. Il primo fattore che incide sull'accuratezza del sistema è senz'altro il setup del sistema: il numero di videocamere, la loro posizione, il volume di misura, la dimensione e forma dell'oggetto da acquisire. Nel 2001 uno studio di Schmid riferisce un'accuratezza dei sistemi commerciali tra $0,09\%$ ed il $1,77\%$ del campo di misura. Tali valori sono comunque indicativi perchè vanno varificate da sistema a sistema.

Gli errori descritti fin ora, sovrapposti alle coordinate globali dei marcatori generano traiettorie errate. Le posizioni e traiettorie errate propagano l'errore al sistema di riferimento dei punti di repere anatomici provocando l'errato calcolo di angoli, momenti e posizioni della parte anatomica sotto esame.

	Peak 5	Ariel APAS	Vicon 370	Elite	Primas	Video Locus
Num. di videocamere	2	2	6	4	2	2
Freq. di campionamento [Hz]	60	60	60	50	100	60
Errore medio [mm]	3,85	11,61	0,94	0,53	1,79	7,73
Dev. standard	2,04	5,36	0,39	0,31	0,14	1,45
Errore massimo [mm]	+8,10 -10,39	+13,47 -24,07	+4,37 -8,57	+1,19 -0,96	+4,09 -6,14	+28,23 -7,19

Tabella 2.1: Errori misurati in alcuni sistemi optoelettronici

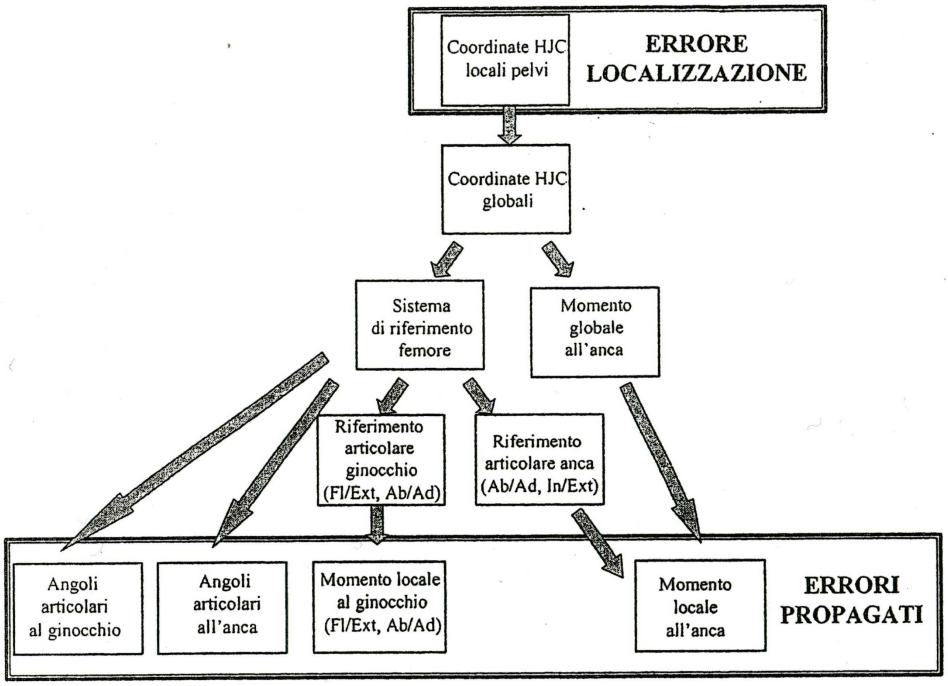

Figura 2.21: Propagazione dell'errore in un sistema optoelettronico per la *gait analysis*

Capitolo 3

Metodi adottati e risultati

3.1 Interfaccia grafica

Per sviluppare le funzioni descritte nel seguente capitolo è stata realizzata attraverso MatLab una piattaforma con interfaccia grafica dove è possibile selezionare l'algoritmo, scegliere i suoi parametri, selezionare una o più immagini da elaborare e procedere con il calcolo. La necessità di un'interfaccia grafica nasce dal fatto che la mole di dati da analizzare e visualizzare è notevole, in più per verificare l'efficienza degli algoritmi sono necessari l'accesso e la modifica dei parametri usati dagli algoritmi stessi.

Le 310 immagini usate sono state acquisite mediante un sistema SMART-D a 60Hz con risoluzione di 640x480 pixel per videocamera. Il sistema SMART ha permesso il salvataggio di una decina di video a 256 livelli di grigio di una videocamera del sistema stereofotogrammetrico nel formato tdf, in seguito tramite un'algoritmo di proprietà BTS e un'interfaccia grafica creata per l'occasione, le immagini sono state selezionate e salvate in bmp. Le immagini rappresentano le situazioni più critiche che si possano presentare durante l'uso del sistema. In alcune acquisizioni sono stati modificati parametri di fuoco e diaframma delle telecamere per peggiorare volutamente le prestazioni del sistema.

Implementata la piattaforma grafica per la gestione delle immagini e

degli algoritmi si sono progressivamente sviluppate le diverse soluzioni .Come già accennato precedentemente Come già spiegato precedentemente prima del riconoscimento di forma viene eseguita un operazione di compressione dei dati. L'aggiunta della codifica RLE non modifica il comportamento dell'algoritmo di blob analysis sensibilmente, infatti come si può osservare dal seguente esempio nelle righe di codice riportato in seguito, con dati non compressi l'algoritmo è costretto ad elaborare elemento per elemento (rappresentato da un ciclo for nel codice 3.1), con dati compressi vengono elaborati blocchi di elementi compressi, che sono elementi sequenziali uguali (rappresentato da un ciclo while nel codice 3.2).

Listing 3.1: Gestione immagini non compresse

```
\\W lunghezza riga;
for i=1:w
        \\ELABORAZIONE elemento in posizione i;
end
```

Listing 3.2: Gestione immagini compresse

```
\\W lunghezza riga;
bloccocompresso=[ripetizione elemento];
indiceblocco=1;
indiceelemento=1;
while indice<w
        \\ELABORAZIONE elemento bloccocompresso(
            Indiceblocco,2);
        indiceelemento= indiceelemento+
            bloccocompresso(indiceblocco,1);
  indiceblocco= indiceblocco+1;
end
```

La compressione RLE permette un miglioramento in termini di prestazioni di calcolo ma non è determinante nella valutazione e scelta dell'algoritmo. Per questo motivo la compressione verrà considerata solo più

avanti per alcune considerazioni affidabilistiche del sistema. D'ora in poi si descriverà l'algoritmo per la blob analysis.

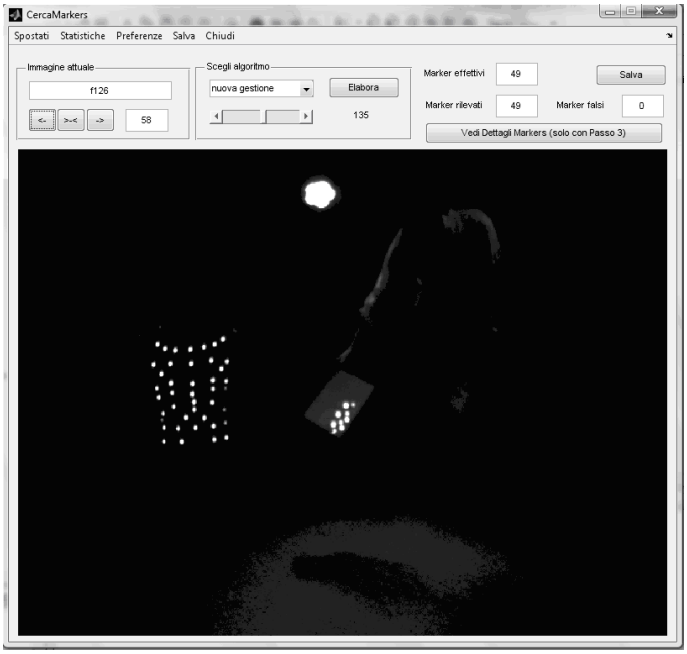

Figura 3.1: Interfaccia per la gestione delle immagini e l'algoritmo

In figura 3.1 si può ossere la schermata principale della piattaforma creata. Tra i comandi principali viene messa in evidenza una toolbar che seleziona il livello di sogliatura, un menù a tendina per selezionare l'algoritmo, una funzione che permette di ottenere informazioni sul blob semplicemente cliccando sopra il blob nell'immagine, la pulsantiera a sinistra serve per scorrere le immagini o raggiungere direttamente un'immagine desiderata.

Una volta che si elabora un'immagine è possibile cliccando sopra un blob o sopra il comando preposto per visualizzare le statistiche dell'immagine. In figura 3.2 si può osservare come per ogni blob si abbia a disposizione l'immagine, l'immagine sogliata, i fattori di forma, area ed altri parametri. In più è interessante osservare il profilo del *marker*, profilo da cui l'algoritmo dedurrà che questo blob descrive 2 *marker* fusi.

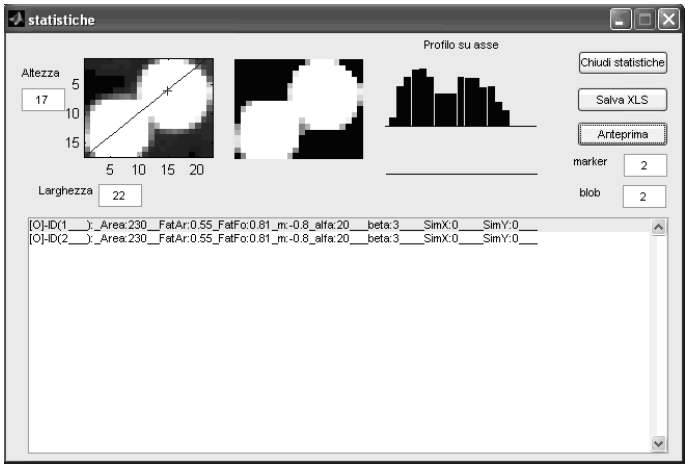

Figura 3.2: Interfaccia per la gestione delle statistiche di una singola immagine

3.2 Descrizione dell'algoritmo

Il cuore della procedura di blob analysis è l'algoritmo di segmentazione, questo a sua volta sfrutta l'algoritmo per il riconoscimento dei *marker* e gestione della fusione tra blob. Nei diagrammi 3.3, 3.4 e 3.5 si può trovare una rappresentazione a blocchi dei metodi appena citati.

Il processo che individua i blob e gestisce il riconoscimento dei *marker* è la segmentazione. L'algoritmo di segmentazione vuole in ingresso l'immagine da acquisire e i parametri di riconoscimento dei marcatori. L'immagine viene analizzata riga per riga, in questo modo è possibile mantenere in memoria solo una riga di pixel e le parametrizzazioni dei blob ricavati. Innescata l'analisi della prima riga l'algoritmo ricercherà il primo pixel soprasoglia e se non viene trovato il pixel che soddisfa la condizione, la riga viene scartata e caricata in memoria la successiva. Nel caso si usi la compressione RLE invece di confrontare tutti gli elementi basterà confrontare solo l'elemento ripetuto.

Raggiunto il primo pixel soprasoglia, questo verrà mantenuto in memoria come i successivi pixel soprasoglia a contatto con il precedente. Su tale serie di pixel viene calcolata la somma delle intensità, che rappresenterà il peso della riga nel blob, in più verrà calcolata la media pesata e

la posizione del primo ed ultimo pixel nell'immagine, da cui ci si ricava la larghezza della riga del blob. La serie di pixel viene cancellata dalla memoria ma viene mantenuta la sua rappresentazione parametrica. Individuata la prima riga di un blob si continuerà la ricerca di una successiva serie di pixel soprasoglia.

Individuata una seconda serie di pixel, e calcolata la sua rappresentazione parametrica si eseguiranno dei confronti per ricavare se la nuova serie di pixel faccia parte di un precedente blob trovato. Detto questo si intende che una riga di pixel fa parte di un blob se uno dei suoi pixel è ortogonalmente connesso all'ultima riga del blob. Una riga sarà connessa al blob solo verticalmente, perché se fossero connesse orizzontalmente verrebbe rilevata una singola riga.

Può capitare però che una riga connetta due blob, in tal caso entrerà in gioco l'algoritmo di fusione tra blob. Riassumendo la riga di pixel ricerca nel vettore contenente i blob, il blob di cui può far parte, se non lo trova crea un nuovo blob. Le rappresentazioni parametriche di un blob vengono memorizzate in un vettore, tale vettore sperimentalmente mantiene in memoria non più di 5 o 6 blob poiché anche in immagini dense di *marker*, difficilmente rimangono tutti allineati. Infatti perché dei blob rimangano in memoria dovrebbero svilupparsi parallelamente nelle stesse righe. Si vedrà in seguito che una volta che il blob è chiuso, cioè le serie di pixel sulla riga non si attaccano al blob, le rappresentazioni del blob verranno analizzate per valutare se rappresentano o no un marcatore. Che un blob venga riconosciuto come *marker* o meno, il blob viene comunque cancellato dal vettore. Una serie di pixel può appartenere ad un blob, anche se appartiene già ad un blob della medesima riga, questo perché un blob si può diramare in diverse direzioni verso il basso, in tal caso la parametrizzazione di quest'ultima serie di pixel influirà sulla precedente perché appartenenti alla medesima righa.

Per alleggerire la memoria del vettore contenente le rappresentazioni dei blob, all'inizio dell'elaborazione di ogni riga viene verificato se alcu-

ni blob siano chiusi. Se un blob è chiuso viene lanciata l'applicazione per valutare se tale blob è un *marker*, in entrambi i casi il blob viene successivamente rimosso dalla memoria. Un blob è chiuso se l'ultima sua riga inserita risale a due righe precedenti (questa operazione non viene effettuata nelle analisi delle prime due righe dell'immagine per ovvi motivi). In seguito alla ricerca di blob chiusi, nei blob aperti vengono ricercate righe in comune, cioè se due blob hanno una serie di pixel in comune appartenenti alla medesima riga viene lanciato l'algoritmo di fusione, anche in tal caso rimarrà un solo blob poiché uno dei due verrà cancellato.

Finita l'elaborazione di tutte le righe, si svuota il vettore dai blob ancora aperti. Questa ultima operazione è abbastanza rara visto nelle acquisizioni si evita l'acquisizione dei marcatori ai margini delle immagini, ma può capitare per soggetti in movimento. Alla fine dell'elaborazione dell'immagine il vettore dei blob sarà vuoto ma esisterà un vettore con le coordinate, i dati statistici e i *marker* individuati; quest'ultimo vettore tornerà in uscita al programma per una visualizzazione grafica.

L'algoritmo per il riconoscimento dei *marker* vorrà in ingresso un vettore contenente la parametrizzazione dei blob e i parametri di riconoscimento. Parallelamente a quanto già illustrato in figura 3.4, ogni blob viene prima processato secondo la propria area. Un *marker* da individuare ha dimensioni note, però variando la distanza dalla telecamera l'area della proiezione del *marker* nel piano immagine varierà sensibilmente. Da questo si può dedurre che non è detto che un blob piccolo corrisponde alla proiezione di *marker* piccolo, perché può corrispondere alla proiezione di qualsiasi *marker* lontano. La conoscenza preliminare della posizione delle videocamere e il volume di studio è determinate per valutare la dimensione minima e massima della proiezione del *marker* sul piano immagine. La prima discriminazione dei blob avviene perciò con un confronto di area, cioè l'area del blob deve essere compreso tra l'area minima e massima della proiezione del *marker* sul piano immagine.

I blob candidati ora si divideranno tra blob di dimensioni molto pic-

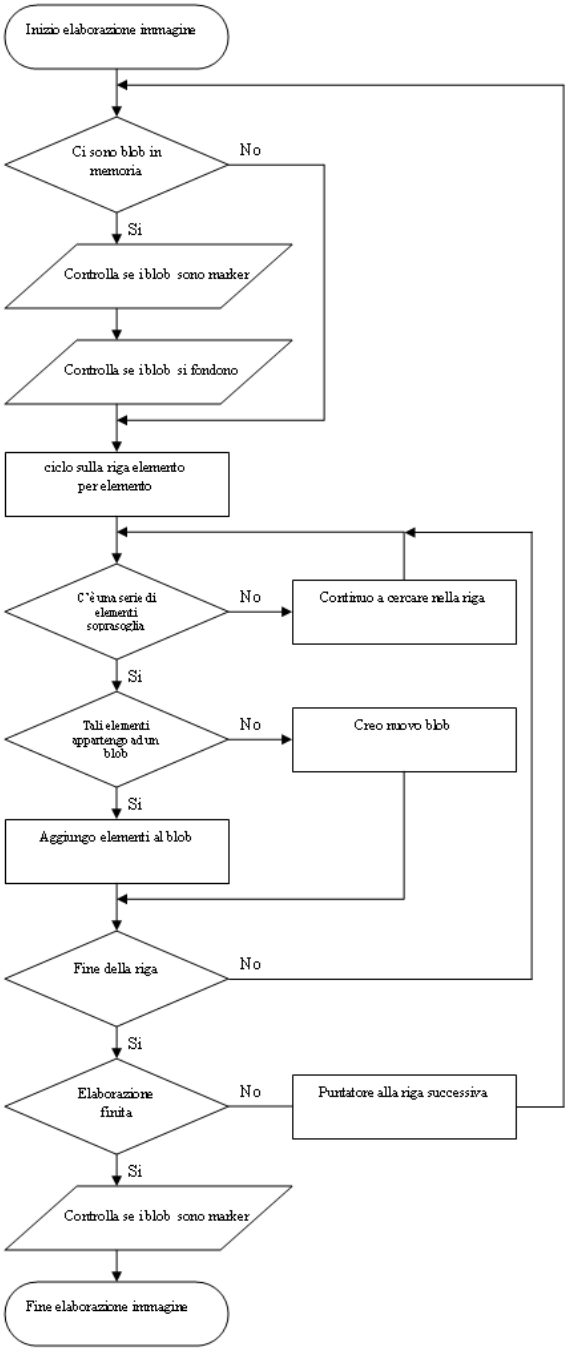

Figura 3.3: Flusso dei dati nell'algoritmo per individuare un blob

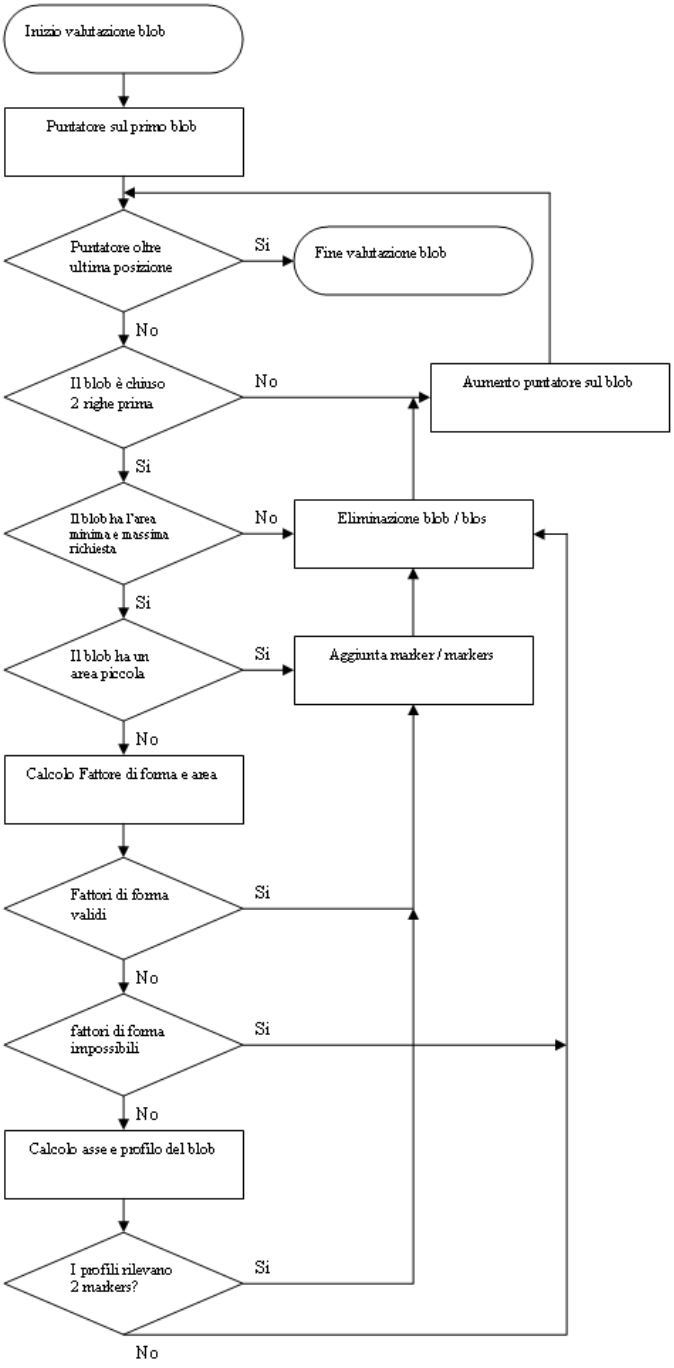

Figura 3.4: Flusso dei dati nell'algoritmo per individuare un marker

cole e le restanti, dove l'area massima per definire un *marker* piccolo è stata definita sperimentalmente tra i 10 e i 15 pixel. Sui blob piccoli infatti non si potranno fare test di riconoscimento di forma e se esistono *marker* riconoscibili con aree cosi piccole, allora i blob con aree piccole vengono promossi automaticamente a *marker*, aggiungendo qualche altro vincolo per eliminare blob sviluppati su una sola riga o colonna. I blob rimanenti avranno dimensioni sufficienti per subire un'analisi più approfondita.

I rimanenti blob verranno riconosciuti come *marker* se i loro fattori di forma e di area sono compatibili con quelli di un *marker* sferico. I blob rimanenti subiranno un'analisi più approfondita: per ogni blob si calcolerà un asse principale, su tale asse si calcolerà la proiezione del *marker*. Come abbiamo visto nel capitolo precedente il profilo, e in questo caso la proiezione del *marker*, ci fornisce informazioni utili sulla forma del *marker*, idealmente a campana. Un singolo *marker* avrà idealmente un profilo con un singolo massimo senza minimi, al contrario la fusione di due *marker* avrà due picchi con un minimo tra i due picchi. Questo ragionamento vale anche per *marker* fusi di diversa dimensione, al contrario non vale per la fusione di 3 o più *marker* poiché 3 *marker* si possono disporre in modi diversi, e non essere allineati. I blob rimanenti vengono irrimediabilmente scartati.

L'algoritmo di fusione dei *marker* confronta i blob due a due per verificare se hanno righe in comune. In tal caso i blob vengono fusi, sommando i pesi delle medesime righe, calcolando delle nuove medie pesate sulle righe e dei nuovi valori di inizio e fine del blob (vengono mantenuti solo gli estremi).

Nel grafico 3.7 viene evidenziato come i *marker* di piccola area superano in numero quelli di area superiore, si ha interesse perciò a processarli molto velocemente. Parte dei restanti marcatori hanno fattore di area e forma compatibile con *marker* sferici e solo un 7% richiede l'analisi di tutte le caratteristiche dei marcatori.

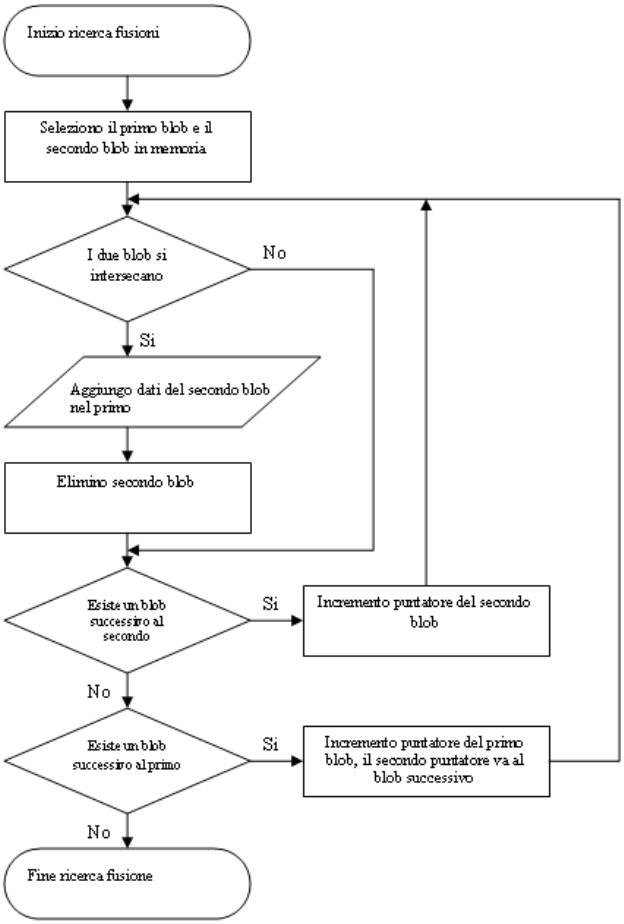

Figura 3.5: Flusso dei dati nell'algoritmo che gestisce le fusioni

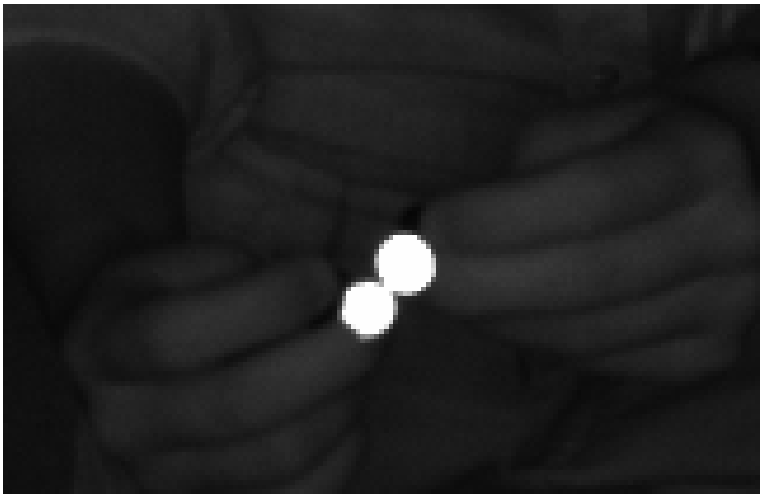

Figura 3.6: Fusione di due marker

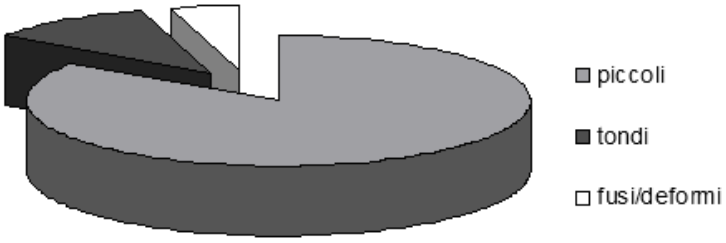

Figura 3.7: Tipologie di marker rilevati

3.3 Ricerca dei parametri ottimi

La procedura appena illustrata risulterà inefficiente se non si pone a priori una rappresentazione corretta dei marcatori. Esistono tre vincoli fondamentali per il riconoscimento dei *marker*: il livello di sogliatura, il fattore di forma e il fattore di area. Di seguito verranno fatto alcune considerazioni nella scelta del valore corretto di questi tre parametri.

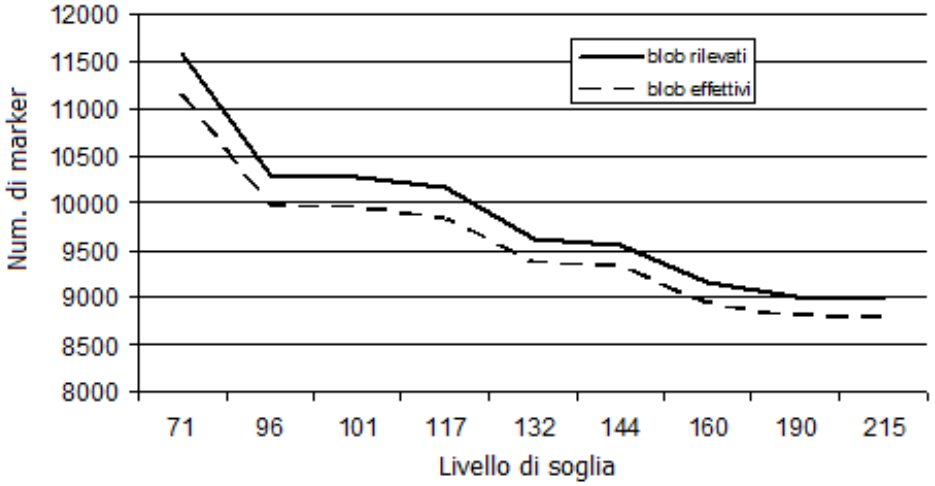

Figura 3.8: Numero di marker rilevati in funzione del livello di sogliatura

3.3.1 Sogliatura

Il livello di sogliatura viene definito trovando il minimo errore commesso nell'individuazioni di falsi positivi e negativi. Precedentemente è stata illustrata la principale tecnica per il calcolo del livello di sogliatura. Ora si volge l'attenzione al grafico 3.9. Nel grafico indicato si osserva che il numero di *marker* individuati varia in funzione del livello di sogliatura, maggiore sarà il livello di sogliatura maggiore sarà l'affidabilità dei *marker* che vengono acquisiti ma quantitativamente in numero inferiore. La sogliatura infatti è in grado di eliminare componenti parassite quali riflessi o rumore, ma può eliminare anche *marker* di piccola intensità luminosa e di ridotta dimensione. In più alzando il livello di soglia si perde l'informazione sull'area del blob, informazione che contribuisce positivamente nel calcolo del baricentro del *marker*.

3.3.2 Dimensioni del marker

Note sul fondatore ed associazione.

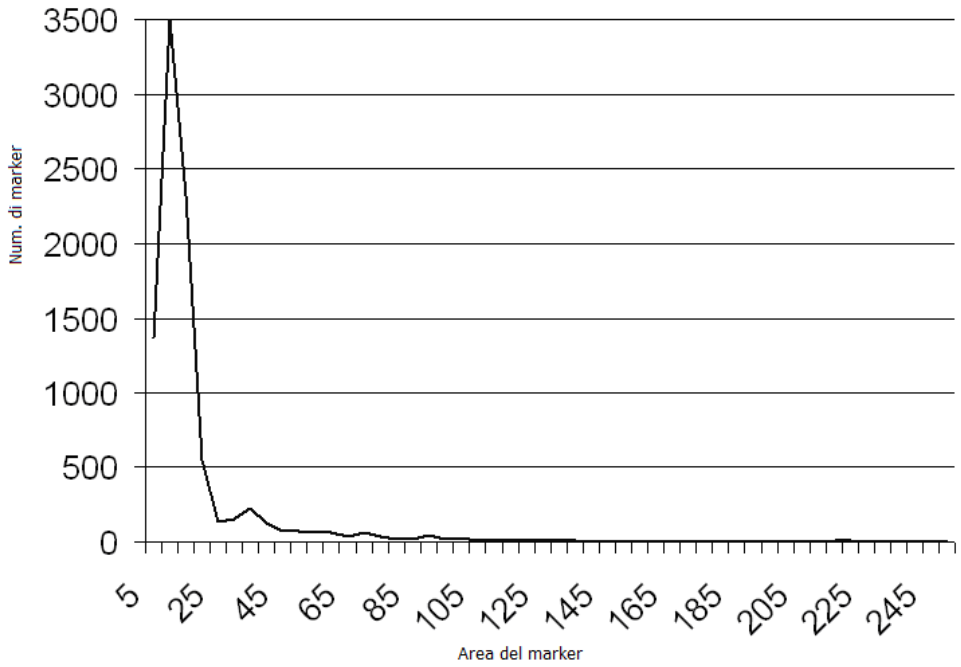

Figura 3.9: Numero di marker rilevati in funzione dell'area misurata in pixel

3.3.3 Fattore di area

Come già accennato precedentemente il fattore di area è un numero compreso tra 0 e 1, dove un *marker* sferico dovrebbe avere un valore idealmente di $0,78$. Per errori introdotti dalla griglia di campionamento rettangolare nel piano immagine la forma corretta viene spesso degradata, di conseguenza anche il fattore di area varierà sensibilmente. Sono state distinte le casistiche di blob di area piccola e non piccola per la ricchezza di informazioni che possiedono nella loro rappresentazione. I blob di piccola area sono caratterizzati da fattori di area generalmente maggiori di $0,78$, succede il contrario per *marker* non piccoli. Nei grafici 3.11 e 3.10 sono rappresentate rispettivamente le densità dei fattori di area per *marker* piccoli e *marker* medio grandi. Da questi grafici si possono dedurre due cose: i *marker* piccoli hanno i pixel concentrati su valori intorno l'uno, perciò un fattore di area generalmente alto, contrariamente i *marker* grandi hanno

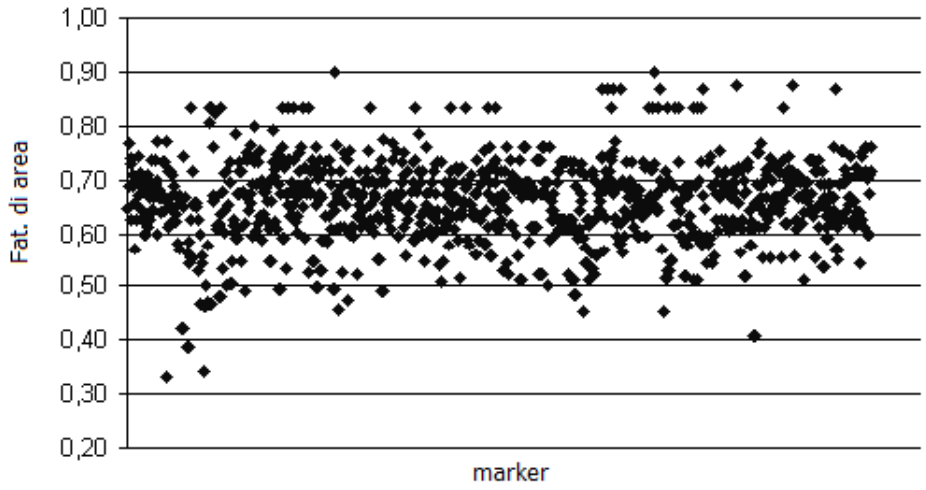

Figura 3.10: Fattore di area dei marker validi con area superiore ai 25 pixel

un fattore di area generalmente intorno allo $0,7$. Generalmente si ottiene un fattore di area inferiore nei *marker* grandi rispetto al valore ideale perché spesso tali *marker* non sono uniformemente illuminati, ragion per cui al momento della sogliatura si perde una frazione di area.

Sperimentalmente si è dedotto che un *marker* viene riconosciuto correttamente imponendo un fattore di area maggiore di $0,65$.

3.3.4 Fattore di forma

Come già accennato in precedenza anche i fattori di forma differiscono per *marker* con area piccola e medio grande. Nei grafici 3.12 e 3.13 si possono fare le seguenti considerazioni: i *marker* con area piccola risentono di una maggiore deformazione dovuta al campionamento della griglia rettangolare nel piano immagine, i *marker* con un area grande raggiungono facilmente fattori di forma compatibili con quello unitario.

Sperimentalmente si è dedotto che un *marker* viene riconosciuto correttamente imponendo un fattore di forma maggiore di $0,8$.

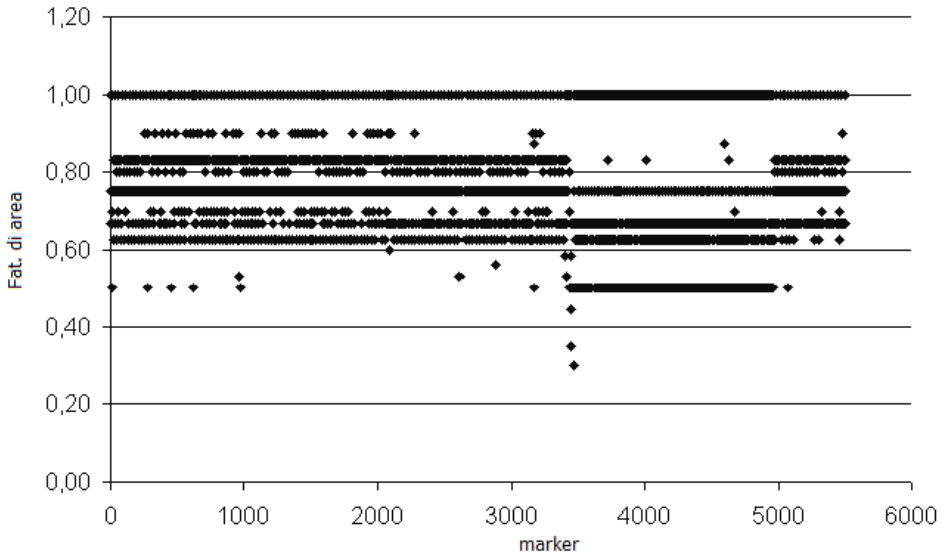

Figura 3.11: Fattore di area dei marker validi con area inferiore ai 25 pixel

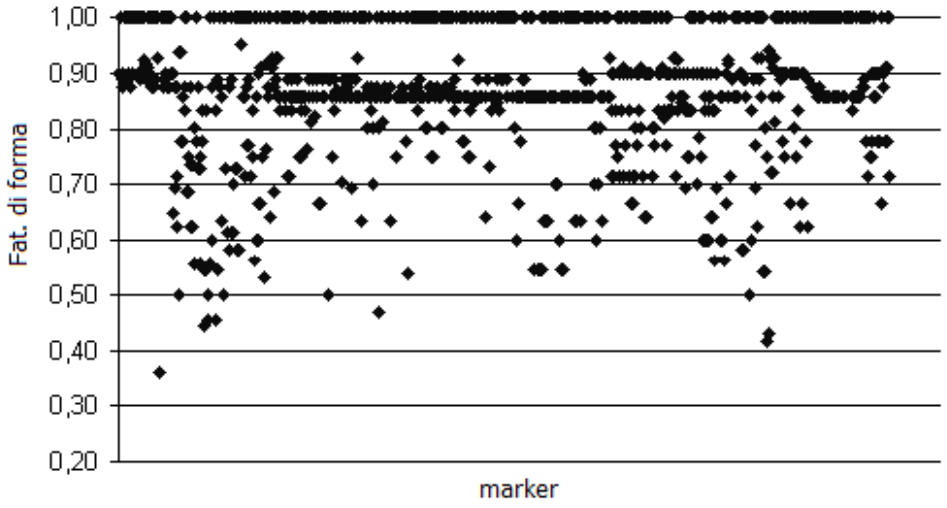

Figura 3.12: Fattore di forma di marker validi con area superiore ai 25 pixel

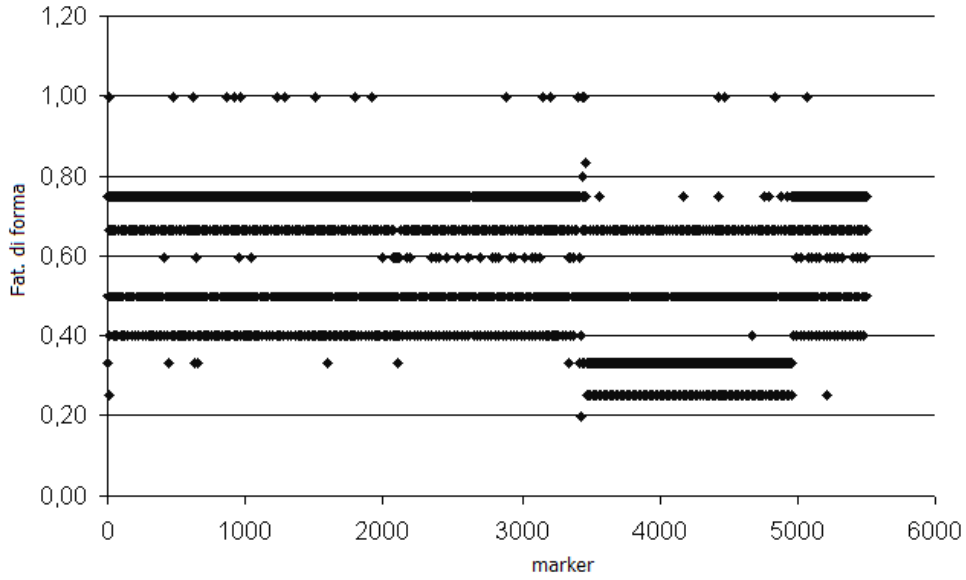

Figura 3.13: Fattore di forma di marker validi con area inferiore ai 25 pixel

3.4 Confronto dei risultati

Di seguito vengono comparati i risultati delle elaborazioni di 310 immagini dove sono presenti 9400 *marker* tramite un classico algoritmo di blob analysis usato attualmente in diversi sistemi stereofotogrammetrici con quello illustrato nel capitolo precedente. In risultati della comparazione sono stati ottenuti usando parametri standard sia per l'algoritmo classico sia per quello sviluppato in questa tesi.

Inizialmente si costruisce un database dei *marker* che si debbono ricavare su tutte le immagini, costruito manualmente poiché ogni tecnica proposta è comunque soggetta a qualche tipo di errore. Succesivamente si elaborano tutte le immagini con una tecnica di riconoscimento sfruttata in diversi sistemi in vendita, questo è basato sulla dimensione dei *marker* che verrà chiamato d'ora in poi 'classico'.Le immagini vengono poi elaborate anche con l'algoritmo proposto in questo elaborato che verrà chiamato 'nuovo'. Si sceglie lo stesso livello di soglia e si mette in relazione l'area massima dell'algoritmo nuovo con la larghezza ed altezza dell'algoritmo

classico, cioè in modo che l'area massima sia uguale al prodotto tra altezza e larghezza. In questo modo si ricaveranno *marker* con le medesime dimensioni prefissate.

Riassumendo l'algoritmo classico processa ogni blob, e secondo le sue dimensioni verrà promosso a *marker* o meno. L'algoritmo nuovo processa ogni blob in base all'area, successivamente secondo il fattore di forma e fattore di area, successivamente valutando il profilo del *marker* proiettato sull'asse principale. Nell'istogramma 3.14 si può vedere il risultato complessivo delle due elaborazioni in relazione al numero di *marker* effettivamente presenti. Con l'algoritmo classico si raggiunge un risultato di circa 8800 su 9400 *marker* presenti. Questo primo risultato rappresenterà un punto di partenza del nuovo algoritmo, infatti sia l'algoritmo classico che quello nuovo usano il medesimo processo di segmentazione perciò individueranno lo stesso numero di blob su cui verrà eseguita l'analisi preliminare di area. L'algoritmo nuovo raggiunge un risultato di circa 9150 su 9400 *marker* presenti, il che significa che individuati gli 8800 blob dell'algoritmo classico, classifica i blob: alcuni blob vengono promossi a *marker*, altri vengono scartati, altri vengono interpretati come fusione di *marker*.

La forza dell'algoritmo nuovo non sta nell'eliminazione dei blob che non sono marcatori perfetti, ma nel riconoscimento dei marcatori imperfetti e fusi. Si ricorda che l'errore di falso negativo è quando un *marker* che non viene riconosciuto come tale, un falso positivo è quando un blob viene riconosciuto come *marker* anche se non lo è. Come già detto in precedenza i falsi negativi sono molto più dannosi nelle successive fasi di *tracking* e labeling rispetto i falsi positivi, ragion per cui l'esigenza primaria è quella di ridurre al minimo i falsi negativi. Il nuovo algoritmo riesce a dimezzare l'errore nell'individuazione dei *marker* grazie al riconoscimento delle fusioni tra *marker*.

Interessante osservare l'elaborazione di un'immagine tramite algoritmo classico: nell'immagine 3.15 si può distinguere sulla sinistra lo sciame di *marker* semisferici di 0,5 cm posti sul un busto di un manichino, in

alto un emettitore infrarosso di una telecamera, a destra l'operatore che sovrappone due *marker* attaccati a due bacchette. Sulla destra si osserva che i due *marker* seppur di dimensioni diverse, sono fusi ma distinguibili. L'algoritmo classico è in grado di individuare solo il blob e non la fusione, invece se si osserva nell'immagine 3.16, dove è stata compiuta l'elaborazione con il nuovo algoritmo, sono stati individuati i due *marker*. Il blob in alto generato dall'emettitore viene scartato in entrambi i casi per dimensioni eccessive.

Bisogna sottolineare che le casistiche di fusioni combinate di 3, 4, 5, 6 o più *marker* potrebbero essere individuate solo tramite algoritmi di blob analysis ad alta complessità di calcolo come cross-correlazione o trasformata di Hough. Il limite di entrambi gli algoritmi consiste proprio nella gestione delle fusioni multiple, ma in *real time* soluzioni complesse risultano ingestibili per la complessità di calcolo elevata. In figura 3.17 si osserva ad esempio una casistica non gestita di 4 *marker* non gestiti.

Il nuovo algoritmo può soffrire di falsi positivi, però sui 350 *marker* che vengono rilevati in più su nuovo algoritmo rispetto a quello classico solo 15 risultano essere falsi *marker*, errore accettabile considerato il fatto che viene introdotto un falso positivo che è un errore che non disturba gli algoritmi di *tracking* e *labelling*. Il risultato di 15 *marker* calcolati erroneamente va messo in relazione con i 9150 calcolati, e rispetto gli 8800 *marker* ricavati con il precedente algoritmo rispetta una quantità di errore trascurabile.

3.5 Affidabilità dei prodotti

Inizialmente si definiscono alcune proprietà di un sistema: quali l'affidabilità, tolleranza ai guasti, efficienza e robustezza.

L'affidabilità è la probabilità che un qualsiasi dispositivo o sistema funzioni correttamente per un dato tempo e in certe condizioni operative. Il problema dell'affidabilità nei prodotti BTS è stato affrontato seguendo di-

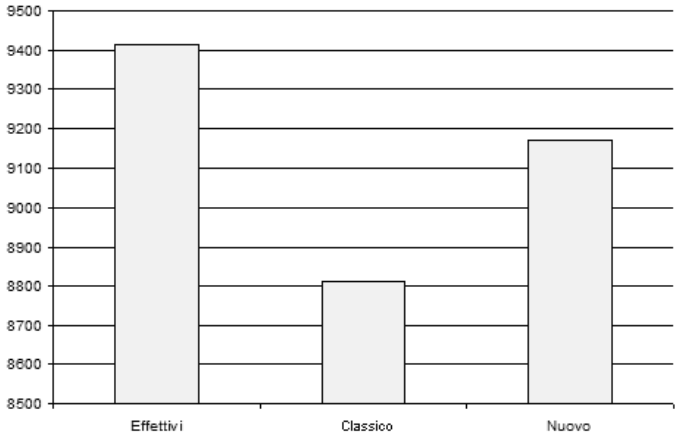

Figura 3.14: Confronto marker rilevati con algoritmo classico e quello proposto

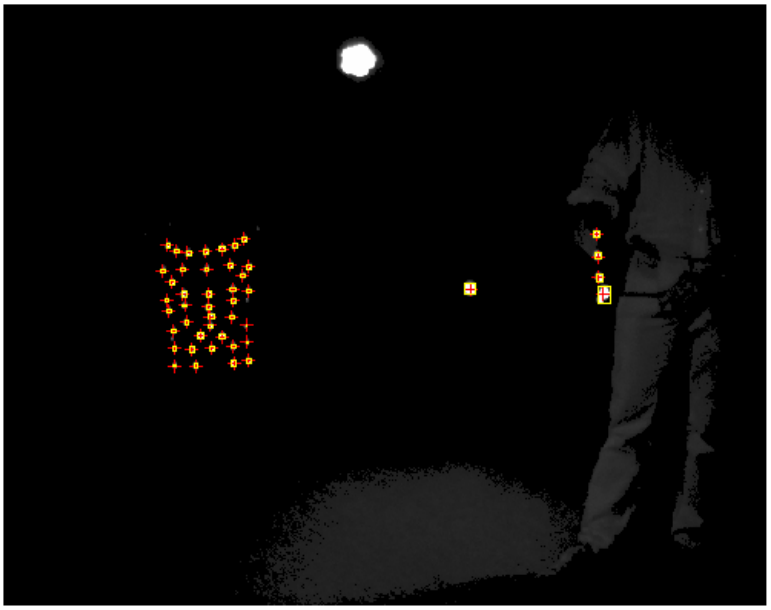

Figura 3.15: Elaborazione dell'immagine con un sistema classico basato sulla sogliatura

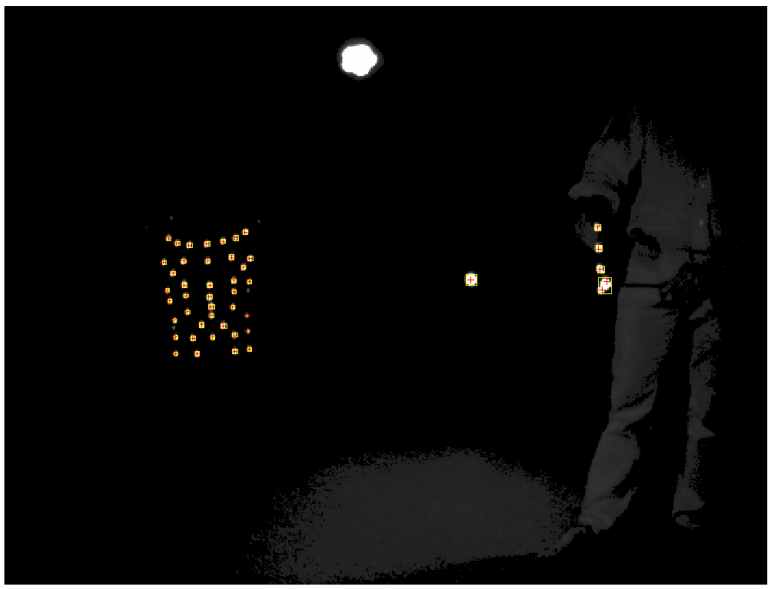

Figura 3.16: Elaborazione dell'immagine con il nuovo algoritmo

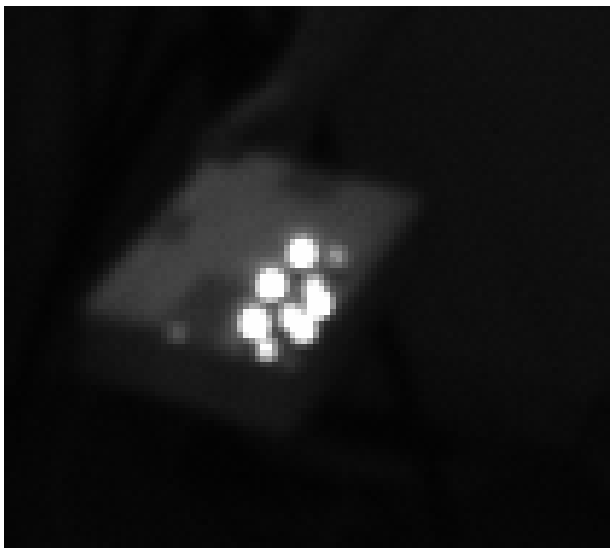

Figura 3.17: Esempio di fusioni multiple non individuabili da un algoritmo in real time considerato fin ora

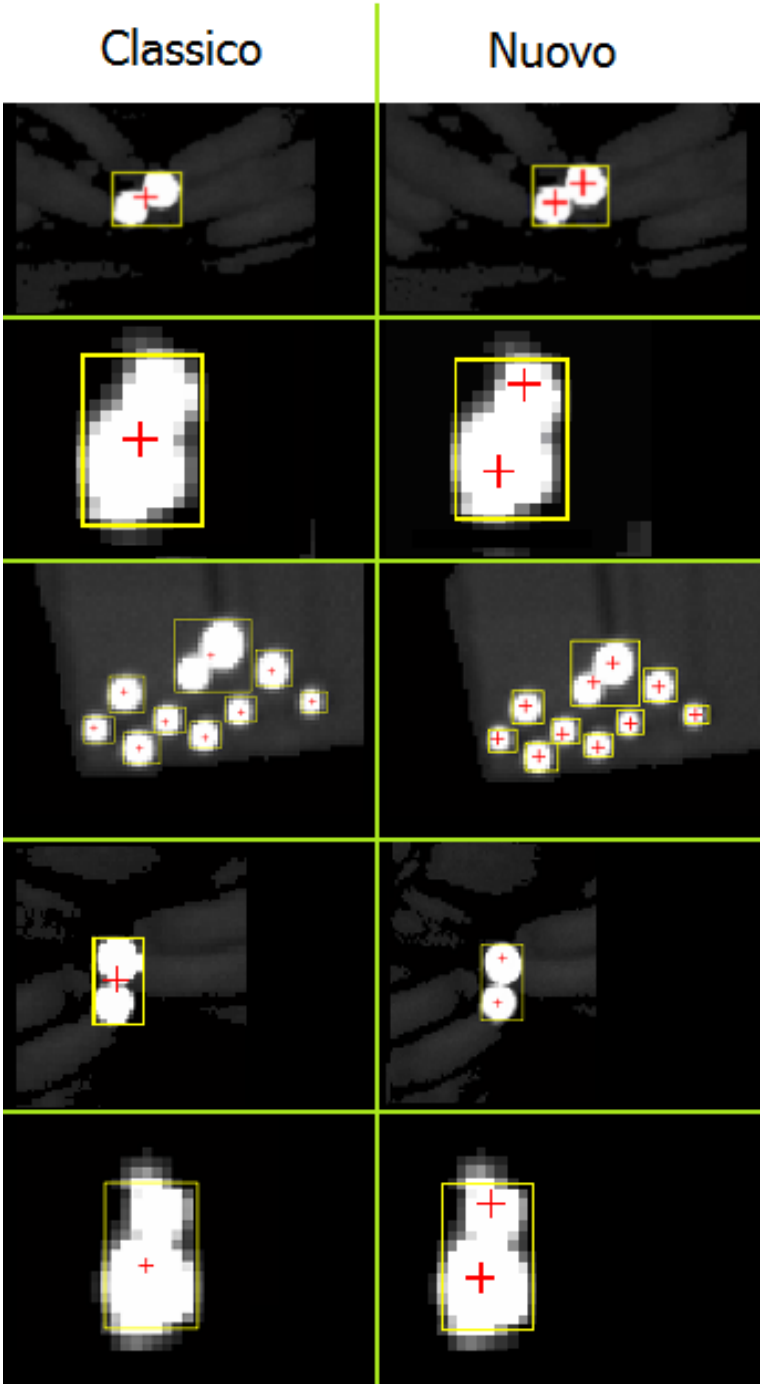

Figura 3.18: Alcuni esempi di fusioni tra marker individuati con il nuovo algoritmo e non il vecchio

verse filosofie di progetto, assecondando il rapido sviluppo delle tecnologie informatiche e il loro sistematico uso nella fase di progettazione.

Un'altra definizione dell' affidabilità è basata sul concetto di MTBF (Mean Time Between Failure, il tempo medio che intercorre fra due fallimenti successivi). Questa seconda definizione tiene conto del fatto che malfunzionamenti gravi si considerano solitamente più influenti di errori minori.

La tolleranza ai guasti, o fault tolerance, è la capacità di un sistema di non subire interruzioni di servizio anche in presenza di guasti. È importante sottolineare che la tolleranza ai guasti non garantisce l'immunità da tutti i guasti, ma solo per i guasti per cui è stata progettata. In più la tolleranza ai guasti può portare al peggioramento di altre prestazioni per cui nella progettazione di un sistema è necessario trovare adeguate ottimizzazioni e compromessi.

La robustezza di un sistema è la misura delle situazioni impreviste, non contemplate dalle specifiche, in cui il sistema si comporta in modo ragionevole. Situazioni di questo tipo riguardano errori ed eccezioni di varia natura (dati di input scorretti, fallimenti di componenti software o hardware esterni al sistema e interagenti con esso, e così via).

Un sistema è efficiente se usa le risorse di sistema in modo proporzionato ai servizi che svolge. Efficienza e prestazioni sono difficilmente predicibili e quindi non raramente vengono prese in considerazione solo a sistema realizzato; tuttavia, gli interventi a posteriori per migliorare il comportamento del sistema rispetto a questi parametri sono spesso estremamente difficili e costosi.

Tralasciando le caratteristiche del sistema SMART, si valuta come l'applicazione dell'algoritmo proposto influenza il sistema. Si suddivide l'algoritmo proposto in due passi: compressione ed elaborazione dei dati. La compressione dei dati permette al sistema di migliorare le sue prestazioni in termini di efficienza, in particolare non si avrà più un tempo di computazione fissato per ogni immagine ma varierà al variare del numero

di pixel non sequenziali e soprasoglia, il caso peggiore sarà un'immagine con molti blob (ad esempio riflessi, *marker* e falsi *marker* con l'aggiunta di rumore) addensati in un area ristretta dell'immagine.

La compressione dei dati permette in condizioni operative standard di ottenere prestazioni migliori, a sfavore di questa tecnica si ha una più alta vulnerabilità ad errori hardware e la necessità dell'uso di algoritmi per la correzione dei dati, tipo 'bit di parità' implementabile anche via hardware. Nei dati non compressi la corruzione del singolo bite generalmente non ha conseguenze rilevanti, ad esempio se un singolo bite dello sfondo viene corrotto ed appare come saturo al più viene identificato come *marker* da una singola telecamera, se un singolo bite appartenente al blob viene corrotto e appare come sfondo al più nel calcolo del baricentro del blob sarà presente un errore inversamente proporzionale all'area del blob considerato. Nei dati compressi la corruzione del singolo bite può avere effetti molto più gravi, ad esempio se viene corrotto il bite (anche solo un bit) che conta le ripetizioni i pixel della riga verranno contati in modo errato, le righe dell'immagine avranno dimensione differente perciò l'intera immagine può essere persa. Nel caso in cui viene corrotto il bite che indica l'intensità di un pixel ripetuto n volte, se questo era sfondo ma appare come valore soprasoglia può unire diversi blob innescando fusioni non volute, se il valore corrotto era soprasoglia e appare come sottosoglia, il blob viene diviso in due parti e conseguentemente verranno individuati due *marker* in posizioni errate e non più uno nella posizione corretta. La compressione può introdurre un abbassamento della tolleranza ai guasti. Si ricorda che diversi protoccoli di trasmissione, anche ethernet, gestiscono il bit di parità per il recupero di dati.

L'elaborazione dell'algoritmo proposto inserisce operazioni aggiuntive, perciò si ha un aumento della complessità computazionale ma si è in grado di classificare i blob ed elaborarli in maniera distinti secondo la complessità. Come succede per la compressione il sistema non ha più tempi prefissati di elaborazione ma a seconda della qualità delle immagini

avranno tempi di calcolo differenti. Il metodo proposto aumenta sensibilmente la robustezza del sistema, infatti la classificazione dei blob permette di controllare il tipo di *marker* che si acquisisce, valutare l'errore che si commette e gestire eventuali anomalie che si possono riscontrare.

Complessivamente il sistema guadagna sia in affidabilità che robustezza applicando l'algoritmo proposto.

3.6 Passi futuri

La prosecuzione naturale di questo lavoro è la trasposizione del codice in linguaggi gestiti dai sistemi quali il C e dei test sulle prestazioni effettive. Cosa ancor più importante è che partendo dalla piattaforma MatLab sarà possibile testare nuovi algoritmi o semplici modifiche, perciò si sono gettate le basi per delle nuove strategie di sviluppo e test del sistema.

Nel panorama della computer vision e stereofotogrammetria stanno emergendo tecniche markerless [19]. Sistemi markerless sono stati sviluppati a Stanford, MIT, e Max Planck Institute e hanno il vantaggio che non richiedono marcatori sul soggetto studiato ed illuminatori ad infrarossi. L'acquisizione si basa su delle riprese video sincronizzate a colori in un ambiente controllato, il sistema riconosciuto lo sfondo sarà in grado di riconoscere persone o oggetti che attraversano il volume di studio. La forza di questa tecnologia sta nella semplicità d'uso, lo svantaggio è quello che in ambienti esterni, con sfondi o illuminazioni mutevoli, il sistema ha difficoltà riconoscere il soggetto e calcolare i volumi studiati. In figura 3.19 viene messa in evidenza la possibilità di segmentare il corpo in volumi differenti, quali testa, arti e busto, e come viene definito il volume del soggetto a partire dalle singole immagini.

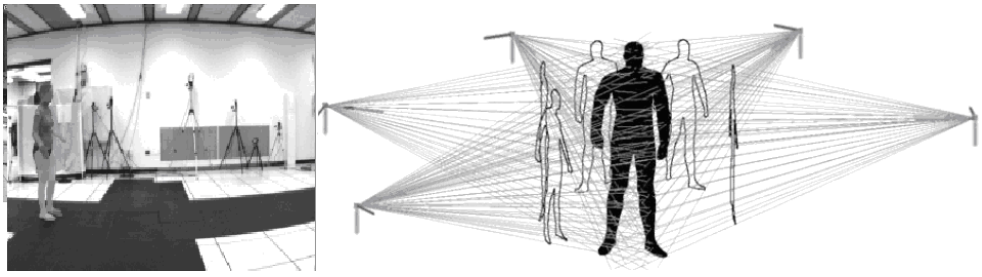

Figura 3.19: La tecnologia emergente markerless

Conclusione

Il presente lavoro di tesi ha permesso l'implementazione di un codice per il riconoscimento dei *marker*dalle immagini stereofotogrammetriche in condizioni di *distraction*, *occlusion* e *fusion*. Innanzi tutto è stata creata una piattaforma in MatLab per la gestione sia dei dati acquisiti tramite stereofotogrammetria, sia dei codici per il riconoscimento dei *marker*. Dopo di che è stato sviluppato un codice modulato in diverse funzioni, in modo da permettere la loro sostituzione o rielaborazione senza che fosse necessario alterare sensibilmente il flusso dei dati nel sistema. La piattaforma sviluppata ha permesso di gestire un set di immagini, in modo che potessero essere elaborate con diversi algoritmi (per sperimentare diverse tecniche di *blob analysis*), e che questi a loro volta fossero modificabili. Ogni algoritmo è stato implementato in modo da richiedere in ingresso un vettore di parametri che servono ai diversi algoritmi per definire quali blob sono *marker*.

L'esigenza di elaborare le immagini in *real time* ad alta frequenza ha imposto un centellinamento delle operazioni di calcolo, è stato quindi necessario eseguire un'operazione preliminare di compressione sulle immagini, per eseguire in seguito la *blob analysis* tramite algoritmi semplificati. Le funzioni di compressione dei dati tramite la codifica Run length encoding, permettendo una riduzione delle operazioni di elaborazione, hanno consentito di sfruttare le operazioni residue per una gestione migliore del riconoscimento dei *marker*.

Per quanto concerne la segmentazione (l'operazione che sta alla base

del riconoscimento di forma) al fine di mettere in evidenza i blob dallo sfondo, e il riconoscimento dei *marker* in *real time* si sono considerate le varie tecniche note in letteratura come cross-correlazione, *template matching*, descrittori di Fourier, momenti geometrici e trasformata di Hough. Un'analsis approfondita delle tecniche sopraelencate ha permesso di evidenziare come queste richiedano una complessità di calcolo troppo elevata nonostante permettano di conseguire ottimi risultati. Si è pertanto scelto di utilizzare i fattori di forma e l'analisi della proiezione sull'asse principale del blob in quanto si sono dimostrati un buon compromesso tra complessità di calcolo e affidabilità dei risultati.

L'adeguatezza di una scelta simile è supportata dai risultati riportati nel terzo capitolo dove si può osservare come l'algoritmo sviluppato consenta di dimezzare il numero di *marker* non riconosciuti rispetto all'applicazione del metodo classico che si basa su un'analisi della dimensione del blob. Al fine di testare l'algoritmo si era infatti costruito un set di immagini contenente 9400, e di questi solo 8800 vengono riconosciuti con tecniche di *blob analysis* classiche basate su sogliatura in *real time*; mentre con l'introduzione dei fattori di forma nell'algoritmo di identificazione si ottiene il riconoscimento di 9150 marker. Infine un ulteriore importante vantaggio introdotto da questa tecnica è la classificazione dei *marker*, questo oltre a migliorare la gestione del riconoscimento dei *marker* è anche utile a fini statistici per valutare nuovi setup di laboratorio. Si può osservare infatti che casistiche di cattiva messa a fuco delle telecamere o una scelta di posizionare le telecamere non ottima possono introdurre immagini ricche di *distraction*, *occlusion* e *fusion*.

Appendice: Codice MatLab

L'applicazione Matlab già illustrata nel capitolo 4 si compone delle seguenti funzioni:

- **CercaMarkers.fig e CercaMarkers.m:** Gestisce tutta l'interfaccia grafica, lo scorrimento delle immagini, il salvataggio delle statistiche, la chiamata alle funzioni statistiche, modificare i parametri di selzione dei marker, salvataggio di immagini elaborate e altro;
- **blob2marker.m:** riceve in ingresso un vettore con le caratteristiche del blob e i parametri di selezione, in uscita restituisce l'eventuale marker con le sue caratteristiche;
- **blob2markerSimple.m:** riceve in ingresso un vettore con le caratteristiche del blob e le dimensioni massime di un marker, in uscita restituisce l'eventuale marker con le sue caratteristiche;
- **cercomaxminprofilo.m, pesicolonna.m, mediapesatapesi.m, calcolaprofili.m e pesimedirigacalcola.m:** in ingresso vogliono un vettore di intensità e restituiscono informazioni sul profilo, peso medio e complessivo;
- **elabMarker.m:** esegue la segmentazione dell'immagine e richiama blob2marker.m per il riconoscimento del marker;
- **elabMarkerBTS.m:** esegue la segmentazione dell'immagine e viene richiamata la funzione blob2markerSimple.m per il riconoscimento del marker;

- **fondiblob.m:** gestisce la fusione dei dati tra due blob
- **quattrocaratteri.m e creavetStatistiche.m:** gestisce alcune funzioni di scrittura di un file XLS per il salvataggio di dati di ogni blob elaborato;
- soglia.m: viene eseguita la sogliatura di un'immagine;
- **statistiche.fig e statistiche.m:** gestisce l'interfaccia grafica per la visione delle statistiche di ogni blob in un immagine;
- **trovam.m:** ricava il coeficiente angolare della retta passante per i punti di media pesata delle righe del blob;
- **vedistat.fig e vedistat.m:** gestisce le statistiche nel caso si lanci la funzione per elaborare tutte le immagini;

Viene riportato di seguito il codice della funzione che esegue la segmentazione dell'immagine, in ingresso vuole l'immagine e i parametri di selezione di un marker, in uscita viene restituito un vettore con le coordinate dei marker e un vettore di statistiche.

Listing 3.3: Codice MATLAB di elabMarker.m

```
%Questa funzione elabora l'immagine IMG e restituisce un
    vettore contenente dati dei marker individuati
function vet=elabMarker(sogliap,img,height,width,areamin
    ,areamax,fatvet,areamax2,fatvetelis,indimage)
deb=0; %per attivare debug mettere a 1
%inizializzo variabili/vettori usate:
nummark=0;   %numero dei marker trovati
coordmark=[]; %coordinate dei marker, ogni riga un
    marker con [y-x][ymin xmin ymax xmax]
vettmp=[]; %vettore contenente prima riga di possibili
    blob= [j,inz,fin,mediapes,som;...]
sizetmp=0; %numero dei pseudo blob su size tmp
```

```
vetblob={}; %vettore di celle contenente blob={[num
    righe,riga inz ,inz[],fin[],mediapes[],som[],area1,
    ultima rigapresente];... }
sizeblob=0; %numero blob
vetblobprofili={}; %vettore contenenti i profili dei
    marker calcolati
statblob=[]; %vettore contenente statistiche tutti i
    blob [ymin xmin ymax xmax m alfa beta]
statnum=0; %numero statistiche

%% Ciclo ogni riga
%cerco elementi di interesse per righe
for j = 1:height; %% %chiudere marker di 2 righe
    precedenti
     if(sizeblob>0 && j>3)
         hh=1; % indice per ciclare i blob per
             individuare marker
         while (hh<=sizeblob ) %inizio con cercare blob
             chiusi di due righe prima
              if(vetblob{hh,8}<(j-2))
             %algoritmo che gestisce la selezione dei
                 marker
              numrighepass=vetblob{hh,1};
              rigainzpass=vetblob{hh,2};
              inzpass=vetblob{hh,3};
              finpass=vetblob{hh,4};
              mediapespass=vetblob{hh,5};
              sompass=vetblob{hh,6};
              area1pass=vetblob{hh,7};
              ultimarigapresentepass=vetblob{hh,8};
             %inizio elaborazione selezione blob e
                 calcolo statistiche
```

```
datimarker = blob2marker( numrighepass,
    rigainzpass, inzpass, finpass,
    mediapespass, sompass,area1pass,
    ultimarigapresentepass ,fatvet,fatvetelis
    ,areamin,areamax,areamax2);
%fine elaborazione selezione blob e calcolo
    statistiche
y = datimarker{1,1};
x = datimarker{1,2};
ymin = datimarker{1,3};
xmin = datimarker{1,4};
w = datimarker{1,5};
h = datimarker{1,6};
markON = datimarker{1,7};
ymax = datimarker{1,8};
xmax = datimarker{1,9};
mcoef = datimarker{1,10};
alfa = datimarker{1,11};
beta = datimarker{1,12};
fattorearea = datimarker{1,13};
fform = datimarker{1,14};
yd = datimarker{1,15};
xd = datimarker{1,16};
simmetriaY = datimarker{1,17};
simmetriaX= datimarker{1,18};
profiloasse=datimarker{1,19};
pesoobliquoperpend=datimarker{1,20};

%inserisco marker se corrisponde alle
    caratteristiche
if(markON== 1)
   coordmark=cat(1,coordmark, [y x ymin xmin
      w h vetblob{hh,7}] );
```

```
    nummark=nummark+1;
elseif(markON== 2)
    duemarker=int32([y x [ymin; ymin] [xmin;
        xmin] [w; w] [h; h] [vetblob{hh,7};
        vetblob{hh,7}] ]);
    coordmark=cat(1,coordmark, duemarker );
    nummark=nummark+2;
end

%riempio vettore contenente statistiche tutti
    i blob
%[ymin xmin ymax xmax mcoef alfa beta area]
%temp=cat(2,profiloasse ,pesoobliquoperpend')
    ;
if(markON== 2)
    tempstatpass={profiloasse
        pesoobliquoperpend';profiloasse
        pesoobliquoperpend' };
    vetblobprofili=cat(1,vetblobprofili,
        tempstatpass );
    duestat=[ double([ymin; ymin]) double([
        xmin; xmin]) double([ymax; ymax])
        double([xmax;xmax]) double([mcoef;
        mcoef]) double([alfa; alfa]) double([
        beta;beta ]) double([vetblob{hh,7};
        vetblob{hh,7}]) double([fattorearea;
        fattorearea ]) double([fform;fform])
        yd xd [simmetriaY;simmetriaY] [
        simmetriaX;simmetriaX ] ];
    statblob=cat(1,statblob,duestat );
    statnum=statnum+2;
else
```

```
                tempstatpass={profiloasse
                    pesoobliquoperpend' };
76              vetblobprofili=cat(1,vetblobprofili,
                    tempstatpass );
                statblob=cat(1,statblob, [double(ymin)
                    double(xmin) double(ymax) double(xmax)
                     double(mcoef) double(alfa) double(
                    beta) double(vetblob{hh,7}) double(
                    fattorearea) double(fform) yd xd
                    simmetriaY simmetriaX]);
                statnum=statnum+1;
            end
            %elimino blob aggiunto (o ignorato per area)
81          vetblob(hh,:)=[];
            sizeblob=sizeblob-1;
             end %fine if
            hh=hh+1;
        end; %fine while
86  %GESTIONE FUSIONI
    hh=1; %posiziono hh su ultimi marker
    while (hh<=sizeblob && sizeblob>1 && vetblob{hh
        ,8}<=(j-1)) %ora cerco blob da unire
        inizio=vetblob{hh,3}(1,vetblob{hh,1});
        finee=vetblob{hh,4}(1,vetblob{hh,1});
91      %rigiro i blob
        hhh=sizeblob;
        if(deb==1)
            strcat('sto analizzando fusioni alla riga:
                ',int2str(hh))
        end;
96      while (hhh>=1 ) %&& vetblob{hhh,8}<=(j-2)
            mediarif=vetblob{hhh,5}(1,vetblob{hhh
                ,1});
```

```
            inizio2=vetblob{hhh,3}(1,vetblob
                {hhh,1});
            fine2=vetblob{hhh,4}(1,vetblob{
                hhh,1});
        if(deb==1)
            strcat('sto analizzando fusioni in
                while2 alla riga ',int2str(hh),'
                confrontando con ',int2str(hhh)
                )
        end;
        %se trovo due marker attaccati ma non
            con lo stesso indice
        if(((mediarif>=inizio && mediarif<=
            finee) || (inizio2>=inizio && inizio2
            <= finee) || (fine2>=inizio && fine2
            <= finee) )&& hhh ~=hh)
        %condizione in cui i due blob si fondano
            , in
        %tal caso devo fonderli
            if(deb==1)
                strcat('FUSIONE tra ',int2str(hh),
                    ' E ',int2str(hhh))
            end;

            cella=fondiblob( vetblob, hhh , hh )
                ;
            %elimino blob aggiunto(o ignorato
                per area)
            if(hh>hhh)
                vetblob(hhh,:)=[];
                vetblob(hh-1,:)=[];
            else
                vetblob(hh,:)=[];
```

```
                    vetblob(hhh-1,:)=[];
                end;
                sizeblob=sizeblob-1;
                vetblob=cat(1,vetblob, cella);
                %trovato perciò ne cerco un altro
            break;
            end;
        hhh=hhh-1;
    end; %fine while dentro
    hh=hh+1;
    end;%fine while blob da unire esterno
end; %fine if di cerca marker chiusi

%% % ciclo sulla riga elemento per elemento (in
    orizzontale)
k=1;
while (k<=width)
    if(deb==1)
        strcat('inizo analisi della riga:',int2str(j), '
            e di k: ',int2str(k))
    end;
%se elemento è nullo vado avanti
    if( img(j,k)>= sogliap ) %trovo dimensione della
        riga del marker
        inz=k;
        if(k<width) %gestisco ultimo pixel ai margini
            k=k+1;
        end;

        while(img(j,k)>=sogliap && k<width) %procedo con
            il conteggio
            k=k+1;
        end;
```

```
fin=k-1;
area=fin-inz+1;
%possiedo num riga-inzio-fine-peso-media pesata
[mediapes som]=mediapesatapesi(inz:1:fin,img(j,
    inz:1:fin));
if(deb==1)
    strcat('primo elemento trovato: ',int2str(
        inz),' e l ultimo ',int2str(fin))
    strcat('hanno media: ',int2str(mediapes),'
        e peso ',int2str(som))
end;
%se flag=1 allora vuol dire che la riga è già
    stata elaborata ed inserita
flag=0;

hh=1; %indice di ricerca
%cerco marker che contenga attuale baricentro
    nell ultimo inz e fin. su blob
%flagint che gestisce il ritrovamento di + raggi
    del marker
while (hh<=sizeblob) %scorro su blob
iniz=vetblob{hh,3};
fine=vetblob{hh,4};
medpesrif=vetblob{hh,5};
if(j==vetblob{hh,8} && inz> vetblob{hh,4}(1,
   vetblob{hh,1}) && inz< vetblob{hh,4}(1,vetblob
   {hh,1}-1) ) %gestione fusioni sotto
      flag=1;    %ho trovato marker che si
          attacca sotto un blob con ramificazioni
          sotto
      vetblob{hh,4}(1,vetblob{hh,1})=fin; %pixel
          fine
      %calcolo media pesata nuova
```

```
            nuovamediapes=(vetblob{hh,5}(1,vetblob{hh
                ,1})* vetblob{hh,6}(1,vetblob{hh,1})+
                mediapes*som)/(som+vetblob{hh,6}(1,
                vetblob{hh,1}));
            vetblob{hh,5}(1,vetblob{hh,1})=
                nuovamediapes; %pixel medio
171         vetblob{hh,6}(1,vetblob{hh,1})=vetblob{hh
                ,6}(1,vetblob{hh,1})+ som; %peso di riga
            vetblob{hh,7}=vetblob{hh,7}+area; %area
                blob
        elseif(( (mediapes>=iniz(1,vetblob{hh,1}) &&
            mediapes<= fine(1,vetblob{hh,1})) || (
            medpesrif(1,vetblob{hh,1})>=inz && medpesrif
            (1,vetblob{hh,1})<=fin) || (mediapes>=iniz(1,
            vetblob{hh,1}-1) && mediapes<= fine(1,vetblob{
            hh,1})-1) || ( medpesrif(1,vetblob{hh,1}-1)>=
            inz && medpesrif(1,vetblob{hh,1}-1)<=fin)))
            flag=1; %ho trovato marker che si attacca
                sotto
            %aggiungo dati
176         vetblob{hh,1}=vetblob{hh,1}+1; %num righe
            vetblob{hh,3}=[vetblob{hh,3} inz]; %pixel
                inizio
            vetblob{hh,4}=[vetblob{hh,4} fin]; %pixel
                fine
            vetblob{hh,5}=[vetblob{hh,5} mediapes]; %
                pixel medio
            vetblob{hh,6}=[vetblob{hh,6} som]; %peso di
                 riga
181         vetblob{hh,7}=vetblob{hh,7}+area; %area
                blob
            vetblob{hh,8}=vetblob{hh,8}+1; %ultima riga
            %size blob rimane invariata perchè aggiungo
```

```
        %solo riga che compone marker e non marker
            nuovi
        %esco dal ciclo perchè ho inserito questo
        %gruppo di pixel, non posso metterlo volte
        break;
    end;%fine if
    hh=hh+1;
    end; %fine while

    if(deb==1)
        strcat('la ricerca su blob ha dato esito: '
            ,int2str(flag))
    end;

    %se elemento non appartiene a vetblob cerco in
        vettmp se vettmp non è vuoto
    if(sizetmp>0 && flag==0)
    hh=1; %trovo riga che mi interessa %[j inz fin
        mediapes som]
    while(hh<=sizetmp && vettmp(hh,1)<(j-1))
        hh=hh+1;
    end;
    %cerco marker nella riga
    while( hh<=sizetmp)
        %se trovato marker metto tutto nel blob, se
            media pesata precedente è tra attuale
            inf e max allora MARKER
        %TROVATO ed esco con vero
        if( (mediapes<=vettmp(hh,3) && mediapes>=
            vettmp(hh,2)) || (vettmp(hh,4)<=fin &&
            vettmp(hh,4)>= inz) || (fin<=vettmp(hh
            ,3) && fin>=vettmp(hh,2)) || (inz>=
            vettmp(hh,2) && inz<=vettmp(hh,3)) )
```

```
                flag=1; %aggiungo su blob nuovo
                    marker
                areap=area+(vettmp(hh,3)-vettmp(hh
                    ,2));
                sizeblob=sizeblob+1;
                %vettore contenente blob=[[num righe
                    ,riga inz
211             %,inz[],fin[],mediapes[],som[],area1
                    , ultima
                %rigapresente];... marker2]
                passvetblob={2 (j-1) [vettmp(hh,2)
                    inz] [vettmp(hh,3) fin] [vettmp(
                    hh,4) mediapes] [vettmp(hh,5) som
                    ] areap j};
                vetblob=cat(1,vetblob,passvetblob );
                %cancello su vet
216             sizetmp=sizetmp-1;
                vettmp(hh,:)=[];
                %esco dal ciclo perchè ho inserito
                    questo gruppo di pixel
                break; %esco da ciclo con flag=1
            end;%fine if
221         hh=hh+1;
        end; %fine while
    end; %fine if vettmp>0 flag==0

    if(deb==1)
226         strcat('la ricerca su vettmp ha dato esito
                : ',int2str(flag))
    end;

    % flag=0 allora non ho inserito la riga da
        nessuna parte perciò ne creo di nuovo su
```

```
            vettmp
        if(flag==0)
231             %creo nuova linea
                elem=[j inz fin mediapes som];
                vettmp=cat(1,vettmp ,elem);
                sizetmp=sizetmp+1;
                flag=1;
236         end; %fine if flag=0, ultimo caso

        if(deb==1)
                strcat('l inserimento nuovo tmp ha dato
                    esito: ',int2str(flag))
        end;
241

    end; %fine if img(j,k)~=0
    k=k+1;
end;%fine analisi elemento

246 if(deb==1)
        strcat('all analisi della riga:',int2str(j), '
            ho in size ',int2str(sizetmp),'e blob ',
            int2str(sizeblob))
    end
end;%fine analisi riga

251 %% svuoto blob dagli ultimi marker, mentre non
    considero dati in
%% vettmpperche di una singola riga

%scorro tutti i blob rimanenti in vetblob: elimino i
    blob inadeguati e salvo gli altri
if(sizeblob>0)
256     hh=sizeblob;
```

```
%controllo se l'area è valida
while (hh<=sizeblob)
        %algoritmo che gestisce la selezione dei
            marker
        numrighepass=vetblob{hh,1};
        rigainzpass=vetblob{hh,2};
        inzpass=vetblob{hh,3};
        finpass=vetblob{hh,4};
        mediapespass=vetblob{hh,5};
        sompass=vetblob{hh,6};
        area1pass=vetblob{hh,7};
        ultimarigapresentepass=vetblob{hh,8};
        %inizio elaborazione selezione blob e
            calcolo statistiche
        datimarker = blob2marker( numrighepass,
            rigainzpass, inzpass, finpass,
            mediapespass, sompass,area1pass,
            ultimarigapresentepass ,fatvet,fatvetelis
            ,areamin,areamax,areamax2);
        %fine elaborazione selezione blob e calcolo
            statistiche
        y = datimarker{1,1};
        x = datimarker{1,2};
        ymin = datimarker{1,3};
        xmin = datimarker{1,4};
        w = datimarker{1,5};
        h = datimarker{1,6};
        markON = datimarker{1,7};
        ymax = datimarker{1,8};
        xmax = datimarker{1,9};
        mcoef = datimarker{1,10};
        alfa = datimarker{1,11};
        beta = datimarker{1,12};
```

```
fattorearea = datimarker{1,13};
fform = datimarker{1,14};
yd = datimarker{1,15};
xd = datimarker{1,16};
simmetriaY = datimarker{1,17};
simmetriaX= datimarker{1,18};
profiloasse=datimarker{1,19};
pesoobliquoperpend=datimarker{1,20};

%inserisco marker se corrisponde alle
    caratteristiche
if(markON== 1)
    coordmark=cat(1,coordmark, [y x ymin xmin
       w h vetblob{hh,7}] );
    nummark=nummark+1;
elseif(markON== 2)
    coordmark=cat(1,coordmark, [y x ymin xmin
       w h vetblob{hh,7}] );
    nummark=nummark+1;
end

%riempio vettore contenente statistiche tutti
    i blob
%[ymin xmin ymax xmax mcoef alfa beta area]
if(markON== 2)
    tempstatpass={[profiloasse
       pesoobliquoperpend';profiloasse
       pesoobliquoperpend' ] };
    vetblobprofili=cat(1,vetblobprofili,
       tempstatpass );
    duestat=[double([ymin; ymin]) double([
       xmin; xmin]) double([ymax ymax])
       double([xmax;xmax]) double([mcoef;
```

```
                    mcoef]) double([alfa; alfa]) double([
                    beta;beta ]) double([vetblob{hh,7};
                    vetblob{hh,7}]) double([fattorearea;
                    fattorearea ]) double([fform;fform])
                    yd xd [simmetriaY;simmetriaY] [
                    simmetriaX;simmetriaX ] ];
                statblob=cat(1,statblob,duestat );
                statnum=statnum+2;
            else
                tempstatpass={profiloasse
                    pesoobliquoperpend' };
311             vetblobprofili=cat(1,vetblobprofili,
                    tempstatpass );
                statblob=cat(1,statblob, [double(ymin)
                    double(xmin) double(ymax) double(xmax)
                     double(mcoef) double(alfa) double(
                    beta) double(vetblob{hh,7}) double(
                    fattorearea) double(fform) yd xd
                    simmetriaY simmetriaX]);
                statnum=statnum+1;
            end

316         %elimino blob aggiunto o ignorato
            vetblob(hh,:)=[];
            sizeblob=sizeblob-1;
            hh=hh-1;
        end;%fine while
321 end;%fine if size blob>0

    %pulsico vettmp
    vettmp=[];
    sizetmp=0;
326 %% salvo file con le statistiche dell'immagine attuale
```

```
save ('stat_imm_temp.mat', 'statblob', 'statnum','
    indimage','nummark','coordmark','vetblobprofili');
%torno dati in uscita
vet={nummark coordmark statblob statnum  };
```

Viene riportato di seguito il codice della funzione che riconosce un marker date le rappresentazioni dei blob, in ingresso vuole un vettore con le caratteristiche dei blob e i parametri di riconoscimento di un marker, in uscita viene restituito un vettore con le coordinate dei marker e un vettore di statistiche.

Listing 3.4: Codice MATLAB di blob2marker.m

```
%%definisce se un blob è o meno un marker
function datimarker =blob2marker(numrighe,rigainz, inz,
    fin, mediapes, som,area1,ultimarigapresente, fatvet ,
     fatvetelis , areamin,areamax,areamax2)
markON=0; %valore di marker riconosciuto
%Preparo parametri di selezione marker
ffmin=double(fatvet(1,1)); %fat forma min
ffmax=double(fatvet(1,2)); %fat forma max
famin=double(fatvet(1,3)); %fat area min
famax=double(fatvet(1,4)); %fat area max
altezzamax2=double(fatvet(1,5));%altezza max2
%parametri ora non più usati per marker ellessoidi
% fforvetmin=double(fatvetelis(1,1));
% fforvetmax=double(fatvetelis(1,2));
% ffororizmin=double(fatvetelis(1,3));
% ffororizmax=double(fatvetelis(1,4));
rappalfabetamin=double(fatvetelis(1,5)); %fat alfa/beta
    min
rappalfabetamax=double(fatvetelis(1,6)); %fat alfa/beta
    min
 %calcolo min x max
xmin=min(inz);
```

```
xmax=max(fin);
ymin=rigainz;
ymax=ultimarigapresente-1;
mcoef =0;
alfa =0;
beta=0;
w= xmax-xmin+1;
h= numrighe;
fattorearea=0;
fform=0;
yd=mediapesatapesi(rigainz:1:ultimarigapresente,som);
xd=mediapesatapesi(mediapes,som);
x=int32(xd);
y=int32(yd);
simmetriaX=0;
simmetriaY=0;
profiloasse=0;
pesoobliquoperpend=0;
%orientazione=1; %ipotesi di che marker si sviluppa in
    verticale
if( area1>=areamin && area1<=areamax2) %controllo se l'
    area è valida, divido aree piccole e grandi
    if(area1<4)
        markON=1;
    elseif ( area1>=4 && w>1 && h>1 )
        markON=1;
    end
elseif( w<=altezzamax2 || h<=altezzamax2) %marker
    piccolo con lieve controllo di fat di forma
    fattorearea=double(area1)/((w)*h);%calcolo fattore
        di area
    if(w<=h) %calcolo fattore di forma
        fform=w/h; %marker si sviluppa in verticale
```

```
        else
            fform=h/w;
        end;
51
        if( fform>=0.2 &&  fattorearea>=0.3) %acquisisco
            marker con fattori di forma e area buoni
            markON=1;
        end

56  elseif( area1>areamax2 && area1<=areamax) %area
        compatibile con dimensioni marker
        fattorearea=double(area1)/((w+1)*h);%calcolo fattore
            di area
        if(w<=h) %calcolo fattore di forma
            fform=w/h; %marker si sviluppa in verticale
        else
61          fform=h/w;
        end;

        if( fform>=ffmin &&  fform<= ffmax &&  fattorearea>=
            famin &&  fattorearea<=famax ) %acquisisco marker
            con fattori di forma e area buoni
            markON=1;
66      else %gestisco  marker deformi (calcolo asse
            principale)
            if(fform>0.3 || fattorearea>0.1) %escludo marker
                troppo allungati o troppo sottosoglia
        mcoef=trovam(mediapes,numrighe); %se orizzontale uso
            metodo alternativo per calcolare m
        if((mcoef<0.1 && mcoef>-0.1) || mcoef<-9 || mcoef>9
            ) %marker orizzontale o verticale
            if(fform>=0.3 && fattorearea>=0.4)
71              profiloasse=som; %pesi verticali
```

```
            pesoobliquoperpend=pesicolonna(inz,fin,
                numrighe,xmin, xmax,som, area1); %pesi
                orizzontali
            %gestisco marker fusi orizzontalmente e
                veticalmente
            if(h+1>w) %gestisco marker veticalmente
        deltay=ymin-yd+1;
        y=int16([yd-deltay; yd+deltay]) ;
        x=int16([xd; xd]);
        yd=double([yd-deltay; yd+deltay]);
        xd=double([xd; xd]);
        markON=2;
            elseif(h<w+1)  %marker orizzontale
        deltax=xmin-xd+1;
        y=int16([yd; yd]) ;
        x=int16([xd-deltax; xd+deltax]);
        yd=double([yd; yd]);
        xd=double([xd-deltax; xd+deltax]);
        markON=2;
            else
        markON=1;
            end

        elseif(fform<=0.3)%elimino marker troppo
            lineiformi
            %possiedo già i profili e li uso per il
                calcolo
            profiloasse=som; %pesi verticali
            pesoobliquoperpend=pesicolonna(inz,fin,
                numrighe,xmin, xmax,som, area1); %pesi
                orizzontali
        end
    else %marker obliquo, devo calcolare ellissi
```

```
%calcolo i profili per valutare le forme
pesimediriga = pesimedirigacalcola(fin, inz,som)
    ;
outputprofili = calcolaprofili(x,y,h,mcoef,ymin,
    fin, inz,pesimediriga);
%{pesiprifilo alfa beta x1 x2 y1 y2 prop1 flag};
profiloasse = int16(outputprofili{1,1});
alfa = int16(outputprofili{1,2});
beta= int16(outputprofili{1,3});
%outputprofili = calcolaprofili(x,y,h,(1/mcoef),
    ymin,fin, inz,pesimediriga);
%pesoobliquoperpend= int16(outputprofili{1,1});
if(outputprofili{1,9}==3) %ho marker obliquo in
    cui ho calcolato la fusione
     y=int16([outputprofili{1,6}; outputprofili
         {1,7}]) ;
     x=int16([outputprofili{1,4}; outputprofili
         {1,5}]);
     yd=double([outputprofili{1,6}; outputprofili
         {1,7}]);
     xd=double([outputprofili{1,4}; outputprofili
         {1,5}]);
     markON=2;
elseif(alfa>0 && beta>0 )
     if(alfa>beta)
if(beta/alfa>= rappalfabetamin && beta/alfa<=
    rappalfabetamax )
     markON=1;
end
     else
if(alfa/beta>= rappalfabetamin && alfa/beta<=
    rappalfabetamax )
     markON=1;
```

```
121             end
                end
                end
                end %fine gestione if marker obliqui
                end %fine if di marker troppo allungati
126
        end; % fine if gestione fattore di form aarea
    end; % fine if gestione area

    %% dati in uscita
131 datimarker={y x ymin xmin w h markON ymax xmax mcoef
        alfa beta fattorearea fform yd xd simmetriaY
        simmetriaX profiloasse pesoobliquoperpend};
```

Bibliografia

[1] M. S. Shafiq, S. T. Tümer, and H. C. Güler, "Marker detection and trajectory generation algorithms for a multicamera based gait analysis system," *Mechatronics*, vol. 11, no. 4, pp. 409 – 437, 2001.

[2] F. Dibos, S. Pelletier, and G. Koep, "Real-time video segmentation." University Ren'e Descartes and University Paris Dauphine.

[3] G. Nagy, "Candide's practical principles of experimental pattern recognition," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 5, no. 2, pp. 199–200, 1983.

[4] K. Wall and P.-E. Danielsson, "A fast sequential method for polygonal approximation of digitized curves," *Computer Vision, Graphics, and Image Processing*, vol. 28, no. 2, pp. 220 – 227, 1984.

[5] H. El Bakry and Q. Zhao, "Fast pattern detection using normalized neural networks and cross-correlation in the frequency domain," vol. 2005, no. 13, pp. 2054–2060, 2005.

[6] A. Sluzek, "Identification and inspection of 2-d objects using new moment-based shape descriptors," *Pattern Recognition Letters*, vol. 16, no. 7, pp. 687 – 697, 1995.

[7] S.-S. Wang, P.-C. Chen, and W.-G. Lin, "Invariant pattern recognition by moment fourier descriptor," *Pattern Recognition*, vol. 27, no. 12, pp. 1735 – 1742, 1994.

[8] G. Xu and Y. Lei, "A new image recognition algorithm based on skeleton," 2008.

[9] U. Eckhardt, "A note on rutovitz method for parallel thinning," *Pattern Recognition Letters*, vol. 8, no. 1, pp. 35 – 38, 1988.

[10] P. E. and K. Fu, "Shape discrimination using fourier descriptors," *IEEE Trans. Syst. Man Cybernet*, vol. 7, 1977.

[11] H. Fonga, "Pattern recognition in gray level images by fourier analysis," *Pattern Recognition Letters*, vol. 17, no. 14, pp. 1477 – 1489, 1996.

[12] K. G. MUHARREM MERCIMEK and T. V. MUMCU, "Real object recognition using moment invariants," *Sadhana*, vol. 30, no. 6, pp. 765–775, 1995.

[13] M. K. Hu, "Visual pattern recognition by moment invariants," *IRE Trans. Info. Theory*, vol. IT - 8, pp. 179–187, 1962.

[14] L. Kotoulas and I. Andreadis, "Image analysis using moments," *Laboratory of Electronics, Section of Electronics and Information Systems Technology, Department of Electrical and Computer Engineering, Democritus University of Thrace*.

[15] A. M.-R. A. Padilla-Vivanco and F. Granados-Agustín, "Digital image reconstruction by using zernike moments," *Universidad Autónoma de Puebla*.

[16] L. Caponetti, "Teoria e tecniche di elaborazione della immagine," *Università degli Studi di Bari, Dipartimento di Informatica*.

[17] A. Cappozzo, U. D. Croce, A. Leardini, and L. Chiari, "Human movement analysis using stereophotogrammetry: Part 1: theoretical background," *Gait and Posture*, vol. 21, no. 2, pp. 186 – 196, 2005.

[18] L. Chiari, U. D. Croce, A. Leardini, and A. Cappozzo, "Human movement analysis using stereophotogrammetry: Part 2: Instrumental errors," *Gait and Posture*, vol. 21, no. 2, pp. 197 – 211.

[19] S. Corazza, L. Mundermann, and T. P. Andriacchi, *Markerless human motion capture through visual hull and articulated ICP*. 2007.

[20] B. Genade, "Marker recognition algorithm." Master's Thesis and relates to 3D marker recognition and motion analysis.

[21] Y. Zheng and Y. Liu, "On determining the projected sphere center and its application in optical tracking systems," *International Conference on BioMedical Engineering and Informatics*, 2008.