
# UNIVERSITÀ DEGLI STUDI DI PERUGIA

## FACOLTÀ DI INGEGNERIA

### CORSO DI LAUREA IN INGEGNERIA ELETTRONICA

## TESI DI LAUREA

# "PROGETTO E REALIZZAZIONE DI UN SISTEMA DI VISIONE STEREOSCOPICA PER LA ROBOTICA, CON APPLICAZIONE ALL'INSEGUIMENTO DI CORPI IN MOTO E ALL'AUTOLOCALIZZAZIONE"

*Laureando*
Davide Scaramuzza

*Relatore*
Prof. Paolo Valigi

*Correlatore*
Ing. Stefano Pagnottelli

*Controrelatore*
Prof. Giorgio Umberto Pignatel

Anno Accademico 2003-2004

2 Luglio 2004

# Indice

# Ringraziamenti

*Vorrei ringraziare il Prof. Valigi, che sin dal corso di Teoria dei Sistemi mi appassionò*

*al mondo dell'automazione e quindi alla Robotica, e l'Ing. Pagnottelli,*

*per la pazienza, il tempo, la disponibilità  e gli incoraggiamenti*

*che mi hanno offerto in questa straordinaria avventura.*

*Un grazie ancora a Beatrice, per essermi stata sempre vicina*

*anche nei momenti più difficili,*

*alla mia famiglia, e ancora a Sara, Elisa e Mario,*

*che mi hanno sempre sostenuto.*

# Abstract

In questa tesi vengono descritti il progetto e la realizzazione hardware/software di un sistema di visione stereoscopico; inoltre, sono studiati ed implementati degli algoritmi di visione artificiale per il riconoscimento automatico di oggetti, che hanno reso possibile applicare l'intero sistema ai seguenti problemi robotici:

- riconoscimento ed inseguimento di oggetti in movimento da parte di un robot mobile
- navigazione autonoma di un robot in un ambiente con l'ausilio della sola visione

Sono state pertanto realizzate ed applicate strategie di controllo per la movimentazione di robot mobili.

Nella prima parte vengono presi in esame gli algoritmi sviluppati per l'elaborazione digitale delle immagini e per il riconoscimento automatico di oggetti.

Nella seconda viene descritta la realizzazione del sistema di visione stereoscopico e degli algoritmi di stereovisione che consentono di impiegarlo come strumento "stereometrico" per il calcolo della posizione 3D di oggetti nello spazio.

Nella terza viene descritta l'implementazione e l'applicazione di leggi di controllo per un robot mobile, che, sfruttando le informazioni fornite dagli algoritmi di stereovisione e di riconoscimento automatico, gli hanno consentito di navigare autonomamente all'interno di uno spazio chiuso e di inseguire un oggetto specifico che transiti di fronte ad esso.

L'intero progetto è stato realizzato in **C**, in ambiente **Linux** (distribuzione **Mandrake 9.2**), con l'ausilio del software di sviluppo KDevelop.

Questo lavoro di tesi è stato svolto anche in collaborazione con il centro ricerche dell'**E.N.E.A.**, presso il Dipartimento di robotica, sede di Casaccia (Roma).

Il software sviluppato è stato infatti testato su un hardware stereoscopico professionale in dotazione al centro ricerche (capitolo 3).

La tesi è suddivisa in cinque capitoli.

Il primo è dedicato al progetto e alla realizzazione fisica dell'apparato di stereo-visione. In esso viene giustificata la scelta dei componenti hardware e software e dei parametri di progetto da considerare. Vengono anche illustrate le caratteristiche tecniche del robot mobile utilizzato e il software per gestirne il moto.

Il secondo descrive l'implementazione di algoritmi di elaborazione e trasformazione digitale delle immagini necessari per il riconoscimento automatico di un oggetto. La prima parte

è dedicato allo studio e all'analisi degli algoritmi di Image Processing, necessari per il condizionamento delle immagini; la seconda illustra l'implementazione di tre diverse strategie di riconoscimento di un oggetto, di cui l'ultima costituisce una soluzione innovativa da me proposta.

Nel terzo viene introdotto il problema della stereoscopia e viene discussa la necessità di correggere via software le non idealità intrinseche nel sistema, mediante una trasformazione digitale chiamata "rettificazione epipolare".
Successivamente viene descritta la realizzazione del software necessario all'impiego del sistema di visione come strumento "stereometrico" per la misura della posizione 3D di un oggetto a partire da una coppia di immagini stereoscopiche. Inoltre vengono presentati i risultati sperimentali del test compiuto sul sistema di stereovisione dell'ENEA.

Il quarto capitolo vede l'applicazione degli algoritmi di riconoscimento e di stereovisione, sviluppati nei precedenti capitoli, all'inseguimento di un oggetto in moto da parte di un robot mobile, dotato del sistema di visione stereoscopica in questione. Vengono prese in esame le tecniche di controllo sviluppate appositamente e sono descritte quattro distinte strategie di inseguimento dell'oggetto, di cui l'ultima costituisce una soluzione innovativa da me proposta.

Nel quinto, dopo aver esposto le problematiche connesse alla navigazione e all'autolocalizzazione autonoma di robot, viene descritta l'applicazione degli algoritmi di riconoscimento e di stereometria, sviluppati nei precedenti capitoli, all'autolocalizzazione del robot. Verrà dimostrato che, grazie alla stereovisione e al riconoscimento automatico di caratteristiche artificiali dello spazio, il robot è perfettamente in grado di muoversi autonomamente all'interno di un ambiente chiuso.

# Introduzione alla visione stereoscopica

In natura la maggior parte degli esseri viventi più evoluti è in grado di percepire in qualche modo la realtà mediante l'uso della vista.
La capacità di ottenere informazioni sull'ambiente circostante dagli stimoli luminosi e dal loro cambiamento è così importante in natura che sin dagli albori dell'intelligenza artificiale si è presa in considerazione l'idea di dotare di funzionalità visive anche i sistemi robotici ed informatici. Purtroppo in questi sistemi la capacità visiva è spesso accompagnata da un requisito di elaborazione in tempo reale che fino a poco tempo fa difficilmente poteva essere garantito dalla tecnologia a nostra disposizione. Ancora adesso i sistemi visivi di molti esseri viventi hanno prestazioni irraggiungibili da qualsiasi macchina, basti pensare ad esempio alla potenza di calcolo necessaria per gestire un sistema visivo complesso come quello dell'uomo: non è un caso che ben il 30% circa dei neuroni (circa 60 miliardi) che compongono un cervello umano è dedicato all'elaborazione delle informazioni visive!
Dai numerosi tentativi compiuti dall'uomo per comprendere i principi alla base della visione umana sono emersi negli anni altrettanti progressi per tentare di imitarla a livello computazionale.
Da ciò nasce l'esigenza della Machine Vision, che consiste nello sviluppo di tecniche di visione artificiale di ausilio alla vita dell'uomo.
Negli anni, la visione computazionale ha trovato molteplici applicazioni in:

- Automazione dei processi industriali
    - Riconoscimento di oggetti, ispezioni visuali, controlli di qualità.
    - Navigazione e coordinazione di robot.
- Sorveglianza e tracking
    - Traffico, sicurezza.
- Interazione uomo-computer
    - Rilevamento e riconoscimento di volti.
    - Assistenza ai disabili
- Realtà virtuale e visualizzazione
    - Ricostruzione di scene e modelli 3D.
    - Fotometria
- Applicazioni spaziali e militari

L'idea alla base della visione computazionale consiste nell'avere una o più telecamere connesse ad un computer, il quale deve automaticamente interpretare le immagini di una scena reale, ottenendo informazioni per svolgere determinate azioni.

**immagini** → **rappresentazione** → **percezione** → **azione**

Ma una delle facoltà più entusiasmanti del nostro cervello è sicuramente la capacità di fondere le immagini retiniche che provengono dagli occhi e di percepirle come un'immagine unica.
È proprio quest'ultimo processo, definito **Stereopsi**, che ci consente la localizzazione relativa degli oggetti visivi in profondità, donandoci la percezione della tridimensionalità dello spazio.
La stereoscopia è appunto il leitmotiv di questa lavoro di tesi, che nasce dalla mia volontà personale di espandere le potenzialità della Machine Vision tramite la percezione della terza dimensione, e trarne informazioni utili per la navigazione, la manipolazione ed il riconoscimento.

Fino a qualche decennio fa l'approccio stereoscopico era spesso trascurato dalla scienza a causa delle ingenti problematiche connesse alla realizzazione pratica del sistema di stereovisione, che imponeva l'impiego di due telecamere perfettamente uguali e ben allineate sullo stesso asse, l'uso di due sistemi di acquisizione, in grado di lavorare parallelamente, e la conoscenza esatta dei parametri di calibrazione dei dispositivi.
Infatti, sebbene le leggi prospettiche, che legano le coordinate 3D dei punti dello spazio a quelle delle rispettive proiezioni sui piani immagine, siano note sin dai tempi di Leonardo da Vinci, la possibilità di utilizzare quelle relazioni geometriche era comunque subordinata alla conoscenza esatta dei parametri interni delle telecamere utilizzate, e alla conoscenza quasi perfetta della disposizione geometrica dei sensori (parametri esterni).
Grazie ai progressi della tecnologia, le difficoltà relative all'hardware sono state finalmente superate, mentre lo studio di algoritmi di ottimizzazione e risoluzione di funzioni non lineari complesse ha consentito di misurare con un grado di accuratezza notevole i parametri intrinseci ed estrinseci delle telecamere correggendo via software le non idealità del sistema.
Grazie a tutto ciò i sistemi di visione stereoscopica hanno trovato applicazione nei più disparati ambiti di utilizzo: dalla robotica ai sistemi di video sorveglianza, dai veicoli a guida automatica all'esplorazione robotica su Marte!
I vantaggi della visione stereoscopica risiedono infatti nel recupero della profondità, che non solo permette di ricostruire la tridimensionalità degli oggetti presenti sulla scena, ma anche di percepire la vicinanza di ostacoli o dedurre l'avvicinamento o allontanamento di corpi in moto, e infine di risalire all'intera struttura 3D di una scena.
Infine, un apparato di stereovisione può essere impiegato come un vero e proprio strumento di "stereometria" per la misura della posizione 3D di oggetti nello spazio, permettendo quindi di

costruire una mappa tridimensionale della conformazione fisica dell'ambiente circostante con un livello di precisione notevole.

Da tutto ciò, si comprende, è possibile trarre informazioni utilissime per la navigazione automatica, la manipolazione e il riconoscimento di oggetti da parte di robot autonomi.

Infatti la ricostruzione tridimensionale di una scena consente ad un veicolo autonomo di navigare liberamente in un ambiente, percependo la presenza di eventuali ostacoli da evitare, e di capire in quale punto dello spazio si trovi (autolocalizzazione) grazie alle elevate prestazioni stereometriche della visione stereoscopica.

Ma, oltre che alla robotica mobile, le applicazioni si estendono a robot manipolatori, veri e propri bracci robotici antropomorfi programmati per eseguire compiti specifici in ambienti industriali. La stereovisione consente loro di essere autonomi, di individuare la posizione nello spazio di un eventuale oggetto riconosciuto dal sistema e, conseguentemente, di afferrarlo senza l'egida guida di un programmatore.

Nei sistemi di video sorveglianza, invece, la percezione della tridimensionalità permette di inseguire qualsiasi corpo in moto grazie all'ausilio di software di riconoscimento automatico in grado di fondere le informazioni tratte dalle due telecamere ed estrarre un modello 3D dell'oggetto.

L'ultima sfida tecnologica in merito di stereovisione e riconoscimento automatico applicate alla robotica è stata compiuta quest'anno dalla NASA che ha realizzato due rover per l'esplorazione del pianeta Marte: Spirit e Opportunity.

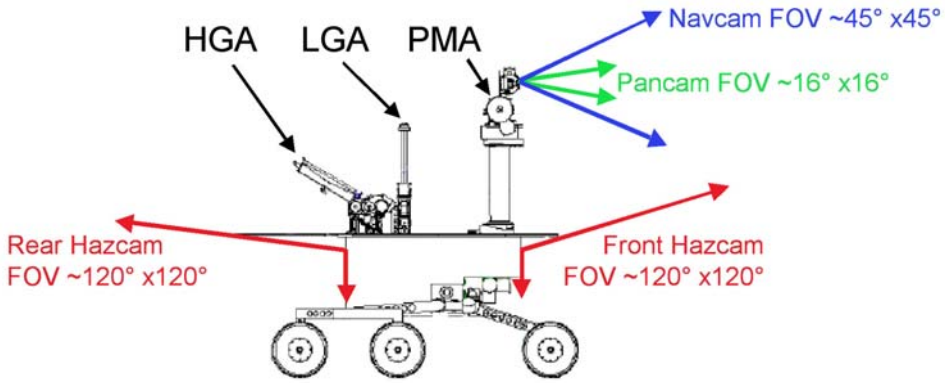
Il rover Spirit costruito dalla NASA. Si avvale di un sistema di visione complesso, composto da 9 telecamere, di cui 4 coppie stereoscopiche per l'"obstacle avoidance" e l'autolocalizzazione

Spirit utilizza un sistema complesso di visione composto da ben nove telecamere, di cui quattro coppie stereoscopiche che gli consentono di ricostruire la mappa tridimensionale del terreno marziano a 360°.

Le due coppie frontali ("Navcam" e "Pancam") hanno una profondità di visione elevata e, proprio per la ristrettezza del campo visivo (45°x45° e 16°x16° rispettivamente), sono impiegate per localizzare il robot nell'ambiente in cui si muove, prendendo come riferimento punti specifici del terreno (detti landmark naturali).

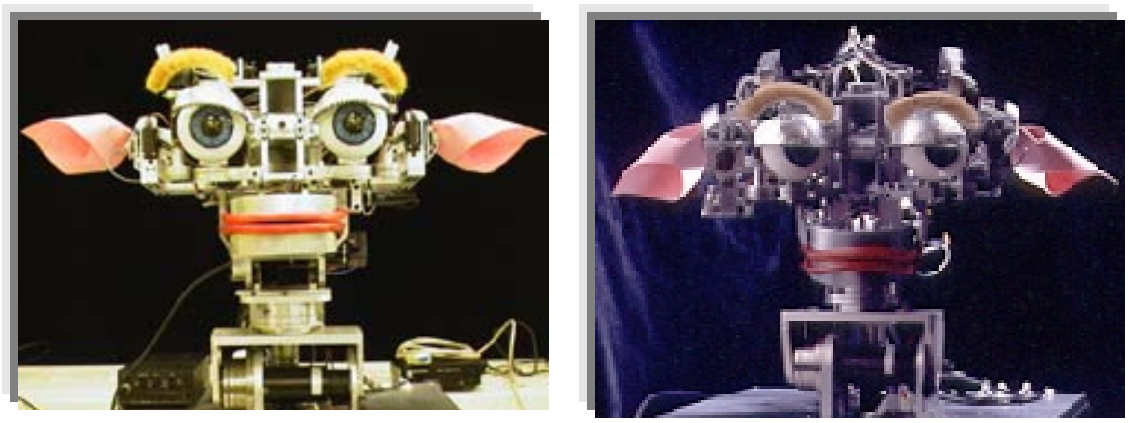
Le due coppie stereoscopiche più in basso ("Rear Hazcam" e "Frontcam") sono caratterizzate da un'ampiezza elevata del campo visivo (120°x120°); esse vengono utilizzate per ricostruire la mappa anteriore e posteriore del terreno marziano; inoltre consentono al rover di percepire gli ostacoli siti in prossimità di esso (rocce, ciottoli e crateri) e di pianificare nuove traiettorie per evitarli ("Obstacle Avoidance").

In questo modo il sistema gode di una certa autonomia di movimento e consente di ridurre così la mole di comandi inviati dal centro di controllo terrestre che, a causa della distanza Terra-Marte, giungono con un ritardo di 20 minuti.




La disposizione delle telecamere di Spirit

Un'applicazione più originale della stereovisione viene dal Media Lab del MIT di Boston, che ha realizzato Kismet, un prototipo di "faccia robot" antropomorfa in grado di emulare le espressioni emotive di chi vi è di fronte, per lo studio delle implicazioni psicologiche e sociali dell'interazione uomo-macchina.



Due espressioni emotive (felicità e tristezza) di Kismet, il prototipo antropomorfo di faccia robot per lo studio delle implicazioni psicologiche dell'interazione uomo-macchina.

Tutti gli esempi appena enunciati utilizzano le informazioni tratte da una coppia di immagini stereoscopiche mediante:

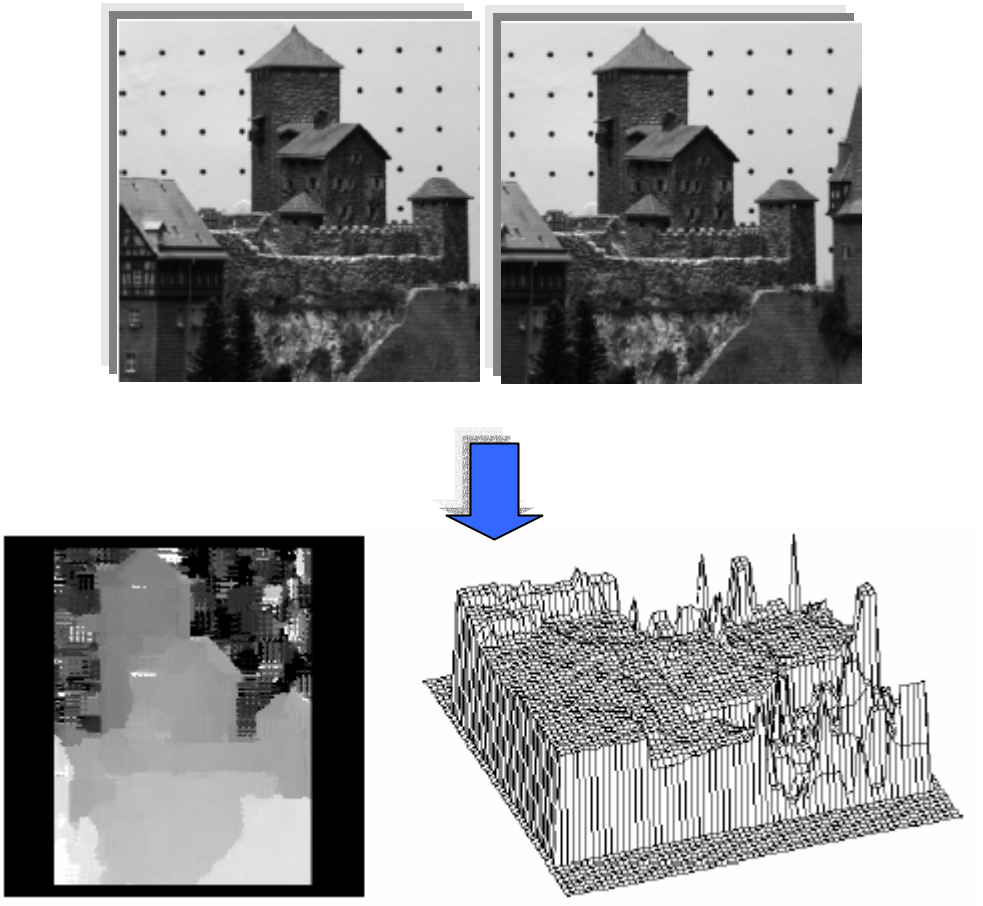
- Costruzione di mappe di disparità
- Calcolo della posizione 3D di oggetti nello spazio fisico

Le mappe di disparità sono delle rappresentazioni bidimensionali della scena ripresa dalle telecamere che codificano con gradazioni di grigio l'informazione sulla distanza e la profondità dagli oggetti presenti. Le due immagini destra e sinistra si fondono in un'unica immagine in scala di grigi, che cambia intensità a seconda della distanza dei punti dall'osservatore: colori più scuri indicano oggetti più distanti, mentre colori chiari oggetti più vicini.

Le mappe di disparità vengono calcolate a partire da punti delle due immagini che sono proiezioni dello stesso punto dello spazio 3D; questi punti sono detti punti corrispondenti o coniugati. La distanza tra due punti coniugati è detta disparità; essa cresce al diminuire della distanza di un oggetto dall'osservatore; diminuisce per oggetti via via più lontani.

Infine, dalla disparità di punti corrispondenti delle due immagini e dalla conoscenza dei parametri delle telecamere (che governano la proiezione prospettica di punti 3D sui piani ottici dei sensori) è possibile calcolare la posizione 3D di oggetti fisici nello spazio e ricostruire l'intera struttura tridimensionale della scena visibile.

Il procedimento è illustrato nella figura della pagina seguente.



Processo di creazione della mappa di disparità (in basso a sinistra) e ricostruzione della struttura 3D di una scena (in basso a destra) a partire da una coppia di immagini stereoscopiche

La stereoscopia è appunto il leitmotiv di questa lavoro di tesi, che nasce dalla mia volontà personale di espandere le potenzialità della Machine Vision tramite la percezione della terza dimensione, e trarne informazioni utili per la navigazione di robot, la manipolazione ed il riconoscimento automatico di oggetti.

# Capitolo 1

# Il Sistema

## 1.0 Introduzione

In questo capitolo verrà descritto il complesso delle architetture hardware e software utilizzate nell'ambito di questo lavoro di tesi.

Il primo paragrafo è dedicato alla realizzazione fisica dell'apparato di stereo-visione. Verrà quindi preso in esame il sistema di acquisizione, composto dall'hardware, dal driver e dal software applicativo necessari alla cattura di immagini.
Un sistema di stereo-visione è tuttavia imprescindibile dalla realizzazione di una struttura meccanica di sostegno, dalla cui configurazione dipendono le potenzialità e le prestazioni dell'intero sistema. Pertanto, alla descrizione dell'hardware seguirà un'analisi dettagliata delle caratteristiche tecniche di quest'ultima.
Infine, sarà presa in considerazione la scelta dei parametri di progetto che riguardano la disposizione fisica delle telecamere.

La seconda parte, invece, è rivolta alla descrizione del robot sul quale è stato montato il sistema di stereo-visione per applicazioni all'inseguimento e all'autolocalizzazione; seguirà una descrizione dell'interfaccia software utilizzata per il controllo.
L'acquisizione video, l'image processing, l'acquisizione dei dati degli encoder e la trasmissione dei comandi da impartire al robot è gestita interamente da un PC portatile *Pentium 4 1700 MHz*, collocato sul piano del robot, al quale è connesso mediante porta seriale.

## 1.1 Il sistema di visione stereoscopica

### 1.1.1 L'hardware

Il sistema di acquisizione video è costituito da una coppia di webcam identiche a colori, per uso domestico, prodotte dalla ditta DiLab.
Le webcam sono entrambe in grado di funzionare in ambiente Linux e di interfacciarsi velocemente al computer tramite due porte USB.
Di seguito sono elencati i parametri tecnici del dispositivo:

| | |
|---|---|
| **Numero bit/pixel** | 32 bit |
| **Risoluzione CCD** | 352x288 pixel |
| **Modalità rappresentazione colore** | RGB |
| **Frame-rate (max)** | 15 Hz |
| **Lunghezza focale dichiarata** | 6 mm |
| **Ampiezza angolare campo visivo** | 45° |
| **Sistema operativo** | Linux |
| **Interfaccia di connessione** | USB |

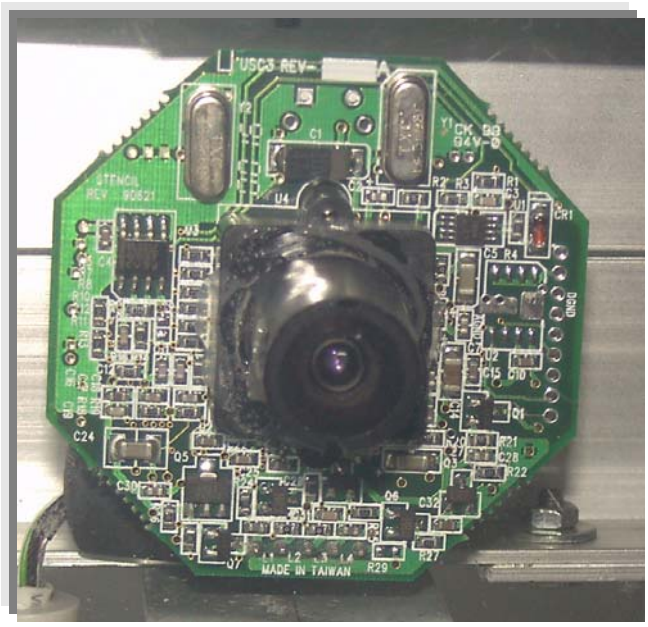



**Figura 1.1** La webcam utilizzata

### 1.1.2 Il software applicativo

Linux consente la possibilità di gestire il frame grabber di qualsiasi dispositivo video. Ciò è possibile grazie all'API Video4Linux, implementata già dalla versione 2.2 del Kernel.

Video4Linux è un'interfaccia realizzata per un gran numero di driver differenti o moduli kernel che supportando diversi tipi di hardware. Tuttavia, poiché Video4Linux è un'interfaccia a livello kernel, essa è governata interamente da chiamate alla funzione ioctl(), e non è particolarmente facile da gestire, oltre che mal documentata.

Allo scopo di acquisire ed elaborare successivamente il segnale digitalizzato dalle webcam USB, ho utilizzato una libreria appositamente implementata: libfb (Frame Grabber Library) [3].

Libfg fornisce una serie di API, open source, implementate interamente in C, che consentono di controllare l'acquisizione da qualsiasi dispositivo video, incluse videocamere analogiche, TV e webcam, mediante semplici funzioni di alto livello.

### 1.1.3 La struttura meccanica

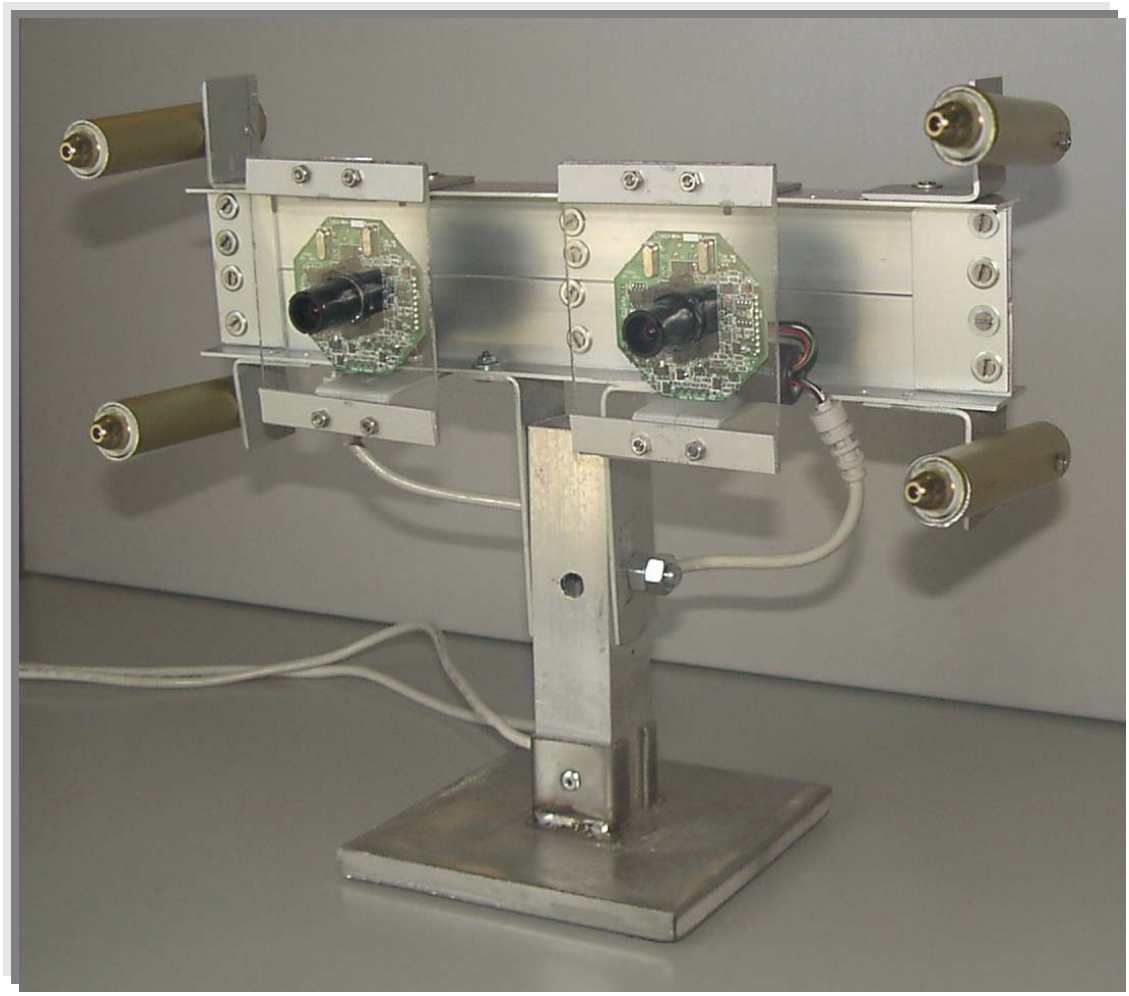


**Figura 1.2** Il sistema di visione stereoscopica

Le due webcam sono state montate su una struttura di appoggio rigida in alluminio, che ho progettato e costruito appositamente per sostenere le telecamere in questione. Per poter fissare stabilmente le due webcam su detto supporto, ho provveduto a rimuovere l'alloggio in plastica che le conteneva, e ad applicare le due schede elettroniche a due basi rettangolari in plexiglas, predisposte allo scopo di proteggerle da possibili urti.

Il supporto di appoggio è caratterizzato da una struttura a T, in cui la parte superiore possiede un grado di mobilità (beccheggio), che le consente di essere orientata manualmente con qualsiasi angolo che intercorre tra il pavimento e la sua normale.

La base quadrata è in acciaio (spessore 1 cm) ed ha un peso di circa 1,5 Kg, allo scopo di assicurare la stabilità e la rigidità della struttura anche durante le variazioni rapide del moto del robot.

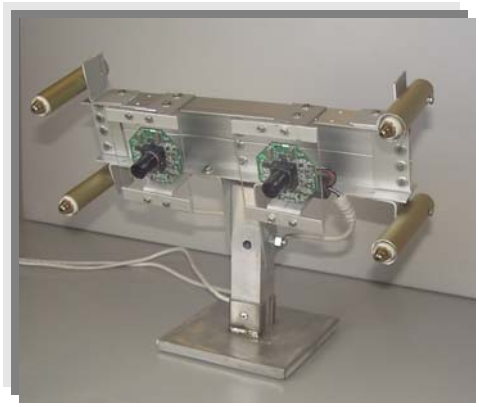


**Figura 1.3**

Le due webcam sono fissate, tramite viti, sulla parte superiore della T in modo da essere orizzontali, parallele e collimate. Quest'ultima ipotesi, come vedremo nel capitolo 3, non può essere mai verificata nella pratica, neanche per apparati stereoscopici professionali, e sarà dunque necessario effettuare la calibrazione del sistema di visione al fine di avvicinarci a tali ipotesi di idealità (che verranno discusse e giustificate nel capitolo 3, relativo alla stereoscopia).

La parte superiore della T presenta un certo numero di fori, che ho predisposto per offrire la possibilità di aumentare la distanza tra le webcam, detta *baseline*.

Ai quattro vertici della T sono fissati tramite dei giunti sferici quattro puntatori laser necessari per la calibrazione automatica del sistema (capitolo 6)

### 1.1.4 Scelta dei parametri di progetto

La *baseline* è il parametro di progetto più importante in un sistema di visione stereoscopico perché da essa dipende il grado di accuratezza con cui è possibile stimare la distanza di oggetto o eventuali ostacoli. Maggiore è la lunghezza della baseline, migliore è la precisione del sistema a grande distanza.

Si consideri la figura 1.4, in cui *b* è la lunghezza della baseline, *f* la focale ed *H* la distanza fra il centro delle telecamere ed il punto P di cui si vuole determinare la distanza.

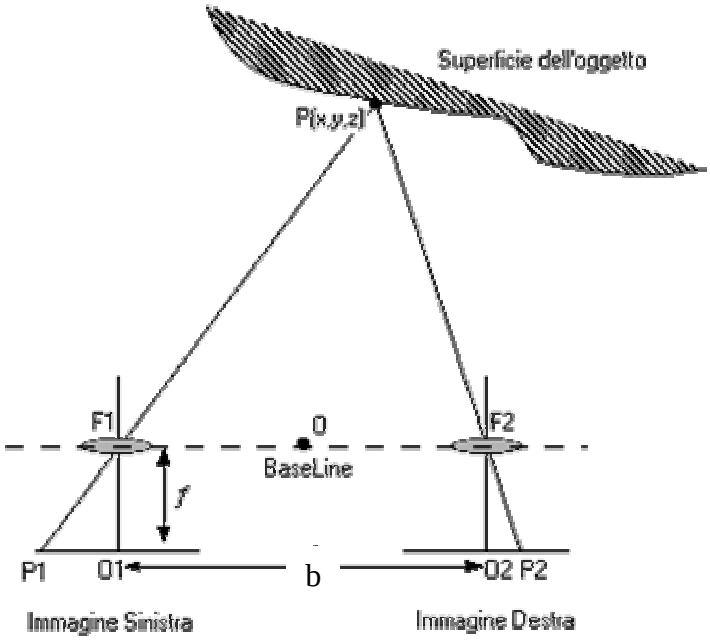



**Figura 1.4**

Da semplici considerazioni geometriche segue:

$$H = \frac{b \cdot f}{\overline{P1O1} + \overline{O2P2}} = \frac{b \cdot f}{D} \tag{1.1}$$

Dove D è detta *disparità* delle proiezioni di P nelle due immagini.

Derivando H rispetto a D si ottiene:

$$dH = \frac{b \cdot f}{D^2} \cdot dD = \frac{D^2}{b \cdot f} \cdot dD \tag{1.2}$$

cioè, aumentando la baseline diminuisce l'incertezza assoluta sulla misura di *H*.

Tuttavia, prima di scegliere *b* è bene considerare l'effetto dell'ampiezza angolare del campo visivo delle due webcam, pari a 45°, sulla minima distanza che può essere calcolata.

La figura 1.5 illustra l'intersezione delle regioni di visibilità dei sensori:

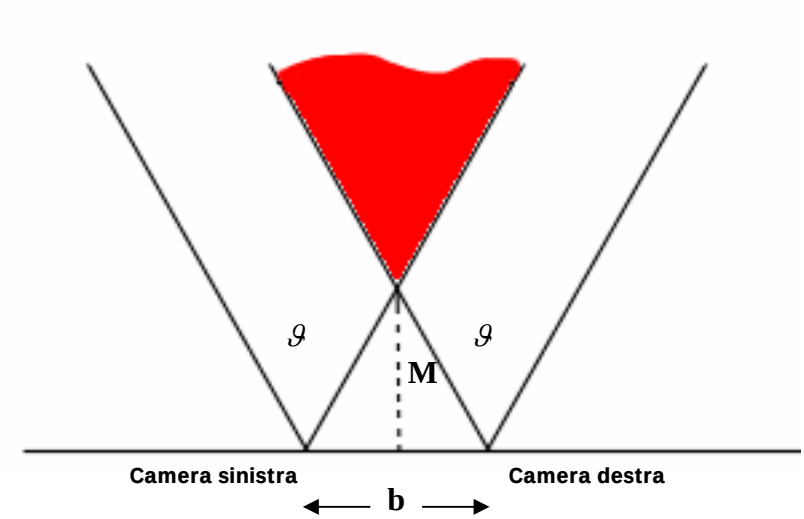



**Figura 1.5**

La lunghezza del segmento di minima distanza M deriva da semplici considerazioni geometriche:

$$M = \tan(90° - \frac{\vartheta°}{2}) \cdot \frac{b}{2} \qquad (1.3)$$

Poiché $\vartheta = 45°$, si è scelta una baseline di 12 cm, che garantisce una distanza minima misurabile pari a 14.5 cm.

Un'ultima attenzione va infine dedicata ai quattro cilindri posti ai vertici della T, che costituiscono gli alloggi di eventuali puntatori laser da usare per inizializzare l'inclinazione del supporto rispetto al piano verticale, oppure per l'eventuale calibrazione automatica del sistema nella fase di inizializzazione (capitolo 6).

## 1.2 Il Robot

### 1.2.1 Caratteristiche tecniche

Il robot utilizzato in questo lavoro di tesi è un Pioneer 2DX, prodotto dalla ActivMedia Robotics.
Il Pioneer 2DX è un veicolo differential drive, a due ruote motrici, ciascuna attuata da un motore elettrico in corrente continua.
Le prestazioni che si possono ottenere consistono in una velocità lineare massima di 1.6 m/sec ed una velocità di rotazione intorno all'asse del robot, ottenuta imprimendo velocità distinte sulle due ruote, di 300 gradi/sec.
Su ogni ruota motrice è calettato un encoder incrementale, utilizzato sia per il controllo del motore, sia per dare una stima della posizione del robot per via odometrica (*dead-reckoning*).
Inoltre il Pioneer è dotato di un array di 16 sonar Polaroid Serie 600, disposti sul perimetro secondo la struttura in figura 1.6. La frequenza di acquisizione di ogni sonar è di 25*Hz*. La distanza dagli oggetti che riesce a rilevare ogni sonar va da 10 cm a circa 5 m.
La gestione dei sensori e dei motori viene svolta da un microcontrollore integrato Siemens 8C166, cui è assegnata anche la comunicazione con un elaboratore remoto che può avvenire mediante un cavo seriale o una connessione via radio-modem (in questo lavoro, mediante cavo seriale connesso ad un PC).
Le variabili di controllo del robot sono la velocità di traslazione e la velocità angolare, entrambe riferite al centro del robot (punto medio dell'asse delle ruote).
A partire da dette variabili il microcontrollore calcola le velocità da impartire separatamente alle due ruote destra e sinistra per generare il moto desiderato.

**Figura 1.6** Pioneer 2 DX vista laterale

**Figura 1.7** Pioneer 2 DX vista frontale

**Figura 1.8** Pioneer 2 DX vista dall'alto

### 1.2.2 Software applicativo

Ai fini del controllo del robot si è deciso di utilizzare il Player [2], un software basato sull'architettura client/server. Esso permette di avere un controllo semplice e completo sui sensori fisici e sugli attuatori del nostro robot.

Mentre il Player viene eseguito sul robot, il programma client si connette ad esso attraverso un semplice collegamento TCP e la comunicazione viene effettuata tramite l'invio e la ricezione di un set di semplici messaggi.

Il client può essere eseguito su qualsiasi dispositivo che abbia la possibilità di essere collegato al robot e può essere scritto in qualsiasi linguaggio possa aprire e controllare una connessione TCP.

Player è progettato per supportare un numero virtualmente infinito di client e ciò permette di controllare un'intera squadra di robot interagenti tra loro: infatti ogni client può connettersi, leggere dati e controllare gli attuatori di qualsiasi altro robot.

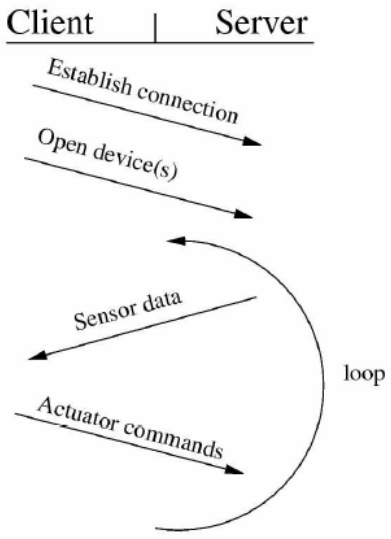



**Figura 1.9** Esempio di interazione client-server

Come esempio d'uso di questo programma si consideri la figura 1.9.

Il sever è eseguito localmente sul computer al quale il dispositivo di interesse (sia esso un sonar, un bumper, un motore) è connesso; di solito questo computer è il robot stesso, ma potrebbe anche essere un PC a cui è connesso il dispositivo. Il client può essere invece eseguito ovunque ci sia una connessione di rete con la macchina che ospita il server. Per prima cosa il client stabilisce una connessione TCP col server, poi invia a quest'ultimo alcuni

messaggi per "aprire" il dispositivo al quale è interessato, dopodichè il server fornisce continuamente dati da tale dispositivo al client che continua a controllarlo con appositi messaggi di ritorno.

# Capitolo 2
# Riconoscimento di un oggetto


## Abstract

In questo capitolo verranno illustrati i vari processi affrontati cronologicamente per l'implementazione finale di un algoritmo efficiente ed efficace di riconoscimento di una palla. Data la molteplicità delle possibili soluzioni al problema, saranno descritte esclusivamente quelle metodologie di riconoscimento che hanno dato luogo allo sviluppo di codice in C. Verranno inoltre confrontate le prestazioni dei diversi algoritmi in chiave di efficienza (tempi di calcolo) ed efficacia (capacità di funzionare in una pluralità di ambienti e condizioni possibili).


## 2.0 Introduzione

L'interpretazione di immagini acquisite da una scena del mondo reale consente di trarre molteplici informazioni utili per la navigazione in un ambiente, la manipolazione ed il riconoscimento di ciò che si ha di fronte. Ma cosa significa "interpretare"? La Psicologia definisce l'interpretazione come il processo di "rappresentazione" della realtà, volto all'estrazione degli "elementi salienti caratteristici" (*features*) da "riconoscere". Tuttavia, se nel caso dell'uomo la problematica del riconoscimento coinvolge associazioni tra gli oggetti correntemente presenti sulla scena e quelli classificati in opportune categorie mentali nel corso dell'esperienza, le macchine "intelligenti" di converso necessitano di un'adeguata fase di addestramento che si esplica attraverso "rappresentazione", "descrizione" e "riconoscimento in sé".

Come asseriva David Marr, il pioniere della Computer Vision: "*Non si può comprendere ciò che si vede e come si è ottenuto senza comprendere il processo sottostante di elaborazione dell'informazione"* [18]. Infatti, dal punto di vista della Teoria dell'Informazione, la visione implica una notevole riduzione dei dati a disposizione; pertanto prima di approdare al riconoscimento effettivo della palla, nei primi paragrafi di questo capitolo illustrerò le fasi di elaborazione fondamentali (*preprocessing*) che caratterizzano i momenti di "rappresentazione" e "descrizione" suddetti. Successivamente descriverò le diverse strategie implementate ai fini dell'individuazione della palla (*ball detection*).

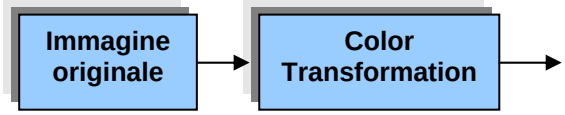
## 2.1 Preprocessing dell'immagine

Il *preprocessing* di un'immagine costituisce un'elaborazione di basso livello che consente di ridurre notevolmente la quantità di informazione, limitandola in tal modo a quella effettivamente utile ai fini del riconoscimento, e diminuendo così ulteriormente il tempo di elaborazione.
Come già anticipato, il sistema di visione acquisisce immagini a colori delle dimensioni di 288x352 pixel in formato RGB, il che equivale a triplicare il numero di punti disponibili portandolo a 288x352x3=304128 punti o byte (giacchè ogni componente cromatica è memorizzata in formato char ad 8 bit). Nell'ipotesi in cui il sistema di visione acquisisca immagini con un frame-rate nelle condizioni di massimo regime dichiarato di 15 frame/sec, ciò comporta un flusso di informazione di 4.35 Mbyte/sec.
Se ogni singolo pixel di colore contribuisse ai fini del riconoscimento finale della palla il tempo di calcolo necessario all'individuazione dell'oggetto (che sarà descritto in seguito) aumenterebbe a dismisura, compromettendo l'obiettivo di una elaborazione real-time!
Al contrario dopo il processo di preprocessing il numero di punti per ogni fotogramma disponibile scende da 304128 a circa 1500 punti, con flusso corrispondente di circa 1.4 Kbyte/sec! Risulta pertanto evidente il notevole risparmio apportato. Prima della fase di riconoscimento ma dopo il preprocessing, una ulteriore fase di elaborazione rimuoverà le cosiddette *false positives* riducendo ulteriormente la mole di dati a soli 100-300 punti!
A questo punto non rimane che spiegare in cosa consista questa "pre-processazione" dell'immagine.

Ecco dunque, nell'ordine in cui sono eseguiti dal codice del programma e nell'ordine in cui verranno esposte di seguito, le seguenti fasi di **preprocessing**:

1. passaggio ad una rappresentazione monocromatica (***color transformation)***
2. filtraggio antirumore (***noise filtering***)
3. estrazione dei contorni (***edge detection***)
4. sogliatura e raffinamento dei contorni (***edge thresholding & edge thinning***).

### 2.1.1 Passaggio alla rappresentazione monocromatica



Il sistema di visione umano (detto Human Vision System HVS) consente di apprezzare tutta la gamma di frequenze dello spettro visibile che si estende fra i 380 nm di lunghezza d'onda (per il violetto) e i 760 nm (per il rosso), consentendo una rappresentazione della scena detta "full color". Tuttavia, i coni e bastoncelli, ovvero gli elementi dell'occhio fotosensibili alla luce, non reagiscono ai cromatismi dell'immagine allo stesso modo: il 65% dei coni infatti sono sensibili al rosso, il 33% al verde e solamente i 2% al blu!

I moderni sistemi di acquisizione video tentano di emulare tale sistema di percezione del colore mediante una matrice di elementi fotosensibili che può essere di tipo CCD o CMOS. Un sensore CCD (utilizzato ormai in quasi tutti i sistemi ad alta risoluzione) non è tuttavia in grado di effettuare un riconoscimento *colorimetrico* in quanto è sensibile alle sole variazioni della luminosità (ossia all'energia dei fotoni che impattano sui sensori).

Sono pertanto necessari dei filtri che permettano la selezione della luce ad alcune lunghezze d'onda tipiche; se si utilizzano i tre colori RGB si ottengono tre immagini che devono successivamente essere assemblate per ottenere l'immagine "true-color"[5].

Le webcam in questione utilizzano la tecnologia CCD e selezionano le diverse componenti cromatiche R, G e B mediante filtri di Bayer disposti di fronte a ciascun elemento come in figura 2.1. Come si vede Il filtro di Bayer alterna una riga di filtri rossi e blu con una di blu e verde. I pixel dunque non sono equamente divisi, e si ha una maggiore presenza di verde. Questo è dovuto al fatto che, come detto sopra, l'occhio umano non è egualmente sensibile ai tre colori fondamentali e per avere una sensazione del colore di tipo true-color occorre una maggiore presenza di verde.

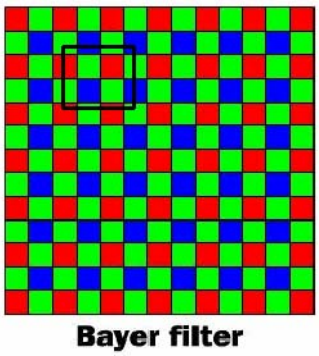



**Figura 2.1**

A questo punto potrebbe sorgere la seguente domanda: poiché le variazioni cromatiche che apprezziamo non corrispondono alla quantità di radiazione elettromagnetica effettivamente riflessa dagli oggetti a causa della natura particolare dell'occhio umano, come utilizzare la pluralità di colori di cui disponiamo? Inoltre, è necessario continuare a rappresentare l'informazione ottenuta dalle telecamere sotto forma di terne di valori RGB?

La risposta dipende dalla finalità dell'obbiettivo che ci siamo prefissi: ovvero il riconoscimento di una palla.

A questo punto sono possibili due tipi di approcci:

1. Riconoscimento basato sul colore (*color based*)
2. Riconoscimento basato sulla forma(*shape based*)

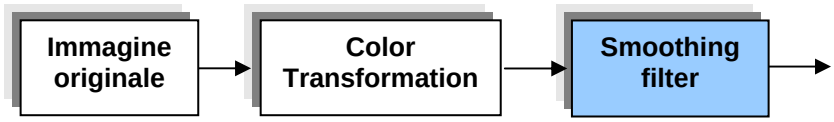
Quest'ultimo è quello effettivamente utilizzato nel programma, in quanto fornisce i migliori risultati. Infatti è in grado di adattarsi alle mutevoli condizioni di illuminazione (interne o esterne), agli effetti delle ombre, ai problemi di occlusione (copertura di una porzione dell'oggetto) nonché a più oggetti di forma sferica ma di diverso colore!

In favore di quest'ultima modalità di approccio, che punta ad desumere la forma degli oggetti presenti nella scena a partire dai loro "contorni" (vedremo come), risulta evidente la necessità di passare ad una rappresentazione monocromatica che preservi o addirittura enfatizzi il contrasto dei corpi sullo sfondo, in modo da facilitare la futura *edge detection*.
A tal fine ho implementato una funzione, che restituisce un valore monocromatico a partire dalla terna RGB associata ai pixel dell'immagine di partenza secondo la seguente formula:

$$dst[i] = \frac{\left(r \cdot R[i] + g \cdot G[i] + b \cdot B[i]\right)}{r + g + b} \tag{2.1}$$

Ogni punto dell'immagine finale risulta così una media pesata delle componenti RGB di ciascun pixel dell'immagine iniziale tramite i coefficienti arbitrari **r**, **g** e **b**.
Nell'assegnare tali coefficienti è importante il rapporto che intercorre tra essi. Scegliere una terna di coefficienti [r , g , b] = [1 , 1 , 1] è la stessa cosa dello scegliere [r, g, b] = [2 , 2 , 2]. In questo modo ad esempio ogni punto dell'immagine finale si ottiene come media aritmetica delle componenti di origine: ovvero si ottiene una rappresentazione in scala di grigi, che non è altro che una misura dell'Intensità di un'immagine.
Dopo diverse prove empiriche effettuate in interno, tra i corridoi della facoltà ed in diverse condizioni di illuminazione, la miglior scelta dei coefficienti di pesatura è risultata essere **[0,1,0],** (ovvero la sola componente **verde**) piuttosto che la consueta scala di grigi!
Infatti nel caso della scala dei grigi (coefficienti di pesatura uguali) si otteneva un maggior rumore di fondo ad opera soprattutto della componente rossa, che essendo evidentemente più vicina all'infrarosso risultava causa di maggior disturbo.
Si osservi che tale scelta preserva ancora il contrasto degli oggetti con lo sfondo ed esercita oltretutto un'attenuazione del rumore, di cui parlerò nel prossimo paragrafo.
Le trasformazioni per passare dalla rappresentazione RGB a quella monocromatica sono molteplici; molte di esse sono tuttavia non lineari (ad esempio logaritmiche per enfatizzare la saturazione o trigonometriche nel sistema HSI) o lineari a meno di una costante additiva (ad es. nella rappresentazione CMYK). Quella da me utilizzata, essendo lineare, gode del principio di sovrapposizione degli effetti; come si vedrà più avanti, il filtraggio del rumore di un0immagine a colori avviene convolvendo bidimensionalmente ogni singola componente cromatica dell'immagine con una maschera di convoluzione; pertanto sarebbe necessario effettuare tale convoluzione per tre volte, al termine della quale l'immagine filtrata (ancora a colori) verrebbe sottoposta alla trasformazione monocromatica. Al contrario essendo quest'ultima lineare è possibile posporre il filtraggio del rumore a valle di tale trasformazione, riducendo in tal modo la complessità computazionale di un fattore 3! Passiamo ora dunque ad esaminare il problema del *noise filtering*.

### 2.1.2 Filtraggio antirumore nel dominio spaziale




Prima di introdurre il filtro di attenuazione del rumore utilizzato dal programma è bene comprendere quali siano le cause principali di tale rumore.

La più comune causa di imperfezione in una immagine digitale è data proprio dall'instabilità della sorgente luminosa, a cui si aggiungono in misura minore le imperfezioni nel sistema di digitalizzazione (trasmissione elettronica dei dati), gli errori statistici di interazione tra elettroni, ecc. Il problema nasce quando il rumore non ha un andamento statistico ben definito ed è dunque random; è tuttavia possibile utilizzare alcune tecniche per ridurlo il più possibile ottenendo i vantaggi ma anche i corrispondenti svantaggi che saranno illustrati.

Innanzitutto si osservi che nel caso siano disponibili più copie di una stessa immagine statica si può ipotizzare di mediarle pixel per pixel in modo da incrementare il rapporto segnale rumore ma ciò contravverrebbe allo scopo di questo capitolo, che mira ad una implementazione efficiente dell'algoritmo oltre che all'elaborazione di immagini dinamiche e non statiche. Pertanto la soluzione migliore è quella che effettua l'operazione di mediatura direttamente sui pixel adiacenti a quello in esame, ovvero effettuando un filtraggio locale di tipo "passa basso".

I rimedi possibili per "eliminare" il rumore da un'immagine contemplano due ambiti di azione: quelli che agiscono del dominio spaziale e quelli che intervengono nel dominio della frequenza tramite trasformata e antitrasformata bidimensionale di Fourier. Nel mio caso mi sono limitato al dominio spaziale, che per filtri FIR di modeste dimensioni risulta computazionalmente più veloce!

Tra questi troviamo sostanzialmente tre categorie di filtro, ciascuno dei quali si adatta meglio a una determinata tipologia di rumore [6, 7]:

1. Filtri lineari (medio e media pesata)
2. Filtri mediani (non lineari)
3. Filtri gaussiani

Il primo e il terzo sono oggetto di analisi ed implementazione come algoritmo e verranno pertanto analizzati; la seconda tipologia è indicata per immagini che manifestano disturbi impulsivi (come *spike* o *salt & pepper noise*) che non interessano le telecamere utilizzate.

#### 2.1.2.1 Convoluzione spaziale

Sappiamo che un'immagine è un segnale definito nel dominio spaziale, e ha quindi due dimensioni $(x, y)$ cui è associato un valore $f(x, y)$ corrispondente all'intensità di colore. Dualmente al caso monodimensionale si definisce convoluzione spaziale o bidimensionale la seguente espressione:

$$g(x, y) = \sum_{k=-w}^{w} \sum_{l=-v}^{v} f(x + k, y + l) \cdot h(k, l) \tag{2.2}$$

dove $h(k,l)$ è la risposta impulsiva che in questo caso è rettangolare di dimensioni $(2 \cdot w+1) \times (2 \cdot v+1)$. Tale definizione purtroppo modifica in generale il valore massimo dell'immagine che rischia di superare il valore massimo ammissibile per ciascun pixel, pari a 255 (poiché ogni pixel è memorizzato in formato `char` a 8 bit); una definizione più pratica è invece la seguente, che non modifica troppo il valor medio di ciascun pixel :

$$g(x,y) = \frac{\sum_{k=-w}^{w} \sum_{l=-v}^{v} f(x+k, y+l) \cdot h(k,l)}{\sum_{k=-w}^{w} \sum_{l=-v}^{v} \cdot h(k,l)} \tag{2.3}$$

La risposta impulsiva $h(k,l)$ è anche detta "maschera di convoluzione". L'algoritmo utilizza maschere di dimensioni quadrate dispari 5x5. La scelta di assegnare dimensioni uguali lungo x e lungo y come vedremo riduce considerevolmente il numero di somme e prodotti da calcolare!
Si osservi inoltre che l'operazione di convoluzione modifica le dimensioni dell'immagine aggiungendo indesiderati effetti di bordo che, a causa delle discontinuità si ripercuoterebbero negativamente sulla futura edge detection. Per questo nell'operazione di convoluzione ai bordi dell'immagine viene effettuato uno ***zero padding*** per addolcire i bordi dell'immagine filtrata e le stesse dimensioni di quella iniziale.

Come detto sopra ho implementato due tipologie di filtri di *smoothing* che mi accingo ora a descrivere:

1. Filtro medio
2. Filtro Gaussiano

### 2.1.2.2 Filtro medio

Il più semplice filtro lineare è costituito dalla media spaziale, e consiste nel sommare i valori di luminosità dei pixel in ogni piccola regione dell'immagine diviso per il numero di pixel "vicini" ed utilizzando tali valori per costruire una nuova immagine.
La maschera utilizzata, già scalata per avere area unitaria, è quindi la seguente:

$\frac{1}{25}\cdot$

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |

$=$

| | | | | |
|---|---|---|---|---|
| 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |
| 0.04 | 0.04 | 0.04 | 0.04 | 0.04 |

Come appena anticipato la convoluzione di tale maschera di dimensioni 5x5 richiede di calcolare 25 somme e 25 prodotti per ogni pixel di immagine. Tuttavia si osservi che essendo la maschera di dimensioni dispari e presentando inoltre due assi di simmetria, $h(k,l)$ può scomporsi nel prodotto di due funzioni indipendenti: $h(k,l) = h(k)\cdot h(l)$ e quindi l'espressione [2.2] diventa:

$$g(x,y) = \sum_{k=-w}^{w} h(k)\cdot \sum_{l=-v}^{v} f(x+k, y+l)\cdot h(l) \qquad (2.4)$$

ovvero posso dapprima convolvere l'intera immagine con una maschera rettangolare siffatta:

| | | | | |
|---|---|---|---|---|
| 0.2 | 0.2 | 0.2 | 0.2 | 0.2 |

Salvare poi l'immagine ottenuta in una temporanea e convolverla nuovamente con una maschera identica ma verticale:

| |
|---|
| 0.2 |
| 0.2 |
| 0.2 |
| 0.2 |
| 0.2 |

In tal modo ho ridotto notevolmente il numero di operazioni per ogni pixel di immagine a 5 somme e 5 prodotti per il primo passaggio, e 5 somme e 5 prodotti per il secondo passaggio.
In generale dunque, se ho una maschera quadrata a doppia simmetria assiale di dimensioni $n \times n$ con una doppia convoluzione passo da una complessità di $n^2$ ad una di $2n$.

### 2.1.2.3 Filtro Gaussiano

Si tratta di un filtro di media pesata che assegna come valori di peso numeri che approssimano il profilo di una gaussiana. Il valore originario viene dunque rimpiazzato dalla media dei valori circostanti moltiplicata per tale distribuzione probabilistica. In questo modo si assegnano i valori più probabili attribuendo un maggior peso al pixel centrale.

Nell'algoritmo ho utilizzato una gaussiana con valor medio nullo e $\sigma = 1.4$, che è quello che offriva i risultati migliori. Applicando la formula seguente si ottiene la seguente maschera di convoluzione nella forma scalata:

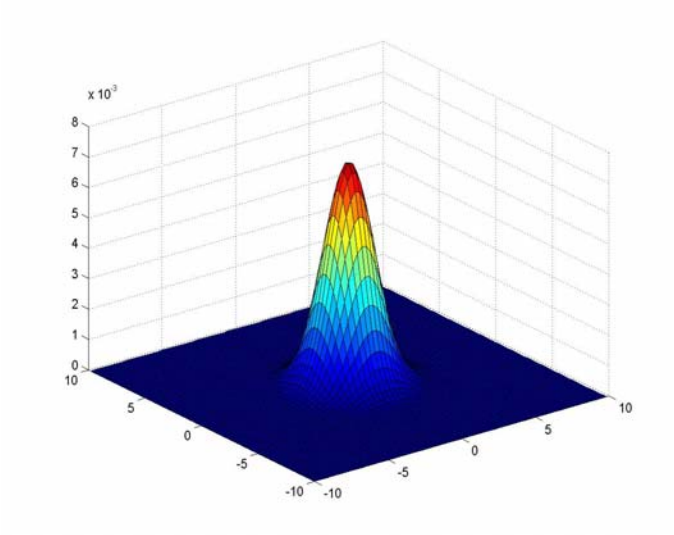



**Figura 2.2**

$$e^{-\frac{(x^2+y^2)}{2\cdot\sigma^2}} =$$

| | | | | |
|---|---|---|---|---|
| 0.0121 | 0.0261 | 0.0337 | 0.0261 | 0.0121 |
| 0.0261 | 0.0561 | 0.0724 | 0.0561 | 0.0261 |
| 0.0337 | 0.0724 | 0.935 | 0.0724 | 0.0337 |
| 0.0261 | 0.0561 | 0.0724 | 0.0561 | 0.0261 |
| 0.0121 | 0.0261 | 0.0337 | 0.0261 | 0.0121 |

Che corrisponde alla seguente rappresentazione discreta 3D in figura 2.3.

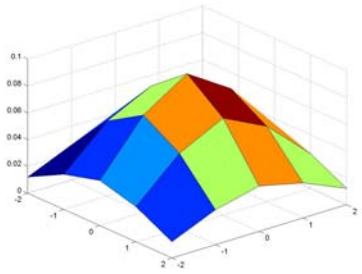


**Figura 2.3**

Come per il filtro medio anche quello gaussiano si presta ad una convoluzione a doppio passaggio osservando che esso è decomponibile in due funzioni indipendenti uguali.
Pertanto le maschere rettangolari da utilizzare risultano:

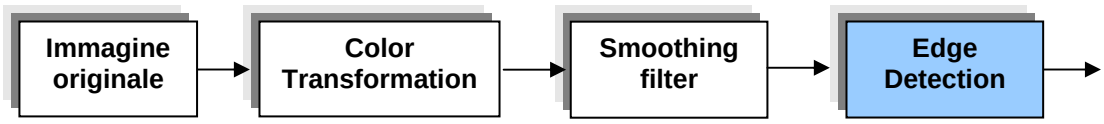
| |
|---|
| 0.1102 |
| 0.2369 |
| 0.3058 |
| 0.2369 |
| 0.1102 |

| | | | | |
|---|---|---|---|---|
| 0.1102 | 0.2369 | 0.3058 | 0.2369 | 0.1102 |

Il cui prodotto restituisce la matrice quadrata suddetta.
In tal modo, ribadisco, si ottiene una notevole riduzione dei tempi di calcolo!

Si osservi che lo *smoothing* in generale ammorbidisce i contorni dell'immagine sfuocandola e pertanto infierisce negativamente sull'individuazione dei bordi di cui mi accingo ora a parlare.

## 2.2 Edge detection




In questo paragrafo analizzerò dettagliatamente il problema della *Edge detection*, ovvero dell'estrazione dei contorni; esso costituisce la fase emblematica ai fini del riconoscimento finale della palla, che abbiamo detto essere basata sulla forma e non sul colore dell'oggetto.

Osservando una scena siamo in grado di discriminare un oggetto dall'altro, o un generico insieme di elementi connessi, valutandone le discontinuità di superficie, o meglio le variazioni di luminosità che la luce incidente determina sulla superficie di essi. Questo processo inconscio di "separazione" viene detto *segmentazione* o *image segmentation* nell'elaborazione digitale. Ma ciò che a prima vista potrebbe sembrarci naturale, è in realtà frutto di un processo conoscitivo della realtà, maturato nel corso dell'esperienza, che non è facile "insegnare" ad una macchina. Occorre infatti tenere conto che anche variazioni accidentali o artificiali del colore di un oggetto, l'ombra che esso proietta o i riflessi che genera su una superficie non del tutto opaca possono essere erroneamente interpretati come contorni di un oggetto che nella realtà non c'è (*false positives*)!

Di seguito esaminerò dapprima sinteticamente la problematica generale *dell'edge detection*, successivamente esporrò le diverse strategie adottate nell'algoritmo.

### 2.2.1 Estrazione del Gradiente

Nelle immagini digitali, un contorno (*edge*) è un insieme connesso di pixel che si estende in corrispondenza della discontinuità (*boundary*) tra due regioni. Dal punto di vista dell'analisi matematica lo strumento di calcolo che consente di descrivere le variazioni di continuità di una funzione dipendente da una sola variabile è la derivata, che per le immagini, viste come funzioni di due variabili, coincide con il gradiente. Equivalentemente alle funzioni reali e continue, nel mondo discreto delle immagini digitali si utilizza la medesima definizione di gradiente, che punta dunque ad approssimarne il modulo e la direzione (secondo $\psi$ ) seguenti:

$$\left|grad\left(g(x,y)\right)\right| = \sqrt{\left(\frac{\partial g}{\partial x}\right)^2 + \left(\frac{\partial g}{\partial y}\right)^2} \qquad \psi = \arg\left(\frac{\partial g}{\partial x}, \frac{\partial g}{\partial y}\right) \tag{2.5}$$

La direzione del gradiente è molto importante e punta proprio nel verso crescente di massima variazione di $g(x, y)$ che, in una scala di 256 gradazioni di grigio, va dal nero (0) al bianco (255). Si osservi inoltre che, per come è definito, esso risulta perpendicolare alla direzione del contorno. Tale proprietà verrà sfruttata nella prima versione dell'algoritmo di individuazione della palla.

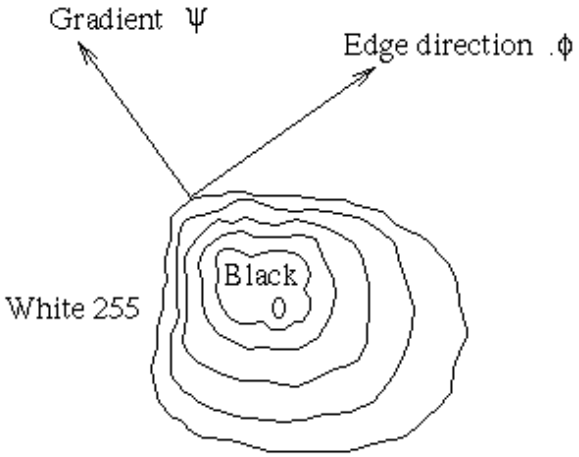
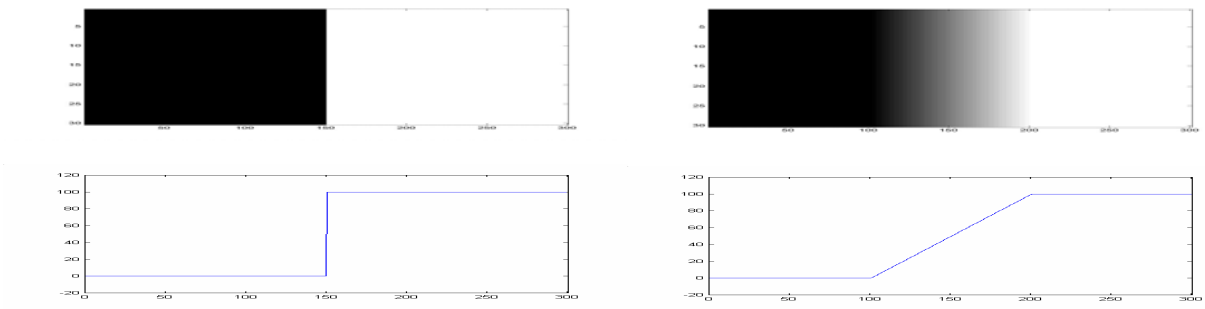



**Figura 2.4**

Nella figura 2.5 è mostrato un modello di contorno ideale (a), in cui la transizione di colore è istantanea e l'andamento della corrispondente derivata spaziale è un gradino. Nella pratica una discontinuità del genere non esiste a causa di problemi connessi all'ottica della telecamera, al campionamento e quantizzazione dei punti e ad altre imperfezioni del sistema di acquisizione, che producono bordi quanto meno sfumati (b).

La pendenza della rampa in figura (b) è inversamente proporzionale al grado di sfumatura del contorno, mentre la sua lunghezza risulta direttamente proporzionale ad esso: in questo senso, contorni sfumati avranno spessore maggiore, contorni più netti tenderanno ad essere fini.



(a) Modello di un contorno ideale

(b) Modello di un contorno a rampa

**Figura 2.5**

Nella figura 2.6 accanto si riportano gli andamenti spaziali di $g(x,y)$ e delle sue derivate prima e seconda al variare di y. Osservando l'andamento della derivata seconda si può evincere che la presenza di un contorno è sempre associata al passaggio per lo zero (*zero crossing*) di questa funzione. In conseguenza di queste ultime osservazioni, nell'*edge detection* si individuano due tipi di approcci:

1. Metodi del primo ordine (*Gradient based*)
2. Metodi del secondo ordine (*Laplacian based*)

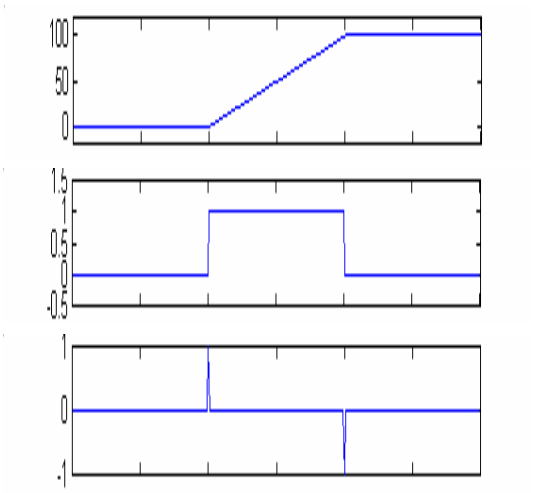



**Figura 2.6**

I metodi del secondo ordine, come appena osservato, rilevano le discontinuità e quindi i contorni applicando l'operatore **laplaciano** alla funzione $g(x,y)$. Successivamente si effettua lo *zero crossing* dell'immagine al fine di individuare i contorni. Ciò riverbera due effetti, uno positivo, l'altro negativo. Come primo effetto i bordi individuati avranno uno spessore di al più un pixel e pertanto i contorni saranno fini; come secondo effetto il rumore di fondo risulterà amplificato a causa delle caratteristiche evidentemente passa alto dell'operatore laplaciano.
I metodi del primo ordine si fermano invece alla derivata prima; le discontinuità vengono rilevate applicando l'operatore gradiente all'intera immagine, ed effettuando successivamente una sogliatura (*thresholding*) allo scopo di trattenere quei valori di gradiente di una certa entità. Tali metodi risultano in contorni più spessi, ma per contro sono sicuramente poco sensibili al rumore. Nel mio caso ho utilizzato un metodo differenziale del primo ordine giacchè le immagini acquisite, nella pluralità delle condizioni di illuminazione in cui si operava, erano fortemente affette da rumore.
Di seguito mi accingo a descrivere il funzionamento dell'algoritmo di estrazione dei contorni.

Il corrispondente discreto dell'operatore gradiente è ovviamente il calcolo delle differenze parziali lungo le due dimensioni x,y dell'immagine:

$$\begin{aligned}\Delta_x g(x,y) &= g(x,y) - g(x-n,y) \\ \Delta_y g(x,y) &= g(x,y) - g(x,y-n)\end{aligned} \qquad (2.6)$$

dove *n* (di solito pari ad 1) è un intero che deve essere scelto abbastanza piccolo da fornire una buona approssimazione della derivata, ma abbastanza grande da ignorare le discontinuità trascurabili. Nella definizione discreta appena data le due derivate parziali vengono calcolate in termini di differenze all'indietro, ma in una funzione immagine, in cui sono disponibili sia i valori precedenti sia quelli seguenti nel dominio spaziale, non c'è ragione per cui non possano essere calcolate anche le differenze in avanti, privilegiando l'una rispetto all'altra! Per questo motivo nella definizione più comune si utilizzano le differenze rispetto al valore centrale:

$$\begin{aligned} \Delta_x g(x,y) &= g(x+n,y) - g(x-n,y) \\ \Delta_y g(x,y) &= g(x,y+n) - g(x,y-n) \end{aligned} \tag{2.7}$$

Da un lato è evidente che tali componenti del vettore gradiente sono esse stesse degli operatori lineari, mentre il modulo no, perché calcolato secondo la [2.4] come radice della somma dei singoli quadrati. Dall'altro però le due componenti **non** sono **invarianti** alle rotazioni (cioè isotrope), mentre il modulo sì.

Poiché dal punto di vista computazionale la valutazione della [2.4] è critica e richiede dunque operazioni in virgola mobile, è pratica comune approssimare il modulo del gradiente utilizzando il valore assoluto, pertanto la [2.4] diventa:

$$|G| = |grad(g(x,y))| = \sqrt{\left(\frac{\partial g}{\partial x}\right)^2 + \left(\frac{\partial g}{\partial y}\right)^2} \approx |\Delta_x g| + |\Delta_y g| \tag{2.8}$$

### 2.2.2 Operatori differenziali di Sobel e Prewitt

La [2.7] è più semplice da calcolare e preserva ancora le peculiarità derivative originarie, tuttavia perde le caratteristiche di isotropia della prima definizione.

È per questo motivo che nella pratica comune le derivate parziali lungo le due componenti vengono calcolate mediante un'operazione lineare di convoluzione tramite maschere di dimensioni maggiori di uno e di coefficienti opportuni. Tali maschere agiscono da operatori differenziali spaziali e lineari.

Nell'algoritmo che ho sviluppato si utilizzano gli operatori di Prewitt o Sobel arbitrariamente:

$$h_x = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \qquad h_y = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

Prewitt masks

$$h_x = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \qquad h_y = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Sobel masks

**Osservazioni**

In relazione a ciascun metodo, $h_x$ risulta adeguata a rilevare transizioni, e quindi contorni, lungo la direzione x, mentre $h_y$ risulta efficace lungo la direzione y. Pertanto nessuna delle due è invariante alle rotazioni, ma la combinazione di esse fornisce ottimi risultati di edge detection. Come si vede la dimensione delle maschere risulta $3 \times 3$. Ciò, oltre a migliorare l'efficienza computazionale rispetto a maschere di dimensioni maggiori, effettua anche un'azione di filtraggio passa basso. Quest'ultima è meglio evidenziata nell'operatore di Sobel dalla presenza del fattore 2, il quale evidentemente attribuisce maggior importanza al valore centrale e pertanto manifesta una maggiore immunità ai disturbi. La differenza di prestazioni in termini di qualità dei contorni sarà mostrata nelle prossime figure. Si osservi inoltre che i coefficienti di tali operatori sono interi, risparmiando così costose operazioni *floating-point* che pregiudicherebbero l'elaborazione real-time come invece accadeva nella fase di *smoothing*. Ai fini di una implementazione più efficiente dell'algoritmo le moltiplicazioni per 2 sono sostituite dagli operatori di *shift* "<<1" disponibili nel linguaggio C.

### 2.2.3 Implementazione dell'algoritmo di Edge Detection

Ecco in breve come funziona l'algoritmo che, a partire dall'immagine iniziale $g(x, y)$, costruisce l'immagine finale $F(x, y)$ contenente solo i punti di contorno di $g$:

1. per ciascuna posizione di pixel $(x, y)$ dell'immagine $g(x, y)$ calcola le corrispettive componenti del gradiente $G_x, G_y$, applicando uno degli operatori suddetti (di Sobel o di Prewitt):

$$G_x(x,y) = \sum_{k=-1}^{1}\sum_{l=-1}^{1} h_x(k,l)\cdot g(x+k,y+l)$$
$$G_y(x,y) = \sum_{k=-1}^{1}\sum_{l=-1}^{1} h_y(k,l)\cdot g(x+k,y+l) \qquad (2.9)$$

2. successivamente computa il modulo del gradiente $|G|$, perdendo in questo modo le caratteristiche di linearità della convoluzione; l'immagine così ottenuta è detta immagine differenziata o gradiente:

$$|G(x,y)| = |G_x(x,y)| + |G_y(x,y)| \qquad (2.10)$$

3. l'immagine finale $F(x,y)$ (binaria perché vale 1 solo in corrispondenza di un contorno) si ottiene applicando all'immagine "differenziata" la seguente funzione non lineare di *thresholding*: ovvero una soglia "globale" che interessa in egual modo ciascun pixel dell'immagine.

$$F(x,y) = \begin{cases} 1 & \text{se } |G(x,y)| >= T \\ 0 & \text{altrimenti} \end{cases} \qquad (2.11)$$

### 2.2.4 Risultati dell'algoritmo di edge detection

Nelle foto che seguono sono illustrati i tre passaggi del processo di *edge detection*, forniti dall'algoritmo che ho implementato; come campione ho utilizzato un'immagine acquisita da una delle webcam oggetto di studio, all'interno del laboratorio di robotica della nostra facoltà. Il seguente diagramma a blocchi mostra la sequenza delle fasi a cui è sottoposta l'immagine.

**1.** È possibile vedere al centro la presenza di una palla (che stata utilizzata sin dall'inizio ai fini della *ball detection*), ed una pluralità di oggetti distinti sullo sfondo. I numeri riportati a lato indicano la posizione x dei pixel, quelli in basso la y.

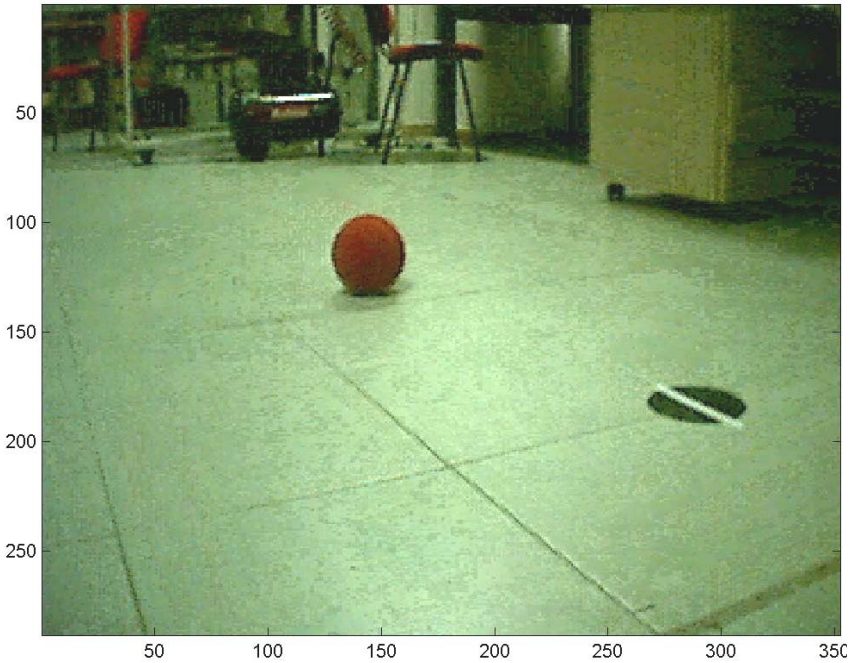



**Figura 2.7**

**2.** Applicando l'operatore di Prewitt all'intera immagine, si ottiene quella differenziata, in cui a ciascuna posizione di pixel è associato un valore intero corrispondente al modulo del gradiente; di seguito ecco il risultato della convoluzione delle due maschere di Prewitt, rappresentato in scala di grigi:

**Figura 2.8**

**3.** A questo punto l'immagine gradiente viene sottoposta alla fase di *Thresholding*. Questa è una delle operazioni più delicate poiché dalla scelta del valore di soglia *T* dipende la qualità dei contorni da utilizzare per il riconoscimento della palla. Un valore di soglia troppo alto può portare a trascurare dettagli utili, mentre uno troppo basso può evidenziare contorni deboli ma anche insignificanti!

Dopo varie prove effettuate non solo in laboratorio, la miglior scelta è risultata T=140.

Ecco quindi il risultato della fase di *Thresholding* avendo imposto T=140:

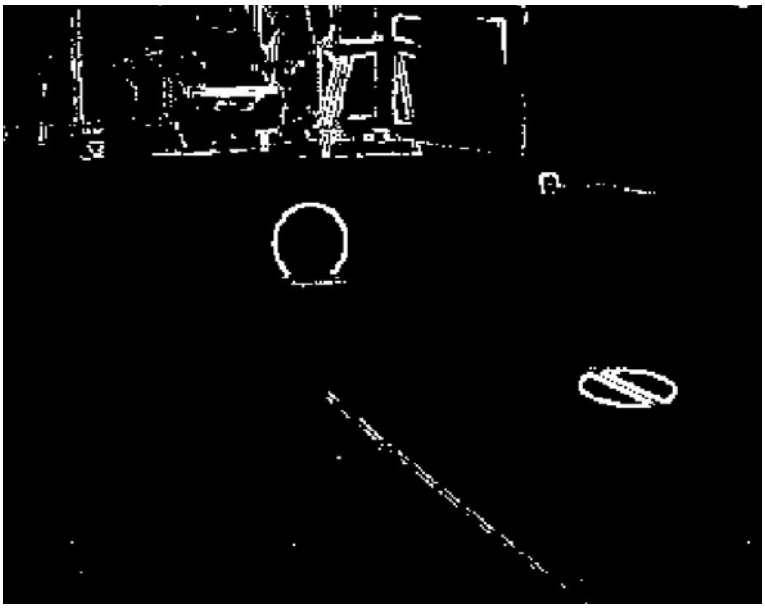


**Figura 2.9**

Si osservi che l'immagine binaria finale presenta dei contorni piuttosto spessi (come evidente dal dettaglio della palla qui accanto); ciò è dovuto al fatto che un maggior numero di punti in generale si trova al di sopra del valore di soglia imposto.

Un'ulteriore fase di filtraggio non lineare è pertanto necessaria al fine di ridurre ulteriormente il numero di punti di contorno; tale processo è detto *edge thinning*.

**Figura 2.10**

### 2.2.5 L'algoritmo di Edge Thinning

Nell'algoritmo ho implementato due metodi diversi per "raffinare" i contorni:

1. Il primo, noto dalla letteratura [8], consiste nell'individuare in seno ai punti ottenuti dal processo di *thresholding* quelli che si trovano in corrispondenza dei punti di massimo del gradiente, pertanto sarà denominato "maxima".
2. Il secondo, da me ideato, trae origine dall'osservazione che ai fini del ritrovamento della palla sono indispensabili quei soli punti che giacciono in corrispondenza di un arco di curva. Linee orizzontali o verticali non sono pertanto necessarie; anzi contribuiscono a rallentare il processo di riconoscimento oltre che ad aumentare inutilmente il numero di punti di contorno. Questo verrà chiamato "fine".

Nell'algoritmo ho implementato entrambe le strategie di *edge thinning* testandone le prestazioni in termini di funzionalità (ai fini del riconoscimento) ed efficienza di calcolo.
Ecco come funzionano i 2 processi:

#### 2.2.5.1 Prima versione di edge thinning: "maxima"

**1.** Dopo aver operato il *thresholding* dell'immagine gradiente, si esamina ciascun punto candidato (avente quindi $|G(x,y)| >= T$ ) .

Consideriamo una regione quadrata di punti di dimensioni 5x5 centrata sul punto di coordinate $(x,y)$ a cui è associato un'ampiezza del gradiente pari a **Z3>=T** (si veda figura in basso).

In forma di pseudocodice ecco come funziona questa versione dell'algoritmo di *edge thinning* :

**SE** Z3**>** {Z1, Z2, Z4, Z5} **OPPURE** Z3>{Z6, Z7, Z8, Z9}.
**ALLORA** $F(x,y)=1$
**ALTRIMENTI** $F(x,y)=0$

| | | | | |
|---|---|---|---|---|
| | | Z6 | | |
| | | Z7 | | |
| Z1 | Z2 | **Z3** | Z4 | Z5 |
| | | Z8 | | |
| | | Z9 | | |

Tale analisi viene ripetuta per tutti i punti dell'immagine gradiente fornendo una $F(x, y)$ finale dello spessore di al più un pixel (figura 2.11 e dettaglio ingrandito).

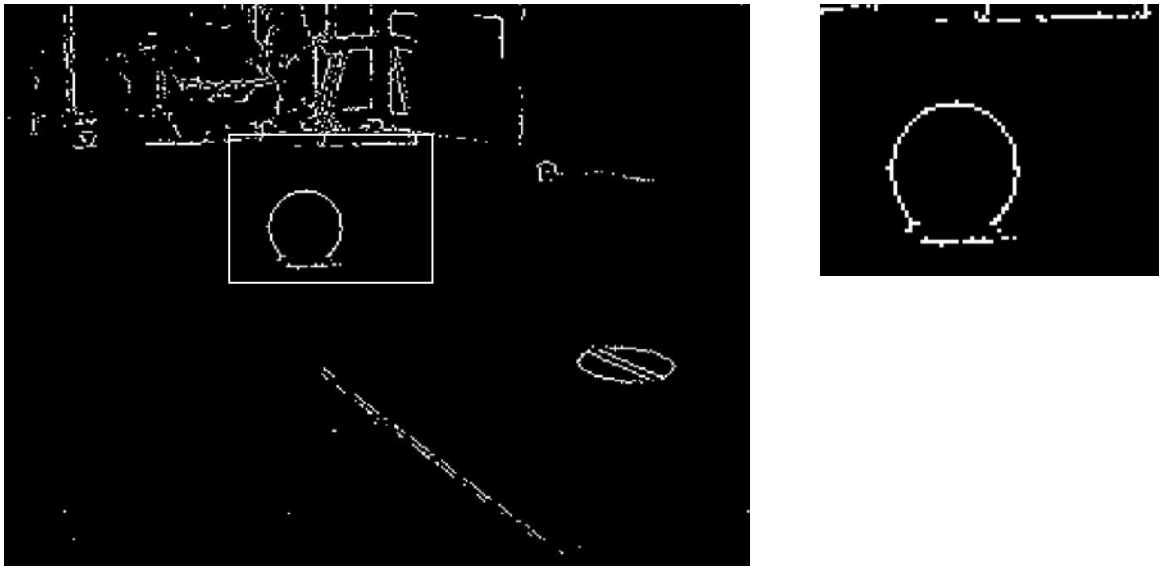


**Figura 2.11**

### 2.2.5.2 Seconda versione di edge thinning: "fine"

**2.** Il secondo metodo di *edge thinning* da me implementato funziona in questo modo: se individua un punto di contorno con un valore di gradiente maggiore della soglia, salta alla posizione di pixel $(x + delta, y)$! Dove $delta$ è un numero intero specificato dall'utente in funzione della qualità dei contorni desiderata: se $delta = 1$ si ottiene l'immagine dai contorni spessi di cui parlavamo. Un valore di $delta = 8$ fornisce invece risultati migliori in termini di efficienza di calcolo ai fini del *ball detection*.

In forma di pseudocodice ecco come funziona questa versione dell'algoritmo di *edge thinning* :

Dapprima si settano a 0 tutti i valori di $F(x, y)$, quindi per ogni $(x, y)$ :

**SE** $|G(x, y)| >= T$

**ALLORA**

$F(x, y) = 1$

**SALTA** ALLA POSIZIONE $(x + delta, y)$

**ALTRIMENTI**

**SALTA** ALLA POSIZIONE $(x + 1, y)$

Ecco il risultato del nuovo metodo di raffinamento dei contorni:

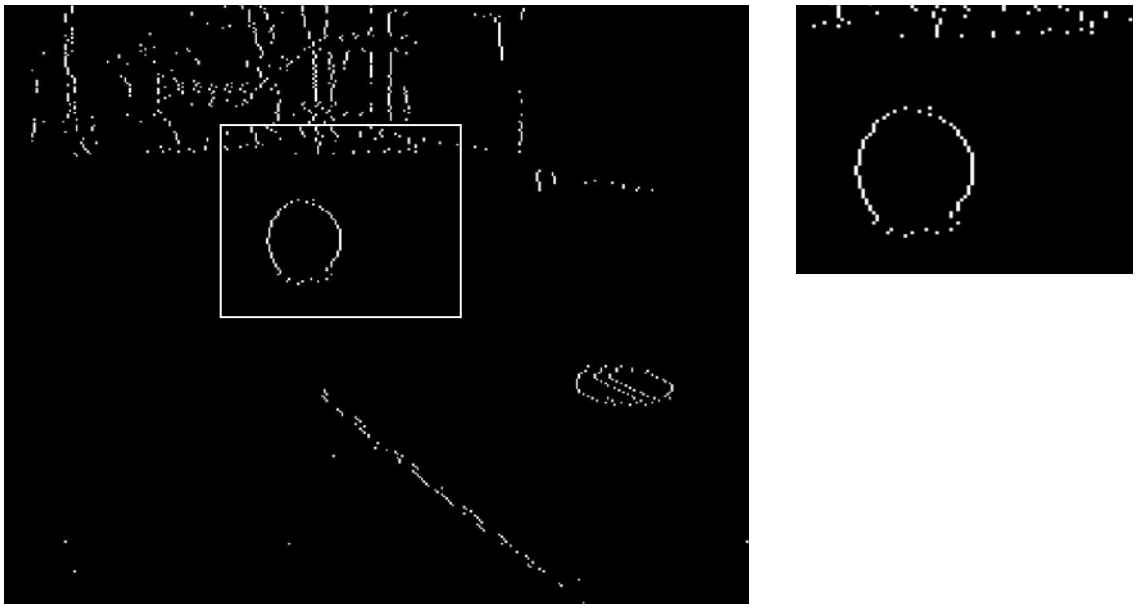
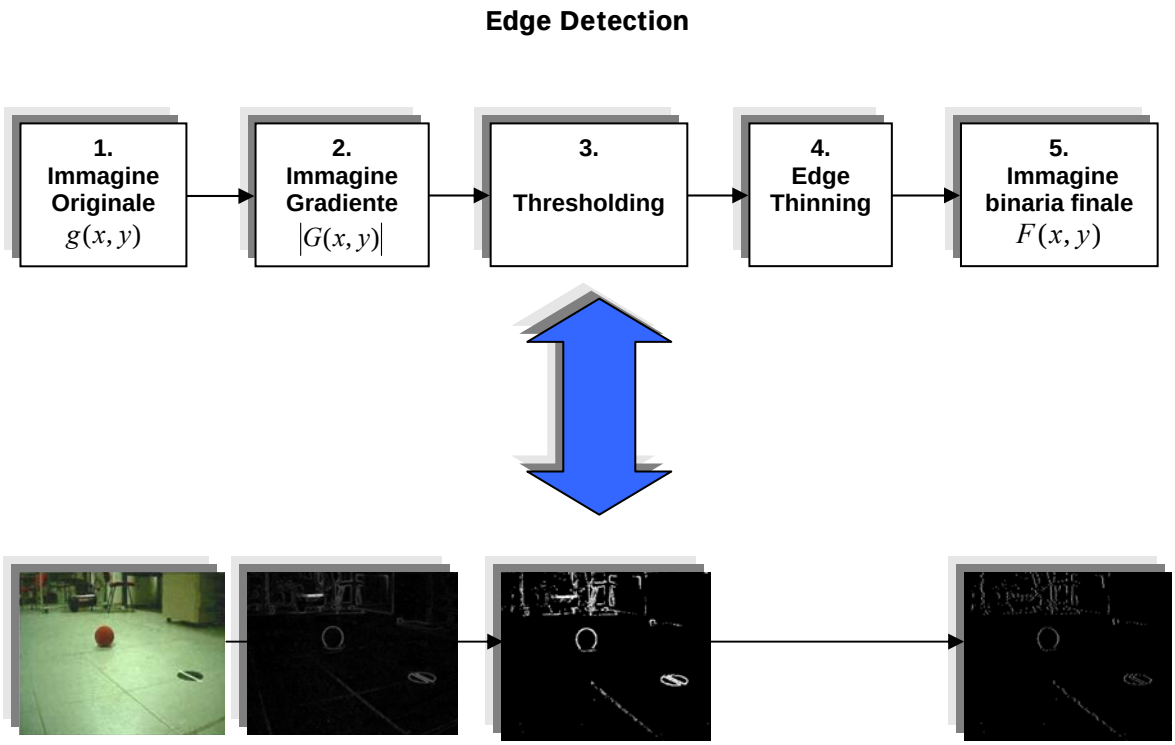


**Figura 2.12**

Si osservi come i contorni, oltre ad avere uno spessore di al più un pixel, risultano visibilmente "corrosi"; ma tale erosione interessa escusivamente i contorni rettilinei (soprattutto quelli orizzontali oltre verticali od oblliqui) senza "contaminare" in alcun modo quelli curvilinei (si osservi il dettaglio della palla, che viene preservato nella sua integrità)!

I due metodi di raffinamento sono stati implementati in due versioni distinte dell'algoritmo di riconoscimento della palla allo scopo di valutarne le prestazioni in situazioni diverse.

Lo schema a blocchi seguente riassume sinteticamente il processo globale di estrazione dei contorni:




Nel prossimo paragrafo entrerò nel merito del riconoscimento effettivo della palla, che opera direttamente sull'immagine a valle di quest'ultima fase descritta, ovvero sui punti di contorno.

## 2.3 Riconoscimento di una palla


Il riconoscimento della palla costituisce l'ultimo stadio della catena di elaborazione delle immagini. Posto a valle della fase di estrazione dei contorni, esso mira ad individuare la presenza di una palla riconoscendone la forma tra una pluralità di oggetti presenti che possono essere causa di confusione.

La scelta di una palla rispetto ad altri possibili corpi è dettata dal fatto che essa appare sferica, o meglio circolare, da qualsiasi posizione venga osservata. Un fatto, questo, non trascurabile se pensiamo che la totalità degli oggetti presenti in natura trasla e ruota apparendo sotto forme sempre diverse.

Se a prima vista il problema di riconoscimento di un oggetto così elementare può apparire semplice, esso risulta in realtà enormemente complicato a causa delle innumerevoli e mutevoli condizioni in cui si opera oltre che dalla necessità di "descrivere" ad una macchina come è fatta una palla. Ciò mi ha dato modo di penetrare più a fondo il mondo della *Machine Vision*, scontrandomi con i problemi reali della visione computazionale, che a differenza di quella umana non distingue un oggetto dalla sua ombra, né un riflesso di luce da quello di una sorgente diretta.

Partendo da quanto appena detto discende che è possibile spostare il problema del *ball detection* al riconoscimento di forme circolari: *circle detection*. Ma non solo, o perlomeno non completamente. Vediamo quali sono dunque gli obiettivi che mi sono prefisso in relazione ai problemi che si riscontrano quotidianamente nella visione computazionale.

Ai fini dell'implementazione di un algoritmo robusto, preciso ed efficiente nel riconoscimento ecco i traguardi che esso dovrebbe raggiungere:

1. deve resistere all'eventuale occlusione di una porzione di superficie, almeno fino a metà di essa
2. non deve soffrire di problemi di scala dovuti all'allontanamento o all'avvicinamento della palla
3. deve essere robusto alla vicinanza di altri oggetti che pur non essendo circolari possano essere causa di ambiguità
4. deve discriminare il cerchio che contorna la palla da quello dell'ombra che essa proietta sul pavimento o su un'eventuale parete di fondo

5. deve poter operare in condizioni di bassa luminosità (illuminazione artificiale scarsa) o in presenza di luce naturale o anche contro luce
6. deve ignorare i riflessi che la luce genera sulla superficie di essa o quelli della palla sul pavimento (entrambi causa di ambiguità)
7. naturalmente deve prescindere dal colore della palla (ma quest'ultimo fatto sappiamo essere ininfluente giacchè il riconoscimento parte solo dai contorni)
8. deve essere computazionalmente veloce, tale da garantire un'elaborazione real-time

Come si vede sono state considerate tra gli obiettivi la capacità di operare in diverse condizioni di illuminazione, giacchè queste introducono forti modifiche sull'apparenza della palla nell'immagine: quando sequenze del moto di essa sono acquisite con luce naturale la palla appare infatti come una coppa sferica, e di ciò bisogna dunque tenere conto!

Veniamo quindi alla descrizione dell'algoritmo. Prima di giungere alla soluzione finale ho effettuato una lunga ricerca di quanto presente allo stato dell'arte in tema di riconoscimento di forme, o di come viene chiamato dalla comunità internazionale *pattern recognition*.
La determinazione di cerchi, nelle immagini digitali, è uno dei problemi più importanti nelle applicazioni industriali di visione poiché oggetti circolari ricorrono frequentemente in ambiti naturali e artificiali. Nel campo medico ad esempio, la diffusione della robotica o dei sistemi avanzati di assistenza medica autonoma, ha favorito notevolmente lo sviluppo di tecniche di *circle detection* [37, 39]. Tra gli esempi in ambito medico posso citare i sistemi di riconoscimento automatico di embrioni, neuroni o dell'iride (quest'ultima ha trovato applicazione nei sistemi di identificazione basati sulla retina) (immagini a lato).
Ma la determinazione di circonferenze investe anche l'ambito astronomico (nei sistemi di localizzazione astronomica) e geologico, per la ricerca di nuovi crateri da immagini satellitari [37].

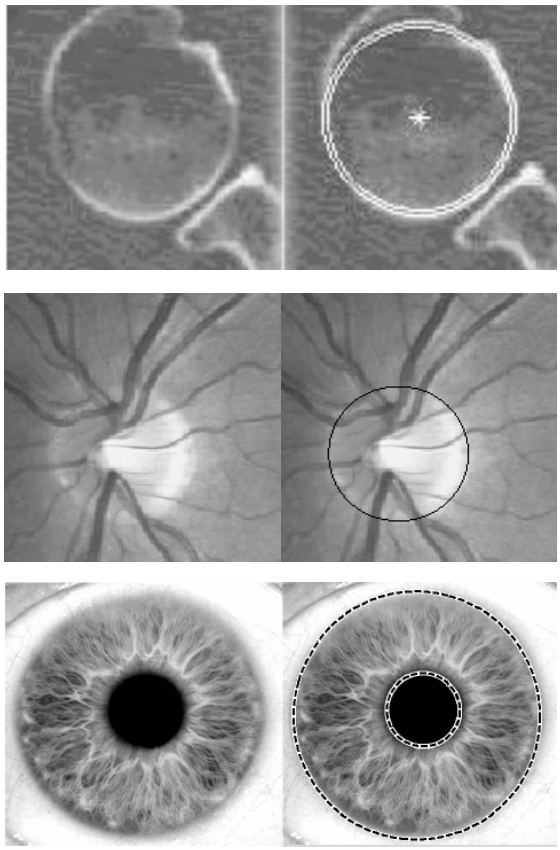


**Figura 2.13**

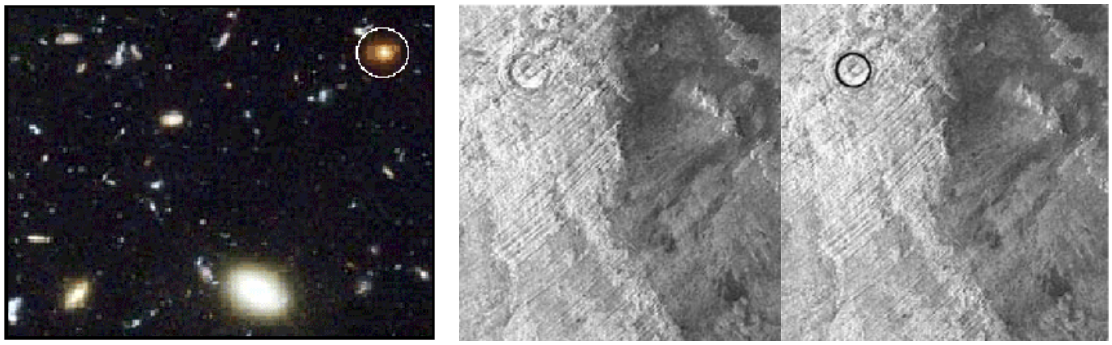


**Figura 2.14**

Ai fini della determinazione di cerchi negli esempi citati sono state sviluppate nel corso degli ultimi dieci anni un gran numero di tecniche. Tra queste la **Circle Hough Transform** (**CHT**) [39] e le sue molteplici versioni modificate sono state riconosciute come tecniche robuste di riconoscimento.

Tuttavia l'implementazione di un algoritmo basato sulla trasformata di Hough per i cerchi risulta computazionalmente molto onerosa ($\propto n^3$ dove $n$ è la dimensione tipica di un lato dell'immagine), pertanto nel corso di questa tesi ho sviluppato diverse tecniche di *circle detection* allo scopo di valutarne l'affidabilità e l'efficienza rispetto a quelle fornite dalla **CHT**.

### 2.3.1 Le tre versioni dell'algoritmo di *ball-detection*

Ho realizzato **tre versioni** distinte di **ball-detection**, che verranno ora prese in esame e descritte accuratamente. Ciascun algoritmo è stato dapprima testato in **Matlab (versione 6.1)** allo scopo di valutare la correttezza di ciò che volevo ottenere, e successivamente scritto in **linguaggio C**. Matlab permette infatti un uso molto veloce ed intuitivo della programmazione grazie anche alla facile gestione delle matrici sotto cui è possible memorizzare qualsiasi lista di elementi. Tuttavia esso risulta ineluttabilmente lento nell'esecuzione di loop, pertanto il passo successivo è stato quello di tradurre in **C** l' "idea embrionale" testata preventivamente in Matlab. Ciascun algoritmo è stato attentamente reso efficiente eliminando le operazioni in virgola mobile (praticamente assenti nell'ultima versione), facendo largo uso di *shift* (ove possibile) e soprattutto utilizzando esclusivamente l'accesso agli elementi tramite **puntatori**, molto più veloce ed efficiente degli array.

In breve ecco le tre versioni dell'algoritmo di *ball- detection* che mi accingo a descrivere:

1. Prima versione, basata sulla **Circle Hough Transform** (**CHT**)
2. Seconda versione, basata sulla **Trasformata di Hough Modificata** sfruttando l'informazione sul gradiente fornita nella fase di *edge-detection*

3. Terza ed ultima versione, da me ideata, basata su una ricerca locale (**Pixel To Pixel**)

La prima costituisce un'applicazione delle tecniche presenti nella letteratura della visione computazionale. Come già antcipato la complessità computazionale della **CHT** è dell'ordine di $n^3$, rendendo praticamente impensabile una elaborazione real-time su hardware non dedicato come il PC da me utilizzato. La seconda versione risponde proprio a questa necessità riducendo l'ordine di grandezza ad $n^2$; questa versione risulta abbastanza veloce ma con il tempo si è rivelata insufficiente a funzionare nelle molteplici condizioni di lavoro ed illuminazione che ho elencato, poiché sfrutta l'informazione contenuta nell'immagine gradiente, di cui parlavamo nella precedente sezione, e dipende quindi fortemente dal contrasto dell'immagine.
Le prima due versioni effettuano una ricerca **globale** dei cerchi nel senso che ogni singolo pixel contribuisce indipendentemente da tutti gli altri ad arricchire il numero di punti in uno spazio di ricerca, detto spazio dei parametri.
L'ultima versione, al contrario effettua una ricerca **locale**, ovvero si limita a quei soli punti che hanno una buona probabilità di appartenere ad un cerchio. La complessità computazionale si riduce ulteriormente a causa sia della diminuzione del numero di punti necessari, che per il minor numero di operazioni da effettuare.

Descriverò ora ciascun algoritmo nell'ordine cronologico con cui è stato affrontato.

### 2.3.2 Prima versione: Circle Hough Transform (CHT)

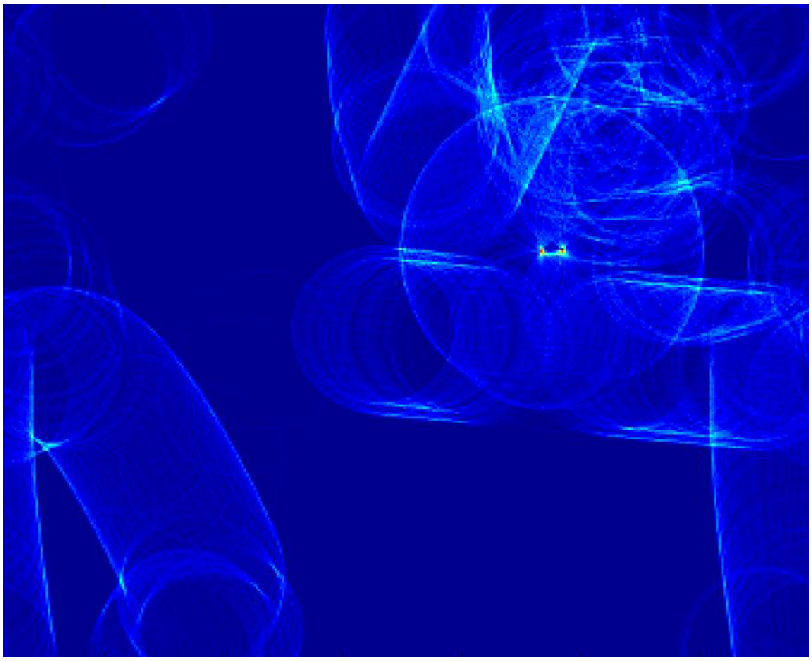


**Figura 2.15** Un'immagine reale della trasformata di Hough per i cerchi

#### 2.3.2.1 La trasformata di Hough per i cerchi

La trasformata di Hough può essere usata per determinare i parametri di un cerchio quando è noto un certo numero di punti che cade in corrispondenza del suo perimetro. Sappiamo che una circonferenza di raggio $R$ e centro di coordinate $xc$ e $yc$ può essere descritta dalle seguenti equazioni parametriche:

$$\begin{aligned} x &= xc + R \cdot \cos(\vartheta) \\ y &= yc + R \cdot \sin(\vartheta) \end{aligned} \tag{2.12}$$

Se un'immagine contiene un gran numero di punti, alcuni dei quali giacciono in corrispondenza del perimetro di alcuni cerchi, il compito dell'algoritmo è quello di trovare le terne di parametri $(xc, yc, R)$ che descrivono ciascuna circonferenza. Il fatto di avere proprio una **terna** di parametri (*3D Parameter Space*) rende l'implementazione diretta della tecnica di Hough, come vedremo, computazionalmente costosa, sia dal punto di vista delle risorse di memoria che per il tempo di esecuzione. La memoria è un dettaglio che non bisogna trascurare soprattutto se l'obiettivo è quello di esportare il codice su un hardware dedicato, magari molto compatto, e in cui le risorse di memoria giocano un ruolo fondamentale!

Se i cerchi presenti in un'immagine tuttavia sono di raggio noto e pari ad $R_0$, allora lo spazio dei parametri si riduce ovviamente a due dimensioni (2D Parameter Space), rendendo così necessario stimare solo le coordinate dei centri $(xc, yc)$.

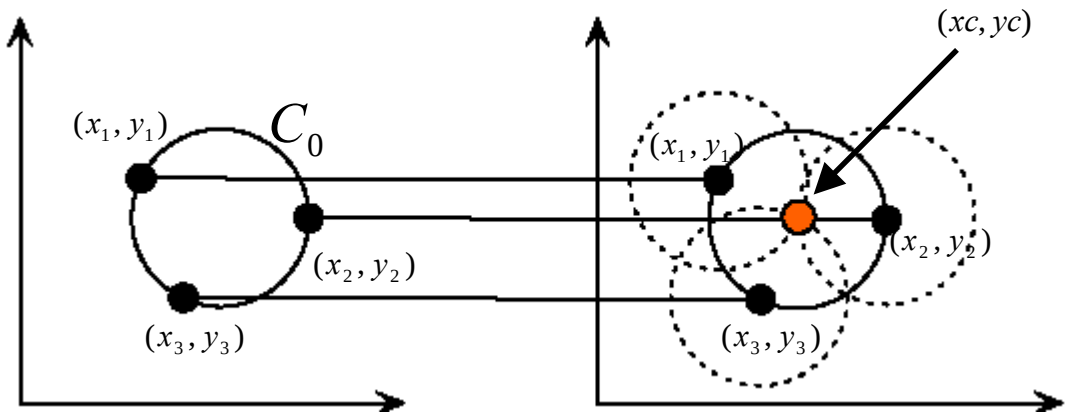
Il nucleo della trasformata di Hough consiste nell'uso di una matrice o array di celle accumulatrici, detto **spazio dei parametri**.

Per ciascun punto nell'immagine di partenza, detta **spazio geometrico**, di coordinate $(x, y)$, la tecnica di Hough richiede di tracciare un corrispondente cerchio nello **spazio dei parametri**, che stavolta ha centro proprio in $(x, y)$ e raggio noto $R_0$. Infatti come si osserva dalle equazioni parametriche della circonferenza [2.9] si ha anche che

$$\begin{aligned} xc &= x + R_0 \cdot \cos(\vartheta) \\ yc &= y + R_0 \cdot \sin(\vartheta) \end{aligned} \tag{2.13}$$

ovvero, il **centro** di un cerchio detto $C_0$ è proprio il punto comune a tutte le circonferenze di raggio $R_0$ centrate su un punto qualsiasi del **perimetro** di $C_0$!

Per comprendere meglio si osservi la figura esplicativa 2.16, che illustra il procedimento appenda descritto.




Ogni punto dello spazio geometrico (sinistra), genera un cerchio nello spazio dei parametri (destra). I cerchi nello spazio dei parametri si intersecano in $(xc, yc)$ che è il centro del cerchio nello spazio geometrico

**Figura 2.16**

In questo modo è evidente come a partire da qualsiasi immagine dello spazio geometrico si ottiene un'immagine di soli cerchi nello spazio dei parametri, detta **trasformata di Hough**.

Nella realtà poiché lo spazio dei parametri non è altro che un array di celle o accumulatori di numeri interi, ciascuna cella, inizialmente posta a zero, viene incrementata di uno al passaggio di un cerchio. In questo modo il centro del cerchio è quello a cui corrisponde un picco di massimo nello spazio dei parametri.
La presenza di più cerchi può essere individuata con la stessa tecnica come nella figura 2.17:

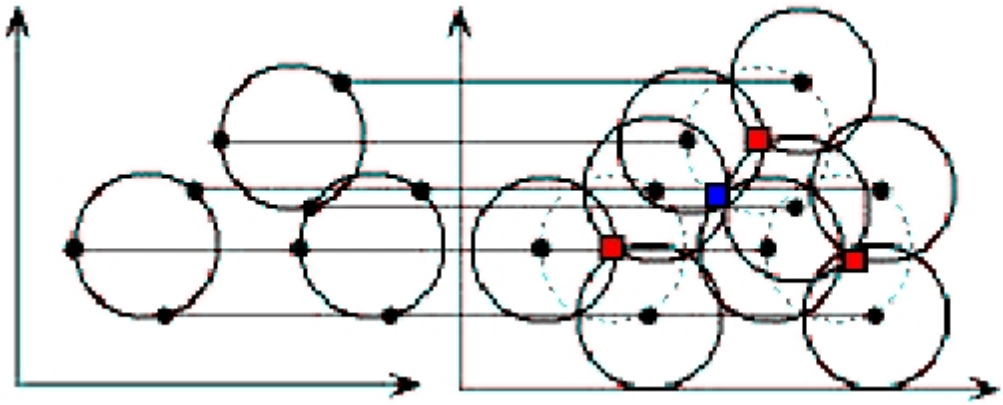


**Figura 2.17**

I centri sono raffigurati in rosso nello spazio dei parametri. La sovrapposizione di alcuni cerchi può causare ambiguità (celle in blu ma con valore di picco minore), che possono essere tuttavia rimosse tramite il *matching* con i cerchi dell'immagine originale.

Sino ad ora abbiamo supposto che il raggio fosse noto a priori, ma nell'applicazione al riconoscimento della palla tale parametro varia in funzione della distanza. Nello spazio dei parametri occorre dunque considerare anche il raggio, tracciando più cerchi, di diametro crescente, per ciascun valore di $(x, y)$; in questo caso il luogo dei punti nello spazio della trasformata giacerà in corrispondenza della superficie di un cono. Ovvero, ogni punto di coordinate $(x, y)$ nello spazio geometrico genera un cono nello spazio dei parametri.

Anche in questo caso, se i punti $(x_i, y_i)$ nello spazio geometrico appartengono ad una circonferenza di raggio $R$, la terna $(xc, yc, R)$ che intendiamo stimare si troverà in corrispondenza della cella accumulatrice in cui si interseca il maggior numero di coni nello spazio dei parametri.
La figura 2.18 illustra il procedimento appena descritto. Come si vede, per ogni $i$ vengono costruite più circonferenze centrate in $(x_i, y_i)$ per ciascun valore di $R$ ammissibile.

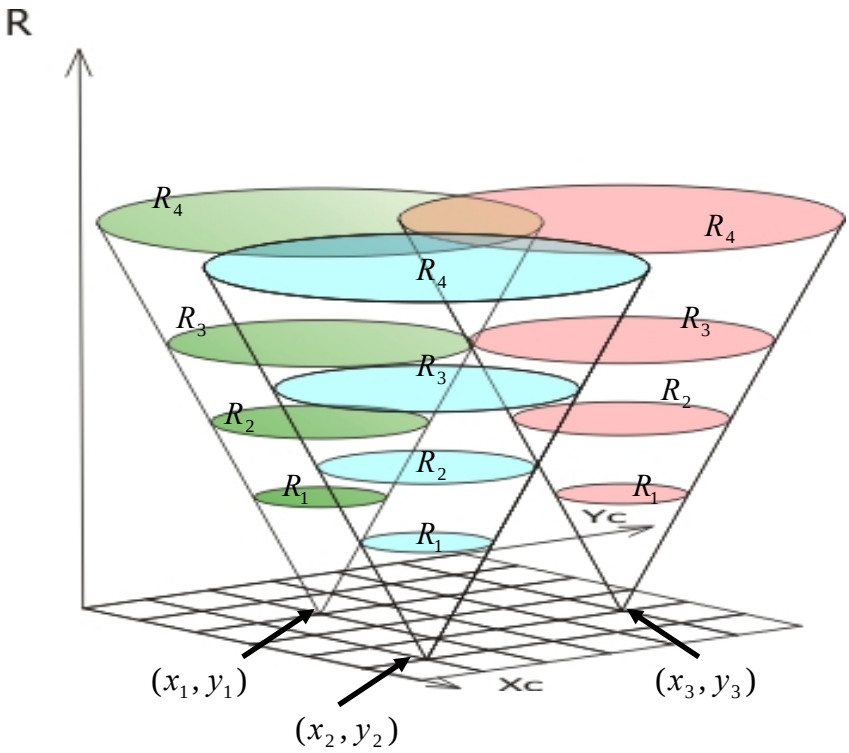



**Figura 2.18**

È evidente che non potendo far variare il raggio in $[0, \infty]$, occorrerà scegliere un range discreto di valori ammissibili: maggiore è la dimensione di tale intervallo, tanto più grande sarà la complessità computazionale e la memoria richieste.

#### 2.3.2.2 Implementazione dell'algoritmo di CHT

Ecco in breve come funziona l'algoritmo:

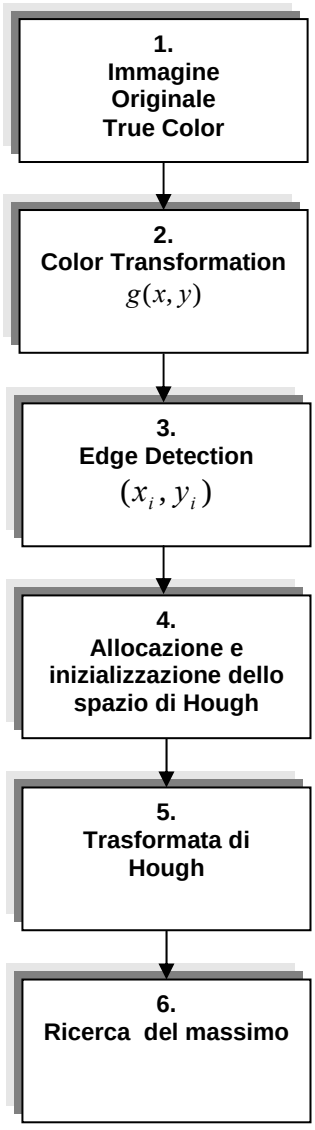
1. Trasforma l'immagine di partenza True-color in una monocromatica (**Color Transformation**).
2. A partire dall'immagine monocromatica richiama la funzione `edge_detection`, che restituisce le coordinate $(x_i, y_i)$ dei punti di contorno secondo uno dei due metodi di raffinamento dei bordi di cui ho parlato: in questo caso è impostata per default la modalità "*maxima*".
3. Viene allocato uno spazio di memoria sufficiente ad accogliere la matrice accumulatrice tridimensionale che costituisce lo spazio dei parametri.
4. Per ciascun valore di $i$ vengono costruite più circonferenze centrate in $(x_i, y_i)$ per ogni valore ammissibile di $R$. NB: Nel caso in cui il punto della circonferenza giaccia al

di fuori de condominio il programma passa direttamente al successivo punto da tracciare proseguendo iterativamente.

5. Ricerca il punto di massimo all'interno della matrice di accumulatori restituendone le coordinate $(xc, yc, R)$

Lo schema a blocchi seguente riassume sinteticamente il processo globale di Circle detection:

**Circle Detection : prima versione**



### 2.3.2.3 Risultati dell'algoritmo di CHT

Il diagramma seguente invece illustra la successione dei passaggi descritti su un'immagine campione acquisita nel laboratorio e utilizzata nelle successive fasi di test proprio per la pluralità di contorni che potrebbero arrecare ambiguità e per la presenza di oggetti (le sedie) aventi lo stesso colore della palla (ma tuttavia ininfluenti).

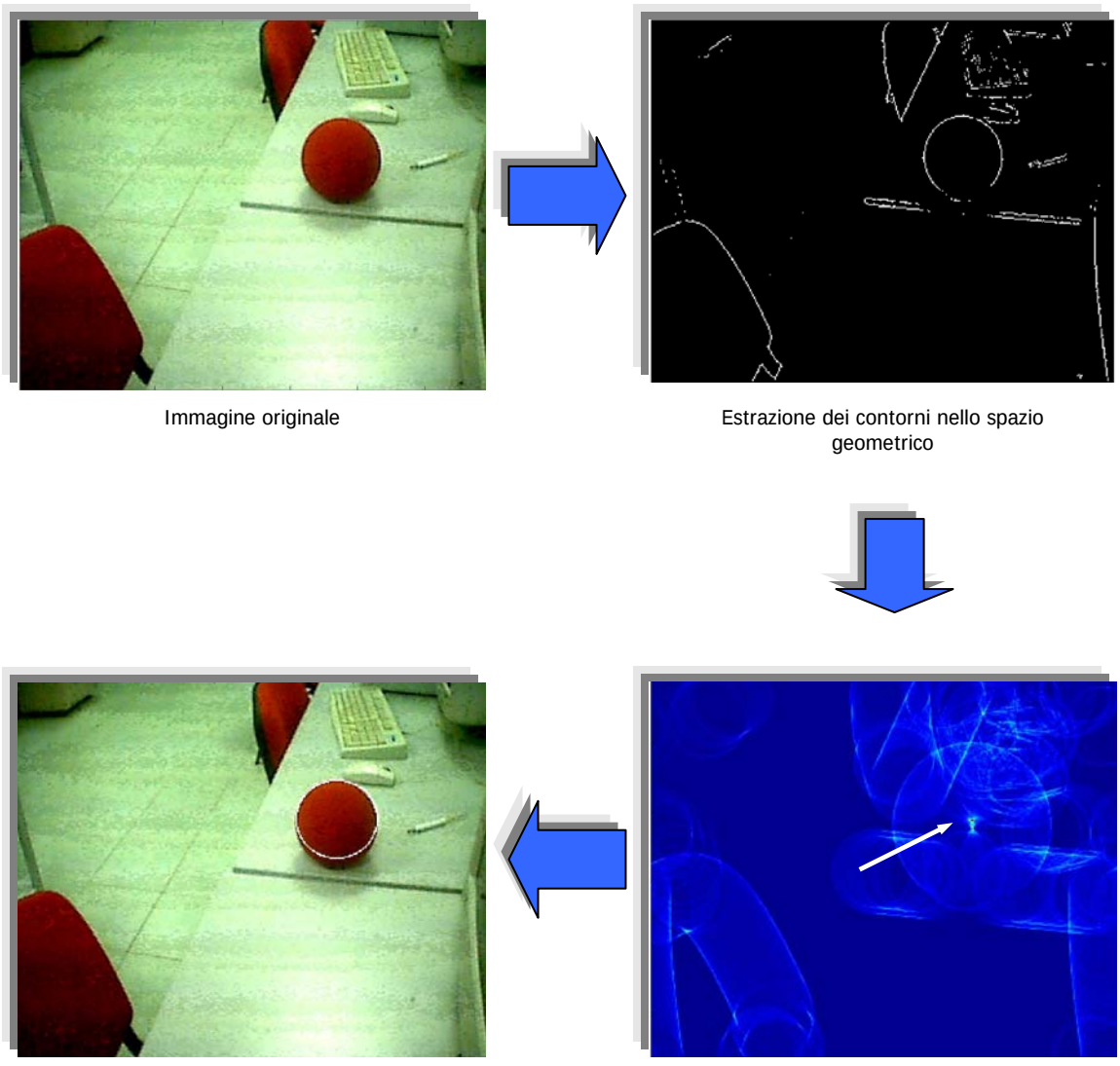


Immagine originale

Estrazione dei contorni nello spazio geometrico

Palla individuata e cerchiata nell'immagine di partenza

Una sezione dello spazio dei parametri corrispondente al punto di picco massimo, scelto come miglior candidato al centro della palla (indicato dalla freccia). Colori più caldi rappresentano celle ad accumulazione maggiore, in una scala che va da 0 (in blu scuro) al massimo (in rosso scuro)

**Figura 2.19**

#### 2.3.2.4 Osservazioni e prestazioni dell'algoritmo

Ciascuna cella accumulatrice è in formato `int` a 32 bit.

Le dimensioni di tale matrice sono `width,height,estr`, di valore rispettivamente: 352, 288, 40, ovvero 352x288x40=4.055.040 celle di memoria, per un totale di circa 15 Mbyte. `Estr=40` è il numero di cerchi che devono essere tracciati per ogni punto nello spazio dei parametri.

Il tracciamento dei cerchi richiede di utilizzare le equazioni parametriche del cerchio, per cui risulta necessario richiamare le funzioni `sin()` e `cos()` che rallentano inequivocabilmente il tempo di calcolo. Per questo ho definito una look-up-table in cui sono memorizzati tali valori al variare di $\vartheta$ in [0,360] ad intervalli di 1° (sufficiente al tracciamento di cerchi alla risoluzione richiesta). La funzione che inizializza la tabella di look-up è `init_SinCos()`.

La funzione per il tracciamento dei cerchi è `drawcircle(int *image, float xy[2], int raggio)`, che disegna in **`image`** un cerchio di centro **`xy`** e raggio **`r`**.

Nonostante l'uso della look-up-table, l'algoritmo effettua comunque operazioni in virgola mobile; il numero di somme e moltiplicazioni richieste è lineare e vale $n \cdot 40 \cdot 360$, dove $n$ è il numero di punti estratti nella fase di *edge-detection* (dell'ordine delle migliaia), 40 è il numero di cerchi tracciati e 360 sono tutti i possibili valori che $\vartheta$ può assumere.

Il tempo per elaborare un singolo fotogramma (dalla fase di edge-detection alla trasformata di Hough) risulta pari a circa **2,5 secondi**, mentre le risorse di memoria necessarie ammontano a circa **16 Mbyte**!

## 2.3.3 Seconda versione: Circle Hough Transform Modificata

### 2.3.3.1 I limiti della precedente versione e l'idea base dell'algoritmo

Dalla precedente trattazione si evince che l'approccio basato sul calcolo della trasformata di Hough completa richiede elevate risorse in termini di spazio e velocità; ciò è dovuto per la maggior parte all'uso di uno spazio parametrico (o matrice accumulatrice) tre dimensioni e alla necessità di tracciare dei cerchi completi. Pertanto una prima semplificazione potrebbe essere quella di non tracciare l'intera circonferenza per ciascun punto di contorno ma di sfruttare l'informazione contenuta nel gradiente sulla direzione dei bordi e tracciare dunque archi di circonferenza solo nelle regioni di spazio in cui ci aspettiamo di trovare il centro! D'altra parte però, sappiamo che il centro di un cerchio è dato dall'intersezione delle rette **normali** alla circonferenza, per cui, se è nota la direzione di un bordo ($\nabla G$), sono noti automaticamente i parametri direttori della retta ad esso perpendicolare. In tal modo un'ulteriore semplificazione è quella di sostituire il tracciamento di archi di cerchio con il tracciamento di rette. Naturalmente non tutte le rette tracciate a partire dai punti della circonferenza si intersecheranno nel suo centro, a causa della conoscenza non esatta della direzione del bordo e della natura discreta dello spazio dei parametri, ma ci sarà comunque una forte accumulazione nell'intorno di esso!

Ovviamente essendo lo spazio di Hough tridimensionale sto parlando del tracciamento di rette in uno spazio 3D: ovvero se nella prima versione per ciascun punto di bordo veniva costruito un **cono** nello spazio della trasformata, ora sono tracciate soltanto le sue **rette generatrici** (nella direzione normale al bordo).

Fino ad ora ho continuato ad assumere che lo spazio parametrico di Hough avesse dimensione tre ma ciò risulta indispensabile solo nei casi in cui alla necessità di individuare il centro si accompagni quella di ricavare il raggio del cerchio. Osserviamo tuttavia che il raggio è un'informazione utile solo nei casi di visione monoscopica, per capire se la palla si sta allontanando o avvicinando dall'osservatore. Ma in caso di visione stereoscopica, binoculare, tale informazione si ricava direttamente dal calcolo della distanza istantanea tra palla e osservatore!

Eliminando dunque il parametro $R$, la dimensione dello spazio dei parametri si riduce a due con una conseguente minor occupazione di memoria.

Rimpiazzando inoltre il tracciamento di cerchi con quello di rette nello spazio bidimensionale di Hough si riduce ulteriormente la complessità computazionale.

Pertanto a ciascun punto di contorno di coordinate $(x_i, y_i)$ nello spazio geometrico corrisponde una retta nel piano dello spazio dei parametri (figura 2.20):

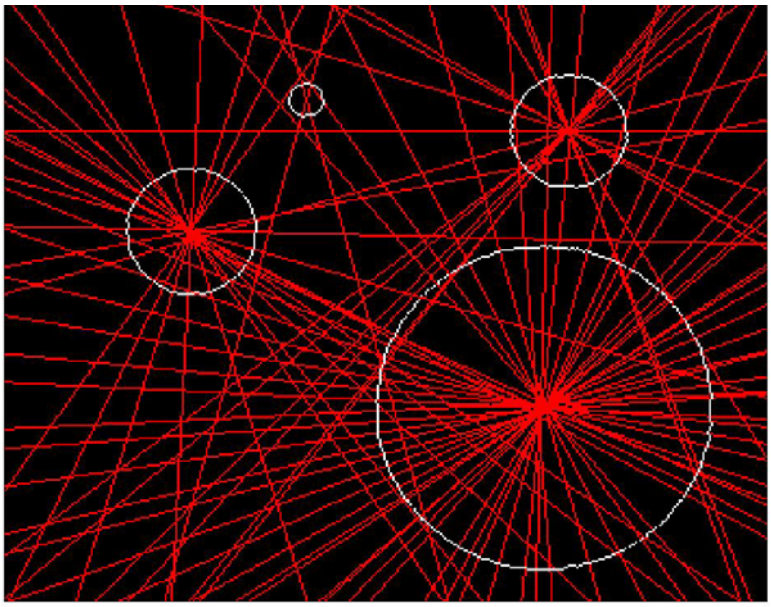


**Figura 2.20**

### 2.3.3.2 Implementazione dell'algoritmo di CHT modificato

Ai fini del tracciamento ho utilizzato l'equazione cartesiana della retta:

$$y - y_i = m \cdot (x - x_i) \tag{2.14}$$

dove $(x_i, y_i)$ sono le coordinate dei punti di contorno ed $m$ è il coefficiente angolare, espresso come vedremo dal rapporto - $\Delta G_x(x, y) / \Delta G_y(x, y)$.

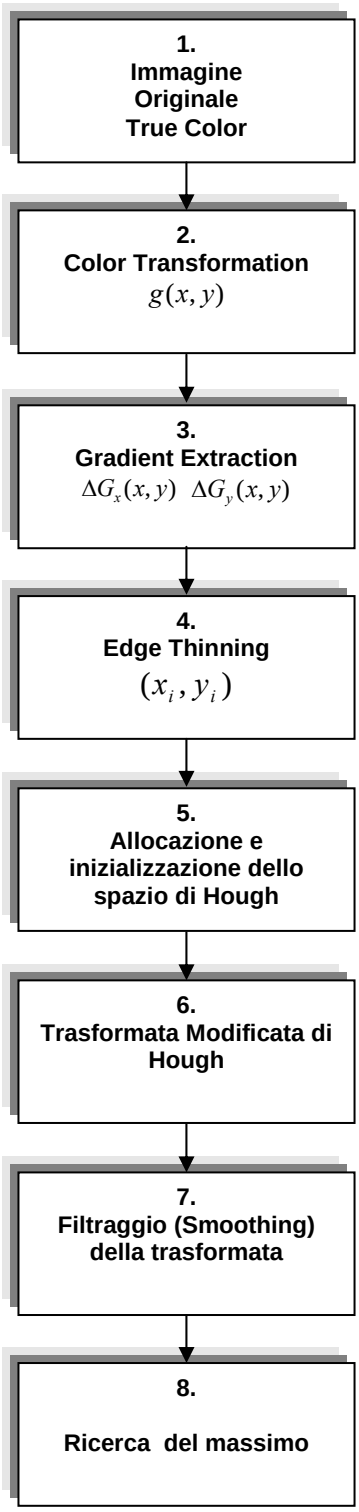
Ecco quindi in breve come funziona l'algoritmo:

1. Trasforma l'immagine di partenza True-color in una monocromatica (**Color Transformation**).
2. A partire dall'immagine monocromatica originale estrae l'immagine **gradiente** $\Delta G(x_i, y_i)$ (che prelude alla fase di *edge-detection* ) secondo uno dei metodi di differenziazione di Sobel o Prewitt già descritti (Prewitt per default) (equazione [2.8]).

3. Richiama quindi la funzione **`edge_detection`**, che restituisce le coordinate $(x_i, y_i)$ dei punti di contorno secondo uno dei due metodi di raffinamento dei bordi di cui ho parlato: in questo caso è impostata per default la modalità "*maxima*".
4. A questo punto sono disponibili in due array distinti le coordinate $(x_i, y_i)$ dei contorni e i corrispondenti valori delle componenti del gradiente lungo *x* ( $\Delta G_x(x, y)$ ) ed *y* ($\Delta G_y(x, y)$ ). A partire da essi, per ciascun punto di contorno $i$, calcola i **parametri direttori** delle rette normali ad esso ($-\Delta G_y, \Delta G_x$ ) .
5. Alloca spazio sufficiente ad accogliere la matrice di accumulatori. Per ogni $i$, traccia (nello spazio bidimensionale di Hough appena allocato) una retta avente i coefficienti direttori appena calcolati, ovvero incrementa di 1 il valore di ciascuna cella accumulatrice (inizialmente posta a zero). Si ottiene cosi la **Trasformata Modificata di Hough**.
6. Applica un filtro **Medio** (par. **2.1.2.2,** eq. [2.2]) alla trasformata di Hough allo scopo di "addolcire" le **forti** discontinuità presenti nella matrice accumulatrice ed "enfatizzare" al contrario i punti di **massima accumulazione**.
7. **Ricerca** il punto di **massimo** nello spazio dei parametri appena filtrato.
8. Calcola le coordinate del centro nell'intorno del punto di massimo restituendo quelle del centro $(xc, yc)$ .

Lo schema a blocchi seguente riassume sinteticamente la seconda versione di Circle detection:

**Circle Detection: seconda versione**



### 2.3.3.3 Risultati dell'algoritmo

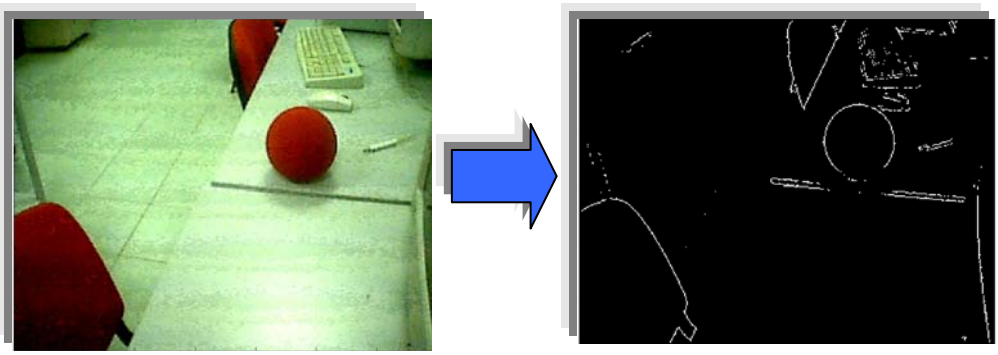
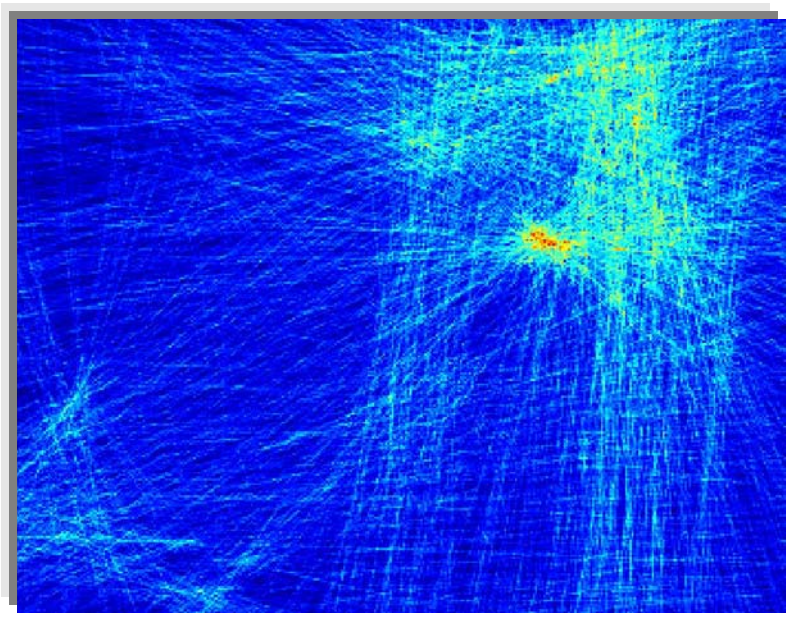
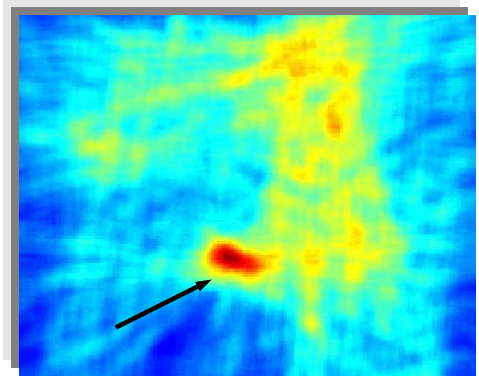


Immagine originale

Localizzazione delle coordinate e delle componenti del gradiente



**Trasformata modificata di Hough**. Colori più caldi rappresentano celle ad accumulazione maggiore, in una scala che va da un minimo di 0 (in blu scuro) al massimo (in rosso scuro) in corrispondenza del picco.



Dettaglio della trasformata modificata di Hough dopo l'applicazione del **filtro medio**. L'immagine risulta evidentemente sfumata dal filtro. Il picco massimo in rosso scuro è indicato dalla freccia.

Una volta individuato il picco massimo, il centro della palla viene marcato dal quadrato in bianco.

**Figura 2.21**

#### 2.3.3.4 La funzione del filtro medio per la massimizzazione

Prima di descrivere le prestazioni di questa versione è opportuno comprendere l'importanza del filtro medio. Osservando l'immagine della trasformata di Hough modificata (al centro della pagina precedente) si evince che il centro della palla non individua un punto di massimo ben **isolato** dal resto dell'immagine ma, al contrario, è caratterizzato da una regione di forte concentrazione di punti con valore prossimo a quello di picco. L'immagine sottostante illustra chiaramente quanto detto mediante un grafo 3D in cui sull'asse ***z*** sono rappresentati i valori della trasformata per ciascuna coppia $(x, y)$.

Se ci limitiamo ad associare il centro del cerchio alle coordinate del punto di massimo globale nello spazio dei parametri, non è detto che questi coincidano. Infatti, applicando l'algoritmo ad immagini acquisite in tempo reale dalle webcam, accadeva, sebbene raramente, che a causa del rumore punti di massimo **spuri** venissero erroneamente scambiati per il centro del cerchio!

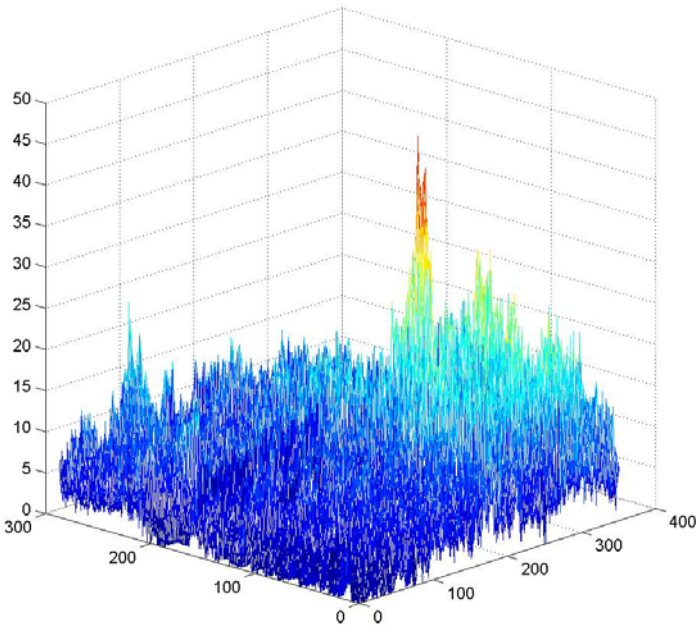



**Figura 2.22**

Detta $f(x, y)$ l'immagine della trasformata di Hough, l'idea è quella di massimizzare non la singola $f(x, y)$, ma la sommatoria de valori che $f(x, y)$ assume all'interno di una maschera quadrata di dimensioni $k \times k$ che viene fatta scorrere su tutta la matrice di accumulatori. Ovvero, si vuole massimizzare la funzione $g(x, y)$:

$$g(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x+i, y+j) \tag{2.15}$$

che è esprimibile anche come:

$$g(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x+i, y+j) \cdot h(i, j) \tag{2.16}$$

se $h(i, j)$ è una matrice di soli 1! Ovvero realizzando una sommatoria di convoluzione con la seguente maschera:

$$h(i,j) = \begin{array}{|c|c|c|c|c|c|c|}\hline 1&1&1&1&1&1&1\\\hline 1&1&1&1&1&1&1\\\hline 1&1&1&1&1&1&1\\\hline 1&1&1&1&1&1&1\\\hline 1&1&1&1&1&1&1\\\hline 1&1&1&1&1&1&1\\\hline 1&1&1&1&1&1&1\\\hline\end{array}$$

dove $(i.j)$ sono le coordinate del pixel centrale e la dimensione della matrice è $7 \times 7$. Dal paragrafo **2.1.2.2** riconosciamo che la 2.10 non è altro che l'espressione del filtro medio.
Facendo eseguire il programma di riconoscimento su immagini acquisite in tempo reale, ho potuto osservare che l'applicazione del filtro, a valle della trasformata modificata di Hough, ha fornito ottimi risultati con alte probabilità di individuazione della palla.

### 2.3.3.5 Osservazioni e prestazioni dell'algoritmo

Anche in questo caso ciascuna cella accumulatrice è in formato `int` a 32 bit.
La matrice di accumulatori stavolta è però bidimensionale e di ampiezza pari a $width \times height$, di valore rispettivamente: 352, 288, ovvero 352x288=101.376 celle di memoria, per un totale di appena 386 Kbyte. L'occupazione di memoria pertanto è esigua rispetto alla prima versione.
Proviamo a stimare la complessità computazionale dell'algoritmo.
Supponiamo di aver estratto $n$ punti di contorno nella fase di *edge-detection* (si osservi che tale valore di $n$ è lo stesso menzionato nella stima della complessità della prima versione e dipende esclusivamente dalla modalità di estrazione dei contorni "*maxima*" o "*fine*").
Per ciascun punto di contorno nello **spazio geometrico**, di coordinate $(x_i, y_i)$, viene tracciata una retta passante per esso nello **spazio dei parametri**. Poiché le dimensioni di tale spazio coincidono con quelle dello spazio geometrico e ogni retta annovera **al più** (cioè nel caso peggiore) un numero di punti pari a *width*, il numero di somme e moltiplicazioni richieste vale $n \cdot width = n \cdot 352$!
A questo punto è evidente il risparmio computazionale rispetto al calcolo della trasformata completa di Hough della prima versione che era pari a: $n \cdot 40 \cdot 360$.
Dal rapporto

$$\frac{n \cdot 40 \cdot 360}{n \cdot 352} \cong 40$$

si evince che questa seconda versione comporta un risparmio della complessità computazionale fino a 40 volte!

Effettivamente, testando l'algoritmo in **assenza del filtro medio** ho verificato un tempo di elaborazione medio di circa $\approx 60ms$ , che è all'incirca 40 volte inferiore a quello della **CHT**!

Lo ***smoothing*** della trasformata incrementa leggermente il tempo di calcolo a causa dell'operazione di convoluzione, che comporta di effettuare $k$ somme e $k$ prodotti (stavolta fra numeri interi) per ogni $x$ ed $y$ , per un totale di $k^2 \cdot width \cdot height$ somme prodotti (complessità quadratica), che può ridursi ulteriormente a $2k \cdot width \cdot height$ ( complessità lineare) con l'osservazione fatta nel paragrafo **2.1.2.2**. Nella pratica l'algoritmo utilizza un valore di k=7.

Il tempo di elaborazione totale, in presenza del filtro medio, sale dunque a $\approx 90ms$ .

## 2.3.4 Terza versione: ricerca locale “Pixel to Pixel”

### 2.3.4.1 I limiti della precedente versione

Prima di passare alla terza ed ultima versione di riconoscimento del cerchio analizziamo innanzitutto i limiti della precedente:

1. Come abbiamo visto, ogni singolo punto di contorno nello spazio geometrico contribuisce **indipendentemente** dai suoi “vicini” al tracciamento di una retta nello spazio dei parametri.
2. Ciascuna retta viene tracciata **per intero** all’interno dello spazio parametrico, dall’inizio al termine dei confini che delimitano tale spazio.
3. Anche tratti o **segmenti** di contorno **rettilinei**, **gruppi isolati di punti** o addirittura **punti angolosi** contribuiscono al tracciamento di rette.
4. I parametri direttori di ciascuna retta sono calcolati a partire dalle componenti direzionali del **gradiente di colore** $\Delta G_x(x,y)$ e $\Delta G_y(x,y)$.

Dall’esame delle caratteristiche appena esposte (1., 2., 3.) emerge che il limite principale della seconda versione è di non tenere **affatto** conto della disposizione spaziale dei singoli punti di contorno, né della loro correlazione spaziale o della loro eventuale predisposizione a far parte di un arco di cerchio: ciò si può riassumere dicendo che ciascun pixel dello spazio geometrico contribuisce **globalmente** allo spazio di Hough a prescindere della locazione e distribuzione dei pixel limitrofi!

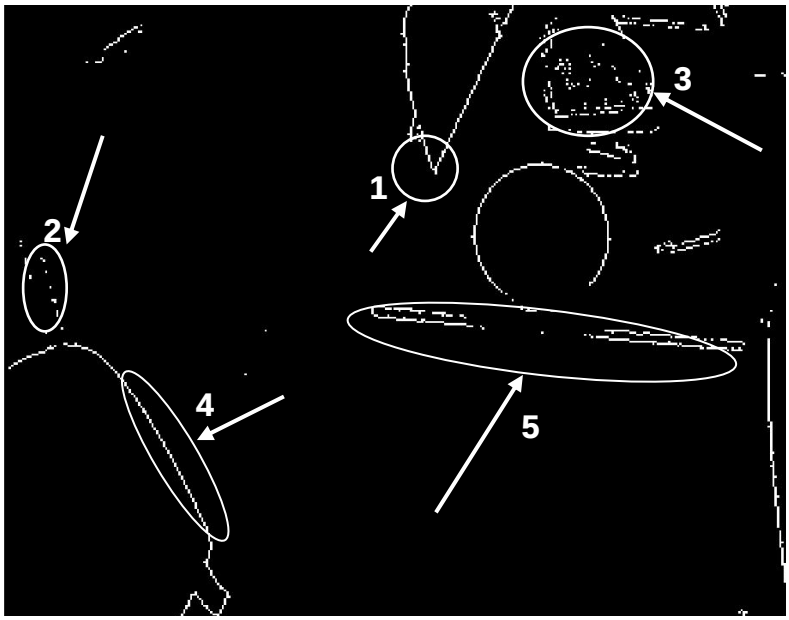



**Figura 2.23** Immagine campione binarizzata, dopo la fase di edge-detection. Sono cerchiate alcune delle caratteristiche spurie che disturbano l’individuazione della palla.

Per comprendere meglio, si osservi la figura 2.23, che riporta i contorni estratti dall'immagine originale dopo la fase di edge-detection: in figura sono cerchiate e indicate dalle frecce le regioni di punti che possono, come si è appena detto, essere causa di ambiguità nella determinazione del cerchio.
Il punto **1**, ad esempio, mostra uno spigolo (o punto angoloso) che proprio per definizione non ha ragione di appartenere ad una circonferenza; in **2** e **3** si individuano invece dei gruppi di punti isolati dal resto dell'immagine, che possono quindi essere interpretati come rumore di fondo; le regioni **4** e **5** identificano invece dei tratti rettilinei di punti piuttosto estesi, che quindi non possono appartenere al perimetro di un cerchio.
Le caratteristiche evidenziate sono solo alcune di quelle informazioni contenute nell'immagine binaria che possono essere ritenute **spurie**, perché non necessarie ad individuare il centro della palla e sono inoltre causa di ambiguità.

Un'altra fonte di disturbo è il modo in cui viene calcolato il coefficiente angolare delle rette a partire dalle derivate parziali lungo *x* ed *y*.
In virtù della [2.8] tali componenti sono date da:

[2.11]

$$\Delta G_x(x,y) = \sum_{k=-1}^{1}\sum_{l=-1}^{1} h_x(k,l)\cdot g(x+k,y+l) = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} g_{x-1,y-1} & g_{x-1,y} & g_{x-1,y+1} \\ g_{x,y-1} & g_{x,y} & g_{x,y+1} \\ g_{x+1,y-1} & g_{x+1,y} & g_{x+1,y+1} \end{bmatrix}$$

$$\Delta G_y(x,y) = \sum_{k=-1}^{1}\sum_{l=-1}^{1} h_y(k,l)\cdot g(x+k,y+l) = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} g_{x-1,y-1} & g_{x-1,y} & g_{x-1,y+1} \\ g_{x,y-1} & g_{x,y} & g_{x,y+1} \\ g_{x+1,y-1} & g_{x+1,y} & g_{x+1,y+1} \end{bmatrix}$$

dove l'operatore ***** indica il prodotto di convoluzione, $h_x$ , $h_y$ sono gli operatori differenziali di Prewitt (par. 2.2.2) mentre $g$ è l'immagine monocromatica, che assume valori tra 0 e 255.
Pertanto i parametri direttori della retta vengono a dipendere esclusivamente dal **valore** dei soli **otto** pixel adiacenti, non dalla **posizione** con cui sono distribuiti intorno al pixel centrale, né dalla disposizione di altri pixel vicini! Otto punti sono quindi insufficienti per poter affermare che questi giacciano su un arco di curva, inoltre, il fatto di calcolare le componenti tramite "gradiente di colore" adombra la possibilità che le variazioni di luminosità lungo il perimetro della palla impediscano di calcolare correttamente la direzione del bordo!
La direzione di un bordo, infatti, non dipende dal valore associato ai singoli pixel ( $g(x,y)$ ) **ma** dalla loro relativa disposizione spaziale.

Per illustrare chiaramente i limiti nel calcolo dei parametri direttori delle rette, riporto di seguito l'immagine binaria a cui ho sovrapposto per maggior chiarezza solo **alcune** delle rette tracciate dall'algoritmo a partire da punti appartenenti al perimetro del cerchio. Questa immagine è stata ottenuta mediante la versione Matlab dell'algoritmo.

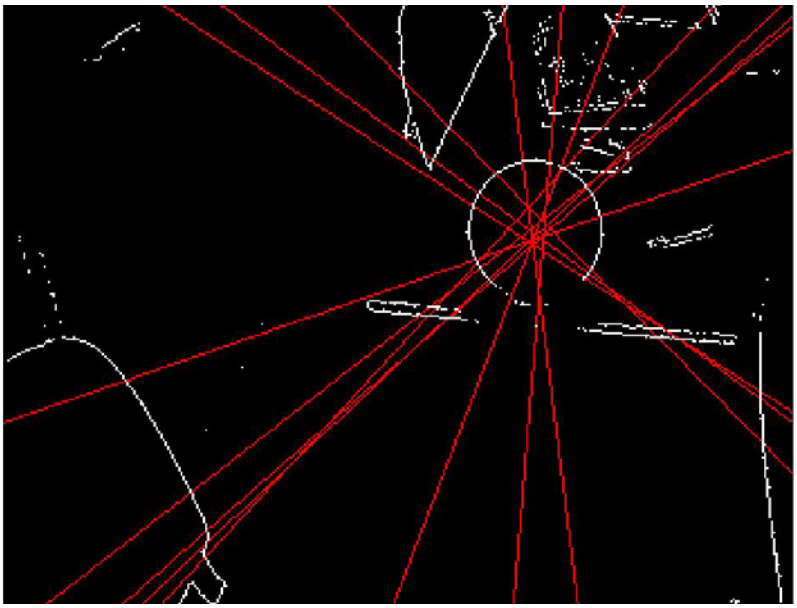


**Figura 2.24** Immagine binaria a cui sono sovrapposte alcune rette tracciate a partire da punti appartenenti al perimetro del cerchio

Si vede chiaramente che non tutte le rette risultano perpendicolari ai punti della circonferenza a partire dai quali sono state tracciate ed è inoltre evidente che non esiste un unico punto di intersezione.
Le sfide di questa terza ed ultima versione dell'algoritmo consistono dunque in:

1. eliminare, o quanto meno attenuare, tutte le possibili fonti di disturbo per la *circle-detection*
2. stimare correttamente la direzione di un bordo
3. tenere conto della disposizione spaziale dei pixel attigui a quello in esame

#### 2.3.4.2 L'algoritmo Pixel to Pixel

Nella figura 2.25 è illustrata l'immagine campione che verrà d'ora in poi utilizzata. Accanto si riporta un ingrandimento di una regione quadrata del perimetro di uno dei cerchi; l'immagine binaria non è altro che una matrice che vale 1 in presenza di un punto di contorno, 0 in assenza di esso.

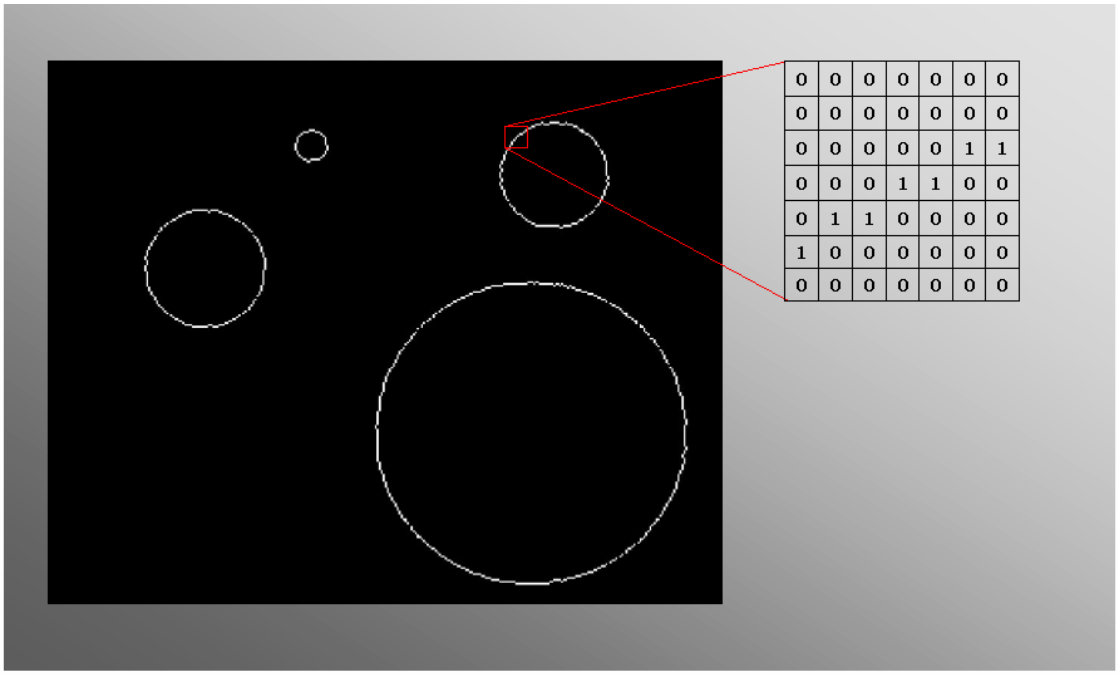
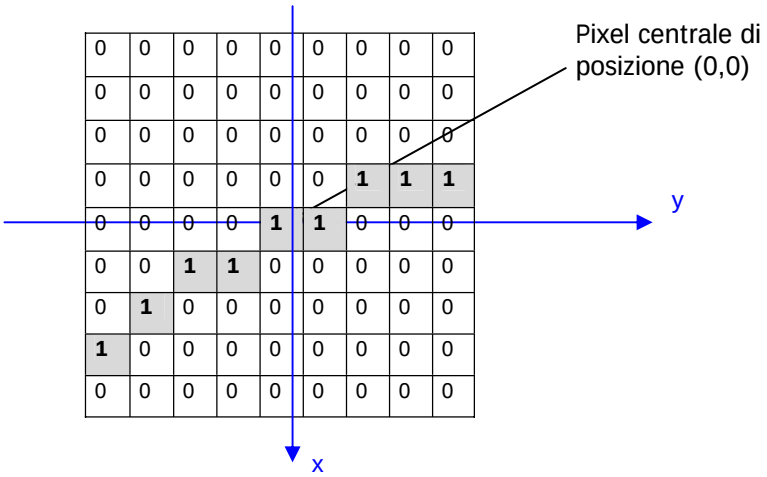



**Figura 2.25**

L'idea che si cela dietro questa nuova versione dell'algoritmo da me ideata consiste nell'esaminare esclusivamente quei gruppi di punti situati all'interno di una regione quadrata di dimensioni $(2k+1)(2k+1)$, con $k$ selezionato dall'utente, tali per cui però il punto in posizione centrale abbia valore 1. Nella tabella che segue si riporta una regione (relativa sempre all'immagine campione) di dimensioni 9x9 che soddisfa i requisiti addotti.




| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | **1** | **1** | **1** |
| 0 | 0 | 0 | 0 | **1** | **1** | 0 | 0 | 0 |
| 0 | 0 | **1** | **1** | 0 | 0 | 0 | 0 | 0 |
| 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Figura 2.26**

A questo punto, prendendo come "bordo" di riferimento la traccia di "1" in figura, occorre stimare la direzione di tale bordo, ovvero quella della retta tangente al pixel centrale.

Se gli "1" appartengono ad una circonferenza, secondo un principio di "**Maximum Likelihood"** la miglior stima della direzione tangente al punto centrale coincide con quella della corda di lunghezza massima, indicata in nero in figura 2.27.

Pertanto il problema della stima della direzione si risolve nella ricerca della corda congiungente i pixel più distanti fra loro.

Assumendo convenzionalmente che il pixel centrale giaccia in posizione (0,0) l'algoritmo effettua una ricerca all'interno della regione quadrata in esame **calcolando la distanza** di ciascun "1" dall'origine e scegliendo come estremi della corda quelle coppie che avendo distanza maggiore dall'origine abbiano però almeno una delle **coordinate di segno opposto**. Quest'ultima condizione serve a scartare i punti spuri che pur trovandosi a distanza maggiore sono situati dalla stessa parte dell'uno o dell'altro estremo (spigoli o punti angolosi).

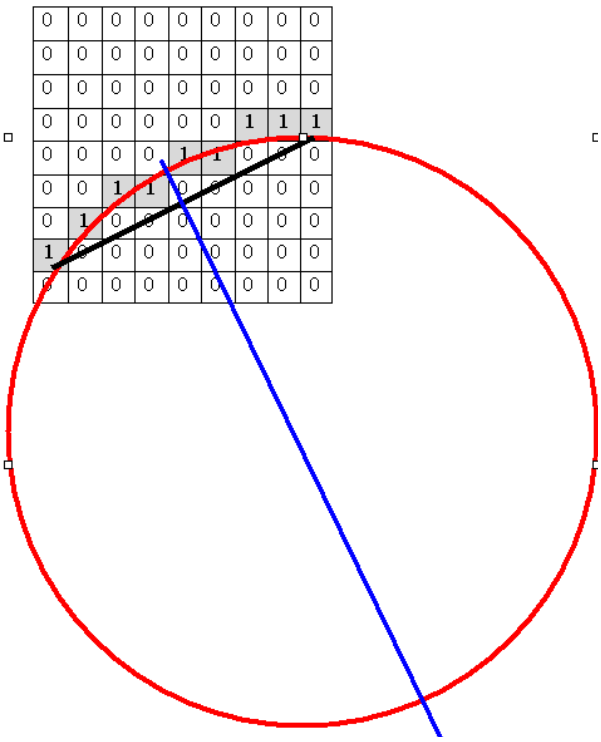


**Figura 2.27**

Poiché in generale il calcolo della distanza dall'origine richiederebbe almeno di calcolare la funzione quadratica $x^2 + y^2$, ho definito una nuova funzione distanza $d(x, y)$, in grado di fornire lo stesso risultato:

$$d(x, y) = |x| + |y| \tag{2.17}$$

che non richiede alcuna moltiplicazione ma solo gli operatori di somma e valore assoluto, ottenendo così un notevole risparmio computazionale!

Una volta ottenute le coordinate degli estremi $A \equiv (x_A, y_A)$ e $B \equiv (x_B, y_B)$, i parametri direttori della retta normale e il coefficiente angolare saranno:

$$a = y_B - y_A,\ b = x_A - x_B,\ m = \frac{b}{a} \tag{2.18}$$

Tale retta dovrà passare per il punto medio $C$ della corda, di coordinate:

$$cnci = \frac{x_A + x_B}{2},\ cncj = \frac{y_A + y_B}{2} \tag{2.19}$$

**N.B.** Si osservi inoltre che il punto medio della corda individua la direzione della concavità dell'arco, rendendo così possibile tracciare il segmento di retta soltanto dalla parte in cui è situato il centro del cerchio con un discreto risparmio computazionale! Tuttavia per valori di ***k*** piccoli ($k \leq 3$) risulta difficile sfruttare l'informazione sulla concavità perché, date le esigue dimensioni della regione in esame, la concavità può indurre a tracciare il segmento di retta dalla parte opposta a quella desiderata!

Fino ad ora abbiamo ampiamente soddisfatto le richieste 2. e 3. avanzate al termine del precedente paragrafo; a questo punto non rimane che scartare quei punti che appartengono a gruppi isolati di pixel o sono parte di un segmento rettilineo (come i solchi tra le mattonelle del pavimento).

Per eludere quelle concentrazioni di pixel ma anche punti isolati che possono essere causa di ambiguità è sufficiente contare il numero N di "1" situati all'interno della regione quadrata in esame. Se N=2k+1 allora l'algoritmo procede nella ricerca degli estremi della corda, altrimenti li esclude del tutto!

Il motivo per cui ho scelto di confrontare N con la dimensione della maschera di ricerca è dovuto al fatto che se i punti che sto esaminando appartengono ad un arco di cerchio, allora con ottima probabilità questi saranno esattamente 2k+1 perché lo spessore dei contorni è al massimo 1 pixel! Nell'esempio che ho riportato nella pagina precedente e che deriva dall'immagine sottoposta a test, si può verificare che N vale proprio 9, che è la dimensione della regione quadrata.

**N.B.** Si osservi che questa condizione permette di eludere vantaggiosamente anche le code finali di un arco di cerchio, le quali non forniscono punti sufficienti a dedurne la direzione (cerchiati e indicati dalla freccia in figura 2.28) ma darebbero solo un'informazione errata.

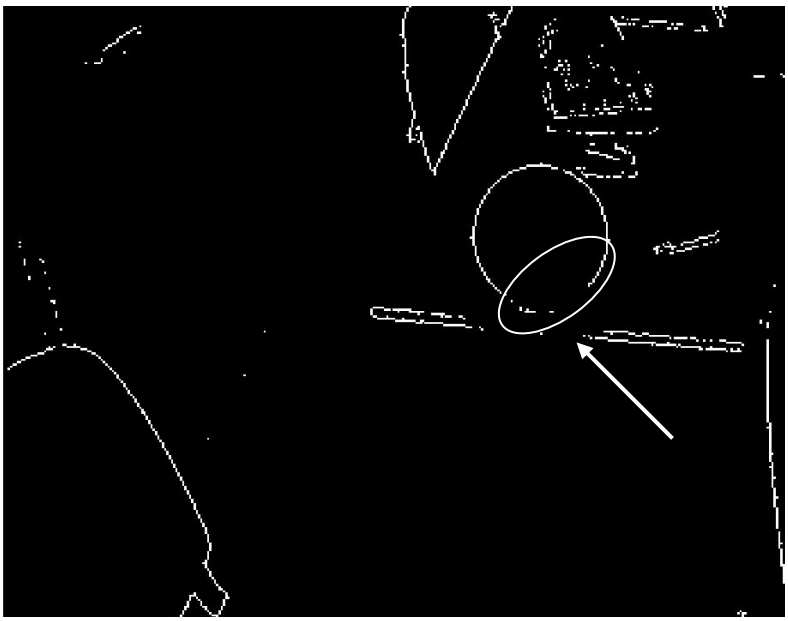


**Figura 2.28**

Per escludere porzioni di punti appartenenti a segmenti rettilinei è invece sufficiente osservare che, in tal caso, il punto medio $C$ della corda giacerà in prossimità del pixel centrale, come evidente dalla figura 2.29.
Ciò si può tradurre imponendo che **cnci** e **cncj** siano ad una distanza dal centro maggiore di 0.5. Per cui sarà sufficiente verificare che la seguente espressione condizionale in linguaggio C restituisca un valore **true**.

```
!(fabs(cnci)<=0.5 && fabs(cncj)<=0.5)
```
(2.20)

Ad esempio, la tabella seguente riporta un insieme di punti siti in corrispondenza di un tratto rettilineo.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | **1** | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | **1** | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | **1** | 0 | 0 | 0 | 0 | 0 | 0 |

**Figura 2.29**

In questo caso la funzione **bord**() fornirebbe **cnci=0, cncj=0**, che non verifica la condizione suddetta. Tali punti verrebbero pertanto esclusi.
**N.B.** Tuttavia per valori di ***k*** piccoli ($k \leq 3$) la 2.14 fornisce un valore **false** anche in presenza di un arco di cerchio; questo perché, date le esigue dimensioni della regione in esame, è facilmente confrontabile con un segmento rettilineo. Pertanto per $k \leq 3$ l'algoritmo salta la verifica della condizione 2.14.

Quanto detto finora è espletato dalla funzione **bord()**, il cui prototipo è:

```
bool bord(int grad[2], float *cnci, float *cncj, int *ind, int k)
```

la quale riceve in ingresso la matrice **ind** di dimensioni (2**k**+1)(2**k**+1) contenente i punti di contorno da esaminare, e restituisce i parametri direttori **grad[0]** , **grad[1]** e le coordinate del centro della corda **cnci**, **cncj**.

**N.B.** È importante notare che tale funzione restituisce un valore **`true`** nei soli casi in cui i pixel esaminati verificano i requisiti detti sino ad ora, ovvero se e solo se sono vere le seguenti espressioni:

- **`N==2k+1`** (2.21)
- **`!(abs(cnci)<=0.5 && abs(cncj)<=0.5)== true`** (N.B. se $k > 3$)

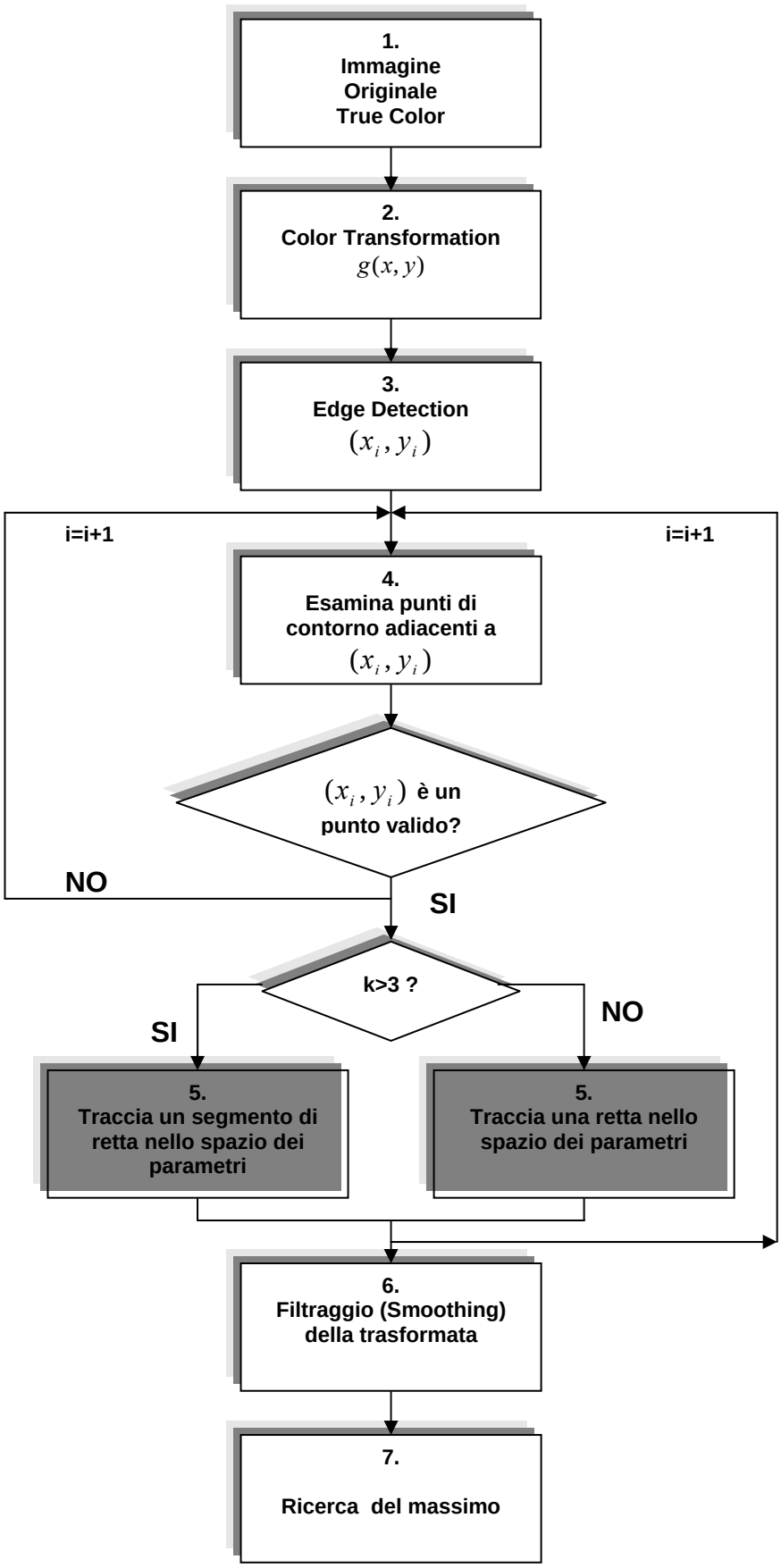
#### 2.3.4.3 Le fasi dell'algoritmo Pixel to Pixel

Dopo aver compreso le modalità con cui viene stimata la direzione dei contorni, veniamo alle fasi principali dell'ultima versione dell'algoritmo di circle-detection:

1. Trasforma l'immagine di partenza True-color in una monocromatica (**Color Transformation**).
2. A partire dall'immagine monocromatica richiama la funzione `edge_detection`, che restituisce le coordinate $(x_i, y_i)$ dei punti di contorno secondo uno dei due metodi di raffinamento dei bordi di cui ho parlato: in questo caso è impostata per default la modalità "*maxima*".
3. Viene allocato uno spazio di memoria sufficiente ad accogliere la matrice accumulatrice bidimensionale che costituisce lo spazio dei parametri.
4. Per ogni $i$, richiama la funzione **`bord()`** ottenendo:
    - I parametri direttori della retta da tracciare
    - Il centro della corda da cui tracciare la retta
    - La concavità dell'arco all'interno del quale tracciare il segmento di retta
5. Partendo dai parametri **`cnci`** e **`cncj`** valuta la condizione 2.15 decidendo se considerare (**`true`**) o scartare (**`false`**) l'informazione
6. Se **`bord()`** ha restituito **`true`** traccia nello spazio parametrico una retta ($k \leq 3$) o un segmento di retta nel verso della concavità dell'arco ($k > 3$), avente i coefficienti appena calcolati, ovvero incrementa di 1 il valore di ciascuna cella accumulatrice (inizialmente posta a zero).
7. Applica un filtro **Medio** (par. **2.1.2.2,** eq. [2.2]) all'immagine ottenuta (per i motivi di cui al par. 2.3.3.3).
8. **Ricerca** il punto di **massimo** nello spazio dei parametri appena filtrato.
9. Calcola le coordinate del centro nell'intorno del punto di massimo restituendo quelle del centro $(xc, yc)$.

Lo schema a blocchi nella pagina seguente riassume sinteticamente la terza ed ultima versione di Circle detection:

**Circle Detection: terza versione**



## 2.4 Risultati sperimentali e conclusioni

Dopo aver accuratamente descritto le fasi dell'algoritmo Pixel-to-Pixel ed analizzato la sua predisposizione a dirimere le principali cause di disturbo, non rimane che verificarne l'effettiva funzionalità.

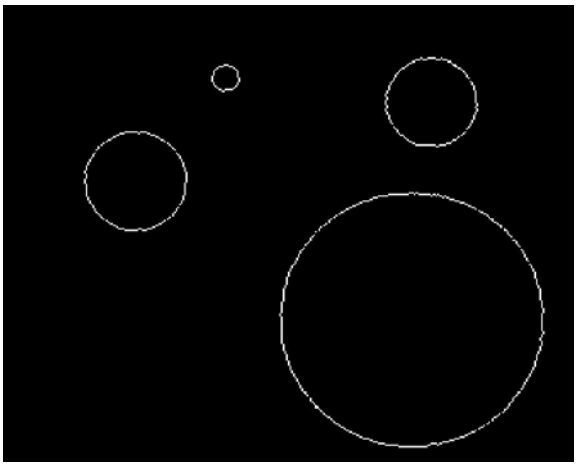
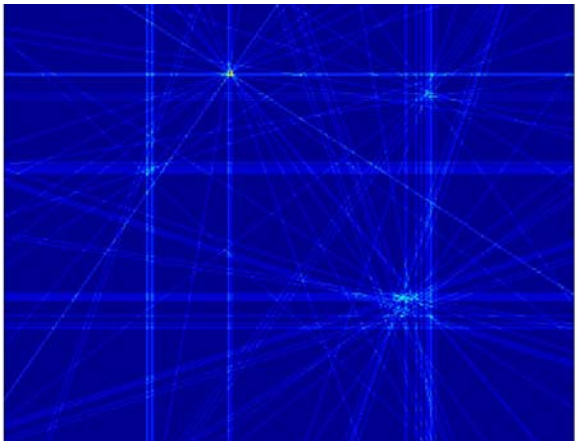


**Figura 2.30** Immagine campione impiegata per testare la funzionalità dell'algoritmo Pixel to Pixel

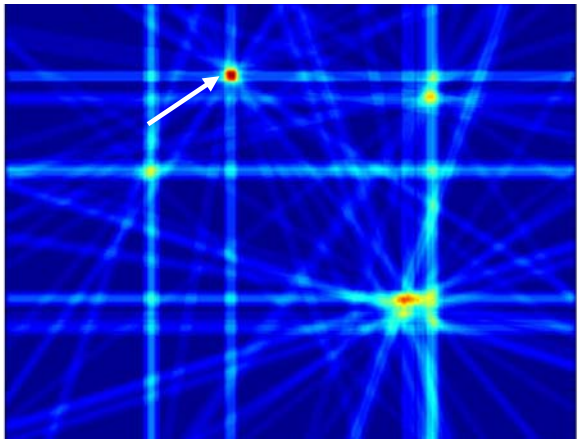
Nella pagina successiva si riportano le rappresentazioni grafiche dello spazio parametrico (o spazio delle rette) e le corrispettive dopo l'applicazione del filtro medio (di dimensioni sempre 7x7).
Le frecce bianche indicano la posizione corrente del picco massimo che individua il centro di uno dei cerchi dell'immagine originaria qui sopra.

L'algoritmo è stato testato per diversi valori di k crescenti: k=3, k=5, k=8, k=15.
Come si può osservare , variando le dimensioni della finestra di ricerca $(2k+1)(2k+1)$ e quindi il valore di k, cambia la definizione dello spazio delle rette: valori di k piccoli consentono di individuare meglio cerchi più piccoli; valori di k grandi consentono di identificare circonferenze di raggio maggiore. Infatti, come abbiamo visto, la dimensione della finestra determina l'incertezza nella stima della direzione di un bordo. Più la finestra è grande, minore è l'incertezza, e quindi un maggior numero di rette si intersecherà nel centro; al contrario, se la finestra è piccola, si avrà maggior dispersione delle rette in prossimità del centro!
Ciò si può formalizzare affermando che occorre scegliere k di valore confrontabile con il raggio del cerchio che vogliamo isolare.

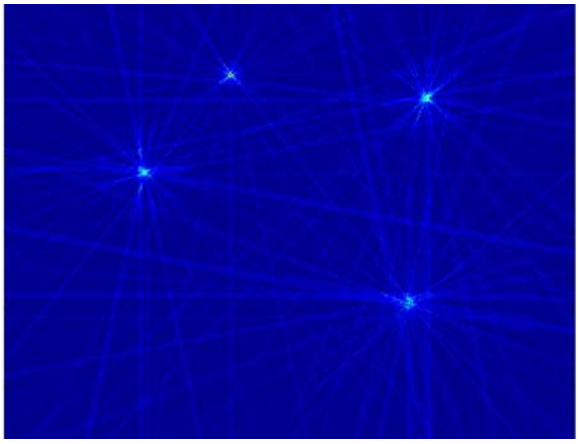
Infatti aumentando k da 3 a 15 come nelle immagini che seguono si osserva che il picco massimo (indicato dalla freccia) si sposta dalla circonferenza più piccola (k=3) via via verso quella più grande (k=15).



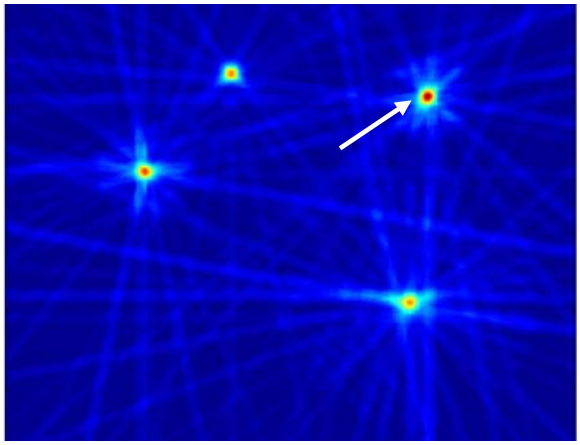
**K=3**. Le rette sono tracciate per **intero**



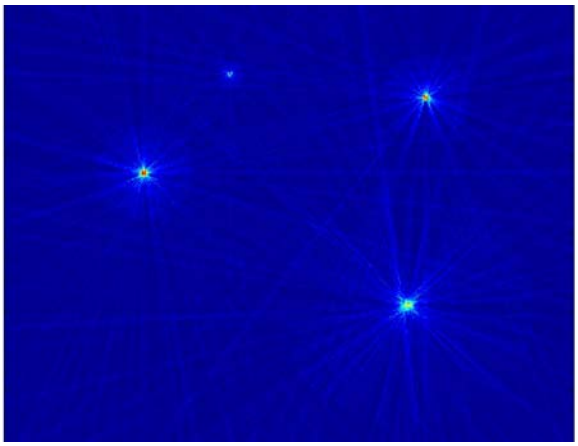
**K=3**. Le rette dopo l'applicazione del filtro medio



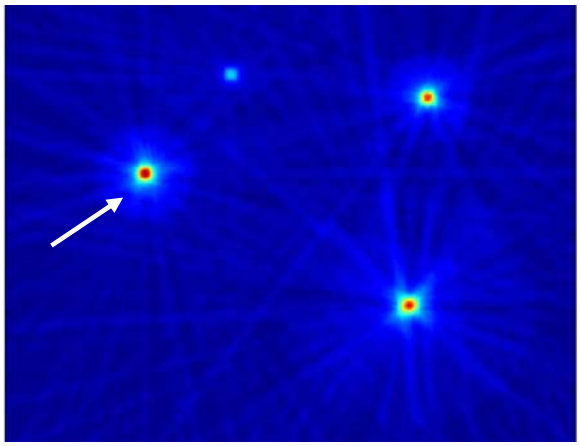
**K=5**. Sfruttata concavità. Sono tracciati **segmenti di rette**



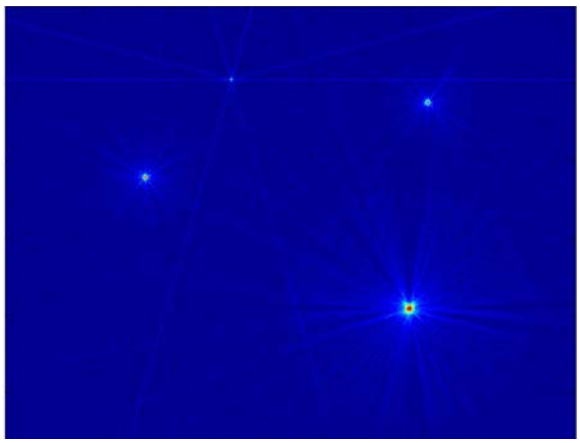
**K=5**. Le rette dopo l'applicazione del filtro medio



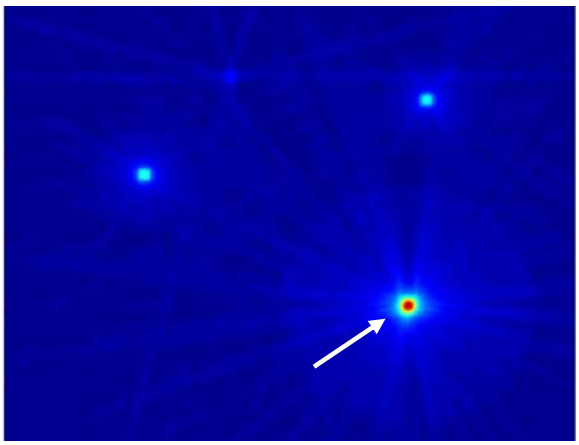
**K=8**. Sfruttata concavità. Sono tracciati **segmenti di rette**



**K=8**. Le rette dopo l'applicazione del filtro medio



**K=15**. Sfruttata concavità. Sono tracciati **segmenti di rette**



**K=15**. Le rette dopo l'applicazione del filtro medio

Quanto osservato tuttavia è valido se ci troviamo in presenza di più cerchi, e quindi occorrerà tarare k al fine di isolare solamente quelli che ci interessano; ma all'atto pratico anche per piccoli valori di k si manifesta comunque un'elevata concentrazione di punti in prossimità del centro del cerchio più grande! Ciò può essere messo in luce dai grafici tridimensionali seguenti, che illustrano lo spazio delle rette in 3D e i valori dei rispettivi picchi massimi in corrispondenza dei centri: anche per k=3 si ottengono valori elevati nelle celle accumulatrici in corrispondenza dei centri.

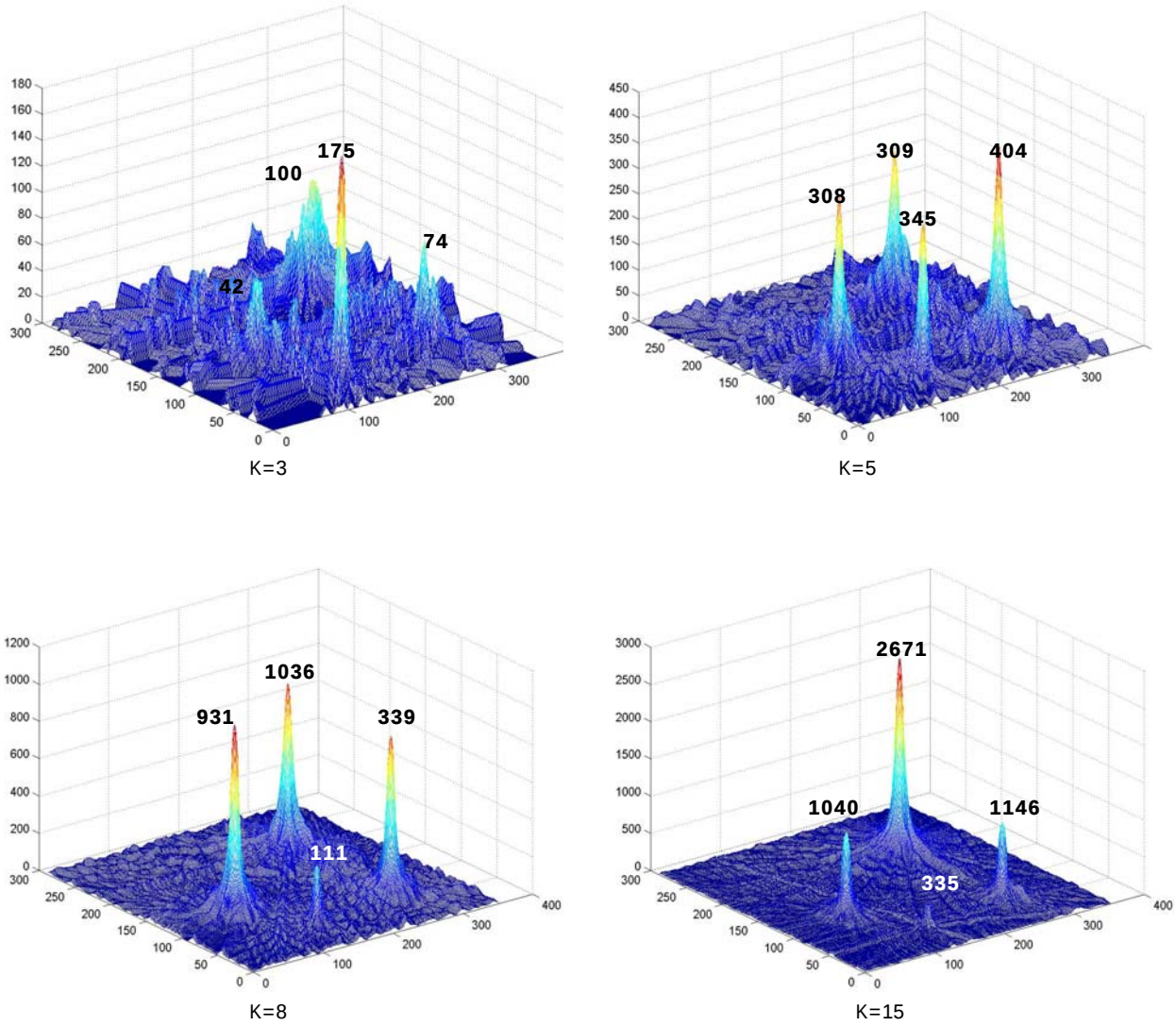



**Figura 2.31**

A questo punto testiamo l'algoritmo su una scena del mondo reale e verifichiamone ancora la funzionalità. L'immagine campione è quella recensita nelle pagine precedenti e che riporto qui per chiarezza.

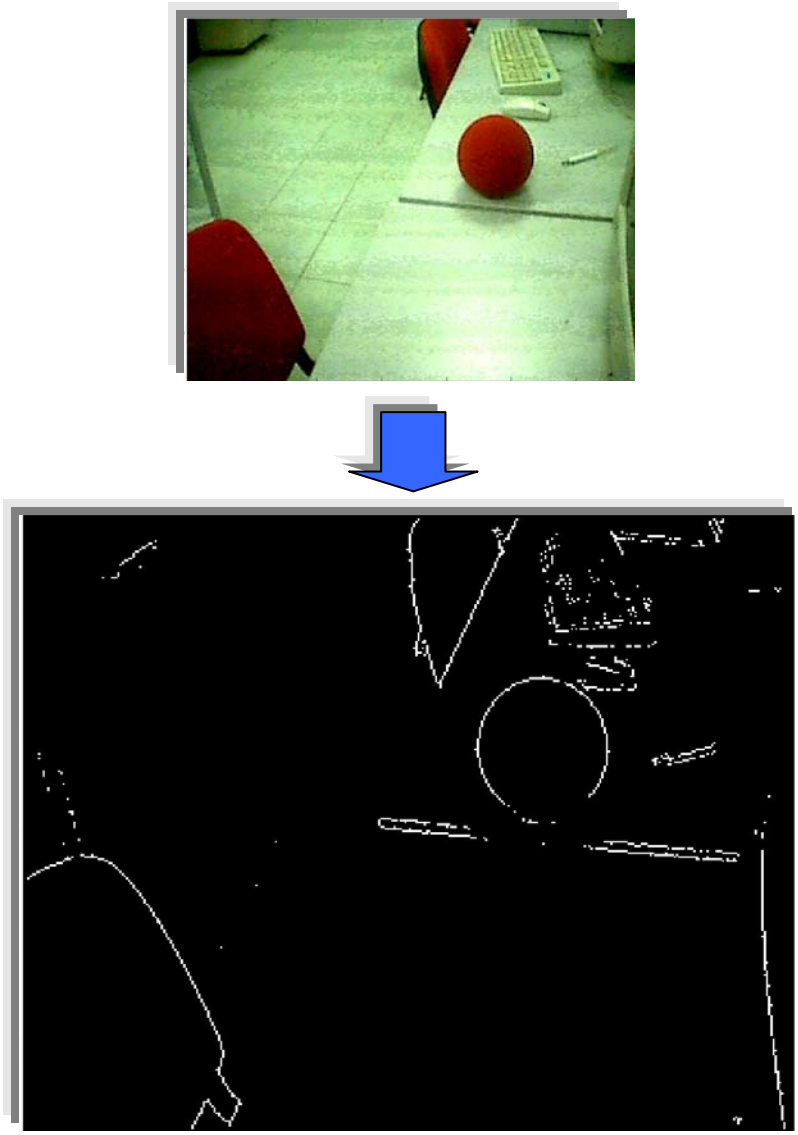


**Figura 2.32**

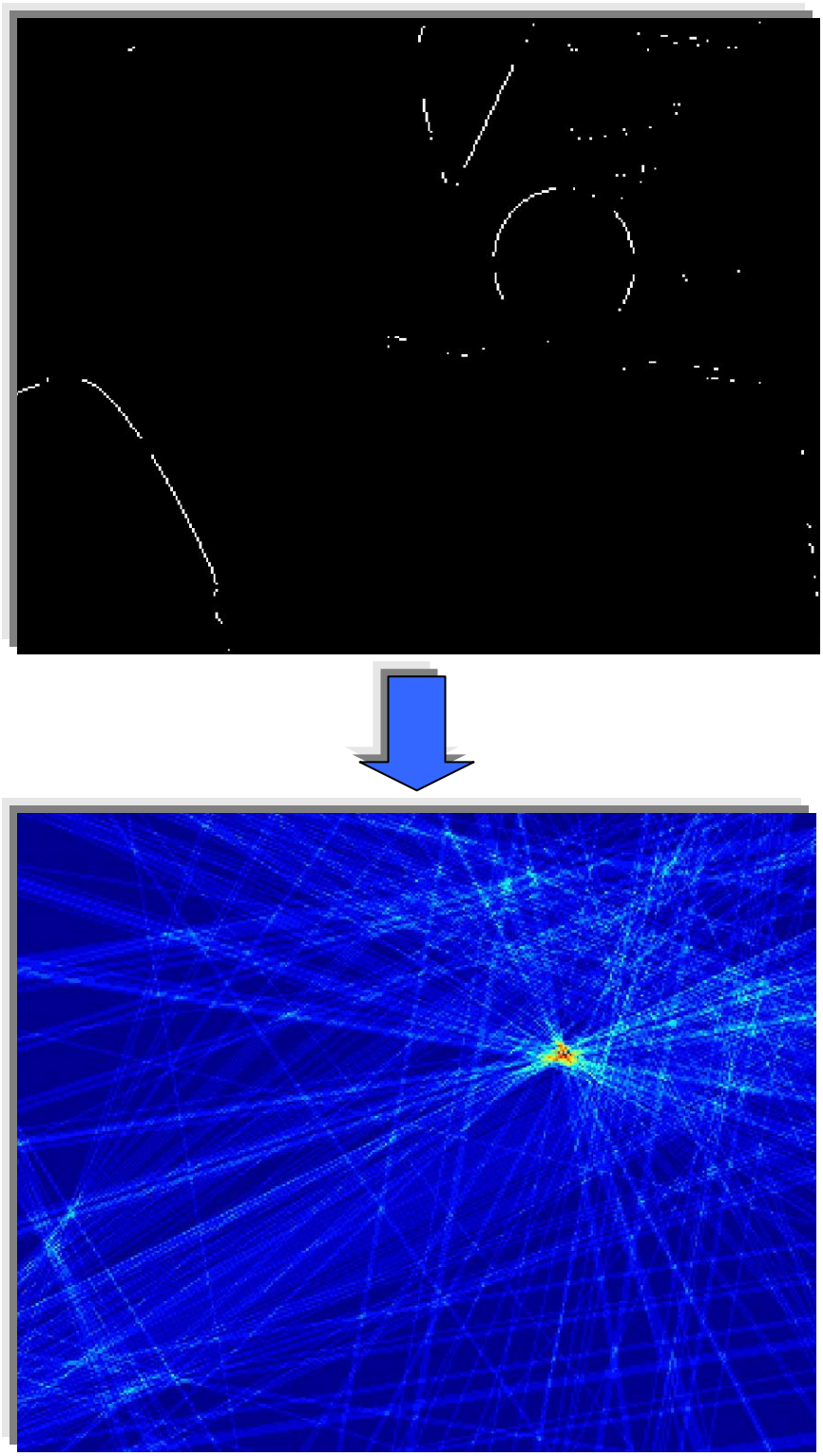
L'immagine binaria qui sopra estratta dall'originale dopo la fase di edge-detection contiene esattamente 1332 punti di contorno!

Nella pagina che segue viene testato l'algoritmo Pixel to Pixel per k=3, k=5, k=10 e si riporta, per ciascun k, l'immagine binaria che rappresenta i soli punti di contorno effettivamente utilizzati per la circle detection che verificano la 2.15**:**

- **`N==2k+1`**
- **`!(abs(cnci)<=0.5 && abs(cncj)<=0.5)== true`** (N.B. se $k > 3$) (2.22)

**k=3**

Per tale valore di k, la 2.15 richiede solo di verificare la condizione N=2k+1, ovvero N=7. I soli punti di contorno che verificano questa condizione e che contribuiscono al tracciamento di rette nello spazio dei parametri sono rappresentati nella figura 2.33 qui sotto e ammontano a **357**, in numero molto minore ai 1332 che avrebbero dovuto essere considerati: tuttavia si osservi che, dato l'esiguo valore di k, permangono i contorni rettilinei **ma** sono stati comunque rimossi i punti angolosi e i gruppi di pixel isolati che erano effetto del rumore di fondo.



**k=10**

Tale valore di k richiede la verifica di entrambe le condizioni 2.15. I punti di contorno che verificano questa condizione sono rappresentati nella figura 2.34 e ammontano a **68**, in numero ancora minore ai 357 considerati nel caso k=3! Dato l'elevato valore di k (cui corrisponde una finestra di ricerca di 21x21 pixel!) stavolta anche i contorni rettilinei sono stati del tutto rimossi e non contribuiscono più al tracciamento di rette nello spazio dei parametri (in basso).

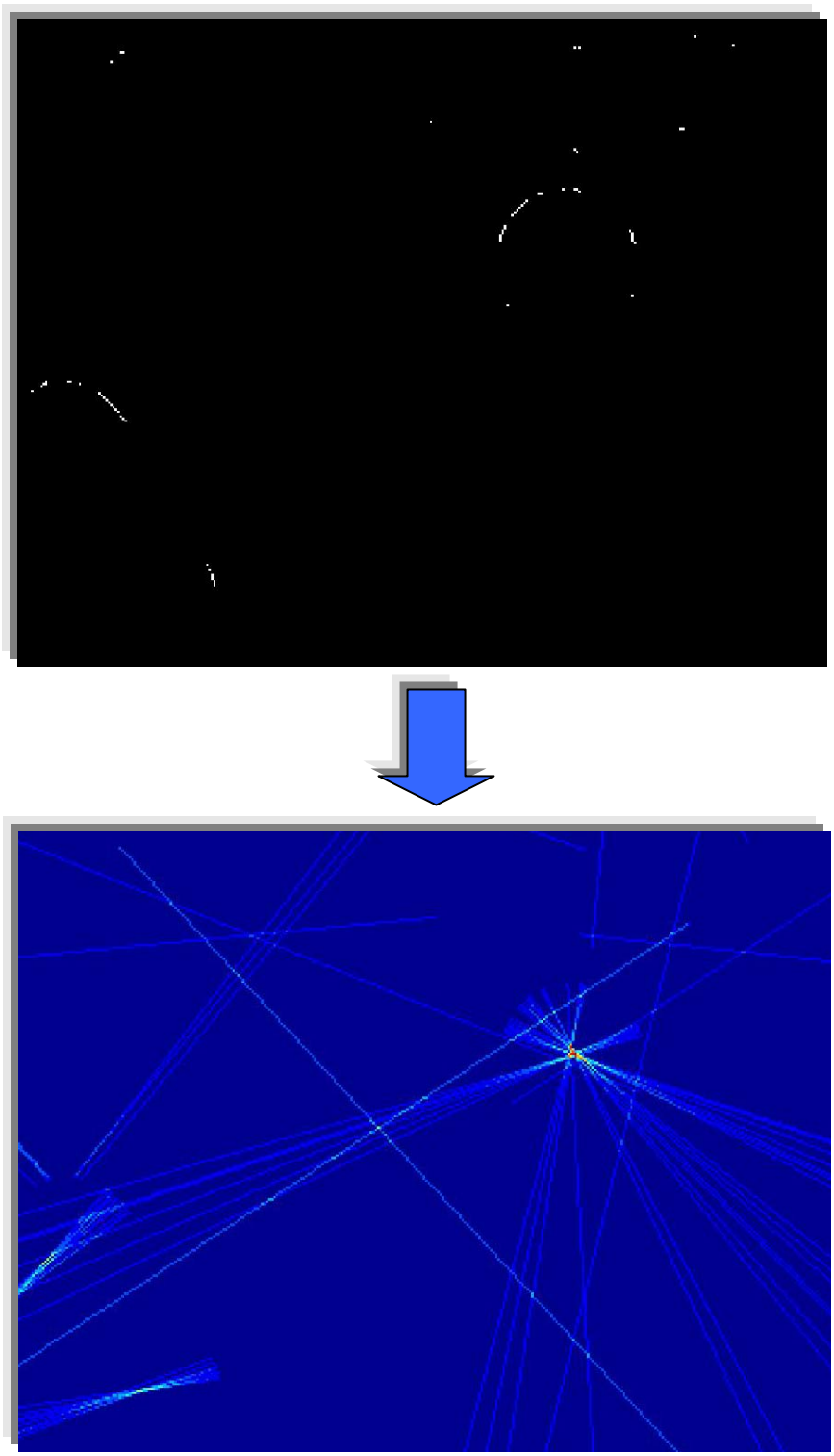


**Figura 2.34**

Confrontando i risultati dell'algoritmo Pixel-to-Pixel con quelli delle altre versioni di Circle detection si ottengono delle prestazioni migliori in quest'ultima versione in termini di:

1. efficienza computazionale
2. resistenza alle occlusioni
3. capacità di individuazione dell'oggetto

Le prestazioni computazionali sono evidenziati dal seguente specchietto, che riassume i tempi di elaborazione di ciascun algoritmo.

| **Trasformata di Hough completa** | **Trasformata di Hough modificata** | **Algoritmo Pixel-to-Pixel** |
|---|---|---|
| 2,5 s | 45 ms | 25 ms |

## 2.5 Guida alle funzioni implementate in C

### Filtraggio antirumore

Le funzioni seguenti consentono di effettuare il filtraggio delle immagini originarie :

```
void smooth(int *dst,int  *src, char *mode)
void smoothRGB(int *dst,int  *src, char *mode)
```

dove:
**`dst`** è il puntatore alla immagine finale filtrata
**`src`** è il puntatore all'immagini originale da filtrare
mode consente di selezionare il tipo di filtro:

- mode = "**`average`**" seleziona il filtro medio
- mode = "**`gaussian`**" seleziona il filtro gaussiano

Le due funzioni `smooth` e `smoothRGB` funzionano allo stesso modo; l'unica differenza di cui tenere conto sta nel fatto che la prima funziona su immagini che hanno già subito il passaggio alla rappresentazione monocromatica e restituisce un'immagine filtrata ancora monocromatica;

la seconda effettua il filtraggio su ciascuna componente cromatica R, G e B e restituisce un'immagine filtrata ancora a colori. Il fatto di intervenire su un'immagine a colori, abbiamo visto, contravviene ai canoni di efficienza suddetti, tuttavia ho voluto introdurla per monitorare la correttezza dello *smoothing* e per confrontare le prestazioni con quella monocromatica.

## Circle detection

```
int *houghcircle(int *image);
```

è la funzione che calcola la trasformata di Hough completa e restituisce l'indice delle coordinate del centro e il raggio del cerchio a partire dall'immagine a colori originale, puntata da `image`.

```
void find_circle1(float xl_circle[2], int *data, int fine)
```

è la funzione che effettua la trasformata di Hough modificata.
**`xl_circle`** è un array che contiene le coordinate del cerchio trovato.
**`data`** è il puntatore all'immagine originale.
**`fine`** consente di impostare la modalità di estrazione dei contorni ("maxima" o "fine").

```
void find_circle4(float xl_circle[2], int *data, int fine, int kk);
```

è la funzione che applica l'algoritmo Pixel-to-Pixel.
**`kk`** è la dimensione della finestra di analisi ( dimensioni (2kk+1)(2kk+1) ).
All'interno della stessa funzione viene richiamata la seguente, che calcola la direzione di un contorno.

```
void bord(int grad[2], float *cnci, float *cncj, int *ind, int kk);
```

dove:

**`grad`** contiene le componenti che individuano la direzione del bordo.
**`cnci, cncj`** cono rispettivamente le coordinate del baricentro del contorno, che individuano anche la concavità.

# Capitolo 3
# Visione stereoscopica


## Abstract

In questo capitolo verrà descritta la realizzazione del software (scritto in linguaggio C) necessario all'impiego dell'apparato di visione come strumento per la stima della posizione 3D di un oggetto partendo da una coppia di immagini stereoscopiche.


## 3.0 Introduzione

Si definisce **Stereopsi** la localizzazione relativa degli oggetti visivi in profondità. Il nostro cervello può fondere (percepire in un'immagine unica) due immagini retiniche che entro certi limiti si formano su punti della retina lievemente disparati, come ad esempio quando, a causa della distanza fra gli assi visivi, i due occhi osservano lo stesso oggetto sotto due angolazioni diverse. È proprio a causa di questa disparità che avviene la ***Visione Stereoscopica***, facendoci percepire l'oggetto come "unico" e "solido".

La stereopsi computazionale è invece il processo che consente di ottenere l'informazione di profondità da una coppia di immagini provenienti da due telecamere che inquadrano una scena da differenti posizioni.

Esso costituisce una branca articolata e complessa della visione computazionale all'interno della quale si possono individuare in generale due sottoproblemi:

1. il calcolo delle corrispondenze
2. la ricostruzione della struttura 3D

Il primo consiste nell'accoppiamento tra punti nelle due immagini che sono proiezione dello stesso punto della scena: detti punti sono chiamati **punti coniugati o corrispondenti**. Il calcolo dell'accoppiamento è possibile sfruttando il fatto che le due immagini differiscono solo lievemente, cosicchè un particolare della scena appare "simile" nelle due immagini. Il concetto di "similarità" si traduce nella minimizzazione di una funzione di costo (tipo correlazione) che si applica a tutte le righe e colonne dell'altra immagine. Basandosi solo su questo vincolo, però, sono possibili molti falsi accoppiamenti ed è inoltre evidente che la ricerca del punto coniugato corrispondente risulta computazionalmente onerosa.

In realtà vedremo che sarà necessario introdurre un altro vincolo che renda il calcolo delle corrispondenze trattabile: il **vincolo epipolare**, il quale afferma che il corrispondente di un punto in una immagine può trovarsi solo su una retta (retta epipolare) nell'altra immagine. Grazie a questo processo, detto **rettificazione epipolare**, la ricerca delle corrispondenze

diventa unidimensionale invece che bidimensionale, diminuendo così la complessità computazionale.

Una volta noti gli accoppiamenti tra i punti delle due immagini e nota la posizione reciproca delle telecamere ed i parametri interni di ciascun sensore , è possibile ricostruire la posizione nella scena dei punti che sono proiettati sulle due immagini, detta **ricostruzione della struttura 3D**.

Questo processo di ricostruzione necessita però della **calibrazione** dell'apparato stereo, ovvero del calcolo dei parametri interni (**parametri intrinseci**) e della posizione reciproca (**parametri estrinseci**) delle telecamere.

Ciò detto, il seguente capitolo è così organizzato:

dapprima verrà introdotto il modello matematico della telecamera, che descrive la proiezione dei punti dello spazio 3D sul piano dell'immagine: questo modello è chiamato modello pin-hole. In questa sede verrà anche illustrata l'equazione completa che modella il fenomeno di aberrazione causato dalla lente, e che introduce una forte non linearità di distorsione dell'immagine, e vedremo che essa può essere ricondotta a due cause: la distorsione radiale e quella tangenziale. Per poter estendere la validità del modello di distorsione anche ad altri apparati di visione, ne verrà adottato uno descritto da cinque coefficienti, capace di adeguarsi alla quasi totalità dei sistemi di lenti in commercio.

Proseguendo, verrà presa in esame la calibrazione delle telecamere come metodo per la misura indiretta dei parametri intrinseci ed estrinseci che descrivono il modello pin-hole dei sensori.

Una volta noti tali parametri passeremo a descrivere l'algoritmo di **rettificazione epipolare** che corregge il fenomeno di distorsione e trasforma la coppia di immagini acquisita dalle telecamere in modo che le rette epipolari suddette diventino parallele e orizzontali in ciascuna immagine. Il caso in cui le linee epipolari sono parallele e orizzontali costituisce infatti un caso molto favorevole per il calcolo delle corrispondenze: se i punti coniugati giacciono sulla stessa riga, la ricerca del corrispondente di un punto nell'altra immagine potrà limitarsi ai soli punti appartenenti alla stessa riga di pixel!

Scopriremo inoltre che la rettificazione realizzata via software è in grado di compensare tutti gi errori e le imperfezioni coinvolte nella costruzione fisica e meccanica dei sensori e della struttura portante, e prescinde anche dall'uso di telecamere uguali o diverse per modello o risoluzione.

Dopo aver rettificato le immagini passeremo alla ricostruzione della struttura 3D e verrà descritto l'algoritmo di **stereo-triangolazione**, che ricostruisce la posizione reale nella scena dei punti che sono proiettati sulle due immagini e ne restituisce la coordinate cartesiane tridimensionali.

Questo lavoro di tesi è stato svolto anche in collaborazione con il centro ricerche dell'E.N.E.A., presso il Dipartimento di robotica, sede di Casaccia (Roma).

Il software di stereometria che ho sviluppato è stato infatti testato su un hardware stereoscopico professionale in dotazione al centro ricerche. I risultati sperimentali verranno illustrati nell'ultimo paragrafo 3.7.

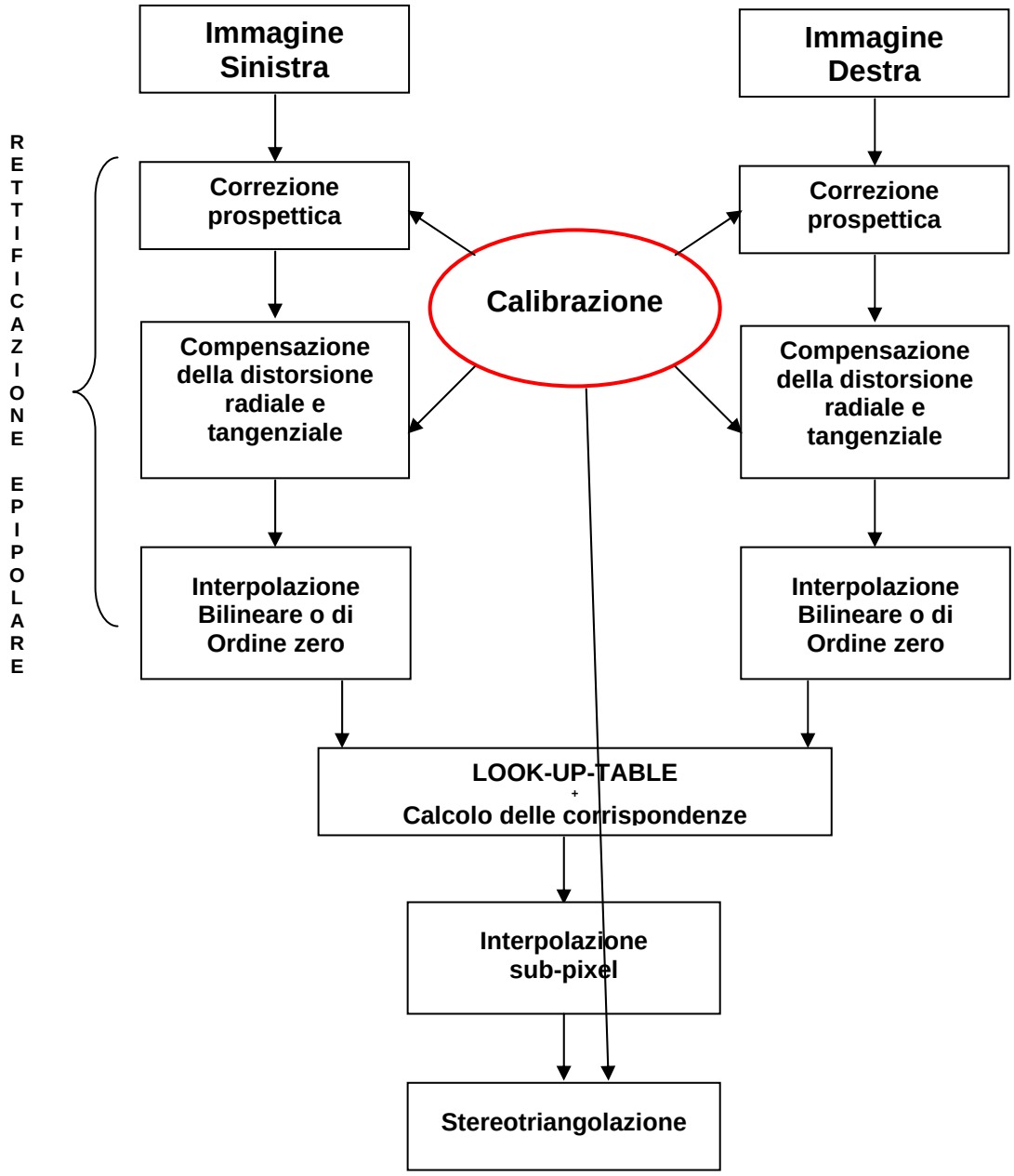



**Figura 3.1** Schema riassuntivo dei blocchi funzionali che descrivono l'architettura del sistema di visione stereoscopica

# 3.1 Il modello Pin-Hole della telecamera

## 3.1.1 Il modello semplificato

L'approssimazione che farò per l'ottica del sistema di acquisizione è quella delle **lente sottile**, che gode delle seguenti proprietà:

1. i raggi paralleli all'asse ottico incidenti sulla lente vengono rifratti in modo da passare per un punto dell'asse ottico, detto *fuoco* f.
2. i raggi che passano per il *centro* C della lente restano inalterati.

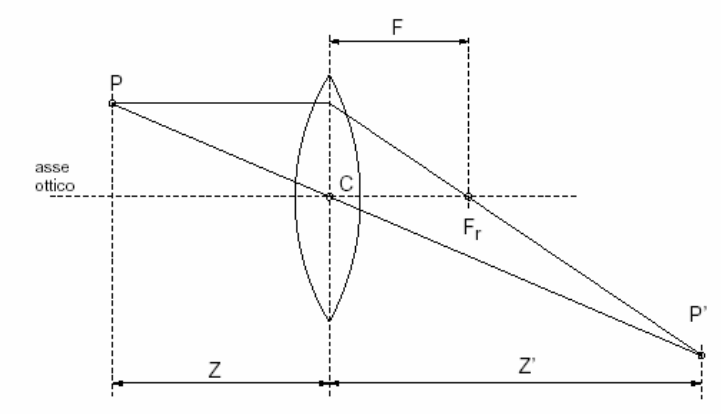



**Figura 3.2** Costruzione dell'immagine di un punto.

Dalle regole geometriche appena citate segue, in virtù dei criteri di similitudine dei triangoli, la *legge di Fresnel*, che esprime la relazione fra la distanza dell'oggetto z e la distanza immagine z' dal centro della lente:

$$\frac{1}{z}+\frac{1}{z'}=\frac{1}{f} \tag{3.1}$$

Nel caso in cui $|z| >> f$ dalla legge di Fresnel risulta:

$$|z'| \cong f \tag{3.2}$$

ottenendo in questo modo un modello geometrico della telecamera molto più semplice: il **modello pinhole**.

Il modello *pinhole* (o prospettico) consiste (fig. accanto) di un piano retina (o piano immagine) $R$ e di un punto $C$, *centro*, distante *f* (*lunghezza focale*)

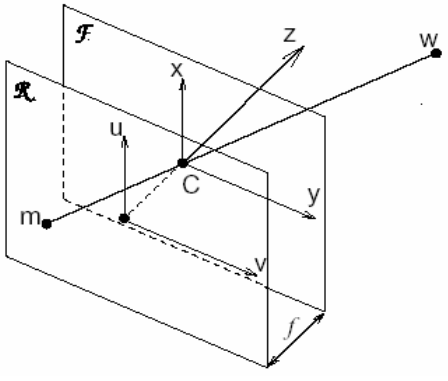



**Figura 3.3** Modello pinhole della telecamera

dal piano. La retta passante per C e ortogonale a *R* è l'*asse ottico* (asse *z* nella figura 3.3) e la sua intersezione con *R* prende il nome di *punto principale*.

Consideriamo ora un punto nello spazio *W,* di coordinate $w = (x, y, x)^T$ , e la sua proiezione su *R* attraverso C , *M*, di coordinate $m = (u, v)^T$ . Mediante semplici considerazioni sulla similitudine dei triangoli possiamo scrivere le seguenti relazioni di **proiezione prospettica**:

$$\begin{cases} u = \dfrac{f}{z} \cdot x \\ v = \dfrac{f}{z} \cdot y \end{cases} \tag{3.3}$$

Poiché la proiezione dallo spazio 3D a quello ottico 2D è evidentemente non lineare (a causa di *z* a denominatore) risulta opportuno passare alle **coordinate omogenee**.

Siano dunque

$$\tilde{m} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \quad \text{e} \quad \tilde{w} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

le coordinate omogenee di M e W rispettivamente, dove l'apice ~ denota proprio il nuovo sistema di coordinate. Con questa trasformazione le equazioni di proiezioni possono essere scritte nella seguente forma matriciale:

$$\tilde{m} = \begin{bmatrix} ku \\ kv \\ k \end{bmatrix} = \begin{bmatrix} f \cdot x \\ f \cdot y \\ x \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \tilde{P} \cdot \tilde{w} \tag{3.4}$$

La matrice $\tilde{P}$ è detta **Matrice di Proiezione Prospettica MPP**.

Si noti che *k* coincide con la terza coordinata di *W*, ovvero con la distanza dal piano *XY*. I punti per cui *k=0* sono punti all'infinito e coincidono col piano focale *F*.

Fino ad ora abbiamo considerato solamente la trasformazione prospettica tra la scena reale e il piano immagine; ma il modello realistico della telecamera deve tenere conto anche di:

1. la **pixelizzazione** o discretizzazione dovuta al sensore CCD (visto come matrice bidimensionale di pixel) e sua posizione rispetto all'asse ottico.
2. la trasformazione isometrica tra il sistema di riferimento Mondo e quello della telecamera: **rototraslazione**

### 3.1.2 La Pixelizzazione

La pixelizzazione tiene conto del fatto che:

1. il centro ottico della telecamera non coincide con il centro fisico del CCD ma ha coordinate $(u_0, v_0)$.
2. le coordinate di un punto nel sistema di riferimento standard della telecamera sono misurate in *pixel* : si introduce quindi un fattore di scala
3. la forma dei pixel non è quadrata: occorre considerare due fattori di scala diversi, lungo x ($k_u$), lungo y ($k_u$)
4. a causa di sfasamenti nella scansione di righe successive dello schermo, gli assi di riferimento immagine *u,v* non sono ortogonali ma inclinati di $\vartheta$

I punti 1), 2), 3) vengono presi in considerazione mediante l'introduzione, nella 3.3, della traslazione del centro ottico e della riscalatura **indipendente** degli assi *u* e *v*:

$$\begin{cases} u = k_u \cdot \dfrac{f}{z} \cdot x + u_0 \\ v = k_v \cdot \dfrac{f}{z} \cdot y + v_0 \end{cases} \tag{3.5}$$

dove $(u_0, v_0)$ sono le coordinate del punto principale; $k_u$ ($k_u$) è l'inverso della **dimensione efficace** del pixel lungo la direzione $u$ ($v$) (le sue dimensioni fisiche sono in $[pixel \cdot m^{-1}]$).

Dopo questo aggiornamento delle equazioni, la **MPP** diventa:

$$\tilde{P} = \begin{bmatrix} \mathrm{fk_u} & 0 & \mathrm{u_0} & 0 \\ 0 & \mathrm{fk_v} & \mathrm{v_0} & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = A \cdot [\,\mathrm{I}\,|\,0\,], \tag{3.6}$$

$$\text{con } A = \begin{bmatrix} \mathrm{fk_u} & 0 & \mathrm{u_0} \\ 0 & \mathrm{fk_v} & \mathrm{v_0} \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

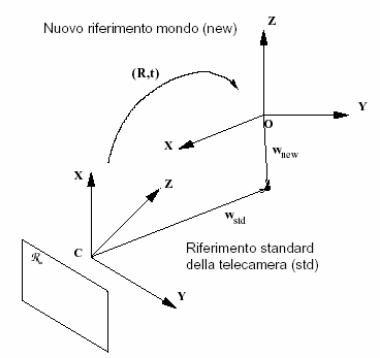
Come anticipato al punto 4) il modello più generale in realtà deve prevedere anche la possibilità che gli assi $u,v$ non siano ortogonali ma inclinati di un angolo $\vartheta$. La matrice $A$ più generale è quindi la seguente:

$$A = \begin{bmatrix} \mathrm{fk_u} & -\mathrm{fk_u} \cdot \cot\vartheta & \mathrm{u_0} \\ 0 & \mathrm{fk_v}/\sin\vartheta & \mathrm{v_0} \\ 0 & 0 & 1 \end{bmatrix} \tag{3.8}$$

Tuttavia nei moderni sistemi di acquisizione si è cercato di attenuare la disomogeneità lungo $u$ e $v$ della matrice di CCD, per cui si può considerare $\theta = \pi/2$ e la 3.8 coincide con la 3.7.

### 3.1.3 La trasformazione rigida tra la telecamera e la scena

Per tenere conto del fatto che, in generale, il sistema di riferimento mondo non coincide con il sistema di riferimento standard della telecamera, bisogna introdurre la trasformazione rigida che lega i due sistemi di riferimento. Introduciamo dunque un cambio di coordinate costituito da una rotazione R seguita da una traslazione T, ovvero:




$$w_{std} = R \cdot w_{world} + T \tag{3.9}$$

che in coordinate omogenee si scrive:

$$\tilde{w}_{std} = G \cdot \tilde{w}_{world} \tag{3.10}$$

dove $G = \begin{bmatrix} \mathrm{R} & \mathrm{T} \\ 0 & 1 \end{bmatrix}$

Pertanto dalle 3.4 e 3.10 la relazione che lega la posizione di $W_{world}$ nello spazio 3D nel sistema di riferimento mondo alla sua proiezione sul piano immagine, M, è

$$\tilde{m} = \tilde{P} \cdot \tilde{w}_{std} = \tilde{P} \cdot G \cdot \tilde{w}_{world} = \tilde{P}_{new} \cdot \tilde{w}_{world} \tag{3.11}$$

in cui $\tilde{P}_{new}$ è la nuova MPP in coordinate omogenee di dimensioni 3x4:

$$\tilde{P}_{new} = A \cdot [\,\mathrm{I}\,|\,0\,] \cdot \mathrm{G} = A \cdot \left[R \,|\, T\right] \tag{3.12}$$

Questa è la forma più generale della matrice di proiezione prospettica: la matrice A codifica i **parametri intrinseci** ($f, k_u, k_v, u_0, v_0$, che caratteristici solamente della telecamera), la matrice G codifica invece i **parametri estrinseci** (R e T).
Ponendo:

$$\alpha_u = fk_u,\ \alpha_v = fk_v,\ T = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix},\ R = \begin{bmatrix} r_1^T \\ r_2^T \\ r_2^T \end{bmatrix} \tag{3.13}$$

si ottiene la seguente espressione per $\tilde{P}$ in funzione dei parametri intrinseci ed estrinseci:

$$\tilde{P} = \begin{bmatrix} \alpha_u r_1^T - \dfrac{\alpha_u}{\tan\theta} r_2^T + u_0 r_3^T & \alpha_u t_1 - \dfrac{\alpha_u}{\tan\vartheta} r_2^T t_2 + u_0 t_3 \\ \dfrac{\alpha_v}{\sin\vartheta} r_2^T + v_0 r_3^T & \dfrac{\alpha_u}{\sin\vartheta} t_2 + v_0 t_3 \\ r_3^T & t_3 \end{bmatrix} \tag{3.14}$$

Nell'ipotesi spesso verificata in cui $\theta = \pi/2$, si ottiene:

$$\tilde{P} = \begin{bmatrix} \alpha_u r_1^T + u_0 r_3^T & \alpha_u t_1 + u_0 t_3 \\ \alpha_v r_2^T + v_0 r_3^T & \alpha_v t_2 + v_0 t_3 \\ r_3^T & t_3 \end{bmatrix} \tag{3.15}$$

La 3.14 costituisce la Matrice di Proiezione Prospettica (MPP) che governa la proiezione dei punti nello spazio (riferiti al sistema mondo) sul piano immagine. È da notare che $\tilde{P}$ è composta da 12 elementi, ma dipende da 11 parametri indipendenti; infatti è definita a meno di un fattore di scala arbitrario (il 12° parametro) che si può eliminare ricorrendo ad una normalizzazione della matrice. Gli 11 parametri indipendenti determinano univocamente $\tilde{P}$ ma la loro conoscenza è ignota in maniera esatta anche al costruttore, che deve comunque ricorrere ad un accurato processo di calibrazione per poterli stimare.

Poiché l'ipotesi $\theta = \pi/2$ è praticamente sempre verificata $\tilde{P}$ necessità in conclusione di **10 parametri**.

### 3.1.4 Modello della distorsione radiale e tangenziale a 5 parametri

Il modello pinhole della telecamera ad 11 (o 10) parametri che abbiamo fin qui ricavato non è ancora sufficiente a descrivere in maniera completa la trasformazione geometrica che subiscono le immagini acquisite.

Le considerazioni finora fatte riguardo al modello pinhole della telecamera sono state desunte dal modello delle **lenti sottili** esposto all'inizio del 3.1.1.

Ma le lenti sottili costituiscono un'astrazione matematica; per quanto i progressi compiuti dall'ottica abbiano portato alla costruzione di lenti sempre più prossime a queste ipotesi di idealità, in esse è sempre presente una certa componente dominante di distorsione **radiale** (dovuta alle dimensioni reali della lente) ed una lieve distorsione **tangenziale** (causata dal decentramento dell'asse della lente).

Ragionando in un sistema di coordinate polari con origine nel centro dell'immagine, la distorsione radiale agisce sui pixel dell'immagine in funzione della sola distanza dal centro $\rho$, mentre quella tangenziale dipende anche dall'angolo $\vartheta$.

Nella figura 3.4 a fianco ad esempio è riportata un'immagine affetta da notevole distorsione radiale.

**Figura 3.4** Immagine affetta da una notevole distorsione radiale

Nelle ultime decadi sono stati sviluppati diversi modelli matematici di distorsione al fine di correggerla e tornare così alle ipotesi di lente sottile: tale processo di correzione prende il nome di **compensazione della distorsione**.

La distorsione può essere pensata come una funzione non lineare che si applica ai singoli pixel dell'immagine originaria non distorta di coordinate $(u, v)$, che dipende, oltre che dalle stesse coordinate, anche dalla distanza dal centro ( $\rho = \sqrt{u^2 + v^2}$ ) e dall'angolo ( $\vartheta = \tan^{-1} v/u$ ). Se $(u_d, v_d)$ sono le coordinate distorte, la relazione che esprime la trasformazione da quelle non distorte è:

$$\begin{bmatrix} u_d \\ v_d \end{bmatrix} = f(u, v, \rho, \vartheta) \tag{3.16}$$

La maggior parte dei modelli matematici è basata sulla stima dei coefficienti dello sviluppo in serie di Taylor di $f$ arrestato al secondo ordine e non tengono quasi mai conto della distorsione tangenziale: pertanto forniscono soltanto due parametri di distorsione, che sono

sufficienti a correggere fenomeni di aberrazione molto lievi, risultando inefficaci per alterazioni enormi come quella in figura 3.4.

In vista di future applicazioni del mio sistema di visione stereoscopica a telecamere dotate di una maggiore profondità di campo e di un ampio angolo visuale (caratterizzate da focale piuttosto corta e affette quindi da maggior aberrazione di quella attuale) ho preferito sviluppare un algoritmo di compensazione della distorsione sia radiale che tangenziale basato su **cinque** parametri piuttosto che due: tre per la distorsione radiale e due per quella tangenziale.

Il modello che ho utilizzato è stato sviluppato dalla Oulu University (Finlandia), ed è lo stesso adottato dal Matlab Calibration ToolBox.

La trasformazione operata dall'algoritmo è dunque la seguente:

$$\begin{bmatrix} u_d \\ v_d \end{bmatrix} = \underbrace{(1 + kc_1\rho^2 + kc_2\rho^4 + kc_5\rho^6)}_{\text{Applica la distorsione radiale}} \cdot \begin{bmatrix} u \\ v \end{bmatrix} + \underbrace{\begin{bmatrix} 2kc_3 \cdot u \cdot v + kc_4(\rho^2 + 2u^2) \\ kc_3(\rho^2 + 2v^2) + 2kc_4 \cdot u \cdot v \end{bmatrix}}_{\text{Applica la distorsione tangenziale}} \quad (3.17)$$

dove $kc$ d'ora in avanti sarà convenzionalmente indicato come un vettore di cinque componenti così ordinate: $kc = [kc_1, kc_2, kc_3, kc_4, kc_5]$. Il parametro più importante è $kc_1$ perché è quello che influenza maggiormente l'entità della distorsione; se $kc_1 > 0$ si ottiene una deformazione di tipo mezzaluna, se $kc_1 < 0$ si ha una distorsione a botte, entrambe illustrate in figura 3.5:

**Figura 3.5**

N.B. I 5 parametri di distorsione si aggiungono ai parametri intrinseci della telecamera e verranno presi in considerazione nella calibrazione.

N.B. È da notare che se avessi considerato i soli due termini della sola distorsione radiale (come nella maggior parte dei sistemi di visione) avrei ottenuto una relazione più semplice:

$$\begin{bmatrix} u_d \\ v_d \end{bmatrix} = (1 + kc_1\rho^2 + kc_2\rho^4) \cdot \begin{bmatrix} u \\ v \end{bmatrix} \quad (3.18)$$

computazionalmente più veloce ma con minor precisione, per le ragioni esposte; quest'ultima infatti è la stessa che è utilizzata dalla maggior parte di fotocamere digitali in commercio per correggere velocemente l'aberrazione radiale della lente.
Per ulteriori approfondimenti rimando ai [14, 15] e agli articoli di D.C. Brown [21], che fu il primo a studiare il fenomeno della distorsione delle lenti nel 1966.
L'algoritmo di compensazione e i risultati verranno illustrati più avanti.

## 3.2 Calibrazione delle telecamere

### 3.2.1 Parametri di calibrazione

La calibrazione consiste nel misurare con accuratezza i parametri **intrinseci** ed **estrinseci** del modello della telecamera comprensivo della distorsione. Poichè questi parametri governano il modo in cui punti dello spazio si proiettano sulla retina, l'idea è che, conoscendo le proiezioni di punti 3D di coordinate note (punti di calibrazione), sia possibile ottenere i parametri incogniti risolvendo le equazioni della proiezione prospettica individuate dalla MPP (3.15). Nelle ultime due decadi sono stati sviluppati due metodi *diretti* di calibrazione, come quello di Caprile e Torre (1990) [15] e quello di Tsai (1987) [14], che partono dalla misura delle coordinate 3D di un oggetto di calibrazione e deducono, mediante metodi lineari di inversione della MPP, i parametri intrinseci ed estrinseci.

Nel mio caso mi sono avvalso dei risultati di una ricerca condotta da Zhengyou **Zhang** nel 1999 per conto di **Microsoft** e **Intel Corp** [12, 13]. Tale ricerca ha portato l'università di Oulu (Finlandia) a sviluppare un tool-box per Matlab appositamente dedicato (*Camera & Stereo Camera Calibration Toolbox for Matlab* [11]), che ho utilizzato per stimare i parametri delle due telecamere del sistema di visione stereoscopica in questione.

**I parametri da stimare sono 15**: **10** ricavati dal modello pinhole e **5** dalla distorsione:

1. $\alpha_u = fk_u$ : fattore di scala dell'asse u $\rightarrow$ 1 parametro
2. $\alpha_v = fk_v$ : fattore di scala dell'asse v $\rightarrow$ 1 parametro
3. $(u_0, v_0)$ : coordinate del centro ottico nel sistema telecamera $\rightarrow$ 2 parametri
4. $T = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$ : traslazione dal sistema mondo al sistema telecamera $\rightarrow$ 3 parametri

5. $R = \begin{bmatrix} r_1^T \\ r_2^T \\ r_2^T \end{bmatrix}$ : rotazione dal sistema mondo al sistema telecamera $\rightarrow$ 3 parametri

6. $kc = [kc_1, kc_2, kc_3, kc_4, kc_5]$ : coefficienti di distorsione $\rightarrow$ 5 parametri

Riscrivendo la MPP (3.15) nel modo seguente:

$$\tilde{P} = \left[\begin{array}{c|c} q_1^T & q_{14} \\ q_2^T & q_{24} \\ q_3^T & q_{34} \end{array}\right] \tag{3.19}$$

dalla 3.11 si ottiene:

$$\tilde{m} = \begin{bmatrix} ku \\ kv \\ k \end{bmatrix} = \begin{bmatrix} q_1^T w + q_{14} \\ q_2^T w + q_{24} \\ q_3^T w + q_{34} \end{bmatrix} \tag{3.20}$$

Dati N punti di calibrazione **non complanari** ($w_i$), ciascuna corrispondenza tra un punto dell'immagine $m_i = (u_i, v_i)^T$ ed il punto della scena $w_i$ fornisce una coppia di equazioni:

$$\begin{cases} (q_1 - u_i q_3)^T w_i + q_{14} - u_i q_{34} = 0 \\ (q_2 - v_i q_3)^T w_i + q_{24} - v_i q_{34} = 0 \end{cases} \tag{3.21}$$

La MPP (incognita) consiste di 12. In teoria, quindi, 6 punti **non complanari** sono sufficienti per il calcolo di $\tilde{P}$ e ne serviranno altri 5 per stimare i coefficienti di distorsione; nella pratica ho utilizzato molti più punti di calibrazione e viene risolto un problema di minimi quadrati diminuendo così l'incertezza associata alla stima.

### 3.2.2 Calibrazione del sistema di visione stereoscopica

Poiché il sistema di visione stereoscopica è fondato sull'impiego di due telecamere, la procedura di calibrazione deve ovviamente essere ripetuta per ciascun dispositivo.

Come ho accennato, ai fini della calibrazione ho utilizzato un tool per Matlab appositamente dedicato, che permette l'estrazione dei parametri intrinseci ed estrinseci delle telecamere: il Camera Calibration Toolbox [11].

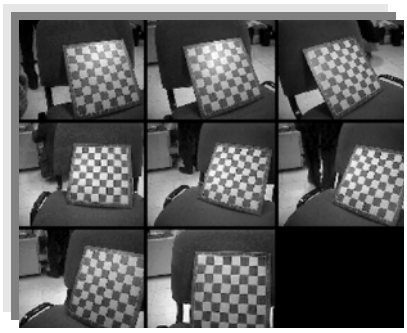
Il metodo di calibrazione adottato dal tool, come accennavo, è proprio quello di Zhang, la cui peculiarità, rispetto ai metodi precedenti, risiede nel fatto che non occorre conoscere la posizione assoluta nello spazio dei punti di calibrazione (come accadeva nei metodi di Tsai e Torre [14, 15]) ma è sufficiente conoscere la posizione relativa di ciascun punto rispetto agli altri sfruttando un vincolo in più: che i punti di calibrazione giacciano tutti su una superficie di vertici noti che viene disposta di fronte alle telecamere con orientamenti diversi. La superficie in questione è una **scacchiera** e i punti di calibrazione costituiscono i vertici di ogni quadrato.
La procedura di calibrazione richiede di acquisire diverse immagini della scacchiera (17 nel mio caso), sotto diversi angoli di vista, in modo che non tutti i punti siano complanari.
Se poi si acquisiscono coppie stereoscopiche della stessa immagine è possibile utilizzare un tool di ottimizzazione che ricomputa i parametri intrinseci ed estrinseci ricavati indipendentemente per ciascuna telecamera in modo da minimizzare **l'errore di riproiezione** di ciascun punto di calibrazione nello spazio! Quest'ultima è possibile grazie allo Stereo Camera Calibration Toolbox.

Ecco in sintesi i passi eseguiti per la calibrazione del sistema di visione binoculare:

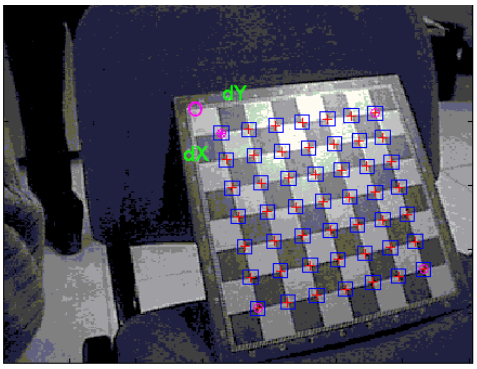
1. Dapprima si acquisiscono molte coppie stereoscopiche della scacchiera sotto angoli di vista differenti. Aumentando il numero di immagini di calibrazione diminuisce l'incertezza associata alla stima dei parametri: ho impiegato pertanto 17 immagini.
2. Si lancia il Camera Calibration ToolBox.
3. La procedura inizia a partire dalle immagini acquisite dalla camera sinistra: si selezionano manualmente i quattro vertici della scacchiera. Un sistema di riconoscimento automatico degli angoli assiste la fase di selezione dei vertici.
4. A questo punto il programma richiede di inserire il numero di quadrati (***Nq***) appartenenti al perimetro selezionato e le dimensioni (***Lx*** e ***Ly*** ) in millimetri: *Nq=36*, *Lx=Ly=32 mm*.
5. Il punto 2 è ripetuto per tutte le immagini. Al termine il programma restituisce i parametri intrinseci ed estrinseci, riferiti al sistema di riferimento standard della telecamera, con la relativa incertezza di stima.
6. Si ripetono i punti 2, 3, 4, 5 anche per le immagini acquisite dalla camera destra.
7. Si lancia lo Stereo Camera Calibration ToolBox
8. Si forniscono i parametri intrinseci ed estrinseci appena calcolati.
9. Il programma ricomputa i parametri intrinseci ed estrinseci appena ricavati minimizzando **l'errore di riproiezione** di ciascun punto di calibrazione nello spazio.
10. Al termine restituisce i parametri intrinseci corretti e la relativa incertezza di stima. I parametri estrinseci (la matrice di rotazione R e il vettore di traslazione T) sono riferiti alla telecamera sinistra.

Il diagramma a immagini in figura 3.6 riassume le fasi della calibrazione.

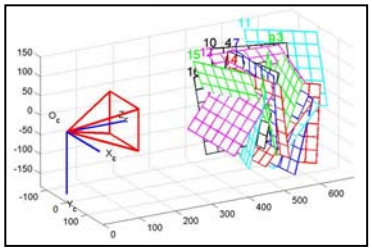
**Telecamera sinistra**



Immagini campione



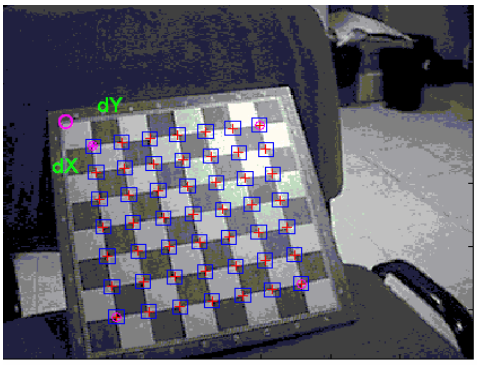

Selezione dei punti di calibrazione



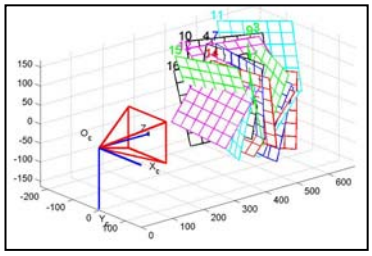
Estrazione dei parametri intrinseci e riproiezione dei punti nello spazio virtuale

**Telecamera destra**

Immagini campione




Selezione dei punti di calibrazione



Estrazione dei parametri intrinseci e riproiezione dei punti nello spazio virtuale

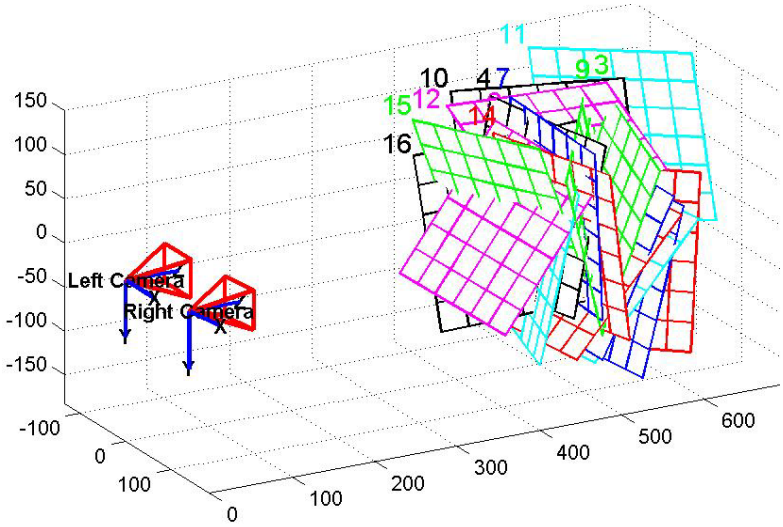



**Figura 3.6** Riproiezione dei punti nello spazio virtuale e minimizzazione degli errori di stima: calcolo dei nuovi parametri intrinseci ed estrinseci e ricostruzione spaziale dell'orientamento delle telecamere e della scacchiera.

### 3.2.3 Risultati della calibrazione

La tabella seguente riporta i parametri intrinseci ed estrinseci del sistema in oggetto:

| | **TELECAMERA SINISTRA** | **TELECAMERA DESTRA** |
|---|---|---|
| **Fattore di scala [pixel]** | $\alpha_u = 393.67$<br>$\alpha_v = 425.31$ | $\alpha_u = 390.87$<br>$\alpha_v = 422.36$ |
| **Centro ottico [pixel]** | $u_0 = 188.56$<br>$v_{0v} = 137.77$ | $u_0 = 191.79$<br>$v_{0v} = 131.30$ |
| **Coefficienti di distorsione [adimensionali]** | $kc_1 = -0.31658$<br>$kc_2 = 0.14950$<br>$kc_3 = -0.00107$<br>$kc_4 = 0.00056$<br>$kc_5 = 0.0000$ | $kc_1 = -0.31955$<br>$kc_2 = 0.11659$<br>$kc_3 = -0.00050$<br>$kc_4 = -0.00316$<br>$kc_5 = 0.00000$ |
| **Matrice di rotazione (riferita alla telecamera sinistra)** | $R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ | $R = \begin{bmatrix} 0.998 & -0.036 & 0.051 \\ 0.037 & 0.999 & -0.023 \\ -0.050 & 0.025 & 0.998 \end{bmatrix}$ |
| **Matrice di traslazione (riferita alla telecamera destra)** | $T = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$ | $T = \begin{bmatrix} -119.089 \\ -3.356 \\ -4.981 \end{bmatrix}$ |
| **Matrice di Proiezione Prospettica MPP** | $\tilde{P} = \begin{bmatrix} 393.67 & 0 & 188.56 & 0 \\ 0 & 425.31 & 137.77 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$ | $\tilde{P} = \begin{bmatrix} 380.44 & -9.40 & 211.53 & -47504.57 \\ 9.35 & 425.29 & 121.12 & -2071.83 \\ -0.05 & 0.02 & 0.99 & -4.98 \end{bmatrix}$ |

### 3.2.4 Valutazione dei risultati

I parametri relativi ai due dispositivi sono confrontabili ma non coincidenti; infatti, come anticipato all'inizio del capitolo, nonostante sia stato impiegato lo stesso modello, esistono sempre delle differenze legate al processo costruttivo.

Il fattore di scala, il centro ottico e i coefficienti di distorsione sono caratterizzati da un'incertezza nulla perché in realtà il programma ricomputa i parametri delle telecamere e l'errore complessivo integrandolo all'interno delle matrici *R* e *T*. Nella matrice R l'incertezza intacca la quinta cifra decimale; nella matrice T l'errore è dell'ordine di $10^{-2}$. L'errore nella stima dei parametri intrinseci viene così ricondotto all'incertezza sulla misura di R e T.

La matrice R della telecamera sinistra è la matrice identità perché in essa è fissato il sistema di riferimento globale. La R della telecamera destra differisce da quella sinistra per meno dell'1% a significare che le due matrici CCD risultano ben collimate.

La matrice T dà invece la posizione del centro della camera destra $C_2$ rispetto a quello sinistro $C_1$. L'origine di tale sistema è il centro ottico della telecamera sinistra; l'asse x congiunge $C_1C_2$ ed è diretto verso $C_2$; l'asse z è perpendicolare al piano immagine ed è diretto nel verso di osservazione (si veda figura 3.7); l'asse y segue per la convenzione della terna levogira.

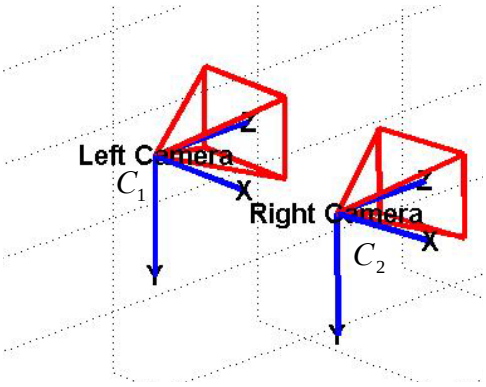



**Figura 3.7** Orientamento cartesiano degli assi di riferimento

Poiché i valori sono espressi in millimetri, la distanza tra le telecamere vale $t_1 \cong 119mm$ e coincide con quella reale misurata direttamente con il decimetro: tale riscontro fornisce quindi un primo conforto dell'accuratezza raggiunta.

Al contrario $t_2$, $t_3$ (pari rispettivamente a *3.35, 4.98 mm*) contrastano con la evidente complanarità degli obbiettivi delle due telecamere, che proprio a tale scopo sono montati su un supporto rigido e dagli assi ben collimati. Ciò, come anticipato, è in parte dovuto al fatto che matrice CCD raramente coincide con il centro fisico dell'obbietto e può essere addirittura decentrato. Dall'altra è dovuto all'errore associato alla stima dei parametri e rientra pertanto nell'intervallo di accuratezza calcolato.

Più avanti, quando parleremo della stereo-triangolazione, ovvero della stima della distanza euclidea a partire da coppie di immagini stereoscopiche, potremo verificare che il livello di accuratezza dei parametri stimati è decisamente notevole: scopriremo infatti che la precisione varia da pochi millimetri su distanze di 1 metro a pochi centimetri intorno ai 3 metri!

Il toolBox di Matlab non si limita solamente a fornire i parametri di calibrazione ma fornisce anche una serie di diagrammi per lo studio della distorsione radiale e tangenziale:

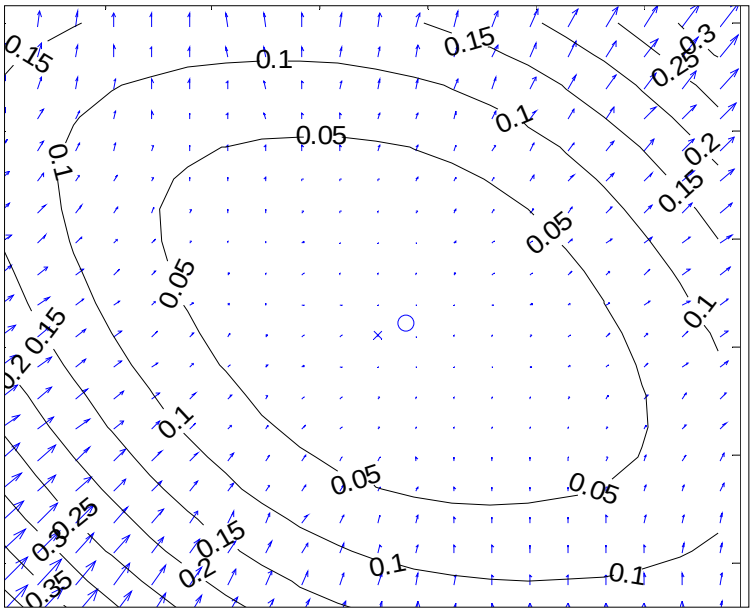



**Figura 3.8** Mostra l'impatto della distorsione tangenziale su ogni pixel dell'immagine. Ogni freccia rappresenta l'effettivo spostamento di un pixel indotto dalla distorsione. L'entità dello spostamento è espresso in pixel e va da 0.05 pixel, vicino al centro, a 0.15 pixel verso gli estremi dell'immagine

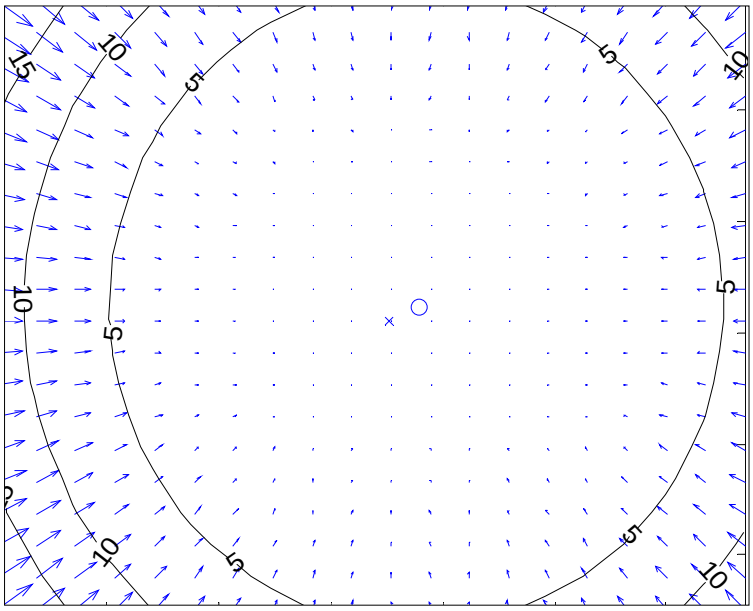



**Figura 3.9** Mostra l'impatto della distorsione radiale su ogni pixel dell'immagine. Ogni freccia rappresenta l'effettivo spostamento di un pixel indotto dalla distorsione. L'entità dello spostamento, espresso in pixel, va da 1 pixel , vicino al centro, a 15 pixel verso gli estremi dell'immagine

In merito ai sensori usati risulta che la distorsione radiale prevale decisamente su quella tangenziale. Ciò era prevedibile anche dalla disparità tra il coefficiente di distorsione radiale kc1 e quelli che governano la distorsione tangenziale (kc3, kc4).

## 3.3 Rettificazione epipolare

### 3.3.1 Cos'è la rettificazione

Abbiamo visto nell'introduzione di questo capitolo che una delle problematiche maggiori della visione stereoscopica computazionale consiste nell'accoppiamento tra punti nelle due immagini che sono proiezione dello stesso punto della scena: detti punti sono chiamati **punti coniugati o corrispondenti**.

Il calcolo dell'accoppiamento è possibile sfruttando il fatto che le due immagini differiscono solo lievemente, cosicchè un particolare della scena appare "simile" nelle due immagini. Il concetto di "similarità" si traduce nella minimizzazione di una funzione di costo (tipo correlazione) che scandisce tutte le righe e colonne dell'altra immagine. Basandosi solo su questo vincolo, però, sono possibili molti falsi accoppiamenti ed è inoltre evidente che la ricerca del punto coniugato corrispondente risulta computazionalmente onerosa.

In questa sessione introdurremo un vincolo che rende il calcolo delle corrispondenze trattabile: il **vincolo epipolare**, il quale afferma che il corrispondente di un punto in una immagine può trovarsi solo su una retta (**retta epipolare**) nell'altra immagine. Grazie a questo processo, detto **rettificazione epipolare**, la ricerca delle corrispondenze diventa unidimensionale invece che bidimensionale, diminuendo così la complessità computazionale.

Nel paragrafo che segue verrà introdotto ed esaminato analiticamente il concetto di retta epipolare ricorrendo all'ausilio della geometria epipolare.

In quello successivo verrà finalmente descritta ed analizzata la rettificazione epipolare e sarà presentato l'algoritmo in C che rettifica le immagini acquisite dalle telecamere.

### 3.3.2 Geometria Epipolare

Vediamo ora quale relazione lega due immagini di una stessa scena ottenute da due telecamere diverse. In particolare, ci chiediamo, dato un punto $m_1$ nella immagine 1 (sinistra), quali vincoli esistono sulla posizione del suo coniugato1 $m_2$ nella immagine 2 (destra). Alcune semplici considerazioni geometriche indicano che il punto coniugato di $m_1$ deve giacere su di una linea retta nella immagine 2, chiamata ***retta epipolare*** di $m_1$.

La geometria epipolare è importante anche (e soprattutto) perché descrive la relazione tra due viste di una stessa scena, dunque è fondamentale in qualunque tecnica di Visione computazionale basata su più di una immagine.

Si consideri il caso illustrato in figura 3.10, dove $R_1$ e $R_2$ sono i piani retinici, sui quali è proiettata l'immagine dei due sensori, e $c_1$ e $c_2$ sono i centri ottici delle lenti.

Dato un punto $m_1$ nella immagine 1, il suo coniugato $m_2$ nella immagine 2 è vincolato a giacere sull'intersezione del piano immagine $R_2$ con il piano determinato da $m_1$, $c_1$, $c_2$, chiamato **piano epipolare**. Questo perchè il punto $m_2$ può essere la proiezione di un qualsiasi punto nello spazio giacente sul raggio ottico di $m_1$. Inoltre si osserva che tutte le linee epipolari di una immagine passano per uno stesso punto, chiamato ***epipolo*** ($e_1$ ed $e_2$), e che i piani epipolari costituiscono un fascio di piani che hanno in comune la retta passante per i centri ottici $c_1$ e $c_2$. Il segmento che unisce $c_1$ e $c_2$ prende il nome di ***baseline***.

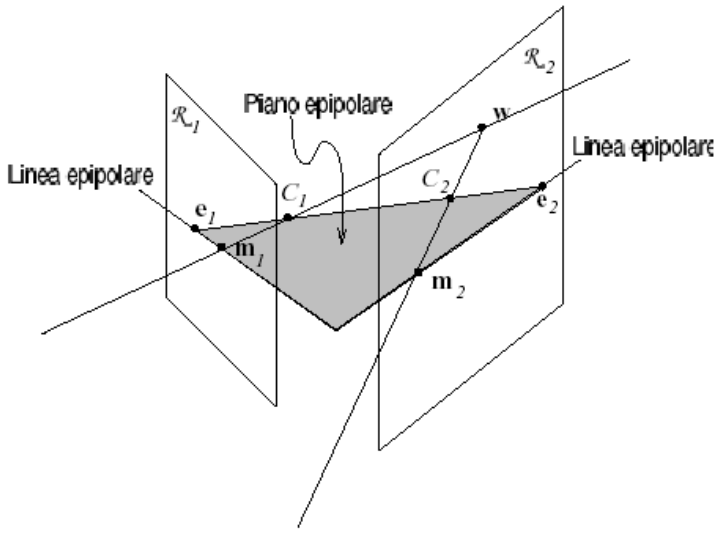



**Figura 3.10** La figura mostra la costruzione della linea o retta epipolare

In conclusione, per differenti punti 3D, il piano epipolare ruota attorno alla baseline e tutte le rette epipolari si intersecano all'epipolo.

Una volta ricavata l'espressione analitica della retta epipolare corrispondente di $m_1$, come dicevamo, la ricerca del punto coniugato $m_2$ è limitata ai soli punti appartenenti alla linea epipolare. Le due immagini in cima alla pagina seguente illustrano questo procedimento, in cui a destra sono disegnate le rette epipolari corrispondenti ai punti marcati con la croce nell'immagine sinistra.

Immagine sinistra Immagine destra


**Figura 3.11** A destra sono disegnate le rette epipolari corrispondenti ai punti marcati con la croce nella immagine sinistra.

Prima di passare a descrivere il processo di rettificazione è opportuno fare le seguenti considerazioni.

Quando $c_1$, $c_2$, è nel piano focale della telecamera di destra, l'epipolo destro è all'infinito e le linee epipolari formano un fascio di rette parallele nella immagine di destra.

Un caso molto speciale si ha quando entrambe gli epipoli sono all'infinito, che accade quando la baseline $c_1 c_2$ è contenuta in entrambi i piani focali, ovvero quando i piani retina sono paralleli alla linea di base! In questo caso le linee epipolari formano un fascio di linee parallele in entrambe le immagini.

Se poi si applica una **trasformazione omografica** alle immagini in modo che le linee epipolari oltre ad essere parallele siano anche **orizzontali** e **collineari** a quelle dell'altra immagine, si ottiene una situazione particolarmente favorevole per il calcolo delle corrispondenze: punti corrispondenti giacciono sulla stessa riga della immagine (vista come una matrice di pixel).

Qualunque coppia di immagini può essere trasformata in modo che le linee epipolari siano parallele, orizzontali e collineari in ciascuna immagine: questa trasformazione prende il nome di **rettificazione**.

### 3.3.3 Rettificazione di una coppia di immagini stereo

Data una coppia di immagini stereoscopiche, la rettificazione epipolare determina una trasformazione di ciascun piano immagine in modo che coppie coniugate di linee epipolari diventino parallele, orizzontali e collineari (dove per collinearità si intende che una retta epipolare dell'immagine destra sia la prosecuzione della sua corrispondente nell'immagine sinistra).
Le immagini così rettificate possono essere pensate come acquisite da un nuovo sistema di stereovisione ottenuto per rototraslazione delle telecamere originarie. Il vantaggio più importante della rettificazione è che il calcolo delle stereo-corrispondenze fra punti coniugati risulta enormemente semplificato perché la ricerca del coniugato di un punto nell'altra immagine passa da bidimensionale a monodimensionale.
È quindi evidente il notevole risparmio computazione: l'algoritmo di ricerca passa infatti da una complessità quadratica ($\propto n^2$) ad una lineare ($\propto n$), se $n$ è una delle dimensioni trasversali (base o altezza) dell'immagine.

La rettificazione va effettuata a valle della calibrazione, ovvero una volta calcolati e resi disponibili i parametri intrinseci ed estrinseci dei sensori che determinano le matrici di Proiezione Prospettica delle due telecamere $\tilde{P}_1, \tilde{P}_2$ e che legano le coordinate omogenee 3D di un punto $\tilde{w}$ nello spazio alle coordinate omogenee delle sue proiezioni sui piani immagine dei due sensori. Dalla 3.11:

$$\begin{cases} \tilde{m}_1 = \tilde{P}_1 \cdot \tilde{w} \\ \tilde{m}_2 = \tilde{P}_2 \cdot \tilde{w} \end{cases} \tag{3.22}$$

Dove la generica $\tilde{P}$ secondo la 3.12 è:

$$\tilde{P} = A \cdot \left[ R \mid T \right] = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \tag{3.23}$$

L'idea che si cela dietro alla rettificazione è quella di definire due nuove matrici $\tilde{P}n_1$, $\tilde{P}n_2$ ottenute per rotazione di quelle originarie intorno ai loro centri ottici fintanto che i piani focali non diventino complanari e contengano la baseline. Ciò, si può dimostrare, assicura che gli epipoli siano punti all'infinito e quindi le linee epipolari risultano parallele.

Per avere rette epipolari orizzontali, invece, la baseline deve essere parallela all'asse X del nuovo sistema di riferimento standard delle due telecamere. In aggiunta, per soddisfare alla richiesta di collinearità tra punti coniugati, le coppie di linee epipolari devono avere la stessa coordinata verticale. Questo viene ottenuto richiedendo che le nuove matrici di prospezione $An_1$, $An_2$ (incluse nella definizione 3.19) abbiano gli stessi parametri intrinseci, ovvero gli stessi valori per le coordinate del centro $(u_0, v_0)$ e gli stessi valori per le lunghezze focali (meglio detti coefficienti di scalatura orizzontale e verticale $\alpha_u$, $\alpha_v$).

Si noti che imporre l'uguaglianza delle lunghezze focali fa sì che anche i piani immagine diventino complanari (immagine seguente):

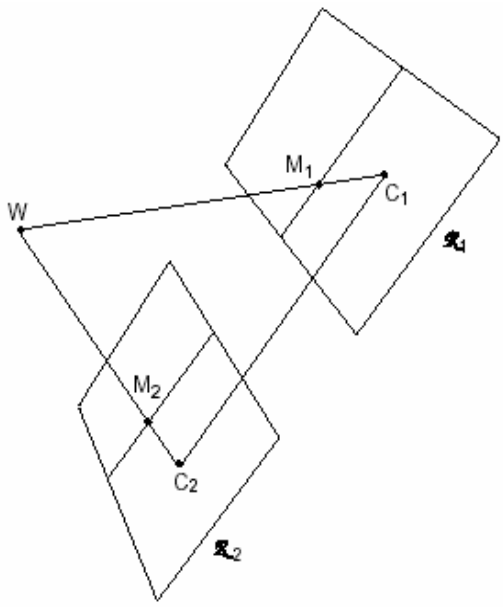



**Figura 3.12** Telecamere rettificate. I piani retina sono complanari e paralleli alla baseline

Per far sì inoltre che oggetti quadrati appiano ancora quadrati piuttosto che rettangolari, bisogna imporre l'uguaglianza tra $\alpha_u$ e $\alpha_v$. Nella pratica si sceglie $\alpha_u = \alpha_v = \alpha = \min(\alpha_u, \alpha_v)$.

In conclusione: la posizioni dei centri ottici $c_1$ e $c_2$ sono le stesse di quelle originarie mentre gli orientamenti delle terne di riferimento di ciascuna telecamera differiscono dai precedenti solo per rotazione di angoli opportuni. I parametri intrinseci delle telecamere vengono imposti uguali e con gli stessi valori dei coefficienti di scalatura orizzontale e verticale. Pertanto le risultanti $\tilde{Pn}_1$, $\tilde{P}n_2$ differiranno solo per le posizioni dei loro centri ottici e le due telecamere possono essere pensate come una singola camera traslata lungo l'asse X del suo sistema di riferimento.

#### 3.3.3.1 Calcolo delle nuove Matrici di Proiezione Prospettica

Per determinare le nuove MPP dobbiamo prima scegliere opportunamente le nuove matrici $An_1, An_2$ , (prospezione) , $Rn_1$, $Rn_2$ (rotazione) $Tn_1$, $Tn_2$ (traslazione), legate alla generica $\tilde{P}$ dalla seguente:

$$\tilde{P} = A \cdot [\, R \mid T \,] \tag{3.24}$$

***Calcolo di Rn***

L'orientamento delle due telecamere nel nuovo sistema di riferimento è determinato dalle rotazioni $Rn_1$ e $Rn_2$ . Per i motivi suddetti risulta $Rn_1 = Rn_2 = Rn$

Una matrice di rotazione è individuata dalle componenti dei coseni direttori ( $r_i^T$ ) della nuova terna di riferimento rispetto a quella originaria. Pertanto la $Rn$ sarà:

$$Rn = \begin{bmatrix} r_1^T \\ r_3^T \\ r_2^T \end{bmatrix} \tag{3.25}$$

Scegliamo innanzitutto l'asse Z del nuovo riferimento coincidente con l'asse Z (individuato dal versore $\hat{z}$ ) del sistema originario; da ciò segue: $r_3^T = \hat{z}$ . (3.26)

Il nuovo asse X deve essere parallelo alla baseline; da ciò segue: $r_1^T = \dfrac{(c_1 - c_2)}{\|c_1 + c_2\|}$ (3.27)

Il nuovo asse Y, per costruzione, è quindi: $r_2^T = r_3 \wedge r_1$ (3.28)

***Calcolo di Tn1, Tn2***

Innanzitutto si osservi che una generica $T$ può essere espressa in funzione del centro ottico $c$ :

$$T = -R \cdot c \tag{3.29}$$

Poiché che le posizioni di c1 e c2 restano inalterate le nuove Tn1, Tn2 saranno:

$$\begin{cases} Tn_1 = -Rn \cdot c_1 \\ Tn_2 = -Rn \cdot c_2 \end{cases} \tag{3.30}$$

***Scelta di An***

Per i motivi detti al 3.3.3 occorre imporre gli stessi valori dei parametri intrinseci, cioè la matrice di prospezione deve essere la stessa per entrambe le telecamere. Sappiamo che la relazione che lega un punto della scena alla sua proiezione (3.3) è non lineare, per via della coordinata z a denominatore. Tuttavia essa è ovviamente lineare per i punti a z costante, che giacciono su un piano parallelo all'asse focale. Per far sì che in questo caso la trasformazione sia omotetica (cioè oggetti quadrati appaiano come quadrati piuttosto che come rettangoli) bisogna imporre gli stessi valori dei coefficienti di scalatura orizzontale e verticale: nella pratica si sceglie

$$\alpha_u = \alpha_v = \alpha = \min(\alpha_u, \alpha_v)$$

***La nuove MPP***

In conclusione le nuove MPP sono:

$$\begin{aligned} \tilde{Pn}_1 &= An \cdot \left[ Rn \mid -Rn \cdot c_1 \right] \\ \tilde{Pn}_2 &= An \cdot \left[ Rn \mid -Rn \cdot c_2 \right] \end{aligned} \tag{3.31}$$

e, come anticipato, differiscono solo per le posizioni di $c_1$, $c_2$.

### 3.3.3.2 Determinazione della trasformazione rettificante

Note le nuove MPP rimane da determinare la trasformazione rettificatrice, in grado di "mappare" i pixel dell'immagine di partenza nel piano immagine finale.
A questo scopo è utile pensare ad un'immagine come il luogo geometrico delle intersezioni tra il piano immagine medesimo e tutti i possibili raggi ottici che proiettano i punti dello spazio 3D e passano per il centro ottico.
Ciò che abbiamo fatto nel passo precedente (Step 1) è stato quello di far muove i piani immagine delle due telecamere nel nuovo sistema di riferimento, lasciando fissi i raggi ottici. Ora vogliamo invece determinare la proiezione dei medesimi raggi nel riferimento appena calcolato.

Ciascun pixel dell'immagine costituisce la proiezione (o intersezione) su detto piano di una infinità non numerabile di punti che giacciono su un'unica retta passante per il centro ottico $c$ ed il punto in questione. Questa retta è chiamata raggio ottico o anche retta di interpretazione. È pertanto impossibile nel caso di visione monoscopica risalire dalla posizione del pixel alla posizione 3D del punto da cui è stato proiettato. Ciò a meno di non far uso della stereoscopia.

Se $\tilde{m}_{O1}$ è il vettore in coordinate omogenee del pixel nell'immagine sinistra, mediante la MPP si ricava l'equazione analitica dei punti $w$ del raggio ottico (espresso nelle coordinate cartesiane):

$$w = c_1 + \lambda \cdot (A_1 \cdot R_1)^{-1} \cdot \tilde{m}_{O1} \quad , \quad \lambda \in R \tag{3.32}$$

Poiché nel cambiamento di riferimento il raggio ottico rimane fisso, nel piano immagine del nuovo sistema di riferimento lo stesso raggio ottico proietta il punto $\tilde{m}_{n1}$, dalla 3.18:

$$\tilde{m}_{n1} = \tilde{P}_{n1} \cdot \begin{bmatrix} w \\ 1 \end{bmatrix} = An \cdot [Rn \mid Tn] \cdot \begin{bmatrix} w \\ 1 \end{bmatrix} \tag{3.33}$$

e quindi insieme alla 3.28 si ricava $\tilde{m}_{n1}$ in funzione di $\tilde{m}_{O1}$:

$$\tilde{m}_{n1} = (An \cdot Rn) \cdot (A_1 \cdot R_1)^{-1} \cdot \tilde{m}_{O1} \tag{3.34}$$

Per un punto $\tilde{m}_{O2}$ dell'immagine destra segue analogamente:

$$\tilde{m}_{n2} = (An \cdot Rn) \cdot (A_2 \cdot R_2)^{-1} \cdot \tilde{m}_{O2} \tag{3.35}$$

Dalle 3.30 e 3.31 si definiscono le seguenti:

$$\begin{cases} Trf1 = (An \cdot Rn) \cdot (A_1 \cdot R_1)^{-1} \\ Trf2 = (An \cdot Rn) \cdot (A_2 \cdot R_2)^{-1} \end{cases} \tag{3.36}$$

### 3.3.4 Interpolazione bilineare

Le matrici di trasformazione $Trf1$, $Trf2$ sono dette matrici di rettificazione e determinano il modo in cui i pixel dell'immagine originaria si mappano in quella finale, **rettificata**, nel nuovo sistema di riferimento.

Si noti tuttavia che se ciascun pixel dell'immagine originale ha coordinate intere ciò non è vero per il suo corrispondente nell'immagine rettificata; l'applicazione delle trasformazioni 3.30 e 3.31 infatti restituisce un vettore di coordinate reali.

Rimane quindi da stabilire il valore cromatico da associare al pixel corrispondente dell'immagine rettificata, le cui coordinate non sono numeri interi. L'approccio classico dell'Image Processing fa uso di opportuni filtri interpolatori **lineari** che tentano di approssimare alla precisione richiesta il valore cercato. Ma il prezzo pagato per una migliore approssimazione si traduce in aumento della complessità computazionale. Per questo nelle applicazioni comuni di image processing si preferiscono le seguenti strategie **non lineari**:

1. Interpolazione di ordine zero: assegna il valore cromatico corrispondente alle stesse coordinate approssimate però all'intero più vicino.
2. Interpolazione bilineare: fa una stima del valore cercato sulla base della conoscenza del valore cromatico relativo ai quattro pixel adiacenti

Il primo è il metodo di interpolazione più semplice, ma è anche quello che causa la maggior perdita di informazione oltre che di qualità dell'immagine. È indicato in tutti quei casi che non richiedano un'elevata precisione; non è indicato quindi nella ricostruzione della struttura 3D di una scena, dove l'accoppiamento dei punti coniugati e la loro posizione effettiva nell'immagine, influenza notevolmente il risultato della misura di stereo-triangolazione.
Per le motivazioni suddette il metodo più usato è quello bilineare.
Nell'algoritmo di rettificazione, che sarà descritto tra poco, ho implementato entrambi i metodi di interpolazione; l'uso dell'uno o dell'altro può essere selezionato arbitrariamente dall'utente.
Vediamo dapprima una breve descrizione dell'interpolazione bilineare.

Come anticipato, poiché è noto il valore cromatico assegnato ai quattro pixel (a coordinate intere) adiacenti a quello in esame (di coordinate non intere $(u',v')$) , il valore cromatico da associare a $(u',v')$, e che denoto con $V(u',v')$, può essere interpolato a partire da suoi quattro "vicini" usando la seguente relazione:

$$V(u',v') = a \cdot u' + b \cdot v' + c \cdot u' \cdot v' + d \tag{3.37}$$

dove i coefficienti $a,b,c,d$ si determinano dalle quattro equazioni in quattro incognite che derivano imponendo il passaggio per i quattro punti adiacenti.
L'equazione 3.37 tuttavia non è quasi mai utilizzata nella pratica, proprio perché richiede la risoluzione di un sistema di Kramer 4x4, con ovvia inversione di matrici.
Esiste infatti una procedura più semplice, e computazionalmente meno onerosa, per il calcolo della 3.37, che è descritta nel prossimo paragrafo.

### 3.3.4.1 L'algoritmo di interpolazione bilineare

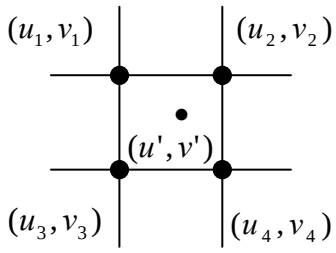



**Figura 3.13**

Siano $(u_1, v_1)$, $(u_2, v_2)$, $(u_3, v_3)$, $(u_4, v_4)$, $V_1$, $V_2$, $V_3$, $V_4$ rispettivamente le coordinate e i valori cromatici dei quattro pixel adiacenti a $(u', v')$ (figura 3.13). A partire da essi si calcolano i parametri:

$$\begin{aligned} dx &= u' - u_1; \\ dy &= v' - v_1; \\ a_1 &= (1-dx)\cdot(1-dy); \\ a_2 &= dx\cdot(1-dy); \\ a_3 &= (1-dx)\cdot dy; \\ a_4 &= dx\cdot dy; \end{aligned} \tag{3.38}$$

E da questi:

$$V(u', v') = [a_1, a_2, a_3, a_4]\cdot\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} \tag{3.39}$$

In conclusione l'interpolazione bilineare richiede la conoscenza di quattro parametri.
Tali parametri devono essere calcolati per ciascuna posizione di pixel nell'immagine, cioè per $width \cdot height$ punti. Al fine di ridurre i tempi di calcolo, tali coefficienti vengono calcolati una sola volta, non appena lanciato il programma, e i risultati sono memorizzati in quattro tabelle di look-up.

**N.B.** Nell'ambito di questo lavoro ho preferito lavorare su immagini interamente a colori, pertanto, affinché le immagini rettificate siano ancora true-color, la trasformazione bilineare deve essere applicata tre volte, per ogni componente RGB di colore.

## 3.4 L'algoritmo di rettificazione epipolare

### 3.4.1 Le fasi dell'algoritmo

Le equazioni 3.30 , 3.31 determinano la trasformazione tra le coordinate dei pixel nell'immagine di origine ($\tilde{m}_{O1}$, $\tilde{m}_{O2}$) e le nuove coordinate ($\tilde{m}_{n1}$, $\tilde{m}_{n2}$) nell'immagine rettificata.
Tuttavia, in tutte le considerazioni fatte finora per ricavare le espressioni delle nuove matrici di proiezione prospettica (3.27) e le trasformazioni rettificanti (3.30, 3.31), abbiamo omesso di considerare l'effetto della distorsione radiale e tangenziale. Per tenere conto di tale effetto l'immagine originaria deve essere preventivamente sottoposta ad un processo di compensazione della distorsione, di cui parlavamo al 3.1.4.
Detto ciò veniamo alla descrizione dei passi dell'algoritmo di rettificazione epipolare.
(Il pedice "1" caratterizza l'immagine sinistra, il pedice "2" l'immagine destra.)

1. inizializza i parametri intrinseci ed estrinseci del modello pin-hole delle telecamere estratti nella fase di calibrazione
2. usa tali parametri per calcolare le matrici di Prospezione (A1, A2), Rotazione (R1, R2) e Traslazione (T1, T2)
3. a partire dalle matrici appena calcolate computa le matrici di proiezione prospettica ($\tilde{P}_1, \tilde{P}_2$)
4. ricava le posizioni dei centri ottici $c_1$, $c_2$ e le usa per fissare l'asse X del nuovo sistema di riferimento
5. calcola le nuove matrici di Prospezione (An), Rotazione (Rn) e Traslazione (Tn1, Tn2), relative alla nuova terna di riferimento
6. a partire dalle matrici appena calcolate computa le nuove matrici di proiezione prospettica ($\tilde{P}n_1, \tilde{P}n_2$)
7. determina le matrici di rettificazione ($Trf_1, Trf_2$) e le rispettive inverse $Trf_1^{-1}$, $Trf_2^{-1}$
8. alloca nella memoria dinamica uno spazio sufficiente ad accogliere la futura coppia di immagini stereoscopiche rettificate: le dimensioni coincidono con quelle delle immagini di origine
9. le celle di memoria appena allocate vengono inizializzate a zero
10. le coordinate $[u_{n1}, v_{n1}]$, $[u_{n2}, v_{n2}]$ di ciascuna cella, vista come elemento di una matrice bidimensionale, vengono tradotte in coordinate omogenee $\tilde{m}_{n1}$, $\tilde{m}_{n2}$

11. a ciascun punto di coordinate $\tilde{m}_{n1}$, $\tilde{m}_{n2}$ applica la trasformazione rettificatrice per risalire alle sue coordinate di origine **non distorte** $\tilde{m}_{O1}$, $\tilde{m}_{O2}$: $\begin{cases} \tilde{m}_{O1} = Trf_1^{-1} \cdot \tilde{m}_{n1} \\ \tilde{m}_{O2} = Trf_2^{-1} \cdot \tilde{m}_{n2} \end{cases}$

12. torna alle coordinate cartesiane e normalizza $m_{O1}$ e $m_{O2}$ rispetto al punto principale e al fattore di scala orizzontale e verticale. Si ottengono i vettori normalizzati: $[u_1, v_1]$, $[u_2, v_2]$

13. a ciascun punto $[u_1, v_1]$ ($[u_2, v_2]$) applica la funzione che computa la distorsione radiale e tangenziale (eq. 3.17). In questo modo passa alle coordinate distorte $[u_{d1}, v_{d1}]$ ($[u_{d2}, v_{d2}]$), che individuano la posizione reale dei punti nell'immagine di partenza:

$$\begin{bmatrix} u_d \\ v_d \end{bmatrix} = (1 + kc_1\rho^2 + kc_2\rho^4 + kc_5\rho^6) \cdot \begin{bmatrix} u \\ v \end{bmatrix} + \begin{bmatrix} 2kc_3 \cdot u \cdot v + kc_4(\rho^2 + 2u^2) \\ kc_3(\rho^2 + 2v^2) + 2kc_4 \cdot u \cdot v \end{bmatrix}$$

14. alle coordinate distorte così ottenute applica uno dei metodi di interpolazione selezionati (ordine zero o bilineare) e calcola i valori cromatici ad essi associati: $V_1(u_{d1}, v_{d1})$ $V_2(u_{d2}, v_{d2})$

15. tutti gli indici che mappano i punti delle coppie stereo di origine nelle immagini finali rettificate (4xcamera) e i coefficienti di interpolazione bilineare (4xcamera), od ordine zero (1xcamera), sono memorizzati all'interno di tabelle di look-up, per un totale complessivo di 18 look-up-table

16. a ciascun punto di coordinate cartesiane $m_{n1}$ ($m_{n2}$), nell'immagine finale, viene associato il valore cromatico RGB $V_1(u_{d1}, v_{d1})$ ($V_2(u_{d2}, v_{d2})$), altrimenti un valore nullo (corrispondente a nero) nel caso in cui $[u_{d1}, v_{d1}]$, ($[u_{d2}, v_{d2}]$), calcolato al punto 13, non appartenga al dominio $D_1$ ($D_2$) dell'immagine:

$$\begin{cases} m_{n1} = V_1(u_{d1}, v_{d1}) \quad \text{oppure} \quad m_{n1} = 0 \quad \text{se} \quad (u_{d1}, v_{d1}) \notin D_1 \\ m_{n2} = V_2(u_{d2}, v_{d2}) \quad \text{oppure} \quad m_{n2} = 0 \quad \text{se} \quad (u_{d2}, v_{d2}) \notin D_2 \end{cases}$$

N.B. Si osservi che i passi $1 \div 15$ sono eseguiti una sola volta e solo all'avvio del programma; questi servono ad indicizzare le tabelle di look-up, che verranno utilizzate solo al passo 16 invocando, come vedremo, la funzione `rectify()`.

### 3.4.2 Risultati della rettificazione

Nelle figure che seguono sono mostrate coppie di immagini acquisite dalle telecamere stereoscopiche all'interno del laboratorio.

Si evidenziano i particolari della scena ripresa

- prima della rettificazione
- dopo la rettificazione
- le differenze qualitative apportate dai due metodi di interpolazione

**Immagini originarie prima della rettificazione**

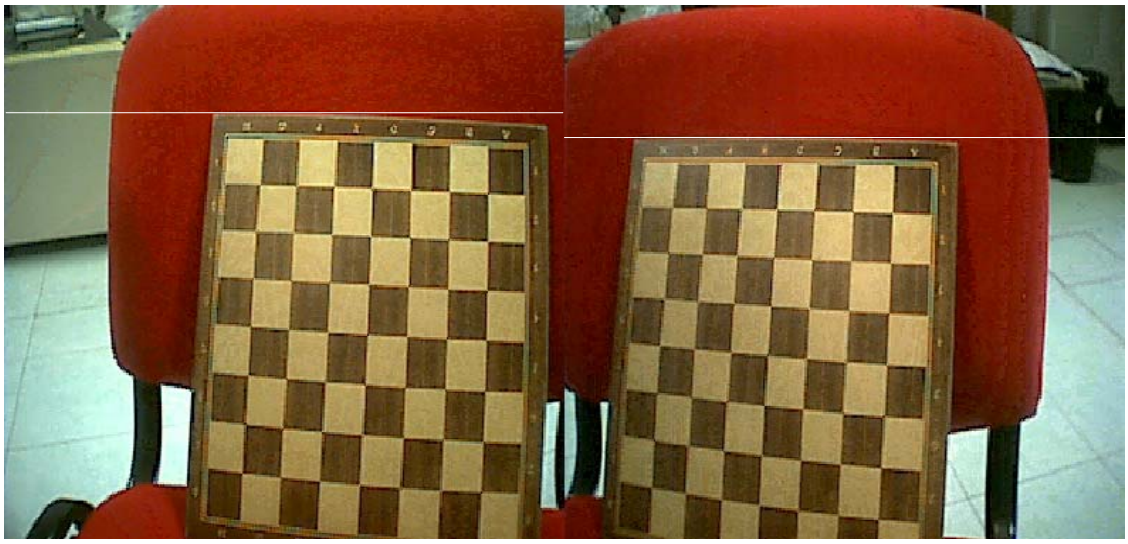


**Figura 3.14** Immagini non ancora rettificate: la rette orizzontali sono state tracciate volutamente per enfatizzare la totale assenza di **collinearità** tra le immagini, nonostante i due sensori siano perfettamente collimati sulla struttura portante. È visibile anche la bombatura della scacchiera a causa del fenomeno di distorsione della lente.

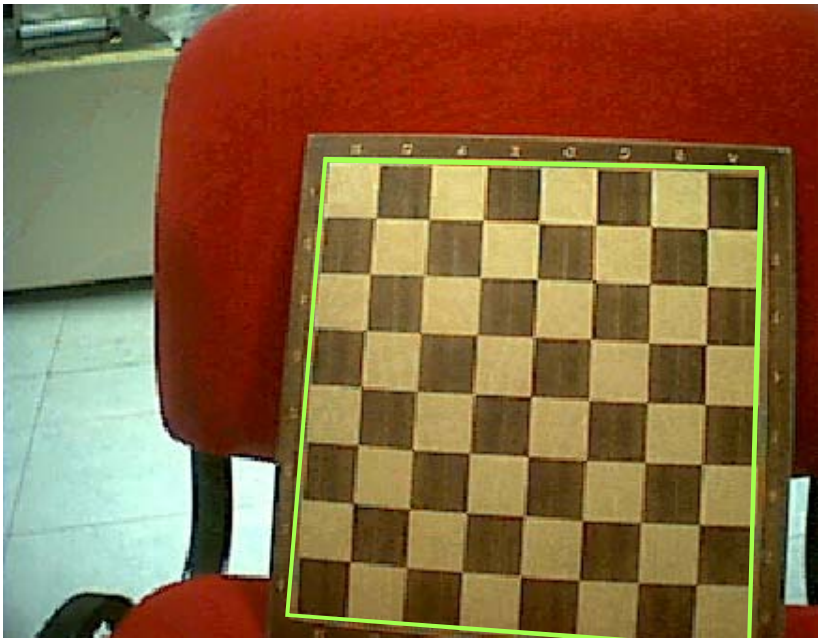


**Figura 3.15** Il quadrato verde è stato tracciato proprio per sottolineare la distorsione a botte causata dalla lente.

**Immagini rettificate con interpolazione bilineare**

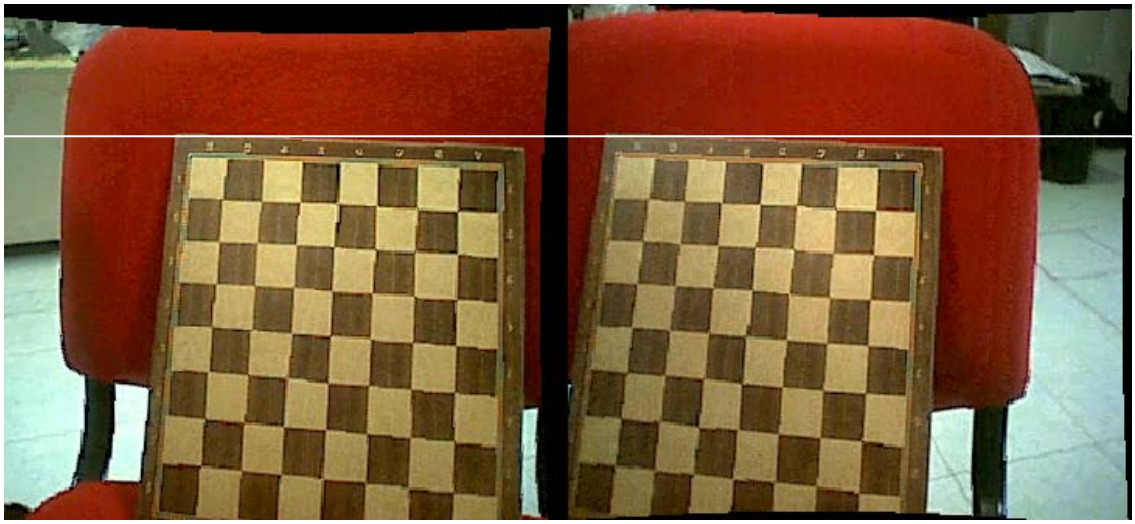


**Figura 3.16** Immagini rettificate con interpolazione di ordine zero. La rettificazione ha allineato le immagini sullo stesso asse. È visibile l'effetto apportato dalla compensazione della distorsione, che è enfatizzato dalle mezzelune in nero presenti ai quattro lati delle immagini. È da notare anche la zigrinatura dei tratti rettilinei (o degli angoli della scacchiera) causata dal tipo di interpolazione usato per determinare il valore cromatico di ciascun pixel.

**Figura 3.17** Ancora alcuni dettagli delle immagini rettificate, appositamente ingrandite per evidenziare l'effetto di zigrinatura dei tratti rettilinei (o degli angoli della scacchiera) causata dall'interpolazione di ordine zero. È evidente la notevole perdita di qualità rispetto alle immagini di origine.

**Immagini rettificate con interpolazione bilineare**

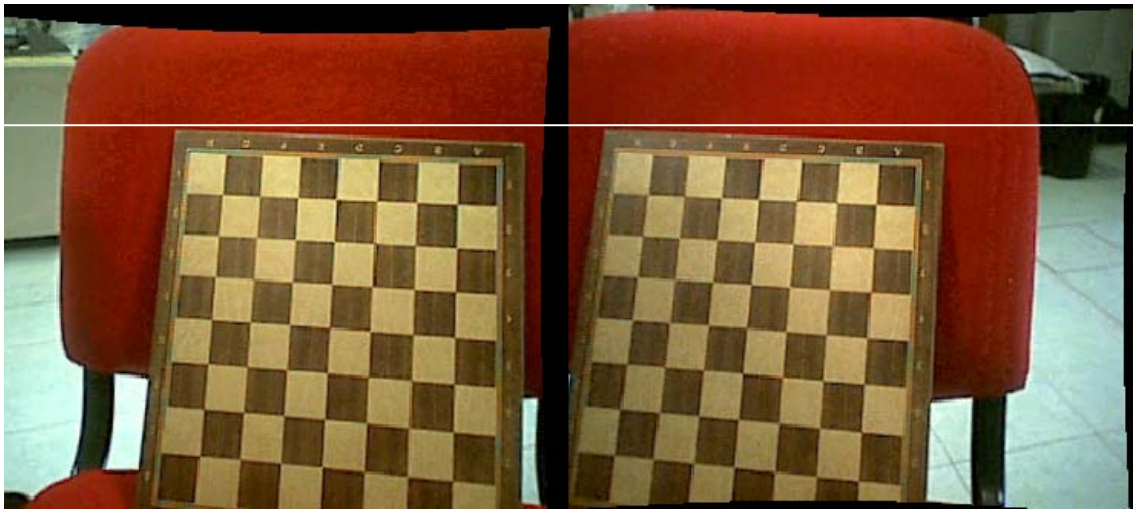


**Figura 3.18** Immagini rettificate con interpolazione bilineare. È visibile l'effetto apportato dalla compensazione della distorsione, che è enfatizzato dalle mezzelune in nero presenti ai quattro lati delle immagini. È da notare inoltre la totale assenza di zigrinatura dei tratti rettilinei (o degli angoli della scacchiera), a manifesto dell'efficacia e della qualità di questo metodo di interpolazione.

**Figura 3.19** Ancora alcuni dettagli delle immagini rettificate, appositamente ingrandite per evidenziare la totale assenza di zigrinatura dei tratti rettilinei (o degli angoli della scacchiera), a manifesto dell'efficacia e della qualità di questo metodo di interpolazione. Si confrontino con i rispettivi dettagli nel caso di interpolazione di ordine zero.

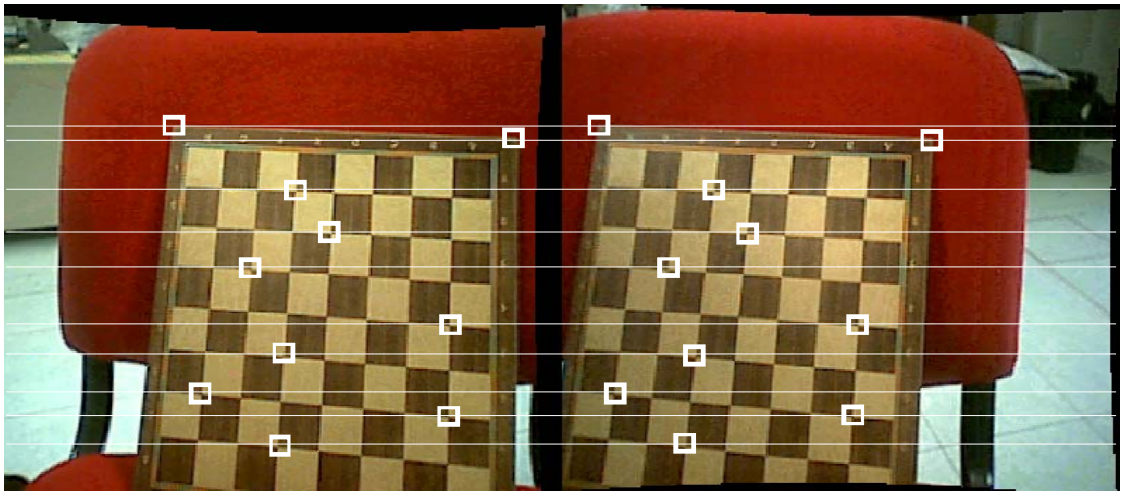


**Figura 3.20** Grazie alla rettificazione, infine, le rette epipolari (tracciate per alcuni punti in quadrato) sono tutte orizzontali, parallele e collineari: la proprietà di collinearità è evidenziata dal fatto che ciascuna coppia di punti coniugati (individuati da quadrati) giacciono sulla stessa retta orizzontale.

Come si evince dalle rette tracciate nell'ultima immagine, la rettificazione ha fatto sì, come previsto dalla teoria, che le linee epipolari risultino parallele, orizzontali e collineari. Quanto ottenuto avrebbe funzionato anche se i due sensori fossero stati montati sulla struttura portante senza alcuna vincolo di complanarità o di orientamento, purchè, naturalmente, essi guardassero dalla stessa parte della scena.

L'inequivocabile correttezza della rettificazione, evidenziata dall'ultima immagine della pagina precedente, denuncia quindi la qualità e l'accuratezza dei parametri intrinseci ed estrinseci stimati grazie alla calibrazione. Senza una conoscenza approssimativamente esatta di tali parametri, le immagini rettificate non avrebbero infatti mai soddisfatto ai tre requisiti.

Si può dire quindi che la rettificazione realizzata via software compensa tutti gi errori e le imperfezioni coinvolte nella costruzione fisica e meccanica dei sensori e della struttura portante.

Il sistema di visione stereoscopica costituisce ora un apparato di visione praticamente "perfetto", che (a meno della risoluzione delle telecamere) , è comparabile con altri sistemi di visione stereo professionali, già tarati e calibrarti dal costruttore.

Vorrei far notare che nel corso di questo lavoro di tesi è stato necessario ripetere la calibrazione soltanto due volte, in seguito alla realizzazione del supporto fisico definitivo, più rigido e stabile di quello provvisorio iniziale.

Si è visto inoltre che il metodo di interpolazione usato influisce sulla qualità delle immagini rettificate. Le funzioni C che realizzano la rettificazione consentono all'utente di selezionare preventivamente il tipo di interpolazione da adottare in funzione della finalità richiesta.

Nell'applicazione all'inseguimento della palla ho utilizzato un'interpolazione di ordine zero, molto più rapida e veloce della bilineare, che invece è stata usata nell'autolocalizzazione del robot per il riconoscimento di landmark sul pavimento. In quest'ultimo caso, vedremo, la precisione con cui stimare la posizione 3D del landmark nel sistema di riferimento del robot costituisce un fattore emblematico ai fini della navigazione in un ambiente.

## 3.5 Stereo-Triangolazione

### 3.5.1 Ricostruzione 3D

La stereo-triangolazione è il metodo che permette di stimare la posizione 3D di un punto nello spazio a partire dalle coordinate in pixel delle sue proiezioni sui piani retinici dei sensori.

Note le matrici di proiezione prospettica $\tilde{P}_{n1}$, $\tilde{P}_{n2}$, che governano la proiezione di un punto di coordinate $w$ nei piani immagine, e note le coordinate omogenee in pixel delle sue proiezioni $\tilde{m}_{n1}$, $\tilde{m}_{n2}$, la posizione di $w$ si può ricavare dall'intersezione dei raggi ottici ad essi associati.

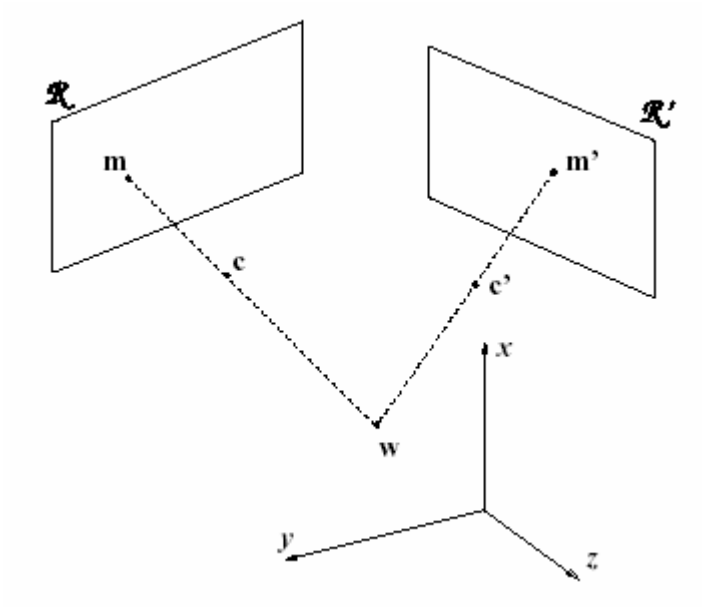



**Figura 3.21** Ricostruzione della posizione 3D di w dall'intersezione dei raggi ottici

Mettendo a sistema le equazioni parametriche delle rette che descrivono i raggi ottici (3.32) segue:

$$\begin{cases} w = c_1 + \lambda_1 \cdot (A_n \cdot R_n)^{-1} \cdot \tilde{m}_{n1} \\ w = c_2 + \lambda_2 \cdot (A_n \cdot R_n)^{-1} \cdot \tilde{m}_{n2} \end{cases} \tag{3.40}$$

Dove $A_n$, $R_n$, $c_1$, $c_2$ sono le matrici di proiezione, rotazione e le posizioni dei centri ottici in cui si possono decomporre $\tilde{P}_{n1}$, $\tilde{P}_{n2}$ secondo le 3.31:

$$\begin{cases} \tilde{Pn}_1 = An \cdot [\, Rn \;\mid\; -Rn \cdot c_1 \,] \\ \tilde{Pn}_2 = An \cdot [\, Rn \;\mid\; -Rn \cdot c_2 \,] \end{cases} \tag{3.41}$$

N.B. Le matrici $A_n$, $R_n$, sono le nuove matrici ricavate a valle della rettificazione, e le coordinate $\tilde{m}_{n1}$, $\tilde{m}_{n2}$ sono le nuove proiezioni di $w$ sui piani retinici dopo l'applicazione delle trasformazioni rettificatrici (3.36) e dopo la compensazione della distorsione radiale e tangenziale.

Le 3.40 costituiscono un sistema **sovradeterminato** di sei equazioni in cinque incognite (tre per le coordinate di $w$ più i due parametri liberi $\lambda_1$, $\lambda_2$), la cui soluzione ai minimi quadrati si può ottenere con il metodo della **pseudoinversa**.
In realtà, poiché la soluzione ai minimi quadrati non ha alcun significato geometrico, per risolvere la 3.40 si preferisce individuare la posizione di $w$ come quel punto per cui è minima la distanza euclidea tra le rette (sghembe) che descrivono i raggi ottici. Infatti estremamente raro che i due raggi ottici si intersechino esattamente in un punto.
A tale scopo conviene scrivere le 3.40 in una forma più semplice.

Siano $(l_1, m_1, n_1) = (A_n \cdot R_n)^{-1} \cdot \tilde{m}_{n1}$ i parametri direttori del raggio ottico associato alla camera sinistra e $(l_2, m_2, n_2) = (A_n \cdot R_n)^{-1} \cdot \tilde{m}_{n2}$ i parametri direttori del raggio ottico associato alla camera destra. Siano inoltre $w = (x, y, z)$, $c_1 = (x_1, y_1, z_1)$, $c_2 = (x_2, y_2, z_2)$.
In questo modo le 3.40 diventano:

$$\begin{cases} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \lambda_1 \cdot \begin{bmatrix} l_1 \\ m_1 \\ n_1 \end{bmatrix} \\ \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} + \lambda_2 \cdot \begin{bmatrix} l_2 \\ m_2 \\ n_2 \end{bmatrix} \end{cases} \tag{3.42}$$

Per determinare $w$ occorre prima calcolare le coordinate dei punti M ed N che costituiscono le estremità del segmento di minima distanza tra le rette; successivamente sarà possibile

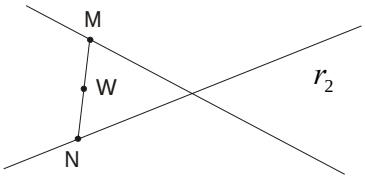



**Figura 3.22**

esprimere $w$ come punto medio di $\overline{MN}$. La figura 3.22 illustra il procedimento descritto. Derivando parzialmente l'espressione della distanza euclidea tra due punti nello spazio 3D si arriva alla seguente soluzione per $\lambda_1$, $\lambda_2$:

$$\begin{bmatrix}\lambda_1\\ \lambda_2\end{bmatrix}=\begin{bmatrix}(l_1^2+m_1^2+n_1^2) & -(l_1l_2+m_1m_2+n_1n_2)\\ (l_1l_2+m_1m_2+n_1n_2) & (l_2^2+m_2^2+n_2^2)\end{bmatrix}^{-1}\cdot\begin{bmatrix}b_1\\ b_2\end{bmatrix} \tag{3.43}$$

dove:

$$b_1=(c_2-c_1)^T\cdot\begin{bmatrix}l_1\\ m_1\\ n_1\end{bmatrix},\quad b_2=(c_2-c_1)^T\cdot\begin{bmatrix}l_2\\ m_2\\ n_2\end{bmatrix} \tag{3.44}$$

Noti a questo punto $\lambda_1$, $\lambda_2$, le coordinate di M, N si calcolano direttamente dalle 3.42:

$$\begin{cases}\begin{bmatrix}x_M\\ y_M\\ z_M\end{bmatrix}=\begin{bmatrix}x_1\\ y_1\\ z_1\end{bmatrix}+\lambda_1\cdot\begin{bmatrix}l_1\\ m_1\\ n_1\end{bmatrix}\\ \begin{bmatrix}x_N\\ y_N\\ z_N\end{bmatrix}=\begin{bmatrix}x_2\\ y_2\\ z_2\end{bmatrix}+\lambda_2\cdot\begin{bmatrix}l_2\\ m_2\\ n_2\end{bmatrix}\end{cases} \tag{3.45}$$

e da queste si ricavano le coordinate di w=(x,y,z):

$$x=\frac{x_M+x_N}{2};\ y=\frac{y_M+y_N}{2};\ z=\frac{z_M+z_N}{2} \tag{3.46}$$

### 3.5.2 Scelta del sistema di riferimento

Nel corso di tutto il capitolo ho considerato le coordinate del punto w espresse in un riferimento puramente arbitrario e fittizio, ma giunti a questo punto del lavoro è opportuno precisare che ho scelto di fissare la terna di riferimento assoluta con l'origine posta esattamente nel centro delle due telecamere e, come orientamento nello spazio, quello della camera sinistra (figura 3.23 nella pagina seguente). Tale scelta è dettata dal fatto che, applicando il sistema di visione su un robot mobile, tutte le misure siano espresse proprio rispetto alla terna mobile del robot piuttosto che in un una terna fissa. La scelta dell'orientamento, invece, è puramente

convenzionale, giacchè le due camere giacciono su un supporto rigido che le rende allineate e parallele fra loro.

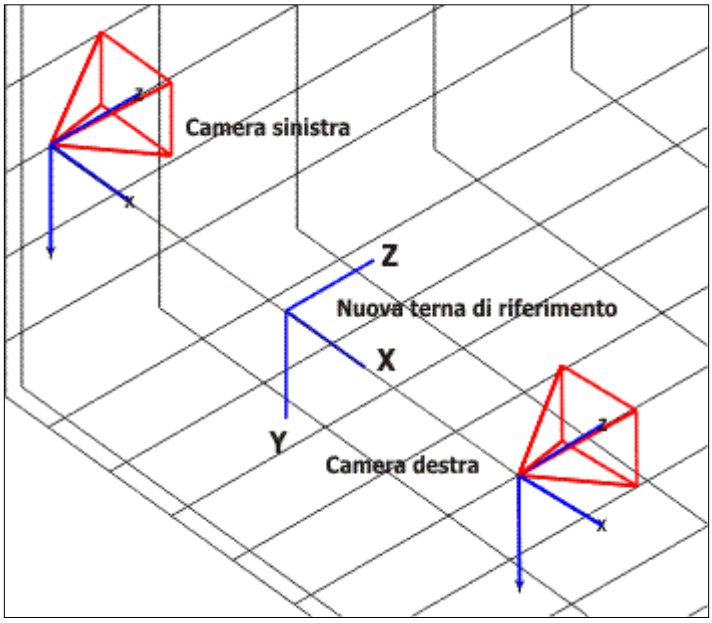



**Figura 3.23** Posizione della nuova terna di riferimento

### 3.5.3 L'algoritmo di stereo-triangolazione

Ecco in sintesi i passi dell'algoritmo di stereotriangolazione che, note le coordinate in pixel dei punti coniugati, restituisce le componenti 3D del punto di cui sono proiezione, espresse nel sistema di riferimento assoluto.

1. Rettifica le immagini fornite dalla stereo-camera
2. Acquisisce le coordinate in pixel dei punti coniugati, $\tilde{m}_{n1}$, $\tilde{m}_{n2}$, e le matrici $A_n$, $R_n$, $c_1$, $c_2$ che descrivono orientamento e posizione dei sistemi di riferimento delle rispettive camere dopo la rettificazione
3. trasforma $\tilde{m}_{n1}$, $\tilde{m}_{n2}$ in coordinate omogenee
4. calcola i parametri direttori delle rette che descrivono i raggi ottici
5. determina il valore di $\lambda_1$, $\lambda_2$ tramite la 3.43
6. calcola le posizioni dei punti M, N che individuano il segmento di minima distanza fra le rette (3.45)
7. restituisce le coordinate 3D del punto w grazie alla 3.46

## 3.6 Architettura finale del sistema di visione stereoscopica

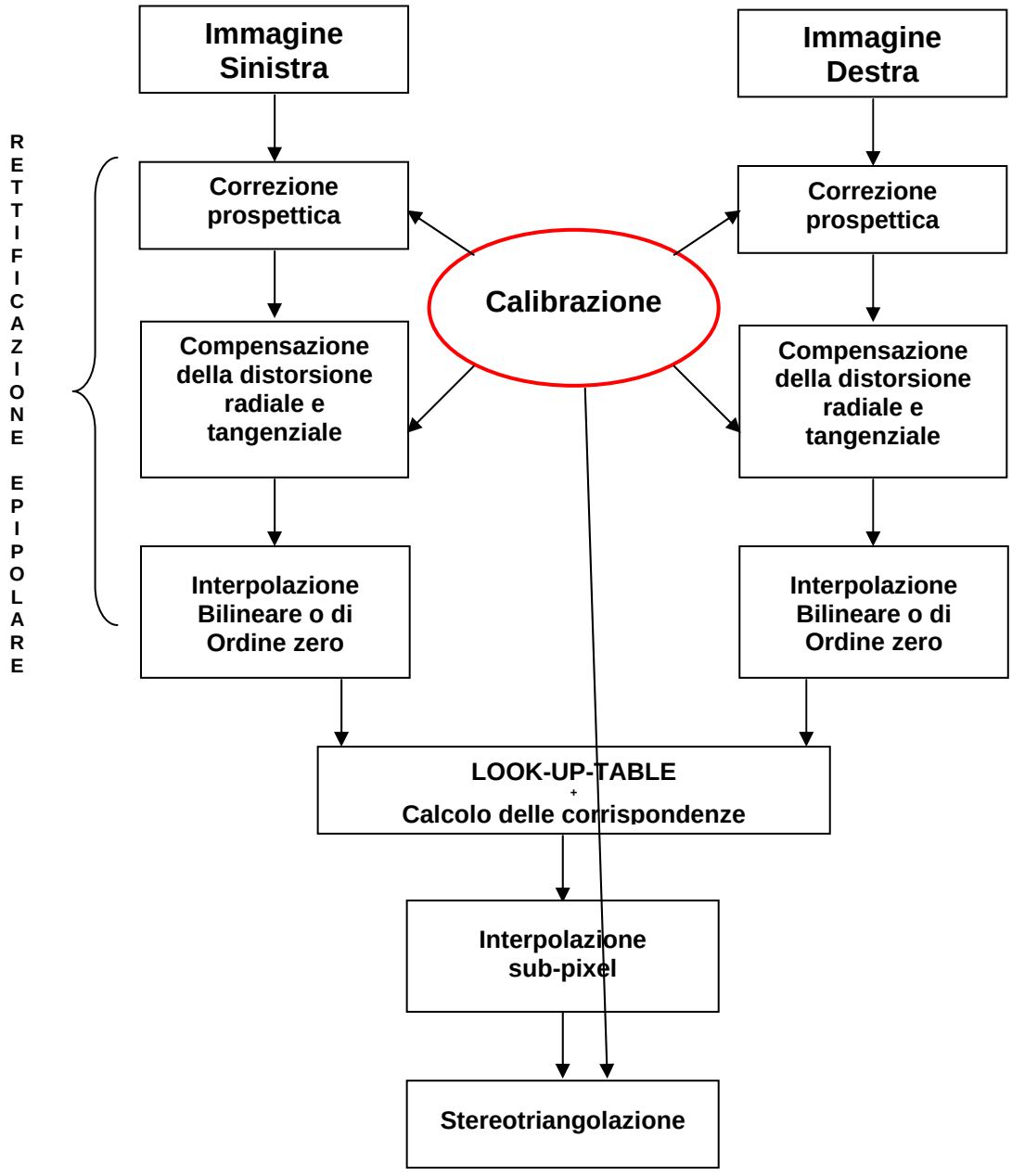



**Figura 3.24** Schema riassuntivo dei blocchi funzionali che descrivono l'architettura del sistema di visione stereoscopica.

# 3.7 Risultati sperimentali e collaborazione con L'ENEA

## 3.7.1 Test del software su hardware professionale

Questo lavoro di tesi è stato svolto anche in collaborazione con il centro ricerche dell'E.N.E.A., presso il Dipartimento di robotica, sede di Casaccia (Roma).

Il software di stereometria che ho sviluppato è stato infatti testato su un hardware differente dal mio per verificare se il grado di precisione del sistema nel calcolo delle distanze dipendesse dal software o dall'hardware.

Di seguito sono riportati i risultati del processo di rettificazione effettuato dal software.

L'hardware prestato dall'ENEA consiste di due telecamere con una risoluzione di 640x480 pixel.

La distanza tra le telecamere (baseline) vale 11,961 cm.


**Figura 3.25** Immagini originali non rettificate: si osservi l'effetto di bombatura della porta


**Figura 3.26** Immagini rettificate

### 3.7.2 Confronto delle prestazioni su hardware differente

Di seguito si riporta un grafico che confronta l'andamento dell'errore nel calcolo della distanza tra i due sistemi di visione stereoscopici utilizzati.
In rosa è rappresentato l'andamento dell'errore relativo al mio sistema di visione; in blu quello relativo al sistema professionale in uso presso l'ENEA.

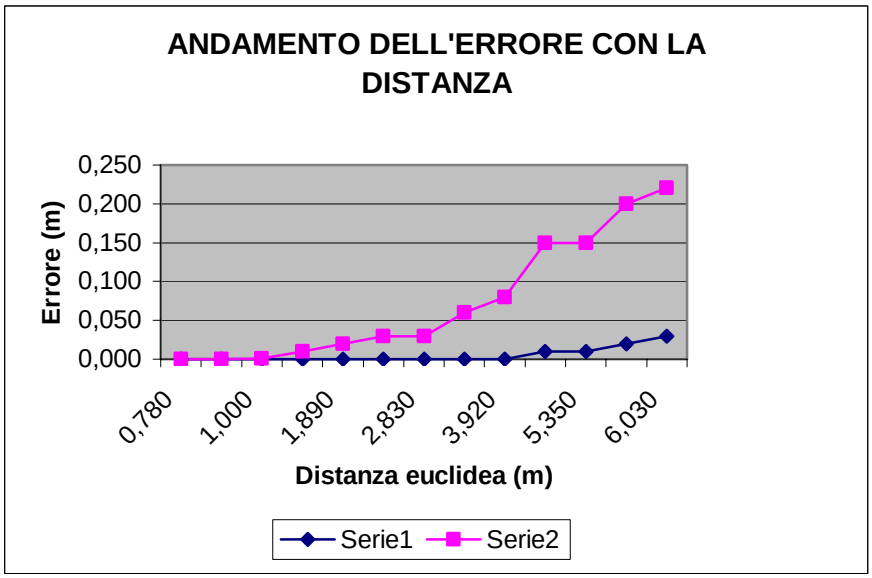



**Figura 3.27**

L'errore è stato calcolato confrontando la misura fornita dal software di stereometria con le misure calcolate direttamente tramite fettuccia metrica.
Si osservi, come era prevedibile, che l'errore aumenta con la distanza dalle telecamere.
Il sistema da me realizzato commette un errore di circa 1 mm su distanze dell'ordine di 1 m; 2 cm su distanze di 2 m; 20 cm intorno ai 6 m di distanza. Ciò è dovuto esclusivamente alla bassa risoluzione dei sensori: 352x288 pixel.
Il sistema professionale dell'ENEA, al contrario, possiede una risoluzione ben quattro volte maggiore ed è pertanto caratterizzato da errori molto più piccoli, che rasentano appena 3 cm su distanze dell'ordine di 6 m.

## 3.8 Guida alle funzioni implementate in C

### Parametri caratteristici della stereo-camera

Tutti i parametri intrinseci ed estrinseci che descrivono il modello matematico della stereo-camera costituiscono i campi della struttura **`stereo_camera`**.

Gli indici delle look-up-table che mappano le corrispondenze tra la coppia di immagini rettificate e quella originaria, nonché i coefficienti di interpolazione bilineare e di ordine zero, costituiscono i campi della struttura **`camera_index`**.

### Compensazione della distorsione radiale e tangenziale

La seguente funzione applica la distorsione radiale e tangenziale alle coordinate non distorte di un punto dell'immagine espresse in pixel

```
void distort(float *ud, float *vd, int index, float fc[2], float
             cc[2], float kc[5], RxMatrix *invTrf);
```

**`ud`** e **`vd`** sono le coordinate float distorte

**`index`** è l'indice delle coordinate intere del punto cui va applicata la distorsione

**`fc, cc, kc`** sono i parametri intrinseci della camera

**`invTrf`** è l'inversa della matrice di rettificazione

### Rettificazione epipolare

La seguente calcola gli indici di 9 look-up-table che governano la rettificazione secondo i due metodi di interpolazione visti. La stessa compensa la distorsione della lente richiamando la funzione **`distort`**.

```
void LUT_rectify(unsigned char **ind_1, unsigned char **ind_2,
                 unsigned char **ind_3, unsigned char **ind_4,
                 int **ind_0,
                 float *a1, float *a2, float *a3, float *a4,
                 int *data1, float fc[2], float cc[2], float kc[5],
                 RxMatrix *Trf);
```

**ind_1, ind_2, ind_3, ind_4** sono usati per l'interpolazione bilineare; sono gli indici delle quattro look-up-table in cui andranno memorizzati i puntatori alle coordinate dei quattro punti da interpolare.
**Ind_0** ha la stessa funzione dei precedenti ma è usato solo per l'interpolazione di ordine zero.
**a1, a2, a3, a4** sono gli indici delle quattro look-up-table in cui andranno memorizzati i quattro coefficienti dell'interpolazione bilineare.
In totale si hanno 9 tabelle di look-up di dimensione $width \cdot height = 352x288$ $[\text{celle}]^2$, dove `celle` può essere di tipo `int, char o float`.

La seguente funzione utilizza i parametri di rettificazione calcolati da **LUT_rectify** e restituisce l'immagine rettificata a colori. Mediante il parametro **order** consente all'utente di selezionare il metodo di interpolazione più conveniente.

```
void rectify(unsigned char **ind_1, unsigned char **ind_2, unsigned
             char **ind_3, unsigned char **ind_4, int **ind_0,
             float *a1, float *a2, float *a3, float *a4,
             int *data_rect1, int order);
```

**data_rect** è il puntatore alla nuova immagine a colori rettificata.
**Order** consente all'utente di selezionare il metodo di interpolazione più conveniente. Se order=0 viene applicata un'interpolazione di ordine zero, altrimenti un'interpolazione bilineare.

## Inizializzazione dei parametri e delle look-up-table

La seguente inizializza i parametri intrinseci ed estrinseci delle telecamere definiti nella struttura **stereo_camera**.

```
void init_stereo_camera(stereo_camera *stereo, float teta, float sf);
```

**teta** è l'angolo di beccheggio che intercorre tra il piano focale delle telecamere ed il pavimento. La sua utilità verrà mostrata nell'ambito dell'autolocalizzazione.
**sf** è il fattore di ingrandimento delle immagini rettificate. Può assumere valori compresi tra zero e infinito. La sua utilità verrà mostrata nell'ambito dell'autolocalizzazione.

La seguente inizializza i campi della struttura **camera_index**.

```
void init_camera_index(camera_index *ind1,camera_index *ind2,
                       stereo_camera *stereo, int *data1, int *data2);
```

## Stereo-triangolazione

Questa è la funzione che espleta la triangolazione stereo restituendo le coordinate 3D del punto a partire dalle coordinate delle sue proiezioni

```
void stereo_triangulation(RxMatrix *XL, float *euclidean_distance,
                          float pl[2], float pr[2], RxMatrix *A1, \
                          RxMatrix *R1, RxMatrix *T1, RxMatrix *A2,
                          RxMatrix *R2, RxMatrix *T2, RxMatrix *Trf1,
                          RxMatrix *Trf2);
```

**XL** è il vettore contenente le coordinate 3D del punto.

**Euclidean_distance** è la distanza euclidea del punto dall'origine del riferimento.

**pl** e **pr** sono le coordinate dei punti coniugati che costituiscono le proiezioni del punto 3D.

# Capitolo 4

# Inseguimento di un oggetto


## Abstract

In questo capitolo viene descritta l'applicazione degli algoritmi di riconoscimento e di localizzazione stereoscopica, sviluppati nei precedenti capitoli, all'inseguimento di un oggetto in moto da parte di un robot mobile, dotato del sistema di visione stereoscopica in questione.
Sono inoltre descritte quattro differenti strategie di inseguimento dell'oggetto, che nella fattispecie è costituito da una palla.


## 4.0 Introduzione

**Figura 4.1**

L'inseguimento di oggetti tridimensionali in ambienti del mondo reale è una delle sfide più ambite nei problemi di visione computazionale. Noto sotto il nome di "3D object visual tracking", esso consiste in un sistema di visione, fisso o mobile, che sia in grado di tracciare il

moto compiuto da oggetti reali presenti nello spazio di visibilità del sistema e reagire ad eventuali cambiamenti di direzione nel moto relativo camera-oggetto, inseguendo quest'ultimo anche quando esso scompaia dal campo di visibilità delle telecamere.
L'object tracking coinvolge entrambi i problemi di processazione spaziale e temporale delle immagini, allo scopo di individuare tutti i possibili cambiamenti tra due frame successivi che inducano il passaggio di un corpo in movimento. La maggior parte degli algoritmi di rilevazione di oggetti in moto sono basati su metodi di stima o rilevazione delle differenze tra frame acquisiti in istanti diversi (motion detection & motion estimation) mediante tecniche di optical flow, basate sull'estrazione dei vettori di moto (vedi MPEG Video).
Il compito svolto in questo lavoro di tesi, tuttavia, non si esaurisce nella sola rilevazione del moto di un generico corpo ma nel riconoscimento di un oggetto specifico, nella fattispecie una palla, e nel suo inseguimento da parte di un robot mobile, dotato del sistema di visione stereoscopica in questione.

I metodi tradizionali di visual tracking sono di solito fondati sull'assunto che gli intervalli di tempo tra due frame successivi siano abbastanza brevi da poter considerare il moto di un oggetto altamente prevedibile. Sfortunatamente quest'ipotesi non è verificata nella maggior parte delle applicazioni. Per esempio, nel caso del robot mobile utilizzato, si richiede che esso collezioni costantemente, e in tempo reale, i dati trasmessi dai sensori (sonar, encoder e visione), sulla base dei quali stimare la sua posizione e decidere eventuali strategie di inseguimento. Di conseguenza non sono garantite prestazioni ad elevato frame-rate a causa delle limitazioni connesse all'acquisizione dei dati sensoriali, alla pianificazione della traiettoria da seguire ed ai ritardi meccanici degli attuatori e dei sensori. Inoltre, anche il tempo di calcolo per l'elaborazione di ogni singolo frame gioca un ruolo fondamentale: infatti, per il riconoscimento della palla, si richiede al sistema di applicare gli algoritmi di circle-detection descritti nel capitolo 2, i quali, come si è visto, richiedono un time processing delle immagini non trascurabile, che raddoppia nel caso di visione stereoscopica.
Per i problemi suddetti, nel corso di questo lavoro ho implementato quattro diverse strategie di inseguimento di una palla, ciascuna delle quali differisce dall'altra per il tipo di controllo utilizzato e per la migliore attitudine a scartare i falsi accoppiamenti che possono indurlo ad inseguire qualcosa anche quando la palla non sia presente.

Al fine di evidenziare i vantaggi apportati dall'uso della visione stereoscopica, la prima strategia fa uso di una sola telecamera (visione monoscopica); tramite questa si controlla il moto di traslazione e rotazione del robot in modo tale che l'oggetto appaia sempre al centro del campo visivo della telecamera.

A causa della totale assenza di profondità, il robot è indotto saltuariamente ad oscillare intorno alla posizione di equilibrio, costituita dalla palla simmetrica rispetto al centro dell'immagine.
Tuttavia, nonostante la visione monoculare faccia perdere il senso della profondità e quindi della distanza che separa l'oggetto dalla telecamera, questa prima modalità di approccio è comunque anche la più flessibile, perché caratterizzata da velocità di processazione dell'immagine chiaramente doppie rispetto a quella stereoscopica, con un frame-rate che raggiunge i 12 Hz rispetto ai 15 Hz massimi con cui la webcam è in grado di acquisire immagini. Vedremo infatti che in questo modo il robot è in grado di reagire prontamente al passaggio di una palla di fronte ad esso, anche quando l'oggetto viene allontanato repentinamente. Tuttavia l'approccio monoculare è anche quello più incline a falsi accoppiamenti poiché, come abbiamo visto, l'algoritmo di circle detection funziona eccellentemente in presenza di cerchi, ma restituisce comunque un valore "true" (in corrispondenza di un picco massimo nello spazio della trasformata) anche quando non sia presente alcun cerchio nell'immagine. Per tale motivo ho implementato degli algoritmi di descrizione e classificazione (descriptors & classifiers) che analizzano il risultato della circle detection mediante analisi della distribuzione statistica dei dati forniti dalla trasformata Pixel to Pixel. Questi algoritmi confrontano i momenti statistici del primo e del secondo ordine dei risultati della circle detection con quelli relativi alla presenza effettiva di un cerchio. Essi, tuttavia, non hanno fornito risultati soddisfacenti nel caso della palla, ma si sono verificati validi nel riconoscimento di landmark (capitolo 5).

La seconda e la terza strategia di inseguimento vedono finalmente l'applicazione della visione stereoscopica, unita alla particolare attitudine del sistema al calcolo preciso delle distanze. In questo caso, infatti, si recupera il senso della profondità grazie alla disparità che esiste tra la posizione della palla nelle immagini destra e sinistra. Successivamente, mediante la stereo-triangolazione, descritta nel capitolo 3, vengono stimati la distanza e l'angolo che separano l'oggetto dal centro delle telecamere, proiettati sul piano del pavimento.

Grazie alla stereoscopia vedremo che è possibile rimuovere molti dei falsi accoppiamenti che affliggevano la visione monoscopica. In virtù della rettificazione epipolare, discussa al capitolo 3, sappiamo infatti che coppie di punti coniugati, che sono proiezione di uno stesso punto nello spazio 3D, devono giacere sulla stessa retta epipolare orizzontale, ovvero devono avere la stessa coordinata verticale: una fatto, questo, che permette di escludere tutte le coppie di punti in cui la differenza tra le ordinate ecceda un certo intervallo di confidenza.
Anche la stereo-triangolazione contribuisce ad escludere i falsi accoppiamenti. Poiché la palla rotola sul pavimento, la sua quota tenderà a rimanere costante rispetto al sistema di riferimento della stereo-camera, a meno di un'incertezza nel calcolo della posizione a grandi distanze.
Come visto nel capitolo 2, allo scopo di aumentare la precisione della misura, che nella realtà sarebbe limitata alla risoluzione di un pixel, le coordinate del cerchio, rilevato in ciascuna

immagine, subiscono un processo di interpolazione sub-pixel. In questo modo, la stima della posizione della palla rispetto al robot raggiunge livelli di accuratezza decisamente elevati (da $\approx 1\,mm$ su distanze di 1 metro a $\approx 2\,cm$ per distanze dell'ordine di 2 metri).
Per cui, qualora si verifichino altri falsi accoppiamenti, nonostante i vincoli imposti dalla rettificazione e dalla quota verticale, vedremo che è possibile considerare spuri quei punti la cui distanza varia repentinamente da un fotogramma all'altro.

Nel caso della stereoscopia il frame rate non supera i 6Hz e sarà dunque necessario compensare la distanza della palla calcolata al termine dell'elaborazione delle immagini con lo spazio percorso dal robot nel tempo in cui era impegnato ad elaborare le immagini.

Le due strategie di inseguimento basate sulla visione stereoscopica differiscono per il tipo di controllo adottato.
Nel primo caso è stato utilizzato un regolatore PID di velocità, ad azione proporzionale-integrale, che tenta di minimizzare l'errore in distanza e quello angolare.
Nell'altro caso si è utilizzata, invece, la pianificazione della traiettoria. Quest'ultimo però richiede di ricalcolare costantemente la traiettoria, risultando in ingiustificato dispendio di tempo.

Verrà mostrato che la strategia di inseguimento che assicura le migliori prestazioni in termini di efficienza computazionale e resistenza ai falsi accoppiamenti è quella stereoscopica con regolazione PID di velocità.
Inoltre, per assicurare che il sistema di visione invii al robot informazioni sulla posizione della palla in tempo reale, ovvero ad intervalli di tempo costanti, sono stati creati due processi multithread, che gestiscono indipendentemente l'odometria (insieme ai comandi da impartire a robot) e l'elaborazione delle immagini stereoscopiche.
In conclusione, gli aspetti innovativi del sistema proposto sono:

1. grazie al riconoscimento della palla basato sulla forma (shape based) piuttosto che sul colore, il sistema ha dimostrato di inseguire la palla anche durante la sovrapposizione di altri corpi o persone che intercorrano tra robot e oggetto.
2. il sistema è in grado di resistere alle frequenti situazioni di occlusione della palla, che comportano l'occultamento di una parte o di un'intera porzione di essa, soprattutto quando si trovi ai limiti del campo visivo delle telecamere
3. la stereoscopia ha inoltre offerto una buona resistenza ai molteplici casi di falso accoppiamento, sfruttando i vincoli sulla costanza della quota, sulla geometria epipolare dei punti coniugati, sulla precisione del calcolo della distanza
4. è garantito inoltre il controllo del robot in tempo reale, grazie alla gestione indipendente di processi multithread separati, per l'image processing e per l'odometria
5. tutte le funzioni sono state implementate con un basso costo computazionale

Infine, nell'ultimo paragrafo di questo capitolo, sarà descritta una quarta ed innovativa strategia di inseguimento e di ricerca della palla, che è in grado di rimuovere fino al 90% dei falsi accoppiamenti presenti nella visione monoscopica sfruttando la correlazione temporale tra frame successivi, adottata nell'algoritmo di ***Motion Estimation dell'MPEG Video***. Per limiti di tempo essa è stata applicata solamente alla visione monoculare.

In quest'ultimo approccio una opportuna fase di training della palla precede al normale inseguimento. Durante il training l'algoritmo memorizza alcuni blocchi di pixel dell'immagine situati in prossimità dei punti in cui è stata localizzata la palla. Una volta "appresa" la conformazione dell'oggetto inizia il normale inseguimento, verificando, per ciascun frame, la "somiglianza" con l'immagine appresa.

Per ciascun fotogramma viene presa in considerazione una finestra di pixel centrata nel punto in cui è stata indicata la presenza di un cerchio; tale finestra viene confrontata con quella memorizzata al frame precedente tramite correlazione. Se l'esito di questo confronto restituisce un valore di "similarità" adeguato, allora la palla è stata individuata, altrimenti il robot ruota su se stesso, nel verso in cui è stata avvistato l'oggetto per l'ultima volta, e continua la ricerca.

In quest'ultima strategia viene conservata memoria delle ultime dieci posizioni della palla per rimuovere eventuali falsi accoppiamenti mediante un approccio convoluzionale.

## 4.1 Prima strategia: visione monoscopica

Prima di effettuare il controllo l'immagine viene divisa in quattro quadranti, come in figura 4.2:

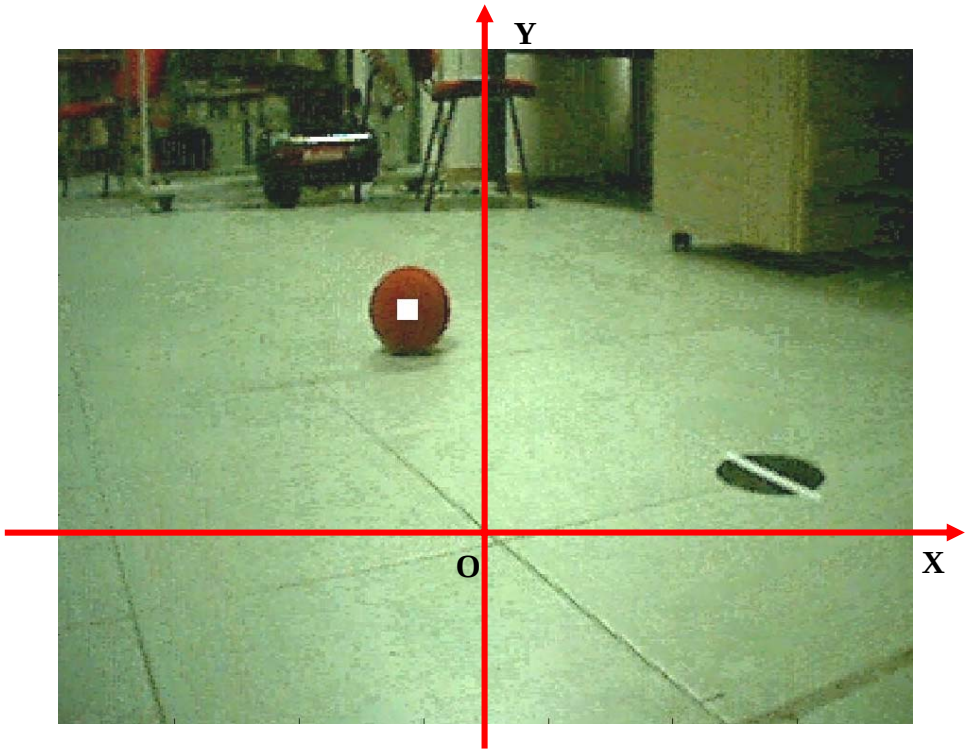



**Figura 4.2**

Una volta calcolate le coordinate del centro della palla, mediante uno degli algoritmi di circle detection già visti, queste vengono convertite nel nuovo riferimento cartesiano in figura 4.2.
In questo modo gli spostamenti longitudinali della palla si traducono in un aumento (la palla si allontana) o diminuzione (la palla si avvicina) della sua coordinata verticale; al contrario, gli spostamenti laterali possono essere codificati con un aumento o diminuzione della sua ascissa.

La strategia di inseguimento che è stata implementata controlla il moto di traslazione e rotazione del robot in modo tale che l'oggetto appaia sempre al centro del campo visivo della telecamera.
Ricordiamo che gli unici ingressi che possiamo imporre al robot per farlo muovere sono le velocità lineare $v$ ed angolare $o$, riferite al centro dell'asse che unisce le ruote. Tali valori di velocità non devono tuttavia essere troppo elevati, per non danneggiare gli attuatori, pertanto a valle del regolatore lineare è stata applicata una rampa a saturazione.
Ai fini del controllo ho adottato un semplice regolatore PID ad azione puramente proporzionale:

$$\begin{cases} v = k_v \cdot y \\ o = -k_o \cdot x \end{cases} \tag{4.1}$$

dove per $k_v$, $k_o$ ho scelto i seguenti valori: $k_v = 0.003 \ \ m/s$, $k_o = 0.003 \ \ rad/s$.

I valori di velocità imposti vengono poi a loro volta saturati tra [-0.2, 0.2].

Il diagramma seguente riassume la strategia di controllo:

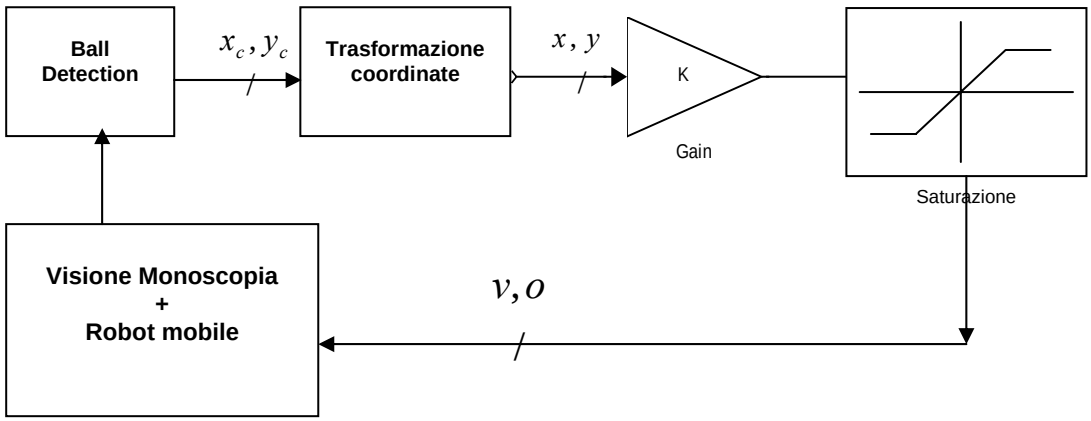



**Figura 4.3**

## 4.2 Seconda strategia: visione stereoscopica con regolazione PID

Questa seconda strategia di inseguimento vede finalmente l'applicazione della visione stereoscopica, unita alla particolare attitudine del sistema al calcolo preciso delle distanze. In questo caso, infatti, si recupera il senso della profondità grazie alla disparità che esiste tra la posizione della palla nelle immagini destra e sinistra.

In questa versione, l'algoritmo di ball detection viene applicato separatamente ad entrambe le immagini, ma solo dopo la rettificazione; una volta note le coordinate del centro della palla nei due frame, si applica la stereotriangolazione, che ricostruisce la posizione 3D dell'oggetto nel riferimento delle telecamere. Dopo una trasformazione di coordinate che permette di passare dal sistema stereo-camera a quello del robot, le coordinate della palla sono proiettate sul piano orizzontale, in un sistema di coordinate polari $(\rho, \vartheta)$ che ha l'origine nel centro del robot e l'asse x allineato nella direzione frontale del robot stesso; $\rho, \vartheta$ individuano la distanza e l'angolo che separano l'oggetto dal centro del robot.

### 4.2.1 Individuazione del target

Come anticipato nell'introduzione, è possibile sfruttare il vincolo epipolare per escludere possibili *false positives*. Dopo la rettificazione delle due immagini, che abbiamo visto essere necessaria anche per la compensazione della distorsione radiale e tangenziale, i centri delle due sfere, eventualmente individuate, devono giacere sulla stessa retta epipolare: se ciò non accade, esiste un'alta probabilità che non sia stato riconosciuto alcun oggetto simile ad una palla.

Da ciò segue che, una volta ottenute le coordinate di possibili cerchi nei due frame, si confrontino le rispettive ordinate. Tuttavia, poiché tali coordinate subiscono un processo di interpolazione sub-pixel per aumentare la risoluzione delle telecamere, è praticamente impossibile che essi abbiano la stessa ordinata. A tal fine nell'algoritmo viene richiesto che la differenza tra le coordinate verticali rientri in un intervallo di confidenza abbastanza piccolo: 15 pixel. Per cui:

$$|y_{sx} - y_{dx}| < 15 \tag{4.2}$$

Se la 4.2 restituisce un valore "true", la prossima condizione da valutare consiste nel verificare che l'eventuale oggetto, riconosciuto come una palla, giaccia sul pavimento. A tal fine è sufficiente verificare che la quota verticale rientri in un ben determinato range.

Lo schema a blocchi nella pagina seguente riassume i passi enunciati finora.

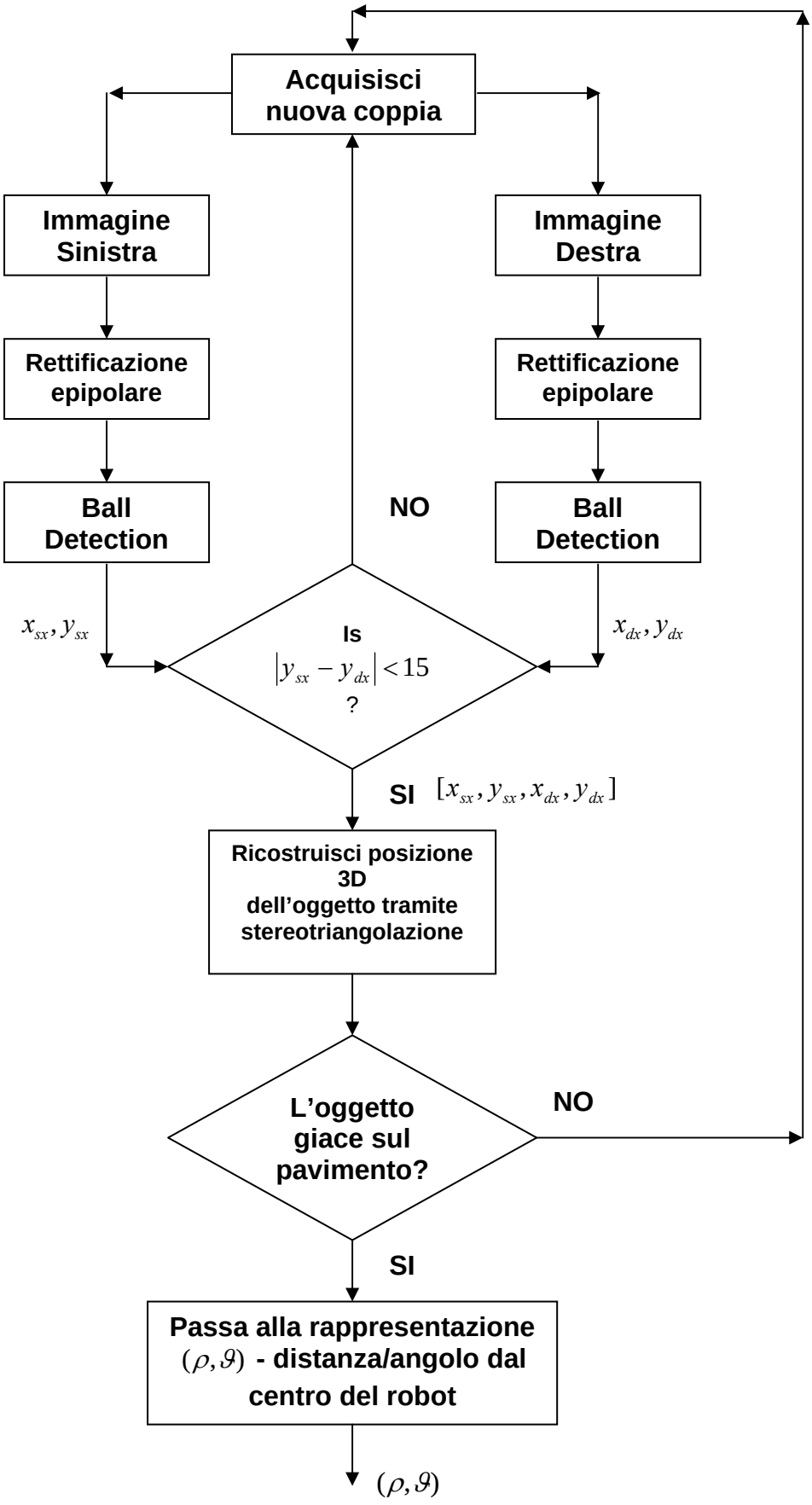



**Figura 4.4** Schema a blocchi che riassume la sequenza di elaborazione delle immagini per il calcolo della posizione relativa della palla rispetto al robot

## 4.2.2 Controllo di velocità

Nota a questo punto la posizione della palla rispetto al robot con un errore relativo dell'1% (capitolo 3), rimane da determinare la strategia di inseguimento .

La regolazione di velocità lineare ed angolare può essere effettuata allo stesso modo della versione monoscopica, utilizzando i valori $\rho$ e $\vartheta$ in luogo di $y$ e $x$ rispettivamente.

Tuttavia, per poter sfruttare pienamente dei vantaggi di una visione stereoscopica, ho scelto di implementare un regolatore PID completo, con azione proporzionale, integrale e derivativa della velocità angolare.

Per le velocità lineari ho scelto invece una semplice azione proporzionale. Infatti, affinchè il robot raggiunga la palla occorre correggere opportunamente l'errore angolare che intercorre tra l'asse del robot e l'oggetto; mentre la velocità lineare ha effetto sul tempo impiegato a raggiungerla. Una volta individuate i parametri caratteristici del controllo, questo è stato applicato alle velocità lineare ed angolare, in maniera indipendente l'una dall'altra.

### 4.2.2.1 Controllo della velocità angolare

Un generico regolatore PID a tempo continuo nella forma standard è caratterizzato dalla seguente espressione nel dominio di Laplace:

$$R_{PID}(s) = K_P + \frac{K_I}{s} + K_D \cdot s \tag{4.3}$$

Dove le costanti di proporzionalità evidenziano l'effetto separato dell'azione proporzionale, integrale e derivativa.

Per limitarne la banda, altrimenti infinita a causa dell'azione derivativa, si introduce un polo lontano, e il regolatore assume la seguente forma:

$$R_{PID}(s) = K_P \left( 1 + \frac{1}{T_i \cdot s} + \frac{T_d \cdot s}{1 + \frac{T_d \cdot s}{N}} \right) \tag{4.4}$$

con $T_i = \frac{K_P}{K_I}$, $T_D = \frac{K_D}{K_P}$ .

Per quanto riguarda N, si scelgono solitamente valori supeiori a 10.

È possibile ricavare un equivalente a tempo discreto applicando la trasformazione di Tustin.
A tal fine definiamo prima le seguenti costanti:

$$C_i = \frac{K_I \cdot T_c}{2};\quad C_d = \frac{K_D \cdot 2}{T_c};$$
$$a = 1 + 2\frac{K_d}{K_P \cdot N \cdot T_c};\quad b = 1 - 2\frac{K_d}{K_P \cdot N \cdot T_c};\quad \rho = \frac{b}{a} \tag{4.5}$$

$T_c$ è il tempo di campionamento.

Da queste si ricava l'equazione alle differenze finite equivalente alla (4.4).

$$u[k] = (1-\rho)\cdot u[k-1] + \rho \cdot u[k-2] + \tag{4.6}$$
$$+\left(1 + C_i + \frac{C_d}{a}\right)\cdot err[k] + \left(\rho - 1 + C_i(1+\rho) - 2\frac{C_d}{a}\right)\cdot err[k-1] - \left(\frac{C_d}{a} - \rho\right)\cdot err[k-2]$$

La 4.6 descrive la variabile di controllo all'uscita del regolatore (u[k]) all'istante k; come si vede questa viene a dipendere dai valori di uscita assunti negli istanti [k-1], [k-2], e dai valori in ingresso err[k], err[k-1], err[k-2], che codificano la differenza tra il valore desiderato e quello in uscita (ad esempio l aveloctà lineare o angolare); pertanto è necessario tenere memoria delle ultime tre posizioni relative palla-robot.
Poiché, come accennato precedentemente, l'errore angolare $\vartheta$ sotto cui viene osservata la palla dal robot influenza l'inseguimemnto in misura maggiore rispetto alla distanza che li separa, il regolatore PID è stato applicato per controllare esclusivamente la velocità angolare $o$ del robot. Da ciò segue la seguente sostituzione:

$$u[i] = o[i];$$
$$err[i] = \vartheta[i]; \tag{4.7}$$

#### 4.2.2.2 Controllo della velocità lineare

Per la velocità lineare ho utilizzato un semplice regolatore proporzionale.
Per far sì che il robot si arresti ad una distanza di 0.5 metri dalla palla (se questa è ferma) ho applicato la seguente legge, che diventa non lineare per distanze inferiori a 0.5 m:

$$v = \begin{cases} v = -k_v \cdot (\rho - 0.5) & \text{se } \rho > 0.5 \\ 0 & \text{altrimenti} \end{cases} \tag{4.8}$$

#### 4.2.2.3 Algoritmo finale di controllo

Grazie alla gestione di processi indipendenti multithread in ambiente Linux, è stato possibile controllare in maniera del tutto indipendente l'elaborazione continua di immagini e la regolazione delle velocità del robot.

Mentre il sistema di acquisizione cattura ed elabora le coppie di immagini fornite dalla stereocamera, un secondo processo, indipendente da questo, lavora sui valori di coordinate della palla fornite dall'ultima acquisizione, e impartisce le relative variabili di controllo del robot

Nel seguito vengono riassunti i passi fondamentali di un ciclo di controllo, che consistono nel calcolo delle variabili di controllo e nell'aggiornamento delle variabili di stato e uscita:

1. La visione stereoscopica fornisce il valore attuale di $\rho$ e $\vartheta$
2. Tramite la 4.6 calcola la velocità angolare di controllo:
$$o = T_1 \cdot o_{k-1} + T_2 \cdot o_{k-2} + K_1 \cdot \vartheta + K_2 \cdot \vartheta_{k-1} + K_3 \cdot \vartheta_{k-2} \tag{4.9}$$
3. Aggiorna i valori dello stato e delle sucite
$$\begin{aligned} o_{k-2} &= o_{k-1}; \\ o_{k-1} &= o_k; \\ \vartheta_{k-2} &= \vartheta_{k-1}; \\ \vartheta_{k-1} &= \vartheta_k; \end{aligned} \tag{4.10}$$
4. Mediante la 4.8 calcola la velocità lineare di controllo:
$$v = \begin{cases} v = -k_v \cdot (\rho - 0.5) & \text{se } \rho > 0.5 \\ 0 & \text{altrimenti} \end{cases} \tag{4.11}$$
5. Applica al robot i valori di velocità o, v appena calcolati

Nelle fasi dell'algoritmo ho omesso la saturazione delle velocità, che devono essere limitate al fine di non danneggiare gli attuatori.

Nella pagina che segue è illustrato l'intero processo di acquisizione, elaborazione e controllo che realizzano questa strategia di inseguimento.

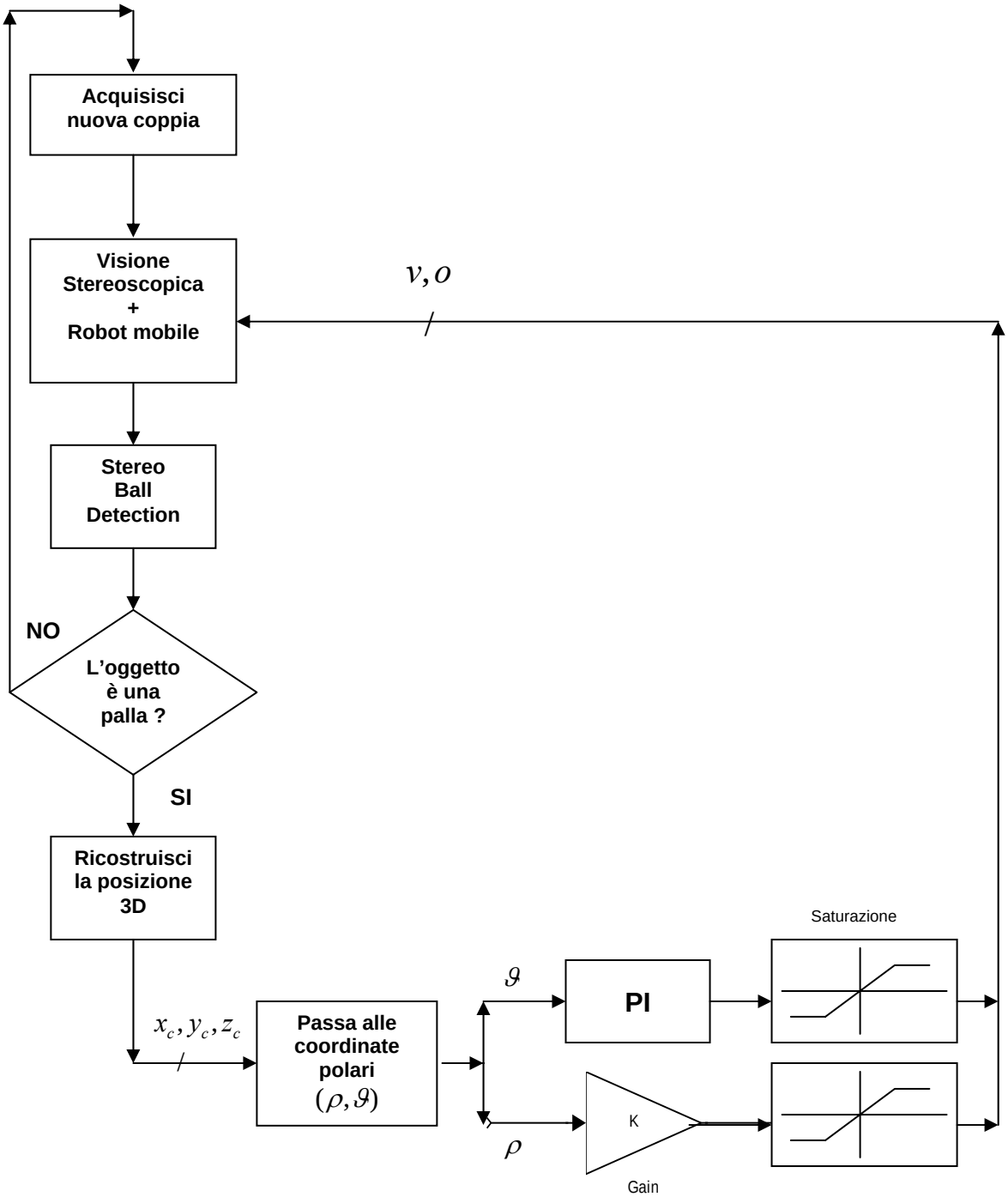



**Figura 4.5** Lo schema illustra l'intero processo di inseguimento della palla: dapprima sono acquisite le immagini; queste vengono elaborate; le coordinate sono sfruttate per calcolare le variabili di controllo.

#### 4.2.2.4 Taratura del PID

Una volta implementate le leggi di controllo del robot, rimangono da identificare i seguenti parametri di controllo:

- $K_P$ = guadagno dell'azione proporzionale
- $K_I$ = guadagno dell'azione integrale
- $K_D$ = guadagno dell'azione derivativa
- $k_v$ = guadagno di velocità lineare
- N = posizione del polo lontano

a partire dai quali possono essere calcolate le costanti che appaiono alla 4.9 e 4.10 grazie alle 4.5, 4.6.

Nonostante sia nota una procedura (Zigher – Nichols [44]) che consente di ricavare empiricamente i coefficienti $K_P, K_I, K_D$ del PID per un generico sistema, essa non è tuttavia applicabile nel caso della visione: tale metodologia richiede infatti di portare il robot ai limiti della stabilità e ricavare successivamente i parametri a partire dal periodo delle oscillaizioni del sistema.

Tuttavia sappiamo che un valore elevato di $K_P$ impone al robot di sterzare rapidamente per correggere il suo assetto in presenza di una palla, mentre valori troppo bassi risulterebbero in movimenti piuttosto lenti. Un valore elevato, inoltre, può indurre addirittura all'instabilità del robot (che oscilla periodicamente di fronte alla palla) o alla posssibilità che l'oggetto fuoriesca dal suo campo di visibilità.

La costante $K_I$ determina l'effetto dell'azione intergrale, che garantisce errore a gradino nullo a regime, e dunque fa sì che, se la palla è ferma, il robot si arresti prima o poi stabilmemnte di fronte ad essa.

La costante $K_D$ introduce un anticipo di fase del sistema, che si manifesta in una capacità del robot di reagire prontamente alle variazioni di posizione della palla. Tuttavia, a causa del ritardo con cui sono disponibili i risultati dell'elaborazione di immagini, esso porta immediatamente instabilità ed è stato posto dunque pari a 0.

Dopo numerose prove sperimentali applicate direttamente al robot in questione, sono risultati validi i seguenti valori per le costanti:

| | |
|---|---|
| $K_P$ | 0.6 |
| $K_I$ | 0.07 |
| $K_D$ | 0.0 |
| $k_v$ | 0.5 |
| N | 10 |

In sintesi, la miglior soluzione di controllo per la velocità angolare è risultata essere un puro PI, ovvero un PID limitato all'azione proporzionale ed integrale.

## 4.3 Terza strategia: visione stereoscopica con pianificazione della traiettoria

Questa strategia, basata sempre sull'uso della visione stereoscopica, differisce dalla precedente solo nella tipologia di inseguimento, che viene applicata a partire dalla conoscenza delle coordinate della palla calcolate nella prima fase.
In questo caso, ogni volta che viene resa nota la posizione della palla nel sistema di riferimento del robot, viene richiesto di pianificare una traiettoria da seguire per raggiungere la palla.
Il tipo di traiettoria scelta è quella rettilinea, cui corrisponde il tragitto più breve da percorrere. Essa consiste nel suddividere lo spazio che separa il robot dalla palla in un insieme di punti allineati ed equispaziati, e nell'assegnare la velocità istantanea (questa volta solamente quella lineare) con cui deve transitare in ciascun punto.

Proprio per garantire la continuità delle velocità nei punti di discontinuità tra due traiettorie, ho imposto al robot di mantenere una velocità costante pari a v=0.12 m/s lungo tutto il percorso.
Questa strategia, al contrario della precedente, richiede di pianificare la traiettoria tutte le volte in cui venga individuata la palla, risultando in un inutile perdita di efficienza. Ciò causa inoltre delle continue situazioni di arresto che si verificano quando la nuova traiettoria calcolata differisce dalla precedente.
Per evitare di progettare il percorso ogni volta, ho richiesto al robot di ricostruire la sua posizione assoluta rispetto al punto iniziale, grazie all'odometria. Mediante semplici trasformazioni di coordinate, in questo modo risulta possibile risalire anche alla posizione corrente della palla nel sistema assoluto. Poiché nell'arco di due frame successivi la palla in generale si allontana di poco, ho limitato la pianificazione della traiettoria ai soli casi in cui la

posizione assoluta della palla risulti modificata di almeno 1 m, in linea d'aria, rispetto alla posizione precedente.

Per riassumere:

1. viene individuata dapprima la posizione della palla
2. viene pianificata la traiettoria di minima distanza congiungente i due punti
3. il robot corregge l'assetto angolare in modo da allinearsi nella stessa direzione della traiettoria
4. comincia a muoversi con velocità costante nella direzione rettilinea suddetta, confrontando con l'odometria che il tragitto che sta percorrendo coincida con quello pianificato
5. nel frattempo continua ad acquisire immagini
6. se la posizione assoluta della palla si è modificata rispetto alla precedente per più di un metro in linea d'aria, pianifica una nuova traiettoria
7. il robot si arresta e riparte nella direzione della nuova traiettoria, inseguendo l'obiettivo

Nella pagina seguente è mostrato il diagramma di flusso che riassume la strategia di inseguimento descritta.

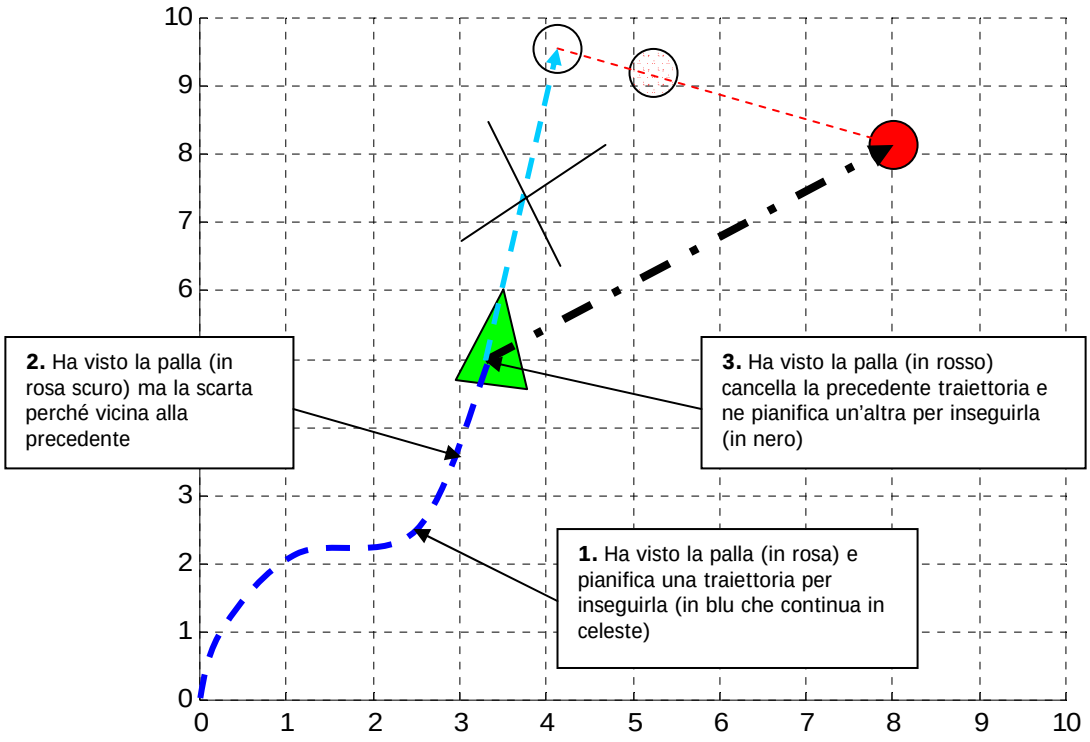



**Figura 4.6** La figura illustra un esempio di inseguimento. La linea blu tratteggiata mostra il percorso compiuto dal robot e ricostruito con l'odometria. Sulla base di esso calcola la posizione assoluta della palla e inizia la traiettoria **(1)**. Nel punto **(2)** vede di nuovo la palla ma la scarta perché vicina alla precedente. Nel punto **(3)** la palla è in posizione valida e può pianificare una nuova traiettoria da inseguire.

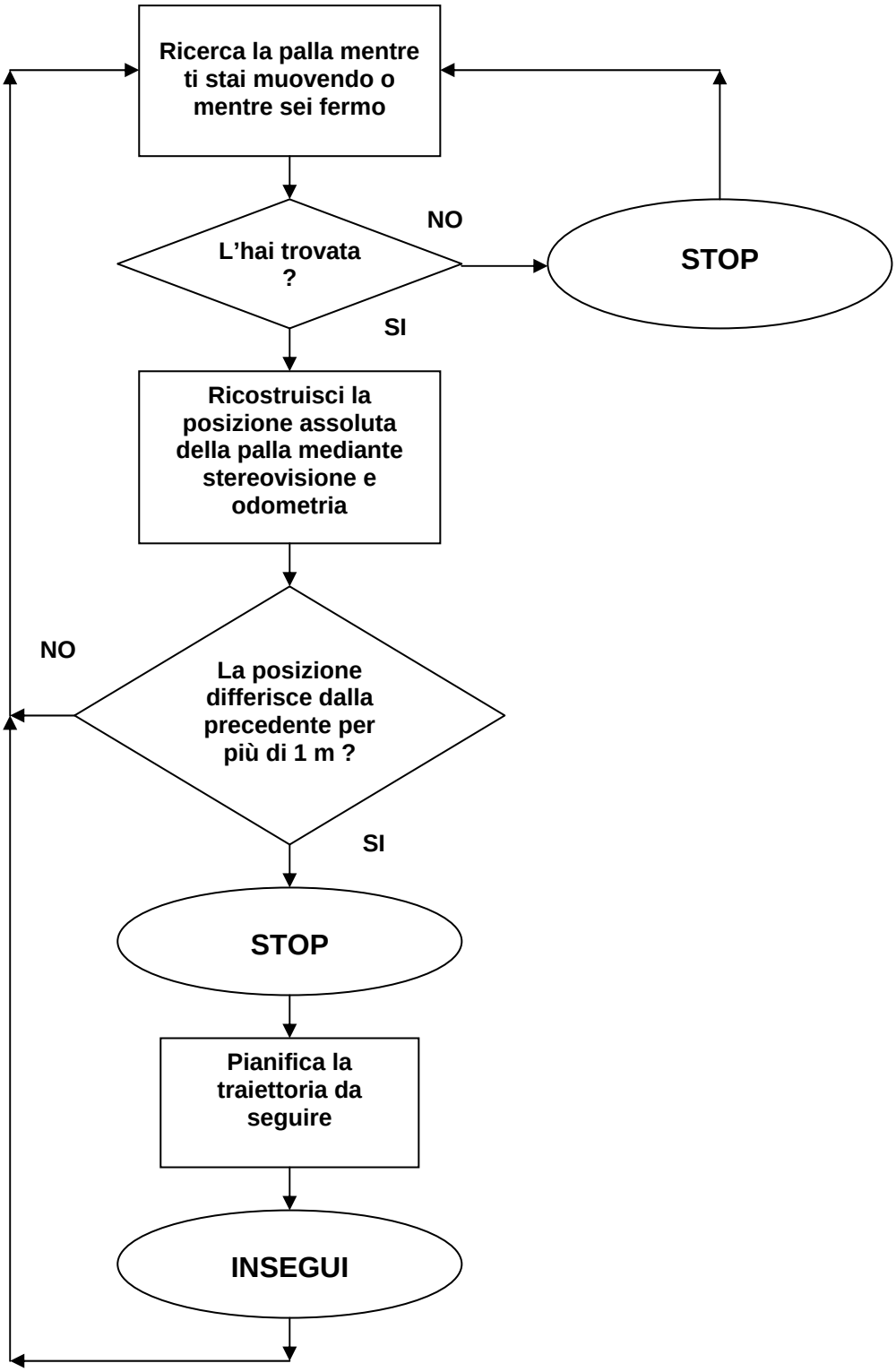


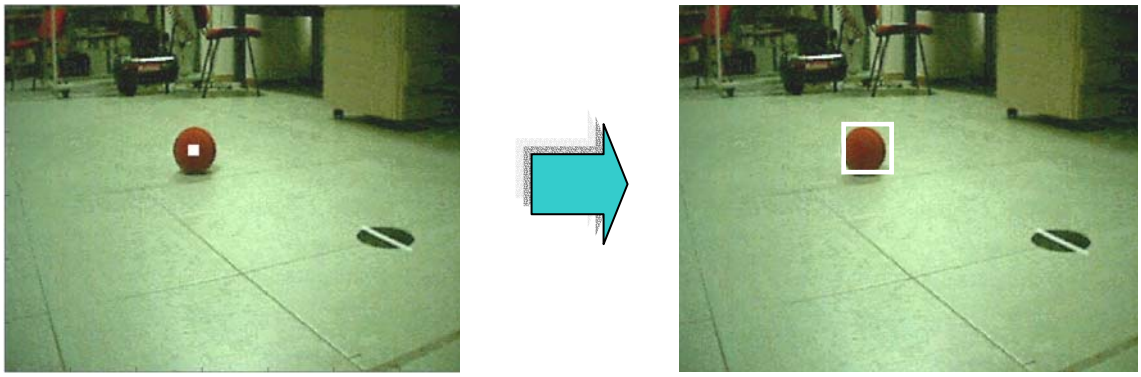

**Figura 4.7** Lo schema illustra l'intero processo di inseguimento della palla: dapprima sono acquisite le immagini; queste vengono elaborate; le coordinate sono sfruttate per pianificare la traiettoria di inseguimento.

## 4.4 Quarta strategia: ricerca e inseguimento della palla con MPEG Motion Estimation

In questo paragrafo viene descritta una quarta ed innovativa strategia di inseguimento e ricerca della palla, che è in grado di rimuovere fino al 90% dei falsi accoppiamenti presenti nella visione monoscopica sfruttando la correlazione temporale tra frame successivi, adottata nell'algoritmo di ***Motion Estimation dell'MPEG Video***. L'aspetto innovativo, rispetto alle precedenti versioni, risiede nel fatto che questo nuovo algoritmo permette al robot di effettuare una ricerca della palla quando questa scompare dal campo suo di visibilità.
Per limiti di tempo essa è stata applicata solamente alla visione monoculare.
In quest'ultimo approccio un'opportuna fase di training della palla precede al normale inseguimento. Durante il training l'algoritmo memorizza alcuni blocchi di pixel dell'immagine situati in prossimità dei punti in cui è stata localizzata la palla. Una volta "appresa" la conformazione dell'oggetto inizia il normale inseguimento, verificando, per ciascun frame, la "somiglianza" con l'immagine appresa.
Per ciascun fotogramma viene presa in considerazione una finestra di pixel centrata nel punto in cui è stata indicata la presenza di un cerchio; tale finestra viene confrontata con quella memorizzata al frame precedente tramite correlazione. Se l'esito di questo confronto restituisce un valore di "similarità" adeguato, allora la palla è stata individuata, altrimenti il robot ruota su se stesso, nel verso in cui è stata avvistato l'oggetto per l'ultima volta, e continua la ricerca.
In quest'ultima strategia viene conservata memoria delle ultime dieci posizioni della palla per rimuovere eventuali falsi accoppiamenti mediante un approccio convoluzionale.

### 4.4.1 Training

Il training è la fase che prelude all'inseguimento dell'oggetto.
In questa fase il robot resta fermo nella sua posizione di origine ed acquisisce immagini della scena. Ciascuna immagine viene sottoposta alla circle-detection per localizzare il centro di eventuali cerchi. Come abbiamo visto al capitolo 2, la funzione restituisce una coppia di coordinate anche quando non sia presente alcun cerchio (palla) nell'immagine. Tuttavia, In questa fase si assume che anche la palla sia immobile di fronte al robot, per cui, dopo alcuni istanti (20 frame) necessari alla stabilizzazione automatica del contrasto, la palla viene perfettamente individuata e marcata. Contemporaneamente viene memorizzata una regione quadrata di pixel centrata nel centro della palla, di dimensioni $(2k+1)\cdot(2k+1) = 21x21 pixel^2$ (vedremo che sono più che sufficienti a verificare l'attendibilità della palla).



Nella fase di training si individua dapprima la palla (marcata in figura)

Successivamente si memorizza un blocco di pixel appartenenti alla regione quadrata

**Figura 4.8**

### 4.4.2 Ricerca con Block Based Motion Estimation

Superata la fase di training, abbiamo una "copia" di come appariva la palla nell'ultimo frame in cui era stata individuata.

Da questo punto in poi ciascun fotogramma viene sottoposto ad un duplice controllo: quello di circle detection e quello di Motion Estimation.

Supponiamo che la circle detection abbia restituito la seguente coppia di coordinate $(x_c, y_c)$, la funzione di Motion estimation confronta i pixel della "copia" $I_1(x, y)$ immagazzinata in memoria (relativa all'ultimo frame in cui era stata trovata una palla) con i pixel contenuti in una finestra di pari dimensioni $I_2(x, y)$ centrata in $(x_c, y_c)$ del corrente frame. Il confronto è effettuato sulle tre componenti cromatiche RGB.

Vi sono vari criteri di somiglianza tra finestre; uno dei più comuni è la cosiddetta SSD (*Sum of Squared Differences*):

$$SSD = \sum_{i=1}^{2k+1}\sum_{j=1}^{2k+1}\left[I_{1RED}(i,j) - I_{2RED}(i,j)\right]^2 + \left[I_{1GREEN}(i,j) - I_{2GREEN}(i,j)\right]^2 + \left[I_{1BLUE}(i,j) - I_{2BLUE}(i,j)\right]^2 \quad (4.12)$$

se il confronto effettuato con la SSD restituisce un valore inferiore a 200.000 la presenza di una palla viene giudicata attendibile, e la "copia" $I_1(x, y)$ viene rimpiazzata da $I_2(x, y)$.

Se, al contrario, il confronto dà esito negativo, occorre impartire al robot un comando di ricerca che lo faccia girare su stesso nel verso in cui era stata avvistata la palla per l'ultima volta. A tale scopo è necessario registrare in memoria una traccia dell'ultima posizione.

N.B. A questo punto è bene osservare che il confronto SSD è molto sensibile all'intensità media di colore di ciascun frame, pertanto, se la palla si allontana troppo dal robot, oppure scompare

dal suo campo di visibilità per riapparire più lontana o vicina è facile che il confronto SSD dia esito negativo e la palla non sia individuata!

In queste situazioni sarebbe opportuno ripetere il training ed acquisire un nuovo "simulacro" della palla. Ma sappiamo che la fase di training funziona bene se non c'è alcun moto relativo tra palla e robot.

Per ovviare a questo inconveniente ho studiato un approccio di ricerca statistica: anziché registrare l'ultima posizione della palla, sono memorizzate le ultime $n$ coordinate $(x_i, y_i)$ restituite dalla circle detection dall'esame degli ultimi $n$ frame. In questo modo dispongo di un maggior numero di campioni su cui poter effettuare una ricerca statistica.

Tale ricerca consiste nel calcolo del baricentro della distribuzione e del valore di media distanza da esso. Se non c'è moto relativo tra palla e robot, la distanza media degli n punti dal baricentro è teoricamente nulla; nella pratica, a causa del rumore, si attesta intorno ad un valore di 0.1-0.5 pixel.

Se invece non è presente alcuna palla la circle detection restituisce dei valori di coordinate praticamente casuali, con una distribuzione ben lontana da quella descritta. Per cui risulta facile isolare la presenza della palla sulla base dell'analisi degli scostamenti medi dal baricentro.

Ma se la palla è in moto ciò non è più vero!

Tuttavia questo vincolo può facilmente essere arginato considerando che, poiché la palla rotola sul pavimento, il moto che viene proiettato dalla telecamera nell'immagine avviene con buona approssimazione su rette orizzontali: per cui ai fini della ricerca statistica possono essere considerati i spostamenti verticali.

L'algoritmo, infatti, prende in esame le ordinate degli $n$ punti e ne il calcola valor medio della distribuzione e la deviazione standard; se questa appartiene ad in intervallo adeguato (nella pratica 2 pixel) viene memorizzata la finestra relativa all'ultima coppia di coordinate individuate, altrimenti il robot prosegue nella ricerca ruotando continuamente si se stesso.

Nella pratica ho assunto $n$=10.

La strategia di inseguimento coincide con quella monoscopica descritta al paragrafo 4.1

Nella pagina seguente è illustrato il diagramma di flusso che descrive l'algoritmo di ricerca e inseguimento.

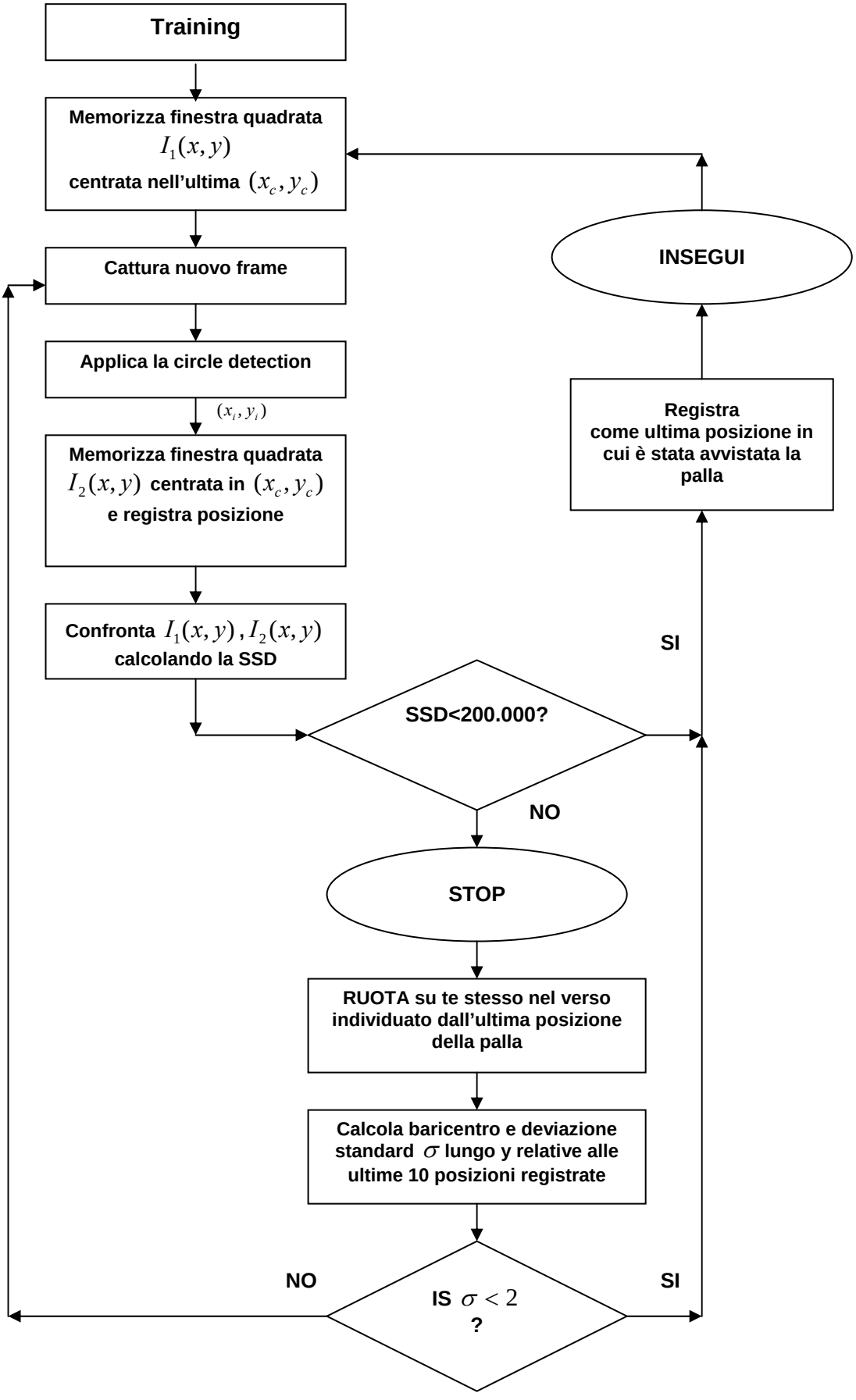



**Figura 4.9** Diagramma di flusso che riassume la strategia di ricerca e inseguimento

## 4.5 Conclusioni e risultati sperimentali

**Limiti della visione monoscopica**

Nella visione monoscopica (4.2), nonostante l'esiguo valore delle costanti di guadagno

$$\begin{cases} k_v = 0.003 \ m/s \\ k_o = 0.003 \ m/s \end{cases} \tag{4.13}$$

il robot risulta spesso instabile (a seconda della posizione relativa della palla) manifestando delle oscillazioni pseudoperiodiche intorno al suo asse di rotazione.
Nonostante la visione monoculare faccia perdere il senso della profondità e quindi della distanza che separa l'oggetto dalla telecamera, questa prima modalità di approccio è comunque la più flessibile, perché caratterizzata da velocità di processazione dell'immagine chiaramente doppie rispetto a quella stereoscopica, con un frame-rate che raggiunge i 12 Hz rispetto ai 15 Hz massimi con cui la webcam è in grado di acquisire immagini. I limiti dipendono ovviamente dall'hardware a disposizione, che nel mio caso è PC portatile con processore Intel Pentium 4 1,7 GHz.
In questo modo il robot dimostra di reagire prontamente al passaggio di una palla di fronte ad esso, anche quando l'oggetto venga allontanato repentinamente.
Tuttavia l'approccio monoculare è anche quello più incline a falsi accoppiamenti poiché, come abbiamo visto, l'algoritmo di circle detection funziona eccellentemente in presenza di cerchi, ma restituisce comunque una coppia di coordinate anche in assenza della palla.

**Limiti della visione stereoscopica con pianificazione della traiettoria**

Il caso di visione stereoscopica con pianificazione della traiettoria richiede di ricalcolare un nuovo percorso ogni volta che venga individuata una palla; questo, come abbiamo visto, risulta in un arresto momentaneo del robot, che, una volta pianificata la nuova traiettoria, torna ad inseguire la palla a velocità costante. Un approccio evidentemente fastidioso a causa delle continue pause che intervengono durante l'inseguimento.

**La strategia migliore**

La strategia di inseguimento che assicura le migliori prestazioni in termini di efficienza computazionale e resistenza ai falsi accoppiamenti è quella stereoscopica con regolazione PID della velocità angolare.

**I parametri di controllo**

Il regolatore applica le variabili di controllo separatamente alla velocità angolare e lineare. Mediante prove sperimentali applicate direttamente all'inseguimento, sono stati ricavati i seguenti parametri di controllo:

| | |
|---|---|
| $K_P$ | 0.6 |
| $K_I$ | 0.07 |
| $K_D$ | 0.0 |
| $k_v$ | 0.5 |
| N | 10 |

Che individuano rispettivamente: i parametri del PID di velocità angolare, il guadagno di velocità lineare, la posizione relativa del polo lontano. Essi sono validi per un intervallo di campionamento $T_c = 0.1s$.

**Processi multithread**

Per assicurare che il sistema di visione invii al robot informazioni sulla posizione della palla in tempo reale è stato scelto un intervallo di campionamento costante $T_c = 0.1s$, e sono stati creati due processi multithread che gestiscono indipendentemente l'odometria (insieme ai comandi da impartire a robot ad ogni $T_c$) e l'elaborazione delle immagini stereoscopiche.

**Precisione nel calcolo delle distanze**

Il sistema stereoscopico, come anticipato, mostra un elevato grado di accuratezza nella stima delle distanze, che va da un'incertezza di 1 mm, su distanze di un metro, a circa 2 cm per distanze nell'ordine di due metri.
Al fine di fornire un feedback in tempo reale del grado di accuratezza con cui il sistema calcola la distanza dall'oggetto, il programma che ho realizzato stima il raggio della circonferenza (misurato in pixel) con cui dovrebbe apparire una palla situata a proprio quella distanza e traccia una circonferenza di egual raggio in corrispondenza dei centri.
Le due immagini che seguono illustrano chiaramente il risultato di questa riproiezione 3D: i due cerchi (disegnati in bianco) si sovrappongono perfettamente al contorno della palla,

dimostrando in maniera inequivocabile il livello di accuratezza raggiunto. (Nell'esempio la palla giace a circa 1.98 m dal robot).

**Figura 4.10**

### Resistenza alle occlusioni

Come accennato, il sistema funziona anche in presenza di occlusione di una porzione di oggetto Tuttavia, nonostante la palla venga individuata, l'occlusione pregiudica l'accuratezza stereometrica del sistema, come evidente dalla riproiezione dei cerchi in funzione della distanza stimata.

Tale errore è dovuto al minor numero di punti di contorno dell'immagine occlusa. Se nel caso precedente i contorni apparivano più simili a cerchi, in quest'altro caso appaiono solamente archi di circonferenza, e la determinazione del centro, di conseguenza, è affetta da maggiore incertezza.

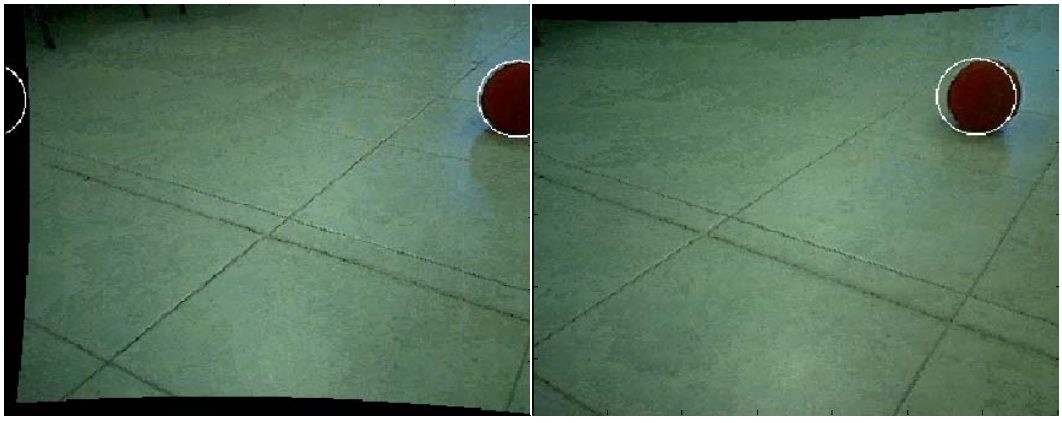


**Figura 4.11**

Quest'altra coppia di immagini ostenta un errore stereometrico maggiore del precedente a causa della maggiore percentuale di superficie occlusa della palla.

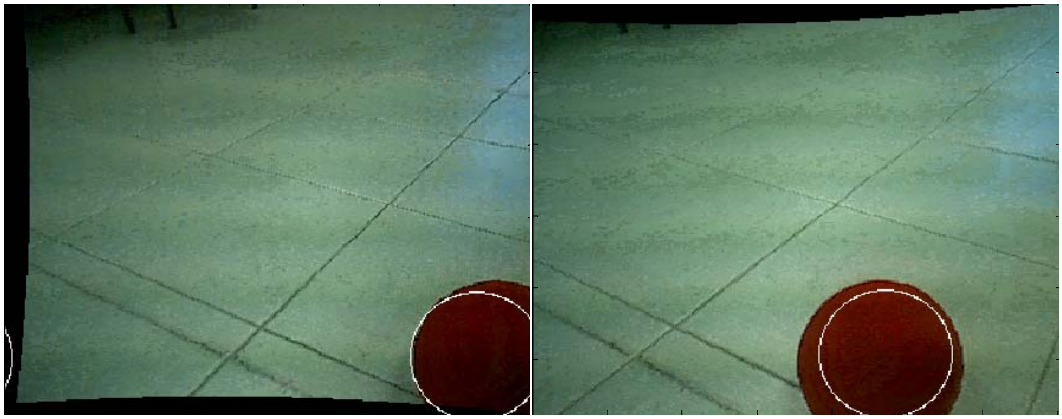


**Figura 4.12**

Ai fini dell'inseguimento, tuttavia, l'occlusione non presenta alcun problema. Infatti essa intacca maggiormente il calcolo della distanza radiale e solo molto debolmente l'errore angolare!
Poiché, come abbiamo visto, è proprio l'errore angolare $\vartheta$ che comanda l'inseguimento, non appena questo viene corretto la palla appare al centro delle telecamere e la distanza stavolta viene determinata con una precisione maggiore, garantendo il raggiungimento de target.

**Aspetti positivi della strategia di inseguimento stereoscopica**

Gli aspetti innovativi del sistema proposto sono:

1. grazie al riconoscimento della palla basato sulla forma (shape based) piuttosto che sul colore, il sistema ha dimostrato di inseguire la palla anche durante la sovrapposizione di altri corpi o persone che intercorrano tra robot e oggetto.
2. il sistema è in grado di resistere alle frequenti situazioni di occlusione della palla, che comportano l'occultamento di una parte o di un'intera porzione di essa, soprattutto quando si trovi ai limiti del campo visivo delle telecamere
3. la stereoscopia ha inoltre offerto una buona resistenza ai molteplici casi di falso accoppiamento, sfruttando i vincoli sulla costanza della quota, sulla geometria epipolare dei punti coniugati, sulla precisione del calcolo della distanza
4. è garantito inoltre il controllo del robot in tempo reale, grazie alla gestione indipendente di processi multithread separati, per l'image processing e per l'odometria
5. tutte le funzioni sono state implementate con un basso costo computazionale

**Aspetti critici della strategia di inseguimento stereoscopica**

Ciononostante permangono alcuni aspetti legati alla visione stereoscopica ancora da risolvere:

1. La velocità di elaborazione delle coppie di immagini stereo (6 Hz) è ancora troppo bassa per permettere al robot di reagire a moti repentini della palla
2. Se la palla non è visibile ad entrambe le telecamere, essa risulta invisibile
3. Permangono alcuni casi sporadici di falsi accoppiamenti che portano il robot ad inseguire un punto ignoto dello spazio
4. Se non vi sono falsi accoppiamenti e la palla non è stata individuata, il robot resta fermo ma non effettua alcuna ricerca.

**Strategia di ricerca e inseguimento**

Come analizzato nel 4.4, questa strategia, nonostante l'approccio ancora monoscopico, fornisce degli ottimi risultati in termini di resistenza ai falsi accoppiamenti: fino al 90% di essi vengono esclusi grazie al Block Based Motion Estimation. Il robot è perfettamente in grado di inseguire la palla, quando questa è presente di fronte ad esso, cercarla, quando assente, e di nuovo individuarla ed inseguirla non appena rientra nel suo campo di visibilità.
In futuro potrebbe essere applicata ad entrambe le telecamere in modo da ampliare il campo visivo dei sensori mediante la fusione di stereo e monoscopia. La stereoscopia entrerebbe in funzione quando la palla sia visibile a entrambi i sensori, mentre la monoscopia interverrebbe nei soli casi in cui la palla appartenga al campo visivo di un solo "occhio", correggendo l'assetto angolare in modo da renderla visibile a entrambi!
La capacità della stereoscopia di rimuovere i falsi accoppiamenti, unita alla block based motion estimation consentirebbe al sistema elevatissime prestazioni.

# Capitolo 5
# Autolocalizzazione


## Abstract

In questo capitolo viene descritta l'applicazione degli algoritmi di riconoscimento e di stereotriangolazione, sviluppati nei precedenti capitoli, all'autolocalizzazione di un robot mobile che naviga all'interno di un ambiente di mappa nota.
L'autolocalizzazione è basata sul riconoscimento di landmark artificiali situati in punti della mappa che sono noti al robot. Quando questo si trova in prossimità di un landmark, ne calcola la posizione relativa al suo sistema di riferimento mobile e, sulla base di essa, ricostruisce la propria posizione nel sistema di riferimento globale, localizzandosi all'interno della mappa.
Grazie al sistema di visione stereoscopica, viene inoltre dimostrata la capacità del robot di navigare nell'ambiente e correggere gli errori sulla posizione fornita da sensori odometrici.


## 5.0 Introduzione

### 5.0.1 Il problema dell'autolocalizzazione

Per autolocalizzazione di un robot mobile si intende il processo autonomo di individuazione e aggiornamento della posizione e dell'orientazione (assetto) del robot, attraverso l'utilizzo dei dati sensoriali. Il fatto di *sapere dove si trova* è fondamentale per il robot, sia quando si deve spostare da un punto all'altro, sia quando deve determinare la posizione assoluta degli ostacoli in un processo di map-building (costruzione della mappa di un edificio).
In generale si individuano due categorie di metodi di localizzazione: *metodi relativi*, che calcolano la posizione del robot rispetto al punto di partenza; *metodi assoluti*, i quali non necessitano di questa informazione e valutano direttamente la posizione in un sistema di riferimento assoluto.
La precisione con cui un robot può conoscere la sua posizione non dipende soltanto dallo specifico metodo di localizzazione che si utilizza, ma anche dai sensori che forniscono i dati in ingresso al sottosistema che si occupa della localizzazione.
Per migliorare la precisione dell'autolocalizzazione è possibile quindi utilizzare tecniche più complesse, ma anche scegliere sensori più sofisticati!
A seconda delle due metodologie d'uso dei sensori, le tecniche di localizzazione possono essere così classificate:

- Misure di posizione relativa
    - Odometria
    - Navigazione inerziale
- Misure di posizione assoluta
    - Guide attive
    - Riconoscimento di landmark naturali e/o artificiali
    - Navigazione tramite mappa

Il robot attualmente in uso presso il dipartimento di Elettronica utilizza due tecniche di posizionamento relativo e assoluto: la prima basata sui dati forniti dall'odometria (encoder), la seconda basata sul Map Matching tramite filtro di Kalman dei dati forniti dai sonar.
La tecnica odometrica è la strategia di posizionamento più semplice.
Il termine odometria consiste nella misura dello spostamento compiuto dal veicolo lungo il cammino, che viene dedotto direttamente da un qualche odometro di bordo: nel nostro caso una coppia di encoder ottici fissati direttamente sugli assi delle ruote.
Poiché l'idea di base dell'odometria è la stima della posizione corrente sulla base dell'integrazione nel tempo dei dati forniti dagli encoder, esso risulta sufficientemente preciso per brevi percorsi, ma su distanze più lunghe gli errori si accumulano ed iniziano ad influenzare in maniera significativa il comportamento del robot.
In alcune situazioni valide per ambienti chiusi esso può correggere gli errori commessi tramite una tecnica di localizzazione assoluta basata su mappe.
Il map-matching è una tecnica in cui il robot usa i dati provenienti dai suoi sonar per creare una mappa dell'ambiente. Questa mappa viene poi confrontata con un modello in memoria e, se viene trovata corrispondenza tra le due (tramite filtro di Kalman), il robot è in grado di risalire alla sua posizione e orientazione globale.
Sebbene quest'ultima tecnica localizzi il robot in maniera assoluta rispetto a quella odometrica, esso funziona bene in ambienti chiusi ed in luoghi in un cui l'ambiente presenta una conformazione della pianta che sia unica e particolare, tale da non arrecare ambiguità. Inoltre la posizione e l'orientazione ricostruite sono affette da un'incertezza non sempre trascurabile.

### 5.0.2 La strategia di autolocalizzazione adottata

La strategia descritta in questo capitolo consiste nell'implementazione di algoritmi di visione artificiale, per il sistema stereoscopico in oggetto, che assistano il robot durante la navigazione.
L'autolocalizzazione è basata sul riconoscimento di landmark artificiali situati in punti della mappa che sono già conosciuti al robot. Quando questo si trova in prossimità di un landmark, ne calcola la posizione relativa al suo sistema di riferimento mobile e, sulla base di essa,

ricostruisce la propria posizione nel sistema di riferimento globale, localizzandosi all'interno della mappa.
I landmark sono stati disposti sul pavimento, a distanze non troppo lontane l'uno dall'altro (4.5 m) in modo che il robot sia in grado di giungervi in prossimità, sulla base dei sensori odometrici, senza troppe difficoltà. Inoltre, sono stati scelti tutti uguali fra loro e, per dedurre quale di essi si trovi nel campo di visibilità della stereo-camera, si utilizza l'informazione sullo spostamento relativo compiuto durante il tragitto, stimato sulla base degli encoder.

Nel primo paragrafo si parlerà del riconoscimento del landmark.
Per poter sfruttare gli algoritmi di circle-detection sviluppati nel capitolo 2, si è scelto di utilizzare un landmark di forma circolare. Tuttavia, poiché un landmark di forma circolare, non è sufficiente per stimare l'orientamento relativo tra il robot e il landmark (da cui dedurre in una seconda fase l'assetto del robot), a questo è stata aggiunta una barra verticale. Pertanto all'algoritmo di circle-detection per l'individuazione del cerchio si è aggiunta la necessità di implementarne uno per la line detection (quest'ultimo basato sul principio della trasformata di Hough per le linee).

Infine, nel secondo paragrafo tratterò la ricostruzione della posizione globale del robot a partire dall'informazione sulla posizione relativa del landmark.
Poiché i landmark giacciono sul pavimento, a causa della distorsione prospettica essi non appariranno circolari, bensì ellittici. Pertanto ciò non rende direttamente applicabile l'algoritmo di circle detection. Dapprima si è quindi proceduto ad un'inversione di prospettiva, che vedremo essere facilmente applicabile modificando un parametro nella trasformazione che governa la rettificazione epipolare della coppia di immagini stereo, descritta al 3.3.
Dopo aver invertito la prospettiva i landmark appariranno circolari e sarà dunque possibile individuarli e stimarne successivamente la posizione grazie all'algoritmo di stereo-triangolazione, descritto al 3.6.
Mediante un cambiamento di riferimento si risale infine alla posizione assoluta del robot all'interno della mappa.
La mappa di cui ho parlato finora risiede già all'interno della memoria del programma e riguarda la ricostruzione bidimensionale della pianta della sezione di Informatica ed Automazione del dipartimento di Ingegneria Elettronica e dell'Informazione.

Allo scopo di offrire una dimostrazione concreta della capacità di autolocalizzazione basata sulla visione stereoscopica, è stato definito un percorso di prova all'interno del Dipartimento di Ingegneria Elettronica e dell'Informazione. Il robot doveva dimostrare di saper navigare all'interno dell'edificio con l'ausilio della sola visione, raggiungendo determinate tappe contrassegnate da landmark artificiali.

Giunto in corrispondenza di ciascun target, il robot ricostruisce la sua posizione globale all'interno della mappa e corregge, annullandoli, gli errori di localizzazione forniti dall'odometria. Una volta riposizionatosi, punta, raggiungendola, alla tappa successiva e prosegue così iterativamente.

Il robot ha dimostrato di sapersi perfettamente localizzare sulla base della stereo-visione, con una precisione dell'ordine del millimetro in distanza, e dell'ordine del di 0.1° per quanto riguarda l'assetto angolare.

# 5.1 Riconoscimento di un landmark

## 5.1.1 Verifica della presenza del landmark

In figura 5.1 è rappresentato il landmark circolare utilizzato per l'autolocalizzazione; come si vede è caratterizzato da una barra centrale necessaria per l'orientamento del robot.

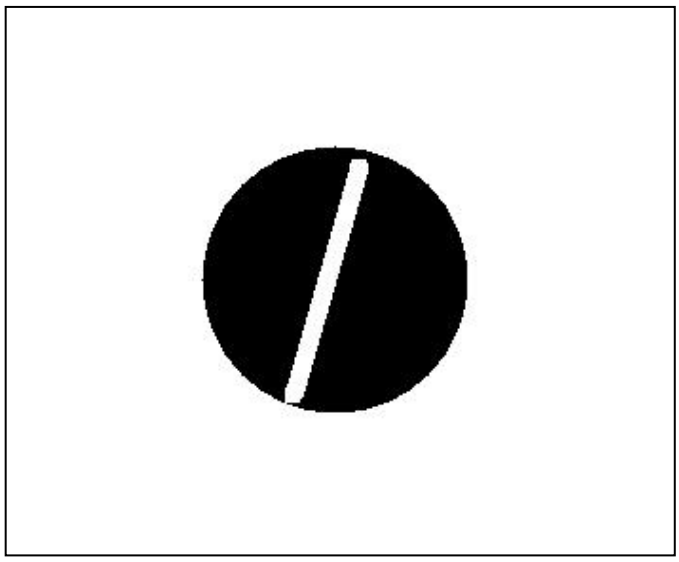


**Figura 5.1**

Per individuare la presenza del landmark si adotta l'algoritmo circle detection presentato al capitolo 2. Come abbiano già detto parlando nella determinazione del cerchio, tale algoritmo consiste nel tracciamento di rette perpendicolari ai punti di contorno, che, nel caso di circonferenze, si intersecano esattamente nel centro della palla. In questo modo il centro corrisponde alla cella con il maggior numero di "voti" nello spazio dei parametri. La figura 5.2 richiama tale procedura, già descritta al capitolo 2.

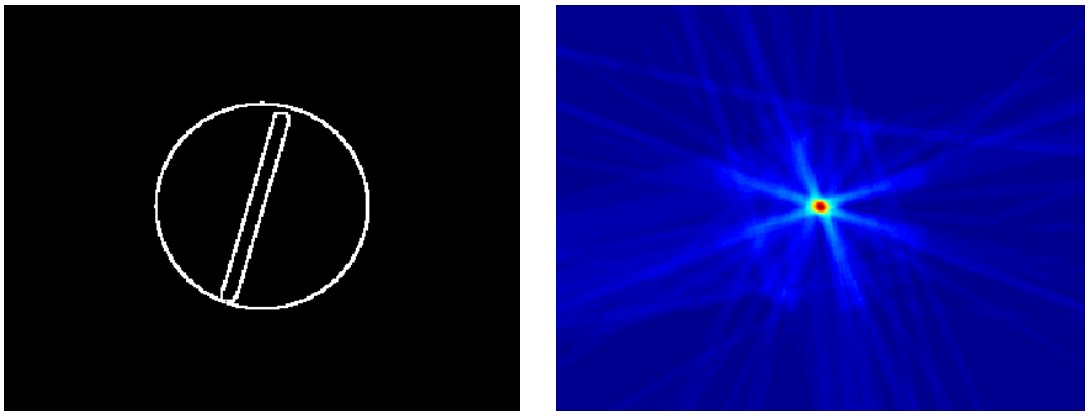


**Figura 5.2** Le fasi dell'algoritmo di circle detection: dapprima si estraggono i contorni dell'immagine; poi si tracciano le rette normali ai punti di tangenza nello spazio dei parametri; il punto col maggior numero di voti corrisponde al centro.

La prima osservazione da fare guardando la figura 5.2 è che la presenza della barra centrale non influenza minimamente l'individuazione del cerchio.

Sebbene l'algoritmo funzioni eccellentemente in presenza di forma circolari, esso restituisce comunque qualcosa anche in assenza di esse, che corrisponde al punto col maggior numero di volti nello spazio dei parametri.

Per far sì che il programma sia certo di aver trovato un cerchio occorre posporre una fase di verifica. A tal fine è bene osservare che dato che i landmark sono stati fissati sulla superficie omogenea del pavimento, in assenza di essi si avranno picchi poco pronunciati nello spazio dei parametri (figura 5.4).

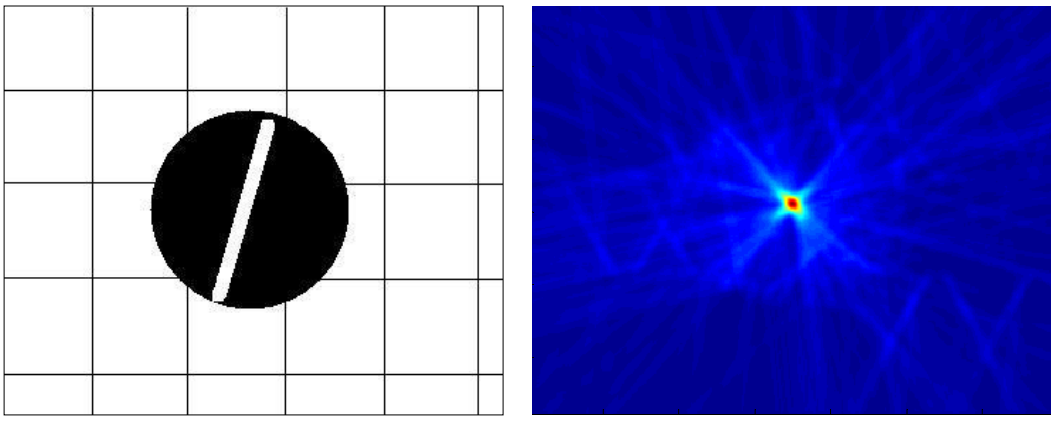


**Figura 5.3** Landmark su una superficie omogenea che simula la presenza dei solchi nel pavimento. Osservando lo spazio della trasformata a destra si evince che il pavimento non influenza la determinazione del cerchio.

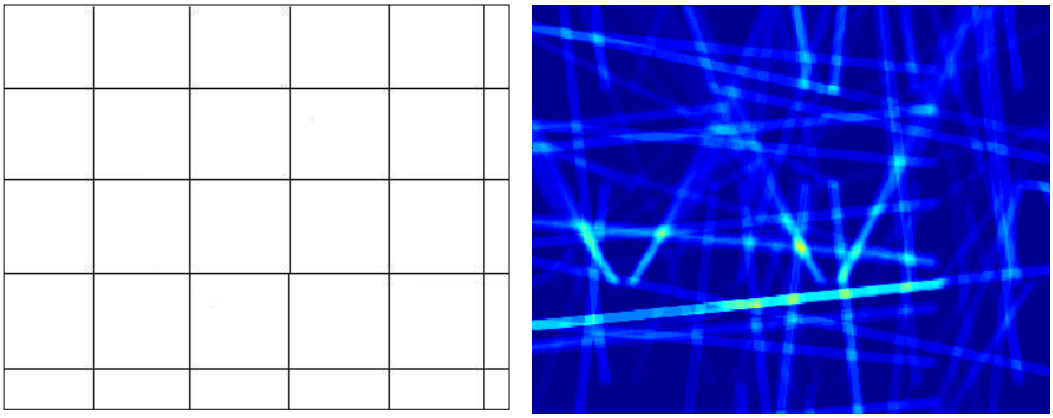


**Figura 5.4** In questo caso è il landmark non è presente. Nello spazio dei parametri si individuano comunque dei picchi ma sono caratterizzati da un minor numero di voti rispetto a quelli che avremmo in presenza di un cerchio.

In questo modo la presenza o meno di un landmark può essere codificata sulla base del numero di voti nello spazio dei parametri.

Nell'algoritmo che verifica la presenza di un landmark ho imposto una soglia pari a 60.

Detto N il numero di voti nello spazio dei parametri si ha:

**Se N>60 allora segnala la presenza del landmark (“true”)**
**altrimenti restituisci un valore “false”**

## 5.2.2 Calcolo dell’orientamento del landmark

Stimare l’orientamento del landmark equivale a calcolare l’angolo con cui è inclinata la barra centrale. Per fare questo occorre però esaminare solo i punti che risiedono all’interno del perimetro stesso del landmark, che giacciono in corrispondenza di un segmento rettilineo.
Una volta determinato il landmark con entrambe le telecamere risulta facile, grazie alla stereoscopia, dedurne il perimetro (si veda capitolo 4) e quindi i punti in esso contenuti.
Ma come determinare quali punti sono parte di un segmento rettilineo e ricavare da questi l’inclinazione della barra?
Sono possibili due soluzioni:

1. calcolo della retta di regressione lineare
2. trasformata di Hough per le linee [5]

### 5.2.2.1 Metodo della regressione lineare

La prima, utilizzata in statistica, permette di individuare la retta che meglio interpola N punti presumibilmente allineati. Questo metodo risulta però fortemente sensibile alla posizione dei punti: è sufficiente che anche solo uno di essi sia lontano dalla retta di regressione perché si commettano errori notevoli nel calcolo dell’inclinazione. Infatti i punti appartenenti al perimetro circolare del landmark partecipano alla determinazione della retta di regressione portando a risultati insoddisfacenti (si veda figura 5.5)

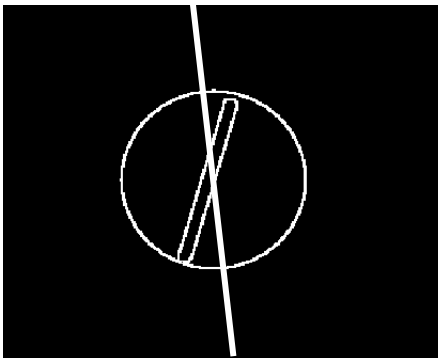


**Figura 5.5** La retta di regressione che interpola i punti appartenenti alla barra centrale

#### 5.2.2.2 Trasformata di Hough per le linee

La seconda consente di isolare i soli punti effettivamente allineati e fornisce i risultati eccellenti, con una precisione nel calcolo della pendenza dell'ordine di 0.1°!
La trasformata di Hough per le linee [5] è analoga a quella usata nella prima versione di circle detection al capitolo 2. Ma stavolta ad ogni punto $(x_i, y_i)$ dello spazio geometrico corrisponde una retta nello spazio dei parametri di equazione polare:

$$\rho = x_i \cdot \cos\vartheta + y_i \cdot \sin\vartheta \tag{5.1}$$

dove $\rho$ è la distanza della retta dall'origine, $\vartheta$ è l'angolo tra $\rho$ ed x.

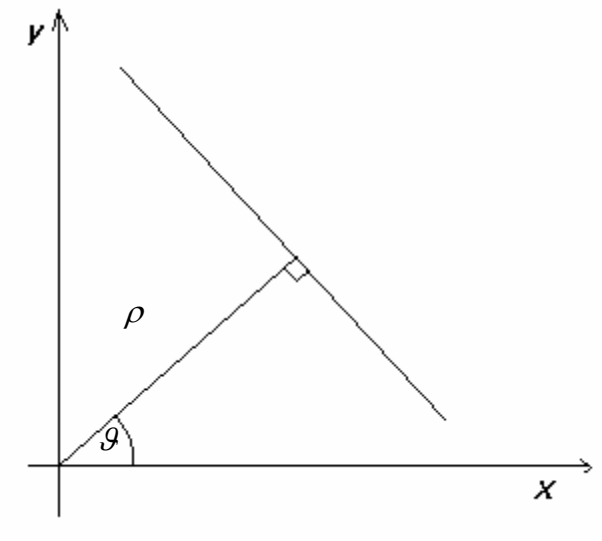
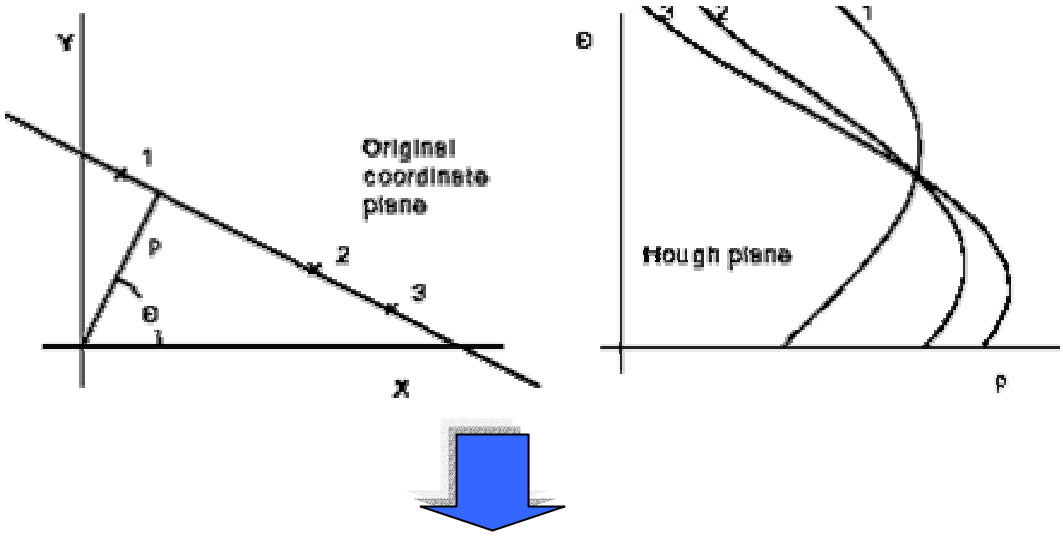



**Figura 5.6**

Facendo variare $\vartheta$ tra -90° e 90° a passi di 1° per ciascuna coppia $(x_i, y_i)$, e calcolando il corrispettivo valore di $\rho$, si ottiene la trasformata di Hough per le linee.



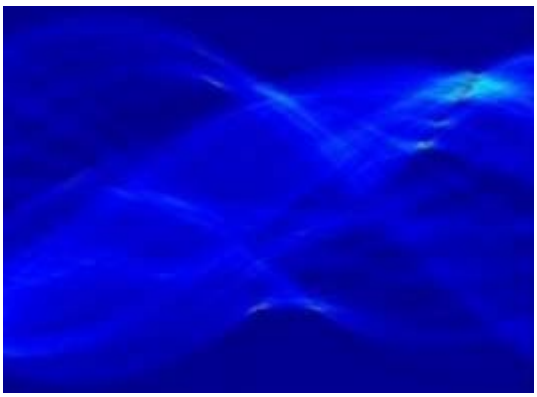



**Figura 5.7** Procedura di calcolo della trasformata di Hough per le linee

La retta che meglio interpola i punti allineati si trasforma corrisponde nello spazio di Hough al punto col maggior numero di voti (picco massimo in figura 5.7).

La coordinata $\vartheta$ del punto di massimo coincide con l'inclinazione della barra del landmark.

Di seguito si mostra il risultato di individuazione dell'orientamento del landmark grazie alla trasformata di Hough. Come si vede, la retta interpolatrice risulta stavolta parallela alla barra.

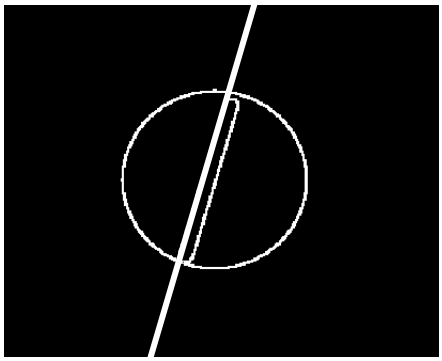


**Figura 5.8** La retta che interpola i punti appartenenti alla barra centrale è stata ottenuta grazie alla trasformata di Hough

Per ridurre la complessità computazionale dovuta al calcolo delle funzioni trigonometriche si è utilizzata una look-up-table con i valori di sin e cos memorizzati a passi di 1°. Per recuperare la perdita di risoluzione nel calcolo del $\vartheta$ (pari proprio ad 1°) dovuta all'entità del passo di quantizzazione, ho interpolato i dati nell'intorno del punto di massimo ottenendo in questo modo una risoluzione finale di 0.1°. Ciò permette anche di stimare l'assetto reale del robot proprio con una precisione di 0.1°.

## 5.3 Inversione della prospettiva

Poiché i landmark giacciono sul pavimento, a causa della distorsione prospettica essi non appariranno circolari, bensì ellittici. Pertanto ciò non rende direttamente applicabile l'algoritmo di circle detection.

Le figure seguenti illustrano la scena vista dal sistema stereoscopico montato sul robot, relative rispettivamente all'occhio sinistro e destro.

N.B Le immagini non sono state ancora rettificate

**Figura 5.9** Vista del robot

Dalle immagini in figura 5.9 risulta evidente la forma ellittica del landmark.

Per far sì che essi appaiano circolari sarebbe necessario avere le telecamere con il piano focale parallelo al pavimento; in questo modo la scena reale e quella ripresa dalla telecamera sarebbero uguali a meno di un fattore di scala.

Come ho anticipato nell'introduzione, è possibile correggere le immagini mediante un processo di inversione della prospettiva: il principio alla base di questo procedimento consiste in una rotazione delle matrici di proiezione prospettica che descrivono i modelli matematici delle telecamere (capitolo 3) in modo da renderle parallele al piano orizzontale del pavimento.

La rotazione avviene lungo l'asse X (che congiunge i centri ottici delle telecamere) ed è caratterizzata dall'angolo di inclinazione $\vartheta$, necessario a rendere il piani focali paralleli al pavimento:

$$Rn = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\vartheta & \sin\vartheta \\ 0 & -\sin\vartheta & \cos\vartheta \end{bmatrix} \tag{5.2}$$

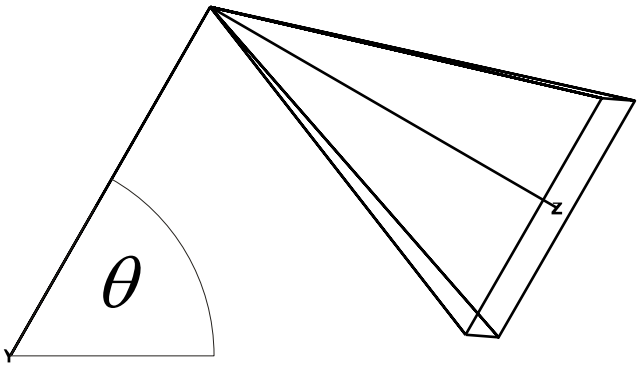



**Figura 5.10** Vista del robot

Poiché l'angolo $\vartheta$ non è noto a priori deve essere inserito dall'utente; la precisione con cui viene invertita la prospettiva dipende dall'accuratezza di con cui si conosce $\vartheta$.

A questo punto, nota la matrice di rotazione Rn, l'inversione di prospettiva può essere fatta direttamente nel processo di rettificazione. Infatti, ricordando quanto detto al capitolo 3, il passo fondamentale della rettificazione consiste nell'applicazione di una rotazione necessaria a rendere i piani focali delle due telecamere perfettamente paralleli fra loro, ma, come si è detto in quella sede, l'inclinazione finale è arbitraria e pertanto era stata assunta verticale.

Ora, invece, modifichiamo direttamente la matrice di rotazione finale della rettificazione tramite la 5.2.

In conclusione, l'inversione della prospettiva può essere effettuata direttamente all'interno della rettificazione; l'angolo di inclinazione necessario ad ottenere la trasformazione prospettica inversa può essere passata come argomento alla funzione `init_stereo_camera` .

## 5.4 Risultati dell'inversione prospettica

Di seguito si riportano i risultati dell'inversione prospettica per diversi valori di $\vartheta$.

**Figura 5.11** Immagini rettificate con un angolo di inversione prospettica ancora troppo piccolo rispetto a quello reale. Come si vede, infatti, le linee del pavimento convergono a due punti di fuga e i landmark appaiono ancora ellittici. Si osservi inoltre come l'inversione prospettica induca anche una deformazione omografica delle immagini.

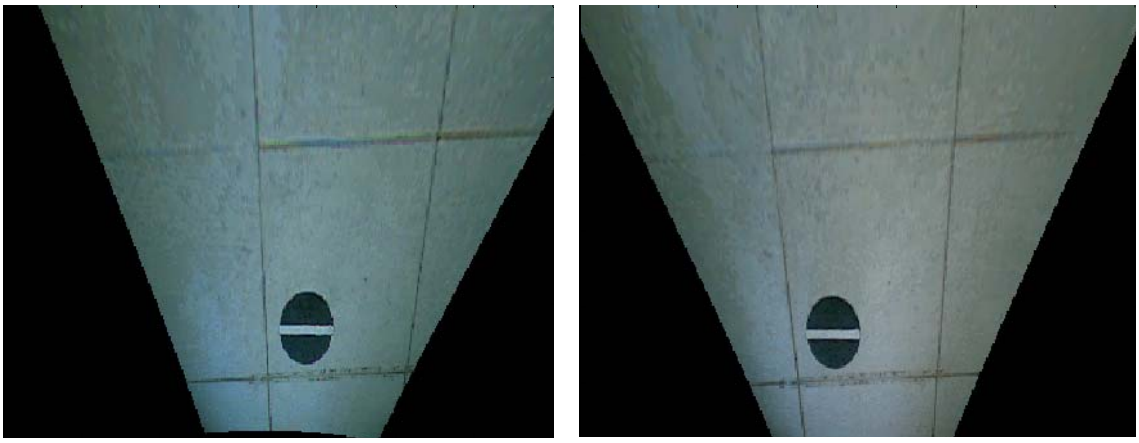


**Figura 5.12** In questo esempio, invece, l'angolo di inversione prospettica è troppo elevato rispetto a quello reale: le linee del pavimento in questo caso divergono dai punti di fuga. Anche in questo caso i landmark risultano ellittici e l'algoritmo di circle detection non individua alcun cerchio.

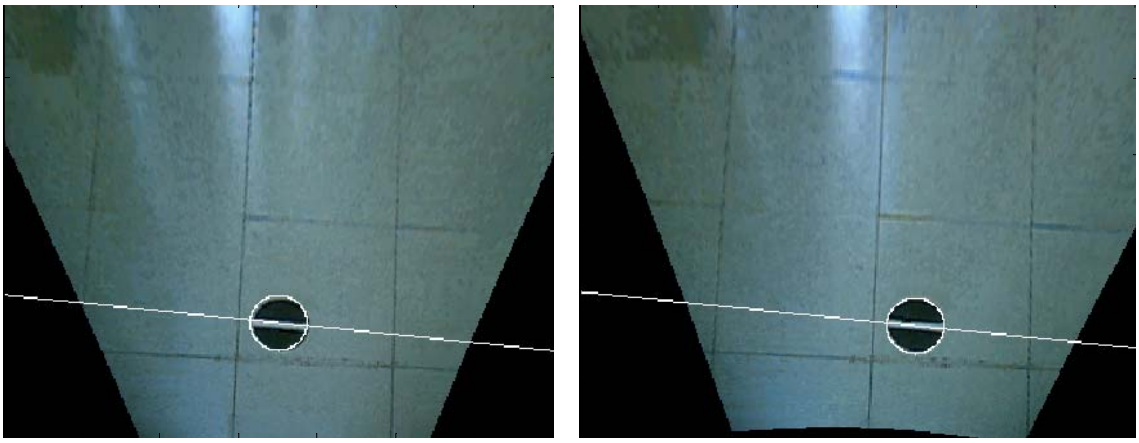


**Figura 5.13** In quest'ultimo esempio l'angolo di inversione prospettica è corretto; ciò è evidenziato dal fatto che le linee di fuga (o anche quelle del pavimento) sono parallele fra loro. Questa condizione si verifica se e solo se l'angolo di inversione prospettica $\vartheta$ coincide con l'inclinazione effettiva delle due telecamere rispetto al pavimento. Si noti inoltre che, essendo ora i landmark perfettamente circolari, risultano facilmente individuabili dall'algoritmo di circle detection.

## 5.5 Localizzazione del landmark

Nonostante l'algoritmo di riconoscimento del landmark sia fortemente robusto a falsi accoppiamenti, è sempre bene aumentare il numero di condizioni che devono essere verificate al fine di escluderli del tutto.

Nel capitolo 4, relativo all'inseguimento di un oggetto, abbiamo visto come è possibile sfruttare il vincolo epipolare per escludere possibili errori di riconoscimento e, quindi, di successiva localizzazione dell'oggetto . Dopo la rettificazione delle due immagini, che abbiamo visto essere necessaria per la correzione delle non idealità sei sensori, i centri dei due landmark, eventualmente individuati, devono giacere sulla stessa retta epipolare: se ciò non accade, esiste un'alta probabilità vi sia stato un errore di riconoscimento.

Da ciò segue che, una volta ottenute le coordinate di possibili cerchi nei due frame, si confrontino le rispettive ordinate. Tuttavia, poiché tali coordinate subiscono un processo di interpolazione sub-pixel per aumentare la risoluzione delle telecamere e ridurre l'effetto del rumore, è praticamente impossibile che essi abbiano la stessa ordinata. A tal fine nell'algoritmo viene richiesto che la differenza tra le coordinate verticali rientri in un intervallo di confidenza abbastanza piccolo: 1 pixel. Per cui:

$$\left| y_{sx} - y_{dx} \right| < 1 \tag{5.3}$$

anche se nella realtà tale differenza ammonta ad appena 0.5 pixel (dopo interpolazione sub-pixel), a dimostrazione del livello di accuratezza degli algoritmi di rettificazione e di riconoscimento di landmark.

Nel capitolo 3, relativo alla misura della posizione 3D, abbiamo visto che le distanza di un punto dello spazio dalle telecamere dipende fortemente dalla disparità delle sue proiezioni sui piani retinici dei sensori. Ovvero, la distanza H longitudinale di un punto dello spazio dal centro delle telecamere è data dalla (1.1):

$$H = \frac{b \cdot f}{D} \tag{5.4}$$

dove D è la disparità, mentre b (baseline) ed f (focale) sono grandezze costanti.

Ricordo che la disparità si calcola lungo le rette epipolari, che sono orizzontali.

Pertanto, piccoli errori nel calcolo della disparità del landmark (dovuti all'ineluttabile presenza di rumore) possono provocare variazioni percettibili nella stima della distanza. Infatti, mentre la (5.3) è un vincolo che deve essere sempre verificato, non è possibile porre alcuna condizione per quanto riguarda D.

Quello che intendo dire è che, anche se il robot è fermo rispetto al landmark, il rumore d'immagine inquina il calcolo preciso della disparità, al punto che la distanza dell'oggetto, valutata grazie alla stereotriangolazione e in istanti diversi, è soggetta a continui errori casuali che raggiungono anche 2-3 cm di lunghezza.

Per ovviare a questo inconveniente, in grado di alterare le misure della distanza in istanti temporali diversi, ho deciso di posporre alla condizione (5.3), relativo alle ordinate, anche un controllo sull'ascissa.

Se è stato individuato un landmark nell'immagine sinistra, esso apparirà simile in quella di destra. Pertanto questa ulteriore condizione di controllo consiste nel verificare che la disparità calcolata applicando separatamente la circle detection nelle due immagini, sia attendibile con la disparità calcolata tramite una ricerca di "somiglianza".

Supponiamo quindi che la circle detection, applicata separatamente alle due immagini, abbia restituito le seguenti coppie di coordinate $(x_{sx}, y_{sx})$, $(x_{dx}, y_{dx})$, e sia dunque

$$D_1 = x_{sx} - x_{dx} \tag{5.5}$$

la disparità calcolata con il primo criterio.

La funzione di somiglianza confronta il valore dei pixel appartenenti ad una finestra centrata in $(x_{sx}, y_{sx})$, che chiamerò $I_{sx}$, con i pixel contenuti in una finestra di pari dimensioni, $I_{dx}(x, y)$, che viene fatta scorrere lungo la corrispondente retta epipolare dell'immagine destra. Vi sono vari criteri di somiglianza tra finestre; uno dei più comuni è la cosiddetta SSD (*Sum of Squared Differences*):

$$SSD = \sum_{i=1}^{width} \left[ I_{sx} - I_{dx}(i, y) \right]^2 \tag{5.6}$$

Il punto corrispondente dell'immagine sinistra nell'altra immagine è quello per cui la (5.5) ha valore minimo: sia $i_{MIN}$ la sua ascissa. La disparità calcolata con la SSD sarà dunque:

$$D_2 = x_{sx} - i_{MIN} \tag{5.7}$$

A questo punto basta confrontare $D_1$ con $D_2$: se i due valori differiscono di poco (nella pratica <5 pixel) allora il riconoscimento del landmark è giudicato attendibile e viene preso in considerazione $D_2$. Pertanto l'ascissa del landmark nell'immagine destra sarà: $x_{dx} = i_{MIN}$.

Lo schema a blocchi in figura 5.14 riassume la procedura di localizzazione del landmark:

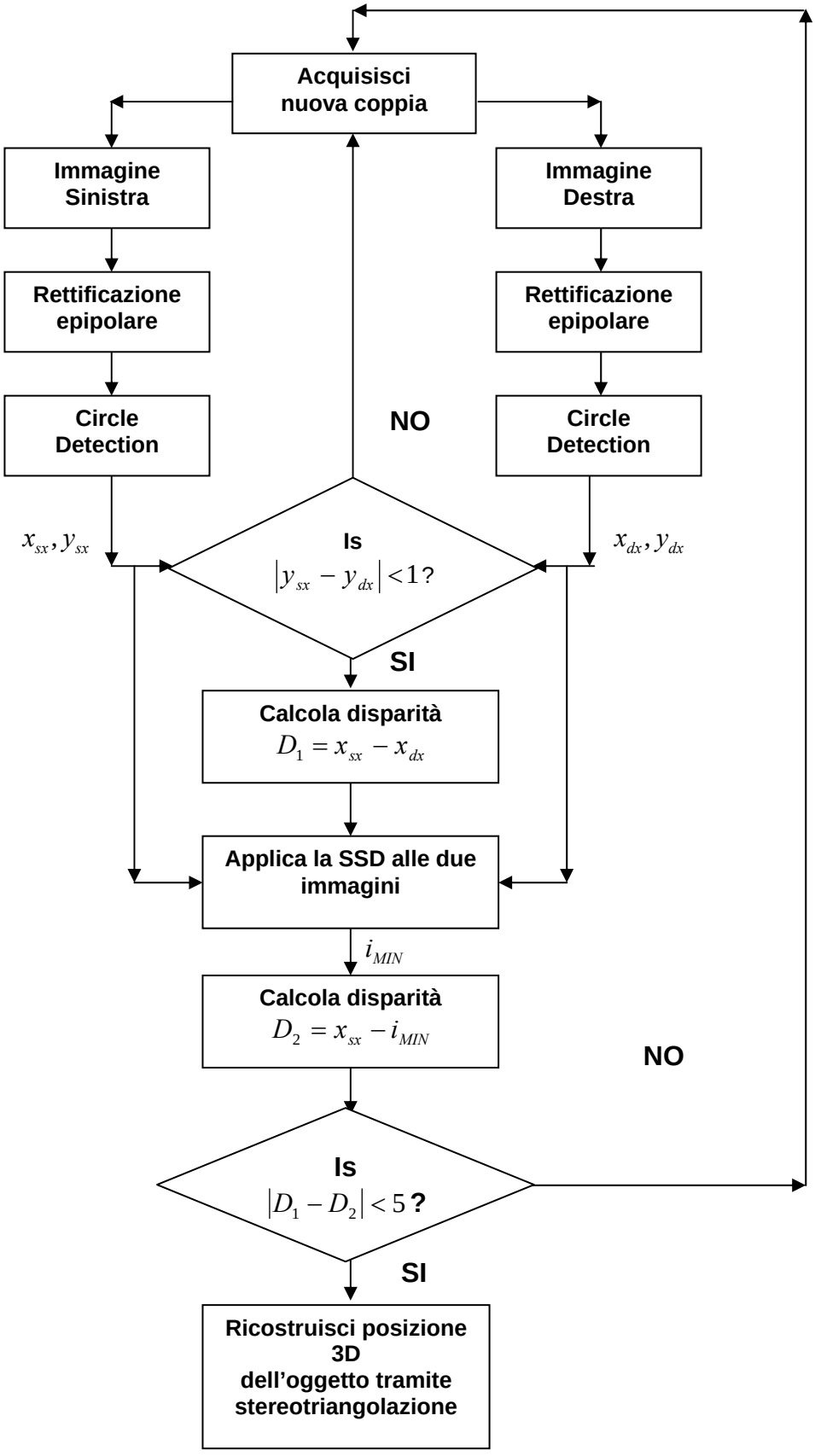



**Figura 5.14** Schema a blocchi che riassume la sequenza di elaborazione delle immagini per il calcolo della posizione relativa della palla rispetto al robot

## 5.6 Autolocalizzazione

### 5.6.1 Ricostruzione della posizione assoluta

L'autolocalizzazione consiste nell'individuazione della posizione assoluta e dell'orientazione (assetto) del robot all'interno di una ambiente di mappa nota grazie all'utilizzo dei dati sensoriali.

Supponiamo che il robot sia giunto in prossimità del landmark e conosca la sua posizione assoluta sulla base dello spostamento totale misurato dall'odometria. Come abbiamo detto nell'introduzione, la posizione del robot rispetto al sistema di riferimento globale non coincide con quello misurato dagli encoder, a causa degli ineluttabili errori accumulatisi durante il tragitto: dovuti ad esempio ad irregolarità della superficie, sterzamenti rapidi o strisciamento delle ruote. Tali errori, inoltre, si accumulano ai precedenti durante il tragitto, al punto che il robot ad un certo punto non sa più dove si trovi.

In questa sede intendo dimostrare che, grazie alla visione stereoscopica e agli algoritmi di riconoscimento di landmark, è possibile ricostruire la posizione assoluta e l'assetto partendo da misure relative della distanza landmark-robot, se e solo se, naturalmente, sono note a priori le coordinate del landmark.

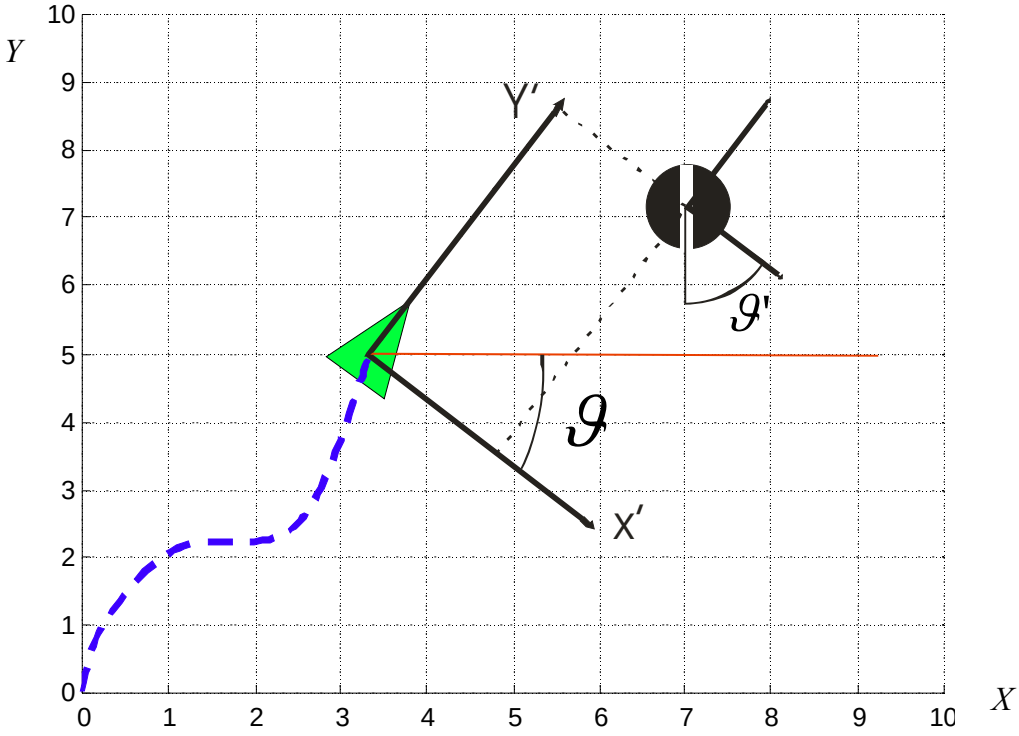



**Figura 5.14** Una schematizzazione della strategia di autolocalizzazione grazie all'utilizzo dell'odometria e della visione stereoscopica. In blu è visualizzato il percorso compiuto dal robot per giungere in prossimità di un landmark di posizione nota. Esso viene misurato dagli encoder e corretto sulla base della visione.

Diamo prima le seguenti definizioni:

- XY è il sistema di riferimento
- $(x_{lm}, y_{lm})$ sono le coordinate note del landmark in XY
- $(x'_{lm}, y'_{lm})$ sono le coordinate del landmark nel sistema robot: esse vengono calcolate grazie alla stereoscopia
- $\vartheta'$ è angolo compreso tra la direzione x' del sistema robot e la barra verticale (bianca) del landmark: questo angolo viene calcolato dall'algoritmo che effettua la trasformata di Hough per le linee
- $\vartheta$ l'angolo compreso tra x' ed x: individua l'assetto del robot nel sistema assoluto e deve essere calcolato
- $(x_{robot}, y_{robot})$ sono le coordinate assolute del robot che dobbiamo stimare

Mediante semplici considerazioni geometriche risulta

$$\vartheta = 90 - \vartheta' \tag{5.8}$$

e quindi la relazione che lega le coordinate assolute del robot a quelle note del landmark si ottiene considerando il cambio di coordinate da X'Y' ad XY.

$$\begin{bmatrix} x_{lm} \\ y_{lm} \end{bmatrix} = \begin{bmatrix} \cos\vartheta & -\sin\vartheta \\ \sin\vartheta & \cos\vartheta \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \end{bmatrix} + \begin{bmatrix} x_{robot} \\ y_{robot} \end{bmatrix} \tag{5.9}$$

In conclusione l'assetto e la posizione del robot nel riferimento assoluto sono:

$$\begin{aligned} \vartheta &= 90 - \vartheta' \\ \begin{bmatrix} x_{robot} \\ y_{robot} \end{bmatrix} &= \begin{bmatrix} x_{lm} \\ y_{lm} \end{bmatrix} - \begin{bmatrix} \cos\vartheta & -\sin\vartheta \\ \sin\vartheta & \cos\vartheta \end{bmatrix} \cdot \begin{bmatrix} x' \\ y' \end{bmatrix} \end{aligned} \tag{5.10}$$

## 5.7 Conclusioni

Allo scopo di offrire una dimostrazione concreta della capacità di autolocalizzazione basata sulla visione stereoscopica, è stato definito un percorso di prova all'interno del Dipartimento di Ingegneria Elettronica e dell'Informazione. Il robot doveva dimostrare di saper navigare all'interno dell'edificio con l'ausilio della sola visione, raggiungendo determinate tappe contrassegnate da landmark artificiali.

I landmark sono stati fissati in determinati punti dello spazio e posti ad una distanza di 4.5 m l'uno dall'altro: una lunghezza sufficiente a garantire che il robot vi arrivasse vicino con l'uso della sola odometria senza apparenti difficoltà.
Giunto in corrispondenza di ciascun landmark, il robot doveva ricostruire la sua posizione e assetto globali all'interno della mappa (pianta dell'edificio), e correggere, annullandoli, gli errori di localizzazione forniti dall'odometria.
Una volta riposizionatosi all'interno dell'ambiente, doveva puntare, raggiungendola, alla tappa successiva, e proseguire così iterativamente verso le altre tappe.

Il robot ha dimostrato di sapersi perfettamente localizzare sulla base della stereo-visione, con una precisione dell'ordine del millimetro in distanza, e dell'ordine del di 0.1° per quanto riguarda l'assetto angolare.
La mappa di cui ho parlato finora consiste nella pianta della sezione di Informatica ed Automazione del dipartimento di Ingegneria Elettronica e dell'Informazione e risiede già all'interno della memoria del programma del robot.

# Capitolo 6
# Conclusioni

## 6.1 Conclusioni

In questa tesi sono stati descritti il progetto e la realizzazione hardware/software di un sistema di visione stereoscopica; inoltre sono stati studiati ed implementati degli algoritmi di visione artificiale per il riconoscimento automatico di oggetti, che hanno reso possibile applicare l'intero sistema ai seguenti problemi robotici:

- riconoscimento ed inseguimento di oggetti in movimento da parte di un robot mobile
- navigazione autonoma di un robot in un ambiente con l'ausilio della sola visione

Sono state pertanto realizzate ed applicate strategie di controllo per la movimentazione di robot mobili.

**Il sistema stereoscopico**

Il primo obbiettivo raggiunto di questo lavoro di tesi ha visto la necessità di sviluppare degli algoritmi che, a partire da immagini stereoscopiche (ovvero immagini della scena catturate da due punti diversi), fossero in grado di stimare la posizione 3D di un oggetto presente nelle due immagini.
La prima fase pertanto ha richiesto la calibrazione del sistema per misurare indirettamente i parametri intrinseci ed estrinseci delle telecamere che governano la proiezione prospettica di punti 3D sui piani ottici dei sensori (per un totale di 10 parametri).
La seconda fase ha richiesto invece di sfruttare la conoscenza di detti parametri per correggere tutte le non idealità del sistema rispetto al modello matematico delle lenti sottili; tra queste: elaborare un modello matematico per la distorsione radiale e tangenziale delle lenti, correggere l'assetto geometrico dei sensori e le differenze tra i fattori di scala delle telecamere.
Tutte le non idealità sono state corrette mediante una trasformazione detta "rettificazione epipolare", che trasforma le immagini di partenza, riprese dalle telecamere originali e diverse fra loro, in altre due immagini, che coincidono con quelle eventualmente riprese da una sola telecamera posta in due punti distinti.
Dopo di ciò sono stati definiti gli algoritmi per la stereotriangolazione, che fossero in grado di ricostruire la posizione 3D di un punto nello spazio a partire dalle sue proiezioni sui piani immagine delle telecamere.

L'intero sistema ha dimostrato un grado di accuratezza notevole nel calcolo delle distanze (un errore di 2 cm su distanze di 2 m), che prescinde dal software sviluppato e dipende esclusivamente dalla risoluzione (numero di pixel) dei sensori.
Il software di stereometria che ho realizzato è stato installato su un hardware diverso dal mio in termini di risoluzione delle telecamere. L'hardware in questione è stato gentilmente prestato dall'ENEA e consisteva di una coppia di telecamere stereoscopiche caratterizzate da una risoluzione doppia rispetto alla nostra.
Il mio stesso software, testato su un hardware differente, ha dimostrato delle prestazioni in termini di accuratezza nel calcolo delle distanze, ben quattro volte maggiori, dimostrando così che non è il software, ma l'hardware a fare la differenza.

**Riconoscimento di oggetti**

Nell'ambito del riconoscimento di oggetti in movimento è stato utilizzato un approccio edge-based, cioè basato sulla forma degli oggetti e non sul colore. Sono stati pertanto studiati ed implementati degli algoritmi di visione artificiale che fossero in grado di riconoscere oggetti di forma sferica. Tra le tre versioni di circle-detection realizzate, l'ultima è quella che ha garantito i migliori risultati perché è in grado di adattarsi a molteplici condizioni ambientali (esterne ed intere), è in grado di resistere alle occlusioni dell'oggetto e assicura prestazioni migliori in termini di complessità, consentendo di raggiungere frame-rate di 25 Hz su hardware dedicato.

**Inseguimento di oggetti**

L'intero apparato stereoscopico è stato montato su un robot mobile differential drive. Dopo aver risolto le problematiche di interfacciamento tra i due sistemi, sono state studiate ed implementate delle leggi di controllo per il robot, che gli permettessero di inseguire un oggetto di forma sferica, individuato tramite gli algoritmi di circle-detection realizzati nella prima fase. L'inseguimento consisteva nel fare in modo che l'oggetto risultasse sempre al centro delle telecamere.
Sono state utilizzate quattro strategie di inseguimento: di tipo monoculare (con una sola telecamera) e di tipo stereoscopico (utilizzando le immagini di due telecamere). Si è visto che la monoscopia non consente di avere una sensazione della profondità degli oggetti e della loro distanza dal punto di osservazione e il robot ha infatti mostrato instabilità nel moto rotatorio (con deboli oscillazioni intorno al punto di equilibrio quando l'oggetto è fermo).
La visione stereoscopica ha invece fornito risultati eccellenti perché, grazie al recupero della profondità, consente di stimare l'angolo di cui deve ruotare il robot per avere l'oggetto sempre visibile.

Le prime tre strategie di inseguimento consentono di inseguire l'oggetto solo finché questo è visibile dalle telecamere (entrambe le telecamere in caso di stereoscopia).
L'ultima, invece, è in grado di avvertire l'assenza dell'oggetto e di cercarlo quando esso scompaia dal raggio di azione delle telecamere.

**Autolocalizzazione del robot**

Nell'ambito dell'autolocalizzazione sono stati dapprima sviluppati degli algoritmi per il riconoscimento automatico di landmark artificiali. Dopo aver collocato i landmark in punti noti dell'edificio del nostro dipartimento, è stato definito un percorso di prova che prevedeva l'attraversamento di porte, corridoi ed ostacoli di posizione già nota lungo il percorso.
L'intero percorso, compresi gli ostacoli, sono stati definiti mediante una mappa che potesse essere interpretata dal programma per far muovere il robot.
Facendo navigare il robot lungo tale percorso senza l'ausilio della visione, si è visto che questo non è in grado di ricostruire la sua posizione esatta all'interno della mappa con l'ausilio dei soli sensori odometrici di distanza: il robot, infatti, percorre meno strada rispetto a quella calcolata dagli encoder; inoltre, a causa degli interstizi tra le mattonelle del pavimento, commette errori notevoli non solo sulla distanza percorsa ma anche sul suo orientamento, dopo appena quattro metri di tragitto.
L'intervento della visione, invece, ha permesso di risolvere ogni problema di localizzazione del robot all'interno della mappa. Infatti, dopo essere giunto in prossimità di ciascun landmark, il sistema di visione è in grado di riconoscerlo e, grazie all'algoritmo di stereometria, è in grado di stimarne la posizione nel riferimento del robot con una precisione di pochi millimetri.
Con ciò il sistema corregge gli errori odometrici sulla posizione e sull'orientamento del robot e permette facilmente al robot di raggiungere ogni singola tappa fino alla destinazione.

## 6.2 Sviluppi futuri

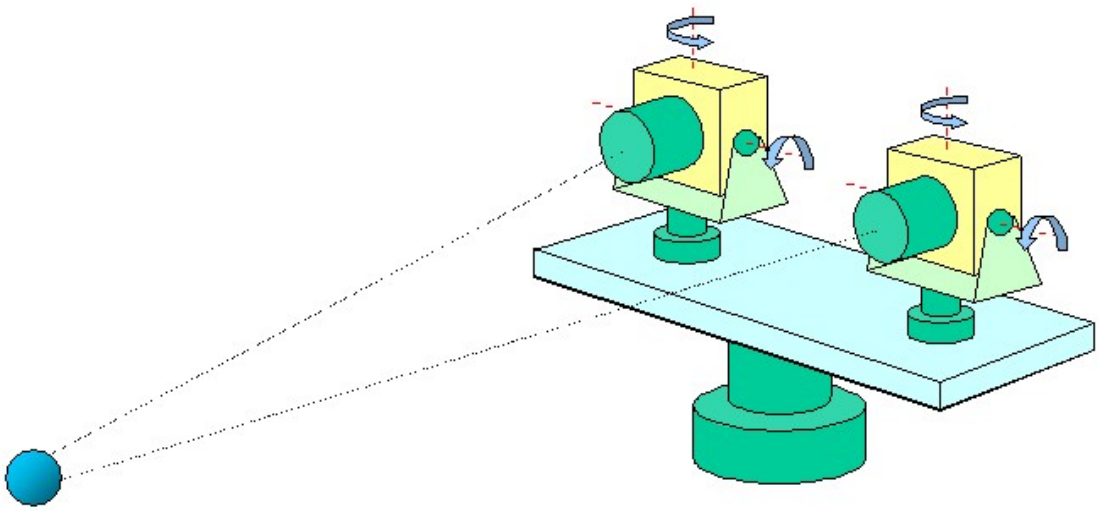
### 6.2.1 Punti deboli del sistema

I punti deboli del sistema riguardano il software di riconoscimento automatico degli oggetti. Nonostante che gli algoritmi di ball-detection funzionino molto bene in presenza dell'oggetto, essi soffrono ancora di falsi accoppiamenti quando questo non sia visibile. Ciò si deve in primo luogo alle modalità di estrazione dei contorni dell'immagine, che prevede un thresholding uniforme indipendentemente dalla luminosità e dal contrasto dell'immagine.

In secondo luogo le cause risiedono nella modalità di ricerca del centro dell'oggetto, che consiste nel massimizzare la funzione che descrive la trasformata Pixel-to-Pixel dell'immagine originale.

## 6.2.2 Applicazioni future

### 6.2.2.1 Pan & Tilt



Un primo miglioramento del sistema sarebbe quello di installare le telecamere su una struttura di tipo Pan & Tilt, in grado di garantire due gradi di mobilità alla vista del robot. In questo modo l'oggetto potrebbe essere inseguito o cercato girando semplicemente la "testa" piuttosto che il corpo del robot.

### 6.2.2.2 Puntamento laser per la calibrazione automatica

Nel capitolo 1 abbiamo visto che la struttura portante del sistema di visione è predisposta per quattro puntatori laser (figura 1.2) posti ai quattro vertici della T.
I puntatori costituirebbero la soluzione ottimale per una calibrazione automatica delle telecamere nella fase di inizializzazione del programma.
Infatti, nel capitolo 5, relativo all'autolocalizzazione, abbiamo visto che la conoscenza dell'angolo di inclinazione delle telecamere rispetto al pavimento costituisce un parametro fondamentale per invertire la prospettiva ed applicare la trasformazione invertente che rettifica le immagini. Dalle figure 5.11, 5.12, 5.13 si è visto che il landmark non viene individuato fintantoché non viene impostato il valore corretto dell'angolo, un'operazione, questa, che è stata finora effettuata manualmente, piegando opportunamente la testa stereoscopica.

Grazie ai laser tale procedura potrebbe essere fatta automaticamente.

Infatti i quattro spike luminosi proiettati dai laser sul pavimento possono essere usati dal sistema stereometrico per calcolare il piano 3D passante per tali punti, e da esso ricavare automaticamente l'angolo di inclinazione. In realtà sono sufficienti solamente tre punti per individuare il piano, ma il quarto potrebbe essere usato per verificare che non vi siano stati errori di misurazione per gli altri tre.

# Bibliografia


**LIBRERIE E MANUALI C**

[1] Shildt, "*C*", McGraw-Hill, 1989.

[2] B.P. Gerkey, R.T. Vaughan, K. Støy, A. Howard, M.J. Matarié, G.S. Sukhatme. *Most valuable Player: a robot device server for distributed control.* In Proc. of IEEE/RSJ Intl. Conf. on Intelligent Robots an Systems (IROS), pp. 1226-1231, Ottobre, 2001.

[3] Doxigen 1.3.4, *"libfg Referemce Manual 1.2",* http://antonym.org, Novembre 2003.

**IMAGE PROCESSING**

[4] A. Fusiello, *"Visione artificiale: appunti delle lezioni"*, Dipartimento di Informatica, Università di Verona, 2003.

[5] R.C. Gonzales, R.E. Woods, "*Digital Image Processing"*, Addison-Wesley Pub., 1992.

[6] Sonka-Hlavac-Boyle**,** *"Image Processing, Analysis and Machine Vision"***,** http://www.icaen.uiowa.edu/~dip/SYLLABUS/syllabus.html, 2004.

[7] *G. E. Christensen, "Digital Image Processing"*, http://www.icaen.uiowa.edu/~dip/.

[8] B. Green, *« Edge detection tutorial », http://www.pages.drexel.edu/~weg22/edge.html*

**COMPUTER VISION**

[9] I. Fantino, *"Computer Vision & Robotics",*

[10] L. Kourtis, M. Philiastides, Constantinos Stratelos, *« A semi-empirical billiards and pool tracker", http://www.stanford.edu/~kourtis/cs223b/cs223b.htm,*

**CALIBRAZIONE DELLE TELECAMERE**

[11] J.Y. Bouguet, *"Camera calibration Toolbox for Matlab"*, http://www.vision.caltech.edu/bouguetj/calib_doc/index.html.

[12] Z. Zhang, "A Flexible New Technique for Camera Calibration", IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 22, NO. 11, NOVEMBRE 2000.

[13] Z. Zhang, "Flexible Camera Calibration By Viewing a Plane From Unknown Orientations", Microsoft Research, One Microsoft Way, Redmond,WA 98052-6399, USA, 1999.

[14] R. Tsai, *"A versatile camera calibration techinique for high-accuracy 3D machine vision metrology using off-the-shelf tv cameras and lenses"*, IEEE Journal of Robotics and Automation, 3(4), 323-344, 1987.

[15] B. Caprile and V. Torre, *"Using vanishing points for camera calibration"*, International Journal of Computer Vision, 1990.

[16] M. Bertozzi, A. Brogli, A. Fascioli, *"Self-Calibration of a Stereo Vision System for Automotive Applications",* Dipartimento di Ingegneria dell'informazione, Università di Parma.

[17] P.I. Corke, *"Machine Vision Toolbox, for use with Matlab (release 1)"*, Division of Manufacturing Science and Tecnology, Queensland Center fir Advanced Technology, January 1999.

[18] D. Marr, *"Vision"*, Freeman, San Francisco, CA, 1982

[19] J. Heikkilä and O. Silvén, *"A Four-step Camera Calibration Procedure with implicit Image Correction",* Infotech Oulu and Department of Electrical Engineering University of Oulu.

[20] P. F. Sturm and S. J. Maybank, "On Plane-Based Camera Calibration: A General Algorithm, Singularities, Applications"

**DISTORSIONE RADIALE**

[21] D.C. Brown, "*Lens Distortion for Close-Range Photogrammetry***"** , Photometric Engineering, pages 855-866, Vol. 37, No. 8, 1971.

**MPEG**

[22] I.E.G. Richardson, *"H264 and MPEG-4, Video Compression, Video coding rof next generation multimedia"*, Wiley, 2003.

[23] I.E.G. Richardson, *"Video Codec Design, Developing Image and Video Compression Systems"*, Wiley, 2003.

**TRACKING**

[24] M. Bertozzi, A. Brogli, A. Fascicoli, G. Conte, *"Automatic Vehicle Guidance: the Experience of the ARGO Autonomous Vehicle"*

[25] S.M. Smith and J.M. Brady, *"Asset-2: Real Time Motion Segmentation and Shape Tracking"*, IEEE Transaction on pattern analysis and machine intelligence, vol. 17 no. 8, Agosto 1995.

[26] A. Hartshorne, *"Object Tracking in 3D",* Progress Repost, Dicembre 2004.

**INVERSE PERSPECTIVE MAPPING**

[27] M. Bertozzi, A. Brogli, A. Fascioli, *"Stereo inverse perspective mapping: theory and applications"*, Image and Vision Computing 16, 585-590, 1998.

[28] T. Bergener, C. Bruckhoff, C. Igel, *"Parameter optimization for visual obstacle detection using a derandomized evolution strategy"*.

[29] Z. Hu and K. Uchimura, *"Action-based road horizontal shape recognition"*, SBA Controle & Automacao vol. 10 no. 02, Giugno Luglio Agosto 1999.

[30] N. Brusco, A. Vidotto, E. Casagrande, A. Toniolo, V. Garbellotto, *"Progetto Khepera: riconoscimento di ostacoli, generazione di un percorso, controllo di un robot lungo il percorso"*, http://www.sitobrusco.com/khepera/index.html, 2001.

**VISIONE STEREOSCOPICA**

[31] A. Fusiello, E. Trucco, A. Verri, *"A compact algorithm for rectification of stereo pairs"*, Machine Vision and Applications 12, 16-22, 2000.

[32] R. Marfil, C. Urdiales, J. A. Rodriguez, F. Sandovai, "*Automatic Vergence Control Based on Hierarchical Segmentation of Stereo Pairs"*, Wiley Periodicals, Inc, vol. 13, 224-233, 2003.

[33] M. Bertozzi, A. Brogli, A. Fascicoli, "A stereo vision system for real time automotive obstacle detection", Dipartimento di Ingegneria dell'informazione, Università di Parma.

**BALL DETECTION**

[34] T. D'Orazio, N Ancona, G. Cicirelli, M. Nitti, *"A Ball Detection Algorithm for Real Soccer Image Sequences"*, IEEE, 1051-4651, 2002.

[35] G. Coath, P. Musumeci, *"Adaptive Arc Fittine for Ball Detection in RoboCup"*, 2002.

**CIRCLE DETECTION**

[36] Ali Ajdari Rad1, Karim Faez2, Navid Qaragozlou1, *"Fast Circle Detection Using Gradient Pair Vectors*", Proc. VIIth Digital Image Computing: Techniques and Applications, Dicembre 2003.

[37] Telespazio, "*Craters*", Telespazio-EO Doc.No. 190.193-SPA-ES-001 – Issue 1.0, Roma Luglio 2002.

[38] B.S. Morse, *"Segmentatio: Edge based Hough Transform",* Brigham Young University, 1998–2000

[39] H. Rhody, "*Circle Hough Transform*", Chester F. Carlson Center for Imaging Science Rochester Institute of Technology, Ottobre 2003

**LOCALIZZAZIONE CON LANDMARK**

[40] I. Shimshoni, *"On Mobile Robot Localization From Landmark Bearings"*, IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 18, NO. 6, DECEMBER 2002

[41] Y. Takeuchi and M. Hebert, *"Evaluation of Image-Based Landmark Recognition Techniques", t*he Robotics Institute Carnegie Mellon University, Luglio 1998.

**ROBOTICA**

[42] L.Sciavicco, B.Siciliano, "*Robotica Industriale*", McGraw-Hill, 2000

[43] V. Cagliati, *"Robotica",* Zanichelli.

**CONTROLLI**

[44] G.Marro, "*Controlli automatici*", Zanichelli, 2002.