UNIVERSITÀ DEGLI STUDI DI ROMA
TOR VERGATA

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA
DELL'AUTOMAZIONE

A.A. 2008/2009

Tesi di Laurea

RICONOSCIMENTO E LOCALIZZAZIONE DI OGGETTI MEDIANTE VISIONE STEREOSCOPICA

RELATORE

Prof. Francesco Martinelli

CANDIDATO

Stefano Pietrosanti

Indice

Ringraziamenti

Ringrazio il Prof. Martinelli per avermi concesso l'opportunità di poter scoprire il mondo della visione artificiale e per l'aiuto che mi ha fornito per questo lavoro di tesi. Un grazie ad Aurora per essermi stata vicina ed ai miei genitori per avermi sempre sostenuto ed incoraggiato.

Introduzione

I primi accenni di visione stereoscopica avvennero milioni di anni fa, quando l'elaborazione di due immagini provenienti da due diverse fonti rese possibile ai primi organismi viventi la percezione della distanza di una preda o di un ostacolo, grazie alla continua triangolazione eseguita dai centri nervosi.

La stessa tecnica è oggi usata utilizzando fotocamere e telecamere con lo scopo di localizzare nello spazio tridimensionale oggetti, ostacoli e riferimenti in contesti che non permettono di utilizzare altri tipi di strumenti come sensori di prossimità e fotocellule, oppure dove altre tecniche non garantiscono la versatilità e la robustezza necessaria (come i radar). La visione artificiale non necessita di intervenire sull'oggetto osservato aggiungendo tag RFID o fonti radio, né ha bisogno di mantenere i sensori in contatto o in prossimità dell'oggetto, per cui è un'ottima soluzione nel caso in cui si debba conoscere la posizione di elementi molto piccoli, caratterizzati da temperature molto elevate o semplicemente non contaminabili, oppure nel caso si debba conoscere la distanza da una serie di ostacoli o di riferimenti. Proprio quest'ultima caratteristica ha fatto sì che la visione stereoscopica venisse utilizzata per i primi esempi di guida automatica di veicoli, in cui una coppia di telecamere fornisce la distanza del mezzo dal riferimento dato dalla segnaletica stradale, con algoritmi che sono stati migliorati col tempo e che tuttora costituiscono l'ossatura del sistema di sensori delle ultime implementazioni di parcheggio assistito e guida assistita.

Il continuo incremento delle capacità di elaborazione dei microprocessori unito alla continua diminuzione del costo dei semiconduttori sta rendendo sempre più conveniente questa tecnica grazie alle enormi potenzialità che possiede, per questo si è scelto di sviluppare un algoritmo che elabori le immagini provenienti da due webcam commerciali per restituire la posizione nello spazio di oggetti di forma nota, essendo questa la base per ogni approccio alla visione stereoscopica.

Nel primo capitolo verranno trattate alcune tecniche di analisi dell'immagine per creare gli strumenti che permetteranno di trovare nelle immagini le posizioni di oggetti di forma nota, inoltre verrà analizzato un modello di sensore ottico così da poter conoscere la relazione tra la posizione degli oggetti e la loro proiezione nelle immagini catturate.
Nel secondo capitolo si passerà ad analizzare le distorsioni che si hanno nelle immagini catturate date dai difetti e dalla non idealità dei sensori utilizzati in questo testo, passando poi allo studio della geometria stereoscopica in cui le due immagini catturate dalle due webcam saranno in relazione tra loro in funzione della posizione dei sensori nello spazio, potendo così ricavare una relazione tra le proiezioni di un punto e le sue coordinate nello spazio tramite triangolazione.
Nel quarto capitolo si applicheranno tutte le tecniche studiate per poter ricavare un algoritmo di riconoscimento e localizzazione di oggetti che verrà poi applicato direttamente sulle immagini delle due webcam, analizzandone i risultati.

Capitolo 1

Algoritmi di analisi dell'immagine

Un problema fondamentale nella visione artificiale riguarda il riconoscimento di oggetti noti, essendo questa la base per ogni processamento di immagini partendo dalla semplice taratura del sensore fino alle più avanzate tecniche di videosorveglianza o guida automatica di veicoli. Le numerose tecniche sviluppate negli anni attingono da delle operazioni basiche di segmentazione, ossia di separazione di regioni nell'immagine

che possiedono caratteristiche o attributi simili, delle quali si possono elencare le principali:

- Segmentazione per **luminosità** nelle immagini in scala di grigi, ovvero dividere l'immagine in diverse parti, ognuna avente un diverso grado di brillantezza
- Segmentazione per **colore**, dove ogni regione è indicata dalla presenza di elementi all'interno di una certa scala cromatica
- Segmentazione attraverso il riconoscimento dei **contorni** di un elemento

L'unione di queste tre tecniche permette di creare algoritmi dall'elevato potenziale utilizzandone i risultati ad un livello d'astrazione superiore.

Esempio Prima di analizzare a fondo le tecniche citate, è utile proporre un esempio di applicazione delle stesse per evidenziare il contributo che possono dare in un contesto reale, quale ad esempio quello dato dalla necessità di dover riconoscere in una sequenza di frame di un filmato tratto da una webcam installata in un veicolo la presenza di segnali stradali, di forma e colore noti.
Si potrebbe partire con un'analisi della luminosità per rilevare se la sequenza di immagini è stata registrata di notte, avendo così un vantaggio dato dalla netta differenza di brillantezza dell'oggetto (dovuta all'illuminazione stradale e del veicolo oppure alla presenza di vernice catarifrangente sull'oggetto stesso) che ci permette di notare due picchi ben separati nell'istogramma dell'immagine, il primo dato dallo sfondo dell'immagine (come ad esempio il cielo, l'asfalto, gli edifici, etc.) permettendoci di estrarre una parte di immagine su cui poi continuare l'analisi.
In un secondo passaggio potremmo sfruttare il fatto di conoscere il colore del cartello stradale da identificare dividendo quindi l'immagine in regioni, ossia unioni di pixel

con caratteristiche simili, restringendo la zona su cui operare.

Nel terzo passaggio potremmo usare degli algoritmi di estrazione dei contorni, al fine di identificare figure geometriche che potrebbero corrispondere ai contorni dell'oggetto, in questo caso potrebbero interessarci delle circonferenze (per segnali di obbligo) o dei quadrilateri (per indicazioni stradali).

1.1 Segmentazione per luminosità

Tecnica utile nei casi in cui si abbiano oggetti dalla luminosità uniforme e con uno sfondo che si differenzi da essi, la segmentazione per luminosità permette di dividere l'immagine in **regioni** (le quali, come già precedentemente detto, sono insiemi di pixel che possiedono caratteristiche simili) o **segmenti** (cioè insiemi di pixel **contigui** con caratteristiche simili) sapendo a priori il livello di luminosità dei singoli oggetti da estrarre.

Nelle immagini bitmap in scala di grigi il valore del pixel ne indica la luminosità, per cui basta separare tutti i pixel tra coloro i quali hanno un valore inferiore ad un certo limite (per esempio appartenenti ad un eventuale sfondo nero) e quelli che invece lo superano (cioè un eventuale oggetto illuminato) oppure che si collocano in un certo intervallo di valori prefissati (oggetto in penombra). Per le immagini a colori, bisogna distinguere tra sorgenti **RGB** e **CMYK** – cioè a colori primari additivi rosso, verde e blu oppure colori primari sottrattivi ciano, magenta, giallo e nero – e sorgenti **HSB** – *hue*, *saturation*, *brightness* ovvero tonalità, saturazione, luminosità[1]. Le prime non esplicitano il valore di luminosità che deve essere quindi calcolato, mentre lo spazio HSB ha come sua terza componente il valore luminosità dato dal massimo tra i valori R, G o B. Per esempio, nelle immagini RGB, si può ricavare l'intensità con questa

[1] chiamato anche HSV, *hue, saturation, value*

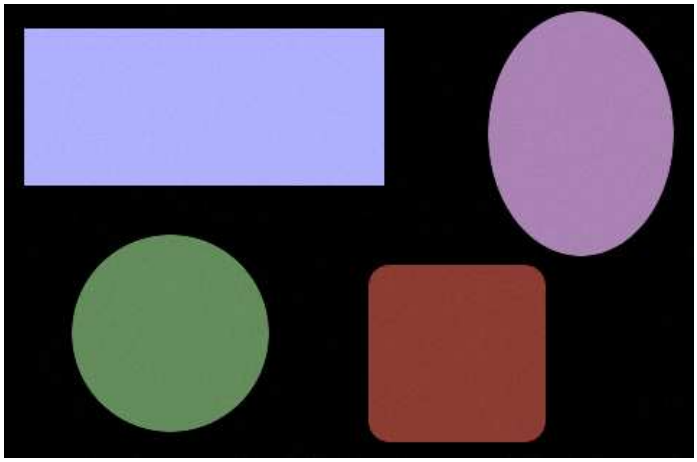


Figura 1.1: L'immagine di partenza, 4 figure geometriche.

formula:

$$I = R \cdot 0.3 + G \cdot 0.59 + B \cdot 0.11$$

Dove R,G,B sono i valori delle 3 componenti del pixel per ciascun colore.
Nelle immagini con metodo di colore HSB invece la componente *brightness* è esattamente il valore di luminosità che quindi si può utilizzare direttamente per l'analisi.

Una volta definito un intervallo di luminosità si può creare una cosiddetta **immagine binaria** in cui i pixel assumono solamente i valori 1 e 0, a seconda se il pixel dell'immagine di partenza rientri in questo intervallo o meno, e questa immagine può essere utilizzata come "maschera" per evitare che l'algoritmo nei passi successivi elabori anche i pixel che sono stati esclusi (perché per esempio appartenenti allo sfondo) o anche per continuare l'analisi solo sulla forma della regione quando non si hanno informazioni utili riguardo colore o trama dell'oggetto.

Esempio Riconoscimento di 4 forme di differente luminosità.

Come immagine di partenza (figura 1.1) prendo una serie di figure geometriche

colorate, su cui è stato applicato del rumore gaussiano per simulare disturbi di acquisizione. Il primo passo è quello di acquisire l'immagine in matlab e convertirla da RGB a 8 bit per colore in una matrice a 3 dimensioni (due spaziali ed una di colore) in codifica *double* con valori compresi tra 0 e 1, quindi ricavo la matrice di intensità (a due dimensioni) con una media pesata tra i tre valori di rosso, verde e blu. In alternativa avrei potuto ricavarla prendendo come valore di intensità il massimo tra i tre valori (la qual cosa sarebbe equivalente al trasformare l'immagine nello spazio HSB). Successivamente mostro l'istogramma per avere un raffronto sulle intensità dei vari oggetti.

```
img=imread('segmentazione1.bmp');
img=im2double(img);
I=rgb2gray(img);
imhist(I);
```

Nel secondo passo creo le quattro immagini binarie valutando se il pixel ha una luminosità tale da appartenere ad una certa figura, prendendo come riferimento le luminosità medie ricavate dall'istogramma. Il risultato però soffre del rumore introdotto artificialmente quindi vi è la presenza di un certo numero di pixel sparsi che vengono facilmente rimossi attraverso un'operazione morfologica di "open" (vedi Appendice A a pagina 75) la quale elimina questi difetti lasciando inalterato il resto dell'immagine.

```
A=(I > 0.65);
B=(I < 0.65 & I > 0.53);
C=(I < 0.53 & I > 0.41);
D=(I < 0.41 & I > 0.25);

A=bwmorph(A,'open');
B=bwmorph(B,'open');
C=bwmorph(C,'open');
D=bwmorph(D,'open');
```

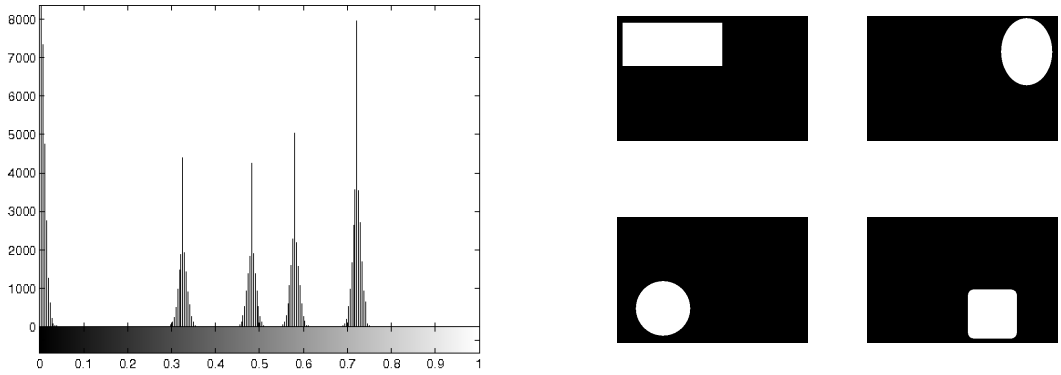


Figura 1.2: Istogramma dell'immagine e le quattro figure rilevate.

1.2 Segmentazione per colore

In caso di presenza di oggetti noti con zone di colore omogeneo è possibile segmentare l'immagine sfruttando questa particolarità, usando sensori che restituiscono immagini a colori. Come vedremo, questo è uno strumento molto potente in quanto il tono di colore tende a cambiare poco anche se in presenza di disturbi od ombre, e quest'ultimi sono sempre presenti nel caso reale.

Molto utile è la trasformazione dell'immagine nello spazio HSB, in quanto contiene direttamente l'informazione sul tono, con un valore che è compreso tra 0º e 360º che come si può immaginare è periodico, difatti lo spazio HSB si può immaginare come un cono e il valore di tonalità indica l'angolo in coordinate cilindriche rispetto all'asse di riferimento (per questo motivo, sia il valore di tonalità 0 sia 359 si riferiscono al colore rosso).

Trasformata l'immagine ed estrapolata la matrice con i valori di tonalità, il procedimento è identico alla segmentazione per luminosità.

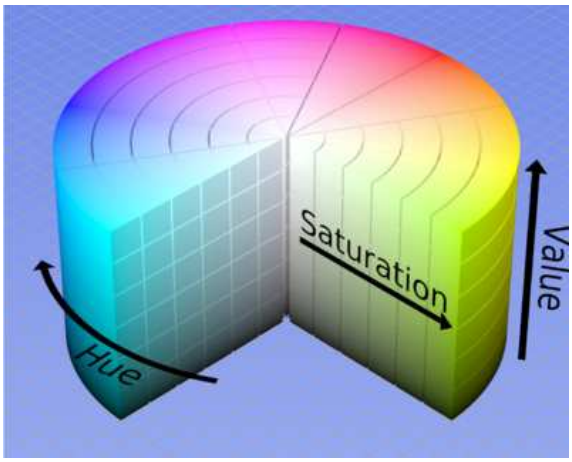


Figura 1.3: Lo spazio di colori HSB.

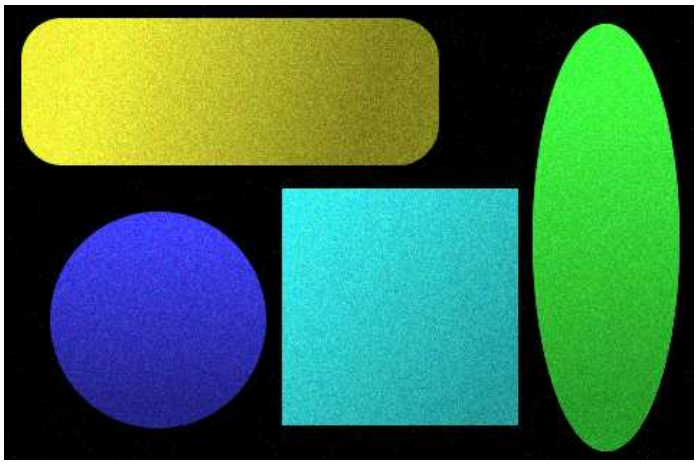


Figura 1.4: Quattro figure geometrice con presenza di rumore.

Esempio Riconoscimento di 4 forme di diverso colore attraverso la segmentazione per colore.

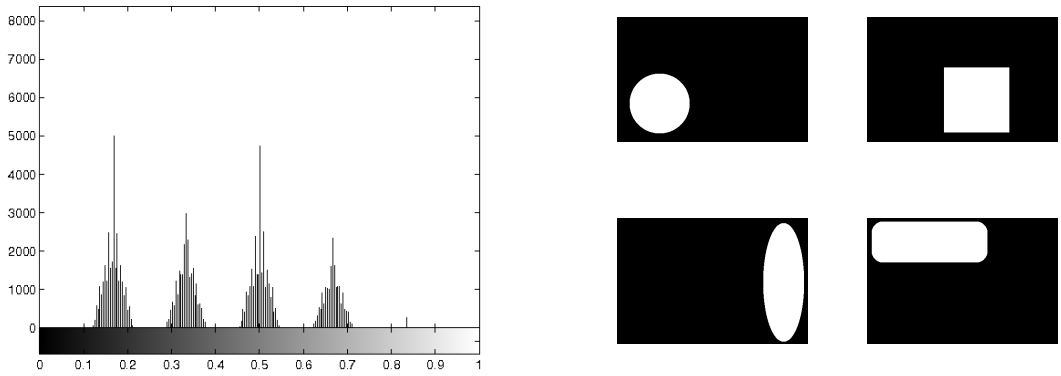
Come nel precedente esempio, si parte da un'immagine con 4 figure di colore diverso e vi ci si applica del rumore (figura 1.4), in questo caso molto più forte vista la migliore robustezza di questa tecnica. Il procedimento è identico alla segmentazione per luminosità, con l'unica differenza nella parte iniziale con la trasformazione nello spazio HSB e la successiva estrazione dei valori di tonalità. La funzione "rgb2hsv" converte un'immagine nello spazio di colori RGB in quello HSB, in modo da ottenere

una matrice "img" la cui prima coordinata sulla terza dimensione "img(:,:,1)" contiene l'informazione sulla tonalità.

```
img=rgb2hsv(img);
I=img(:,:,1);

A=(I > 0.60);
B=(I < 0.62 & I > 0.42);
C=(I < 0.42 & I > 0.25);
D=(I < 0.25 & I > 0.10);

A=bwmorph(A,'open');
B=bwmorph(B,'open');
C=bwmorph(C,'open');
D=bwmorph(D,'open');
```



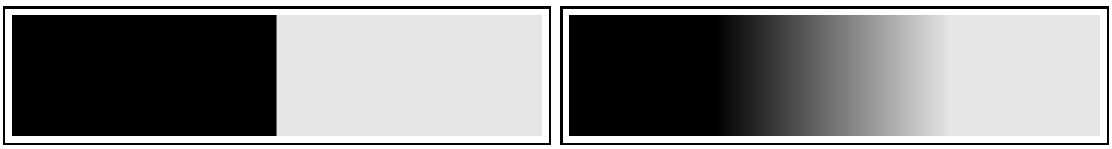
(a) istogramma dell'immagine di partenza (b) immagine risultante

Figura 1.5: Risultato della segmentazione per colore.

Alla stessa maniera che la segmentazione per tono di colore, è possibile segmentare un'immagine attraverso i suoi valori di saturazione del colore, per cui un oggetto potrebbe possedere un colore quasi saturo (tendente al colore puro), a differenza dello sfondo nel quale potrebbero predominare colori poco saturi (quindi tendenti al bianco/grigio)

1.3 Segmentazione attraverso il riconoscimento dei contorni di un oggetto

Le discontinuità nell'ampiezza della luminosità di un'immagine costituiscono una delle caratteristiche più importanti dato che forniscono informazioni riguardanti l'estensione fisica degli oggetti presenti nell'immagine, per questo motivo acquisisce molta importanza la capacità di un algoritmo di rilevare i **contorni** di un oggetto, approssimabili ad uno scalino (nella realtà la presenza di filtri passa-basso anti-aliasing riduce la pendenza dei contorni rendendo più fedele un'approssimazione di tipo a rampa, come indicato nella figura 1.6)



(a) contorno di tipo gradino (b) contorno di tipo rampa

Figura 1.6: Due tipi di possibili contorni di un oggetto.

1.3.1 Riconoscimento dei contorni di immagini in scala di grigi

Gradiente

Come primo passo per il riconoscimento dei contorni ci si può avvalere del calcolo del gradiente dell'immagine il quale avrà un valore basso in corrispondenza di zone con pochi cambiamenti di intensità. Applicando una sogliatura a quest'ultimo si è in grado di generare una nuova immagine contenente solo i contorni degli oggetti raffigurati, ignorando eventuali disturbi.

Primo ordine Il gradiente di primo ordine in un'immagine bidimensionale F(x,y) nel punto (x,y) si definisce come il vettore G:

$$G(F(x,y)) = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{dF}{dx} \\ \frac{dF}{dy} \end{bmatrix}$$

Questo vettore raggiunge la massima ampiezza quando la variazione è massima, quindi lo si può definire come:

$$G(F(x,y)) = ([G_x(x,y)]^2 + [G_y(x,y)]^2)^{\frac{1}{2}}$$

il quale può essere approssimata, per velocizzarne il calcolo:

$$G(F(x,y)) = |G_x(x,y)| + |G_y(x,y)|$$

Data la rappresentazione discreta dell'immagine si può approssimare l'operazione derivata come la **differenza all'indietro**, per cui la derivata $g(x) = \frac{df(x)}{dx}$ è data da:

$$g(x) = f(x) - f(x-1)$$

Quindi il gradiente G può essere definito come:

$$G(F(x,y)) = |F(x,y) - F(x-1,y)| + |F(x,y) - F(x,y-1)|$$

Si può altresì notare che le derivate in entrambe le direzioni possono essere espresse come due matrici di convoluzione spaziale:

$$G_y(x,y) = F(x,y) - F(x,y-1) \longrightarrow \begin{bmatrix} 0 & -1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$G_x(x,y) = F(x,y) - F(x-1,y) \longrightarrow \begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Esempio Applico ad un'immagine di prova, tramite convoluzione, il calcolo del gradiente.

Carico un'immagine binaria, raffigurante le 4 figure del primo esempio (figura 1.7(a)), nella matrice I e creo i filtri spaziali che rappresentano la derivata per differenza all'indietro, quindi applico la convoluzione spaziale tramite il comando "filter2".

```
Gy=[0 -1 0; 0 1 0; 0 0 0];
Gx=[0 0 0; -1 1 0; 0 0 0];
Y=filter2(Gy,I);
X=filter2(Gx,I);
```

Calcolo il gradiente e ne normalizzo i valori (per mantenerli nel range [0,1])

```
G=abs(X)+abs(Y);
G=(G-min(min(G)))/(max(max(G))-min(min(G)));
```

A questo punto applico la *sogliatura*, considerando contorno solo il pixel il cui gradiente in quel punto è superiore al valore normalizzato di 0.1.

```
G=(G>0.1);
```

Nella figura 1.7 si mostra l'applicazione dell'algoritmo su un'immagine di esempio, mentre nella figura 1.8 lo stesso algoritmo lo si è applicato su un'immagine fotografica.

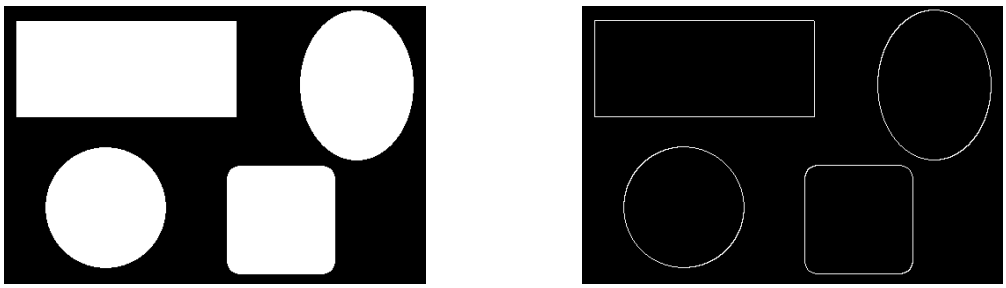
Secondo ordine Per la derivata di secondo ordine si utilizza l'operatore laplaciano, definito come:

$$\Delta f(x) = div(grad(f(x))) = \bigtriangledown(\bigtriangledown f(x)) = \bigtriangledown^2 f(x)$$

$$\Delta = \bigtriangledown^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

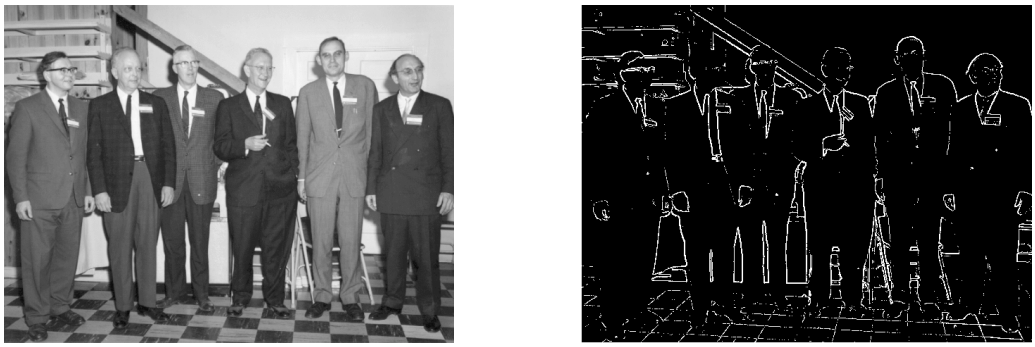
Il quale in un'immagine a due dimensioni nel dominio continuo si definisce come:

$$L(x, y) = \bigtriangledown^2 F(x, y)$$



(a) immagine di partenza (b) immagine risultante

Figura 1.7: Contorni rilevati tramite calcolo del gradiente di ordine primo, immagine d'esempio.



(a) immagine di partenza (b) immagine risultante

Figura 1.8: Contorni rilevati tramite calcolo del gradiente di ordine primo, immagine fotografica.

Utilizzando nuovamente l'approssimazione data dalla differenza all'indietro, posso definire l'operatore come:

$$L(x,y) = [F(x,y) - F(x,y-1)] - [F(x,y+1) - F(x,y)] + [F(x,y) - F(x+1,y)] -$$

$$-[F(x-1,y) - F(x,y)] =$$

$$= 4F(x,y) - F(x,y-1) - F(x,y+1) - F(x+1,y) - F(x-1,y)$$

Nell'immagine digitale si può ottenere con facilità il risultato approssimato attraverso la convoluzione della seguente maschera:

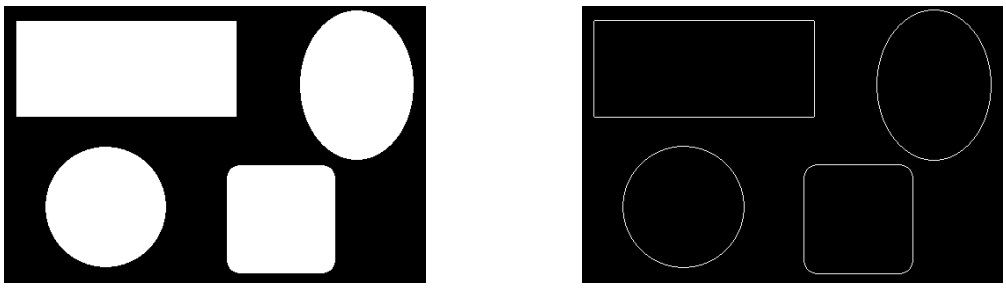
$$H = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Esempio Applico il calcolo del gradiente di secondo ordine.

Il procedimento è identico a quello dell'esempio precedente, con l'unica differenza di applicare all'immagine un unico filtro dato dalla seguente matrice:

```
H=[0 -1 0; -1 4 -1; 0 -1 0];
```

Viene quindi applicata una sogliatura a 0.255 per ottenere l'immagine dei contorni (figure 1.9 e 1.10).

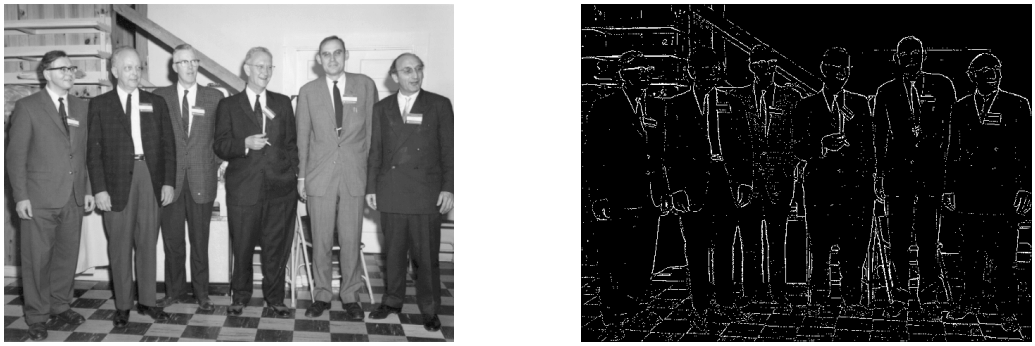


(a) immagine di partenza (b) immagine risultante

Figura 1.9: Contorni rilevati tramite calcolo del gradiente di ordine secondo, immagine d'esempio.

Algoritmo di Canny

Nel 1986 lo statunitense *John F. Canny* progettò un algoritmo per il riconoscimento dei contorni che è ormai definito come lo standard in questo campo, per questo motivo ne viene offerta un'analisi dettagliata preferendolo ad altri algoritmi che nella maggioranza dei casi non offrono risultati migliori. L'algoritmo si divide in quattro fasi:



(a) immagine di partenza (b) immagine risultante

Figura 1.10: Contorni rilevati tramite calcolo del gradiente di ordine secondo, immagine fotografica.

1. Riduzione del rumore
2. Ricerca del gradiente di luminosità dell'immagine
3. Soppressione dei non-massimi
4. Individuazione dei contorni mediante sogliatura con isteresi

Prima fase: riduzione del rumore Un problema primario negli algoritmi di riconoscimento dei contorni è dato dalla presenza di rumore nelle immagini non processate, per cui come primo passo si applica all'immagine un filtro spaziale, tramite il processo di convoluzione già visto, il cui scopo è rimuovere le alte frequenze su cui il rumore interferisce in maniera più problematica. Per far ciò viene scelto un filtro gaussiano il cui valore di deviazione standard può essere modificato in modo da prediligere il riconoscimento di bordi piccoli e netti oppure più grandi e graduali (nell'implementazione che ne fornisce Matlab ha un valore di deviazione standard predefinito $\sigma = 1$, mentre la dimensione del filtro viene scelta automaticamente a seconda del valore di σ, dato che quest'ultimo ne definisce la velocità con cui i valori tendono a 0) ed il risultato di

questa prima fase è un'immagine leggermente sfuocata, in cui nessun pixel è affetto dal rumore in maniera significativa.

Seconda fase: ricerca del gradiente della luminosità dell'immagine È già stata data in precedenza la definizione di gradiente, in particolare i gradienti calcolati sull'asse x ed y (G_x, G_y) di cui si riportano per comodità le definizioni:

$$G_x(x,y) = \frac{dF(x,y)}{dx}$$
$$G_y(x,y) = \frac{dF(x,y)}{dy}$$
$$G = \sqrt{G_x^2 + G_y^2}$$

Nella maggior parte delle implementazioni i valori dei gradienti nelle due dimensioni vengono calcolati utilizzando l'operatore di Sobel:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$
$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Questo operatore garantisce una discreta approssimazione pur mantenendo bassa la richiesta di calcolo. Per migliori approssimazioni sarebbe opportuno utilizzare differenti tecniche che però aumentano enormemente la quantità di tempo necessario per processare un'immagine e necessitano del calcolo di numeri irrazionali.

Si definisce θ l'angolo di direzione del contorno:

$$\theta = \arctan\left(\frac{G_y}{G_x}\right)$$

Il valore di θ viene arrotondato ai valori di 0, 45, 90 e 135 rappresentanti rispettivamente i bordi verticali, orizzontali e lungo le due diagonali.

Questa fase restituisce per ogni pixel la direzione, arrotondata, di massimo gradiente con il valore di quest'ultimo.

Terza fase: soppressione dei non-massimi In questa fase si azzerano i valori dei pixel non considerati parte del contorno, cioè i pixel il cui valore di intensità non è maggiore di quello dei pixel adiacenti situati lungo la direzione data dal valore θ in quel punto. Il risultato è un'immagine binaria con una linea sottile in corrispondenza dei bordi degli oggetti nell'immagine.

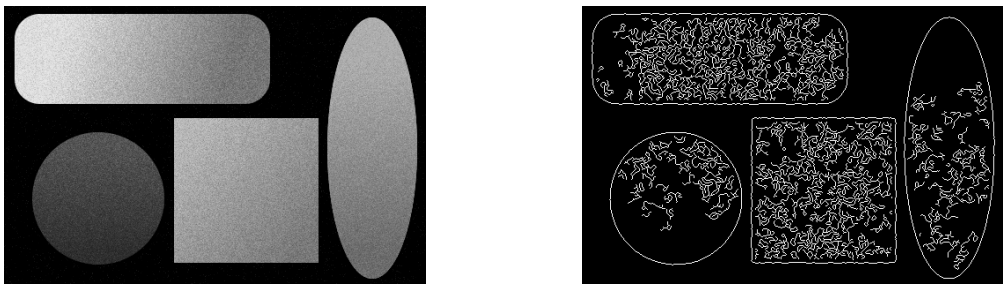
Quarta fase: individuazione dei contorni mediante sogliatura con isteresi Una debolezza riguardante gli algoritmi di riconoscimento di contorni è data dal valore di sogliatura da assegnare per ottenere un buon risultato. Negli esempi dei capitoli precedenti questo valore è stato determinato empiricamente, cercando di ottenere il miglior risultato possibile, ma in generale non si può prevedere quale sia il valore minimo che debba avere il gradiente affinché il pixel considerato sia parte del contorno. Per attenuare questa debolezza si utilizza un metodo di sogliatura con isteresi, che prevede due valori: il valore di soglia alta e quello di soglia bassa. Ogni punto il cui gradiente sia superiore al valore di soglia alta è automaticamente definito parte del contorno (ne viene assegnato il valore 1), inoltre ogni punto contiguo ad un punto del contorno che abbia un valore del gradiente superiore al valore di soglia bassa entra a far parte del contorno, assegnando così valore 0 a tutti i punti rimanenti e quelli al di sotto della soglia bassa. Il risultato è l'immagine binaria dei contorni.

Esempio Matlab implementa l'algoritmo di *Canny* nella funzione "edge", con la possibilità di variare il valore della deviazione standard σ e i valori di soglia. Nella figura 1.11 vengono evidenziati in modo corretto i bordi di un cartello autostradale.

(a) immagine di partenza (b) immagine risultante

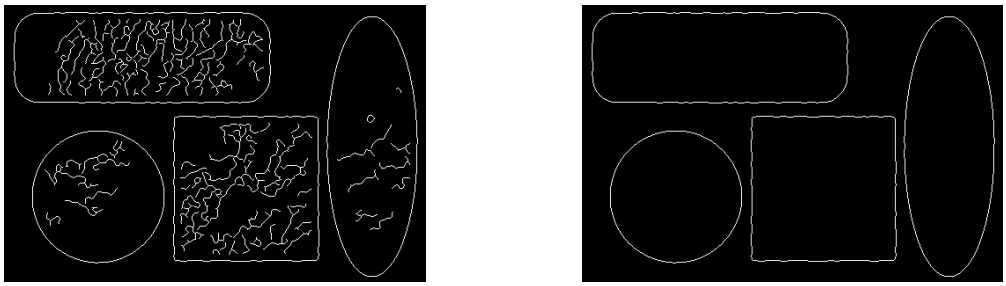
Figura 1.11: Bordi rilevati dall'algoritmo di Canny.

Nella figura 1.12(b) si nota come in presenza di immagini fortemente rumorose (come quelle nella figura 1.12(a)) l'algoritmo non riesca a distinguere i bordi delle figure dal rumore all'interno delle stesse. Modificando il parametro σ a 1.8 otteniamo un miglioramento con una drastica riduzione dei falsi positivi, dato dal fatto che aumentando la deviazione standard si applica un filtro gaussiano più "ampio" con un maggior effetto passa-basso, eliminando così la maggior parte dei disturbi (figura 1.12(c)). Nell'ultimo caso (figura 1.12(d)) oltre al valore di deviazione standard si modifica anche il valore di soglia alta, passando da 0.0469 (scelto automaticamente dall'algoritmo implementato in Matlab) a 0.06 costringendo l'algoritmo a generare il contorno a partire da punti con gradiente maggiore, favorendo i contorni regolari dell'immagine (dati dai bordi delle figure) e sfavorendo quelli irregolari (dati dal rumore inserito artificialmente).



(a) immagine di partenza

(b) parametri predefiniti



(c) sigma = 1.8

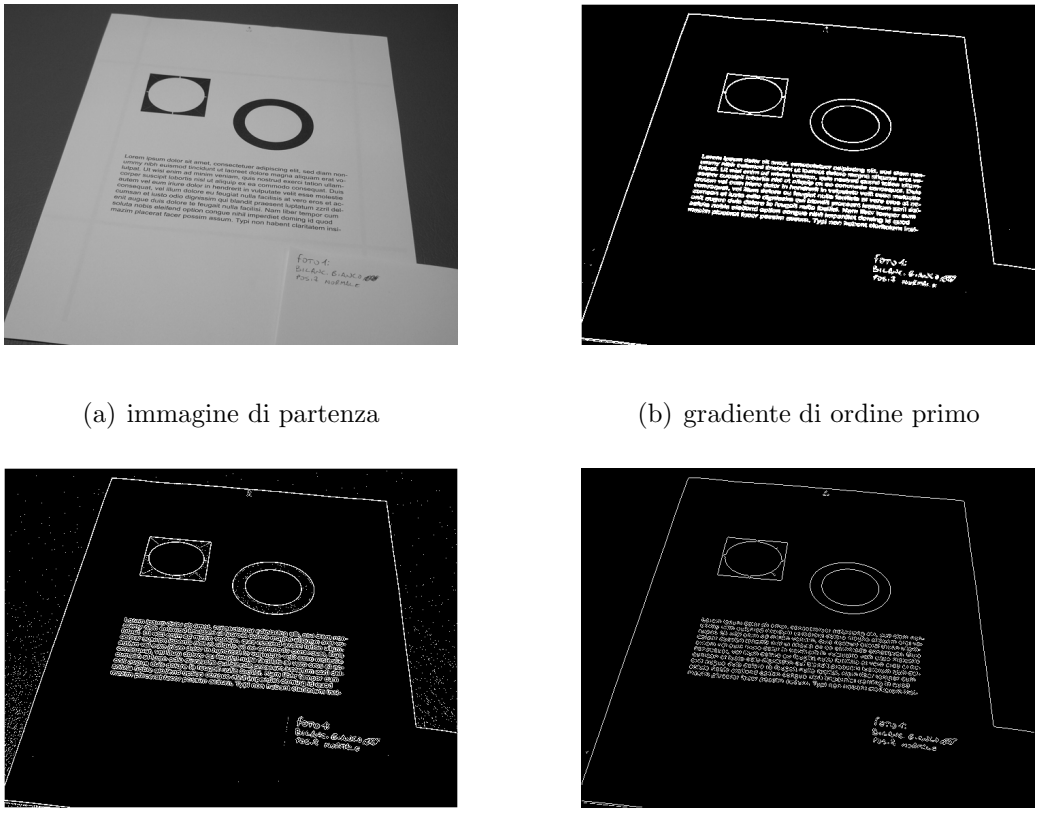
(d) sigma = 1.8, soglia alta a 0.06

Figura 1.12: Risultati dell'algoritmo di Canny in funzione dei parametri di deviazione standard e sogliatura.

Confronto tra gli algoritmi di riconoscimento dei contorni Nella figura 1.13 viene mostrato il risultato dell'applicazione di differenti algoritmi di riconoscimento dei contorni. La (a) mostra l'immagine originale, mentre le (b) e (c) sono ottenute mediante gradiente di, rispettivamente, primo e secondo ordine. Infine nella figura 1.13(d) viene mostrato il risultato dell'applicazione dell'algoritmo di *Canny* che da un'analisi qualitativa risulta decisamente migliore rispetto ai precedenti.

1.3.2 Riconoscimento dei contorni di immagini a colori

Nelle immagini a colori vi possono essere differenti criteri per definire la presenza di un contorno. Esso potrebbe essere dato dai contorni della componente di luminosità,



(a) immagine di partenza

(b) gradiente di ordine primo

(c) gradiente di ordine secondo

(d) algoritmo di Canny

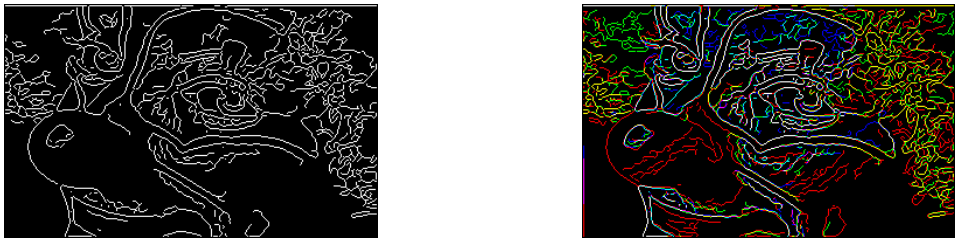
Figura 1.13: Confronto tra i diversi algoritmi di riconoscimento dei contorni.

quindi riconducendo il problema ad un già trattato riconoscimento di bordi in immagini in scala di grigi, escludendo però dei bordi dati dalle variazioni di saturazione e tono del colore. Un altro criterio potrebbe essere quello di rilevare i contorni per ogni componente primaria (RGB, rosso, verde e blu) quindi unirne i risultati, limitando così gli eventuali contorni non rilevati con la singola luminosità.

In generale, se non si prevedono immagini con repentini cambi di saturazione e tono e luminosità costante, conviene applicare gli algoritmi sulla singola matrice di intensità luminosa risparmiando così tempo di calcolo.

Esempio Applico l'algoritmo di *Canny* su di un'immagine a colori (figura 1.14) utilizzando i due criteri citati, prima trasformando l'immagine a colori in scala di grigi e poi ricercando i bordi nelle immagini delle tre componenti RGB.
Possiamo vedere come alcuni contorni dell'immagine vengano rilevati dall'applicazione dell'algoritmo sulle singole componenti (figura 1.15(a)) ma non vengano rilevati sull'immagine in scala di grigi (figura 1.15(b)). Nel secondo esempio, applicando lo stesso algoritmo, si nota come non ci siano differenze tra i due criteri tranne delle imprecisioni nei contorni della fascia arancione (figura 1.17).

Figura 1.14: Immagine a colori di partenza.



(a) scala di grigi (b) componenti singole

Figura 1.15: Risultati dell'algoritmo di Canny per immagini a colori.

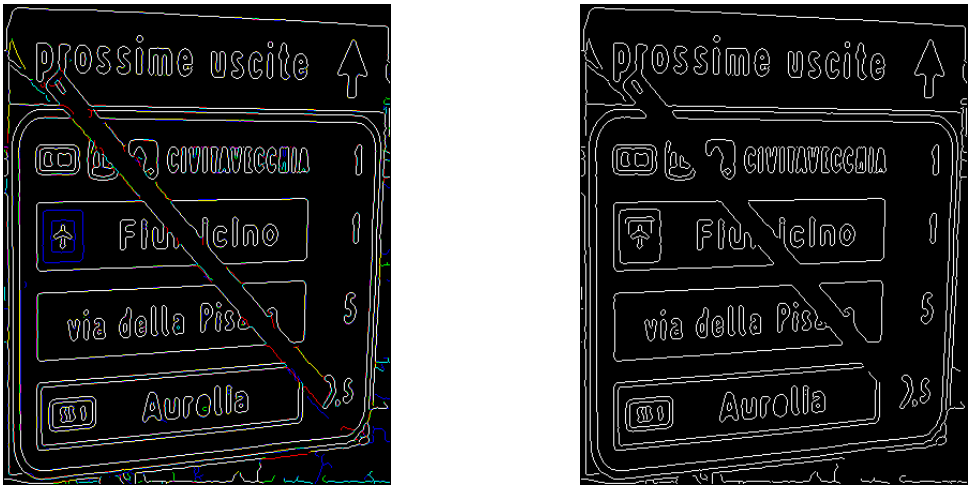
Figura 1.16: Immagine a colori di partenza.

1.4 Trasformata di Hough

Ottenuta l'immagine binaria dei contorni nell'immagine passiamo all'analisi globale il cui obiettivo è il riconoscimento di forme note, assumendo che gli oggetti ritratti possano essere ricondotti a unioni di figure geometriche (il che vale, con minore o maggiore complessità, per ogni oggetto di forma costante), i cui parametri verranno processati ad un livello d'astrazione superiore. Una tecnica molto utilizzata in questo ambito è l'utilizzo della trasformata di *Hough* la quale produce una rappresentazione dell'immagine nello spazio dei parametri delle figure geometriche che ci siamo preposti di cercare, le quali in un primo momento si riducevano alle singole rette mentre successive implementazioni permettono di trovare figure più complesse, generalmente circonferenze ed ellissi.

Riconoscimento di rette

Nelle coordinate cartesiane una retta può essere definita tramite la funzione $y = mx + q$ quindi nello spazio dei parametri essa occupa il punto (m, q), dove m indica il coefficiente angolare mentre q indica la coordinata di intersezione con l'asse y. Ogni



(a) scala di grigi (b) componenti singole

Figura 1.17: Risultati dell'algoritmo di Canny per immagini a colori.

punto (x_i, y_i) ha infinite rette passanti per lo stesso le quali possono essere definite dall'equazione $y_i = mx_i + q$ oppure, in un modo più conveniente, come $q = -mx_i + y_i$ definendole così nel nostro spazio dei parametri. Nelle immagini binarie dei contorni ogni pixel con valore '1' indica che è stato rilevato un contorno in quel punto quindi ci si aspetta la possibile presenza di una retta-contorno di cui quel pixel fa parte, così possiamo aumentare di un valore la mappa di accumulazione bidimensionale sullo spazio dei parametri per tutti i valori di m e q che soddisfano l'equazione delle rette passanti per (x_i, y_i). Se un gruppo esteso di pixel con valore '1' giace sulla stessa retta avremo un massimo in corrispondenza del punto (m_i, q_i) di cui m_i e q_i sono i parametri, potendo affermare con una buona approssimazione che nell'immagine è presente tale retta. Il fatto che si ricorra a cercare un valore di massimo nella mappa di accumulazione ci fa capire che è possibile riconoscere rette anche quando l'immagine dei contorni contenga molti falsi-positivi e falsi-negativi.

Un problema computazionale nella rappresentazione nello spazio (m, q) è dato dal

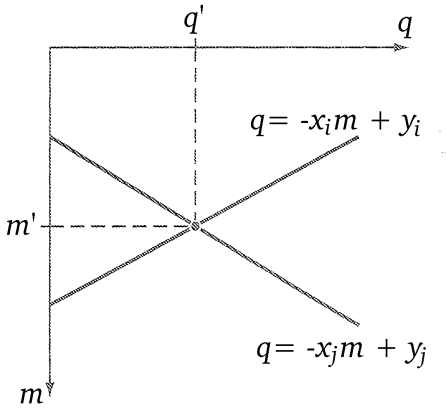


Figura 1.18: Rette nello spazio dei parametri (m,q).

fatto che il valore del coefficiente angolare non è limitato e potrebbe essere infinito, perciò si definisce ogni retta attraverso i parametri ρ e θ, in cui il primo indica la distanza della retta dall'origine mentre il secondo è l'angolo formato con l'asse x dal vettore che congiunge l'origine con il punto della retta meno distante da esso. Usando questi parametri ogni retta è definita come:

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right) x + \left(\frac{\rho}{\sin\theta}\right)$$

oppure:

$$\rho = x\cos\theta + y\sin\theta$$

Quindi ogni retta in questa rappresentazione è definita da un'unica coppia (ρ, θ) con θ compreso tra 0 e π, e $\rho \in R$. Ripetendo il discorso precedente con i nuovi parametri possiamo affermare che per ogni punto (x_i, y_i) passano infinite rette definite da:

$$\rho(\theta) = x_i\cos\theta + y_i\sin\theta$$

cioè una sinusoide, quindi più punti giacenti sulla stessa retta con parametri (ρ_i, θ_i) produrranno differenti sinusoidi che si intersecano nel punto (ρ_i, θ_i). In generale pos-

siamo rilevare la presenza di rette nell'immagine cercando i massimi nello spazio di *Hough*, cioè nello spazio dei parametri.

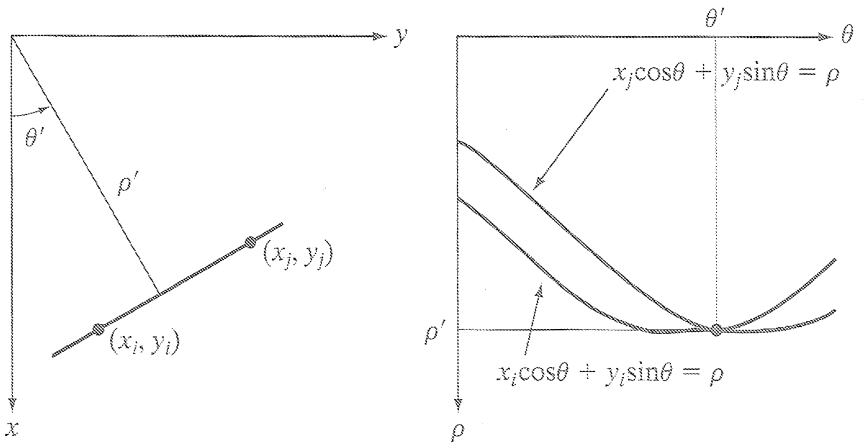


Figura 1.19: Rette nello spazio dei parametri (ρ, θ) e punto di accumulazione in (ρ', θ').

Esempio L'algoritmo di trasformazione nello spazio (ρ_i, θ_i) è già efficacemente implementato in Matlab attraverso la funzione "hough()" che riceve in ingresso l'immagine binaria e altri parametri (come ad esempio la risoluzione angolare) e restituisce l'immagine nello spazio di *Hough* più i vettori theta e rho che contengono i valori su cui è stata generata l'immagine.
Partendo dall'immagine binaria I (figura 1.20(a)) applichiamo la funzione:

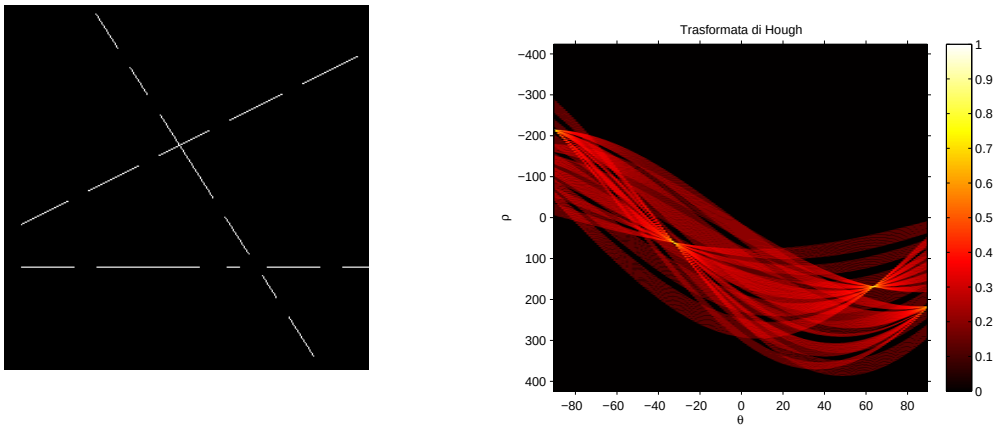
```
[J,theta,rho]=hough(I);
```

Otteniamo quindi direttamente la rappresentazione parametrica (figura 1.20(b)) su cui possiamo osservare i massimi presenti nei seguenti punti:

1. theta = -32° , rho = 57
2. theta = 63° , rho = 167
3. theta = 90° (0°) , rho = 213

Con un'analisi qualitativa possiamo facilmente osservare che questi valori di θ e ρ sono esattamente i parametri delle rette presenti nelle immagini. Per evidenziare le sinusoidi (di valori molto bassi) date dai singoli punti si è mostrato a video il logaritmo naturale dell'immagine di uscita, nella realtà i massimi hanno valori nettamente superiori al valore medio dell'immagine.

```
imshow(mat2gray(log(J+1)),'XData',theta,'YData',rho,...
      'InitialMagnification','fit');
```



(a) immagine di partenza (b) trasformata di Hough

Figura 1.20: Applicazione della trasformata di Hough.

Riconoscimento di circonferenze

Per il riconoscimento di circonferenze si utilizza la cosiddetta *trasformata di Hough generalizzata* che utilizza un numero di parametri maggiore, aumentando anche la complessità. Come dice il nome, questa trasformata può essere adattata al rilevamento di ogni forma geometrica definibile da un'equazione a due incognite ed n parametri, e nel nostro caso questa equazione è:

$$r^2 = (x - a)^2 + (y - b)^2$$

la quale definisce appunto il luogo dei punti di una circonferenza e come si può vedere presenta tre parametri, "r", "a" e "b". Il primo indica il raggio della circonferenza mentre "a" e "b" sono le coordinate, lungo gli assi "x,y", del centro della stessa. La rappresentazione parametrica è la seguente:

$$x = a + r\cos\theta$$

$$y = b + r\sin\theta$$

Lo spazio di *Hough* sarà quindi a tre dimensioni e ciò può portare problemi nel calcolo computazionale dovuto all'enorme quantità di memoria necessaria all'algoritmo, quindi si preferisce limitare l'algoritmo al calcolo di un numero limitato di raggi.
Come nella ricerca di rette, si utilizza una matrice di accumulazione su cui si traccia, per ogni punto del contorno dell'immagine di partenza, una circonferenza con centro in quel punto e raggio desiderato. Quando questo raggio coincide con quello della circonferenza da rilevare, avremo un punto nello spazio di *Hough* con valore massimo che coinciderà con il centro della circonferenza da rilevare, quindi in generale possiamo reiterare l'algoritmo per ogni valore di raggio possibile ed avremo nella matrice di accumulazione un unico massimo nelle coordinate (x,y) del centro della circonferenza presente nell'immagine originale. In presenza di più circonferenze avremo più picchi e basterà un'analisi intelligente del risultato per ottenere le posizioni e le caratteristiche delle circonferenze.

Esempio Partendo da un'immagine con la presenza di 3 circonferenze di raggio dato, estraggo le coordinate dei centri delle stesse. Parto dall'immagine binaria dei contorni I (figura 1.22) ed eseguo l'algoritmo sopra descritto creando un'immagine di accumulazione totale M (figura 1.23(a)) creata dalla somma pesata degli accumulatori

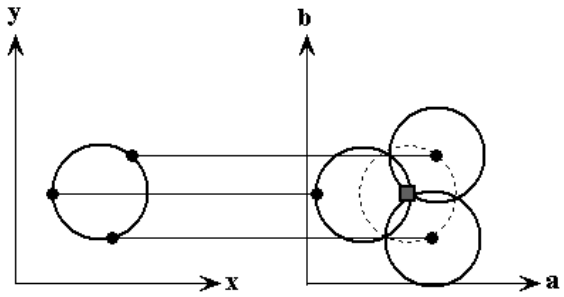


Figura 1.21: Proiezione dei punti di una circonferenza nello spazio di Hough.

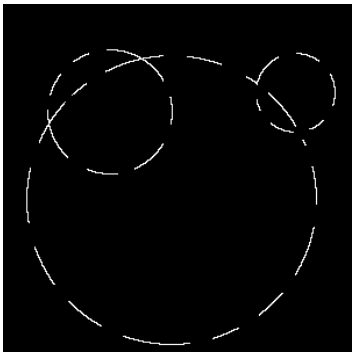
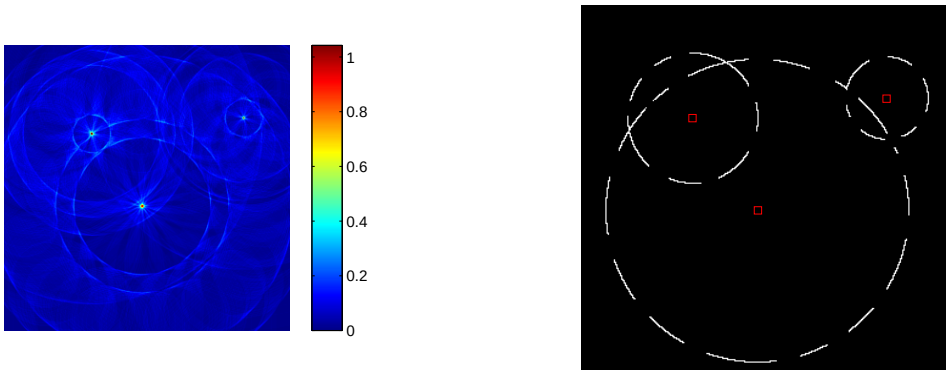


Figura 1.22: Immagine di partenza.

degli output per i differenti valori di raggio (ogni accumulatore pesa in modo uguale nell'immagine finale, con valori compresi tra 0 e 1, ottenendo quindi tre picchi di uguale intensità). La funzione hough_cerchi crea questa immagine di accumulazione, mentre l'output N è una matrice di accumulazione a somma non pesata, che verrà utilizzata in seguito.

```
raggi=[125 34 53];
[M,N]=hough_cerchi(I,raggi);
```

Esempio Data un'immagine binaria dei contorni di una circonferenza di raggio non noto (ma di cui si stima la presenza all'interno di un certo intervallo) ne ricaviamo



(a) trasformata di Hough (b) immagine originale con i centri rilevati

Figura 1.23: Applicazione della trasformata di Hough generalizzata.

le coordinate del centro. Inoltre, per mostrare la robustezza dell'algoritmo, viene aggiunta all'immagine del rumore del tipo "salt & pepper" che crea dei pixel bianchi (valore 1) aggiunti casualmente all'immagine di partenza (figura 1.24). Viene indicato il numero di raggi su cui ripetere l'algoritmo (indicando quindi la risoluzione) e l'intervallo di raggi tra cui cercare, assegnando un minimo di 80 pixel (cioè qualitativamente il raggio di una piccola circonferenza) ed un massimo dato dalla metà della minima dimensione dell'immagine (dato che sappiamo che la circonferenza è completamente all'interno dell'immagine), quindi generiamo il vettore contenente i raggi da analizzare.

```
n_raggi=30;
max_raggio=0.5*min(size(I));
min_raggio=80;
tmp=1:n_raggi;
raggi=min_raggio:round(100/20):max_raggio;
```

Applico quindi l'algoritmo e ne estraggo la matrice di accumulazione data dalla somma di tutte le matrici di accumulazione (figura 1.25(a)) di cui cerco il massimo ricavandone la posizione, quindi ricostruisco una circonferenza sull'immagine originale utilizzando

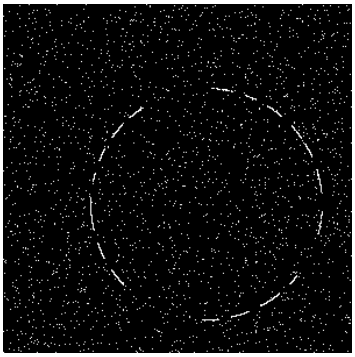
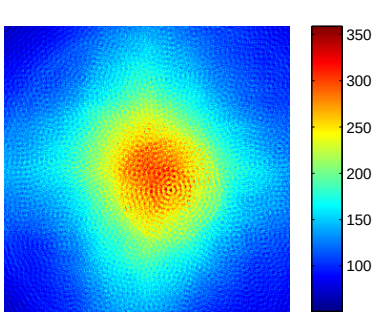
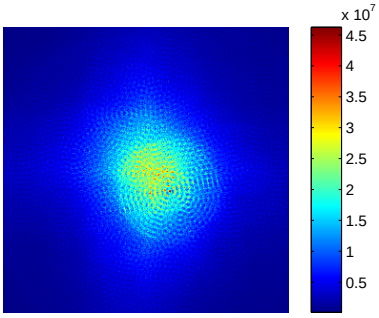


Figura 1.24: Immagine di partenza.

l'algoritmo "Midpoint Circle"[2] con il valore di raggio reale per verificare i risultati ottenuti (figura 1.25).



(a) trasformata di Hough



(b) trasformata di Hough, evidenziando i valori più alti

1.5 Modello del sensore ottico

La coppia di sensori utilizzata per l'acquisizione delle immagini consiste di due web-cam *Logitech Quickcam E2500* con sensore CMOS le quali presentano una lente che

[2] `http://en.wikipedia.org/wiki/Midpoint_circle_algorithm`

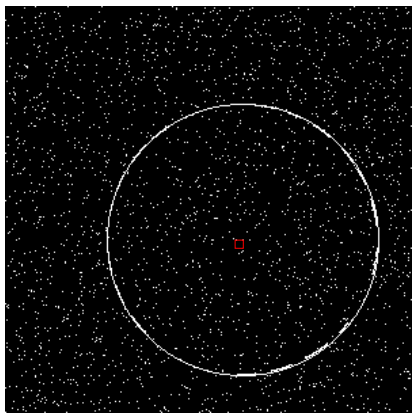


Figura 1.25: Immagine originale con circonferenza ricostruita.

ha lo scopo di mettere a fuoco (manualmente) un oggetto posto su di un preciso piano perpendicolare all'asse ottico. Questo tipo di webcam consente di modificare la messa a fuoco muovendo la lente ma per mantenere costanti i parametri si è deciso di mantenerle fisse per tutte le prove in una configurazione atta a mantenere a fuoco oggetti disposti ad una distanza di circa 30 cm dal sensore.

1.5.1 Modello semplificato

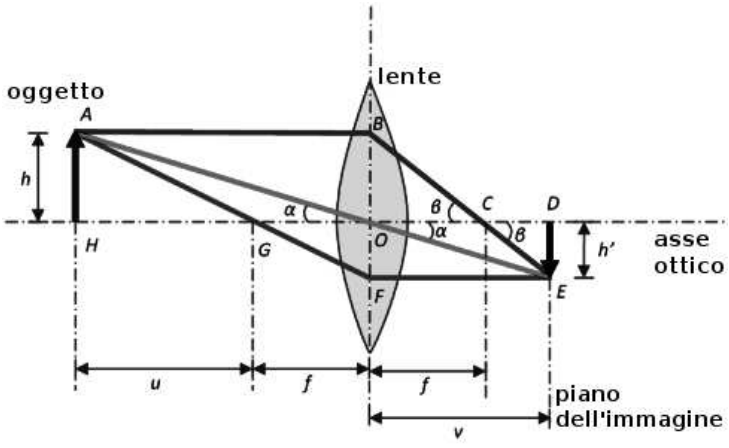


Figura 1.26: Costruzione dell'immagine di un punto.

L'ottica della webcam viene approssimata a una **lente sottile** di spessore infinitesimo che ci permette di assumere che tutti i raggi diretti alla webcam paralleli all'asse ottico vengano rifratti un punto detto **fuoco** tranne i raggi passanti per il centro ottico (che passano quindi inalterati) come viene mostrato in figura 1.26. Grazie alle proprietà di similitudine tra triangoli è possibile stabilire che:

$$\frac{1}{u} + \frac{1}{v} = \frac{1}{f}$$

e dato che la distanza f è presumibilmente di qualche ordine di grandezza minore della distanza $u + f$ dell'oggetto, si può stimare con una buona approssimazione che $u+f \cong u$ e quindi $|v| \cong f$ ottenendo un modello della webcam detto "modello *pinhole*" il quale prevede un punto di dimensioni infinitesime attraverso il quale passano tutti i raggi riflessi dall'oggetto per restituire sul sensore un'immagine rovesciata. Per semplicità prenderemo in considerazione l'immagine non rovesciata ottenuta dalla proiezione di questi raggi su un piano perpendicolare all'asse ottico a distanza f dal centro ottico (quindi un'immagine identica a quella proiettata sul sensore a meno del rovesciamento). Definite come (u, v) le coordinate dell'oggetto sul piano immagine e come (x_i, y_i, z_i) le coordinate di un punto nello spazio secondo gli assi x, y, z come in figura 1.27, vige la seguente relazione non lineare:

$$\begin{cases} u &= \frac{f}{z_i} x_i \\ v &= \frac{f}{z_i} y_i \end{cases} \tag{1.5.1}$$

Passando alle coordinate omogenee ed introducendo un fattore di scala w abbiamo:

$$\begin{bmatrix} wu \\ wy \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = P \cdot w$$

dove P è detta "matrice di proiezione prospettica" e $w = z$ è la distanza dell'oggetto dal piano immagine.

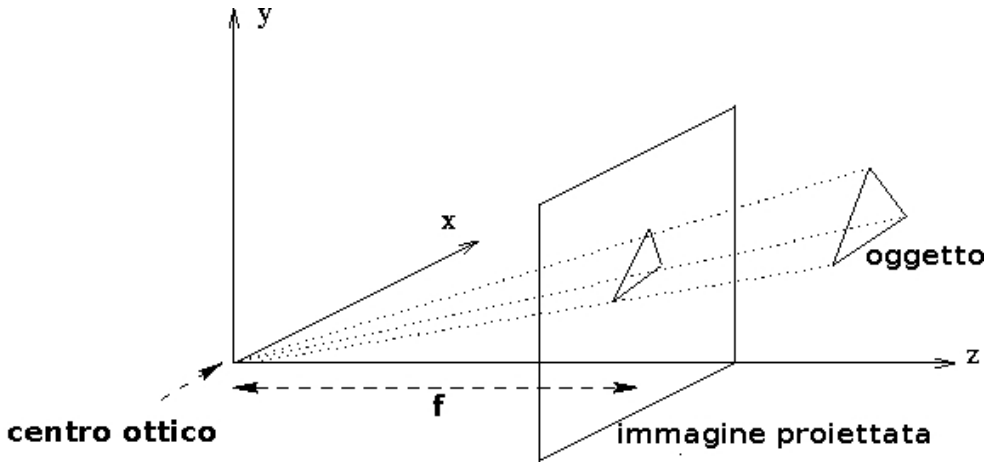


Figura 1.27: Proiezione prospettica.

1.5.2 La Pixelizzazione

La webcam presenta alcune caratteristiche costruttive di cui bisogna tener conto nella modellizzazione, tra queste vi è la pixelizzazione ovvero la discretizzazione dovuta al sensore e alla sua posizione rispetto all'asse ottico. Questo comporta alcune proprietà:

1. Il centro ottico della webcam non coincide con il centro fisico del CCD ma ha coordinate (u_o, v_o) ed è detto "punto principale".

2. Le coordinate di un punto nel sistema di riferimento standard della webcam sono misurate in *pixel*, si introduce quindi un fattore di scala.

3. La forma del pixel non è quadrata: occorre considerare due fattori di scala diversi lungo x (k_u) e y (k_v).

Queste proprietà vengono prese in considerazione modificando la (1.5.1) aggiungendo la traslazione del centro ottico e la riscalatura indipendente degli assi del sensore.

$$\begin{cases} u &= k_u \frac{f}{z} x + u_o \\ v &= k_v \frac{f}{z} y + v_o \end{cases}$$

per cui la matrice di proiezione prospettica diventa:

$$P = \begin{bmatrix} fk_u & 0 & u_o & 0 \\ 0 & fk_v & v_o & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} A & 0 \end{bmatrix}$$

con la matrice di prospezione A che risulta essere:

$$A = \begin{bmatrix} fk_u & 0 & u_o \\ 0 & fk_v & v_o \\ 0 & 0 & 1 \end{bmatrix}$$

1.5.3 La trasformazione rigida tra la webcam e la scena

Si rimuove ora l'ipotesi che il sistema di riferimento spaziale coincida con quello della webcam introducendo la trasformazione rigida che lega questi due sistemi, quindi definendo come $\bar{x}$ il sistema di coordinate della webcam e $\bar{X}$ il sistema di coordinate spaziali, si considera il seguente cambio di coordinate:

$$\bar{x} = R \cdot \bar{X} + T$$

che in coordinate omogenee diventa:

$$\bar{x} = G \cdot \bar{X}$$

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

e la sua proiezione sul piano immagine diventa:

$$\begin{bmatrix} ku \\ kv \\ k \end{bmatrix} = P\bar{x} = P \cdot G \cdot \bar{X} = \bar{P}\bar{X} \tag{1.5.2}$$

dove $\bar{P}$ risulta essere:

$$\bar{P} = A \cdot \begin{bmatrix} R & T \end{bmatrix}$$

Ponendo:

$$T = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}, \; R = \begin{bmatrix} r_1^T \\ r_2^T \\ r_3^T \end{bmatrix} \tag{1.5.3}$$

si ottiene la seguente $\bar{P}$:

$$\bar{P} = \begin{bmatrix} fk_u r_1^T + u_o r_3^T & fk_u t_1 + u_o t_3 \\ fk_v r_2^T + v_o r_3^T & fk_v t_2 + v_o t_3 \\ r_3^T & t_3 \end{bmatrix} \tag{1.5.4}$$

la quale costituisce la matrice di proiezione prospettica che governa la proiezione dei punti nello spazio sul piano immagine e può essere ricavata sperimentalmente mediante taratura del sensore.

Esempio Si è vista la relazione tra un punto nello spazio e la sua proiezione sul piano immagine, quindi ci si è preposti di ricavare la matrice che trasformi un'immagine ricavata dalla proiezione di un oggetto, posto su di un piano Z non necessariamente perpendicolare all'asse ottico, nell'immagine data da un eventuale sensore con il suo asse ottico perpendicolare al piano Z, con una trasformazione prospettica detta "inversione prospettica".

L'obiettivo di questo esempio è di mostrare l'effetto del calcolo della matrice di trasformazione prospettica e per semplificare i conti verranno ignorati i parametri interni della webcam concentrando l'analisi sulla sola trasformazione delle coordinate e verranno anche ignorate le distorsioni non lineari (trattate nella prossima sezione).

Definendo come $\bar{X} = (X, Y)$ le coordinate dei punti sul piano Z e $\bar{x} = (x, y)$ le coordinate nell'immagine, possiamo ridurre la matrice di trasformazione prospettica da 3x4 a 3x3 (in coordinate omogenee):

$$\begin{bmatrix} Xw \\ Yw \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

dove $a, b, c, d, e, f, g, h, i$ sono coefficienti non noti a cui potremo risalire attraverso una taratura a partire da punti noti nello spazio.

Per normalizzare il vettore di coordinate omogenee dell'immagine possiamo dividere

tutto per il coefficiente di scala w:

$$w = gx + hy + 1$$

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \frac{\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}}{\begin{bmatrix} g & h & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}} \tag{1.5.5}$$

da cui ricaviamo le espressioni di X e Y:

$$\begin{aligned} X(gx + hy + 1) &= ax + by + c \\ Y(gx + hy + 1) &= dx + ey + f \end{aligned}$$

$$\begin{array}{rcccccccc} X &=& ax & +by & +c & & & -Xxg & -Xyh \\ Y &=& & & & dx & +ey & +f & -Yxg & -Yyh \end{array}$$

le quali possono essere rese come un prodotto matriciale:

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 & -Xx & -Xy \\ 0 & 0 & 0 & x & y & 1 & -Yx & -Yy \end{bmatrix} \cdot \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix}$$

che verrà indicato come:

$$\bar{X} = B(\bar{X}, \bar{x}) \cdot \lambda$$

L'interesse è rivolto al vettore lambda di cui dobbiamo determinarne i coefficienti, e per fare ciò bisogna necessariamente conoscere le coordinate nello spazio (x) e le coordinate nell'immagine (X) di almeno 3 punti, che definiscono univocamente un piano, con l'ovvia proprietà che ad un numero maggiore di punti con coordinate note corrisponde una maggiore precisione nella conoscenza dei coefficienti. Per comodità verranno utilizzati 4 punti per ottenere una matrice A quadrata 8x8 che potrà essere

invertita senza dover ricorrere alla matrice pseudoinversa. Le coordinate nello spazio dei 4 punti p_i (con $i = 1, 2, 3, 4$) vengono indicate con (X_i, Y_i) mentre nell'immagine saranno (x_i, y_i), quindi si ottiene la seguente matrice B:

$$B = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -X_1x_1 & -X_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -Y_1x_1 & -Y_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -X_2x_2 & -X_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -Y_2x_2 & -Y_2y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -X_3x_3 & -X_3y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -Y_3x_3 & -Y_3y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -X_4x_4 & -X_4y_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -Y_4x_4 & -Y_4y_4 \end{bmatrix}$$

ed il vettore lambda lo potremo ottenere semplicemente invertendo questa matrice e moltiplicandola per il vettore $\bar{X}$ contenente le coordinate nello spazio dei vari punti:

$$\bar{X} = \begin{bmatrix} X_1 & Y_1 & X_2 & Y_2 & \cdots \end{bmatrix}'$$

$$\lambda \;=\; B^{-1}\bar{X}$$

Per ottenere le coordinate dei 4 punti su di un piano si è scelto un riferimento formato dalle intersezioni di 4 rette (figura 1.28), le quali sono state localizzate tramite segmentazione per colore e applicazione della trasformata di *Hough*. Si è quindi creato il vettore $\bar{X}$ inserendo le 4 coordinate nello spazio, si è creata la matrice A e conseguentemente si è ricavato il vettore di coefficienti λ per creare la matrice di trasformazione. Ciò ci ha permesso di creare una nuova immagine N di coordinate X, Y coincidenti con le coordinate del piano Z nello spazio su cui giace il riferimento attraverso l'equazione (1.5.5). Questo equivale a portare l'asse ottico dell'immagine perpendicolare al piano Z, come vediamo nella figura 1.29. A pagina 78 è presente il codice matlab della funzione "prospettiva_inversa".

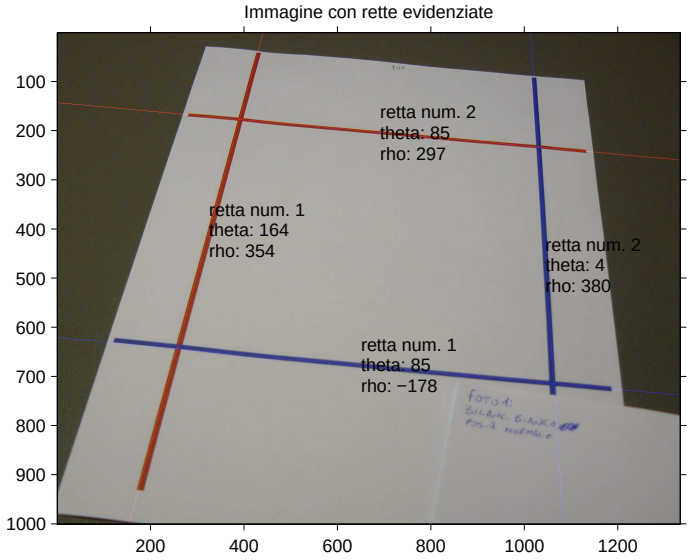
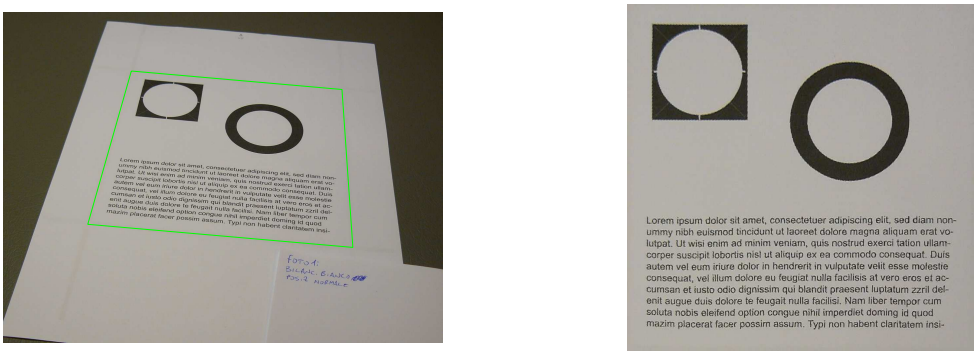


Figura 1.28: L'immagine con le rette di riferimento e i parametri nello spazio di Hough.



(a) immagine di partenza

(b) immagine risultante

Figura 1.29: Inversione prospettica. Nella figura (a) è evidenziata l'area interessata dalla trasformazione.

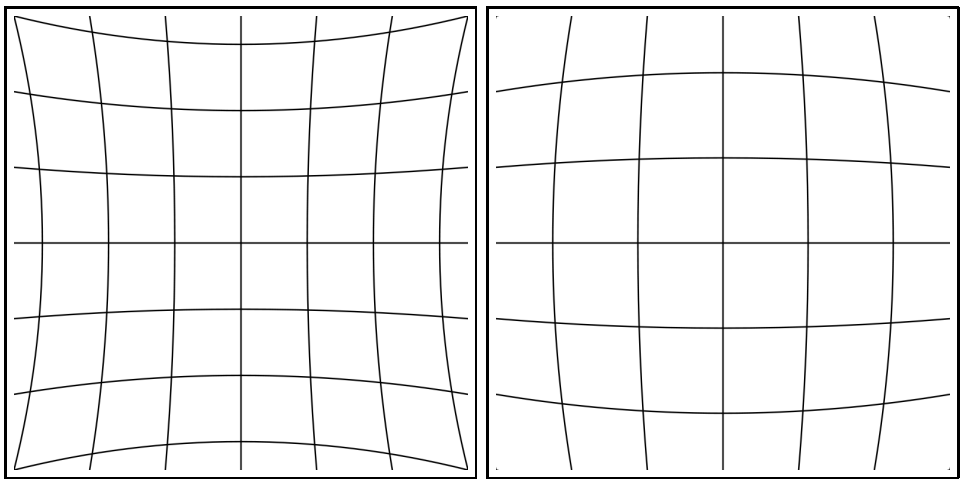
Capitolo 2

Calibrazione e triangolazione stereoscopica

Come accennato nel capitolo precedente, la precisione necessaria per poter ricavare le coordinate spaziali di un oggetto, mediante visione stereoscopica, necessita di un modello del sensore che tenga conto delle imperfezioni presenti nell'immagine acquisita. La webcam utilizzata è stata progettata per scopi in cui non è assolutamente necessaria la precisione, quindi utilizzando materiali costruttivi (per la lente e per i circuiti elettrici) di bassa qualità ed un sensore soggetto ad alti livelli di rumore, perciò si ha bisogno di un modello che tenga conto di queste aberrazioni. Ci si concentrerà quindi sulle distorsioni che maggiormente influiscono sulle immagini acquisite, dovute alla posizione reale del sensore rispetto all'asse ottico e alla lente di spessore non infinitesimale. La prima caratteristica porterà l'immagine a soffrire di distorsione **tangenziale**, mentre la seconda produrrà nell'immagine una distorsione **radiale**.

Distorsione tangenziale Questa distorsione è data dal decentramento dell'asse della lente e dipende dall'angolo α_c tra gli assi u, v del sensore che differisce dal valore ideale di 90^o. Questo scostamento, come vedremo, sarà minimo, in ogni caso verrà preso in considerazione nella compensazione della distorsione.

Distorsione radiale Questa distorsione ha effetti più evidenti nell'immagine facendo si che linee rette nello spazio vengano proiettate come linee curve nell'immagine, e ciò è dovuto al fatto che l'ingrandimento dell'immagine è dipendente dalla distanza dall'asse ottico. Nella figura 2.1 sono presentati i due tipi di distorsione radiale possibili, ossia quella a "cuscino" (figura (a)) in cui le rette che non passano per il centro dell'immagine vengono piegate verso di esso, e la più comune distorsione a "botte" (figura (b)) in cui si ha l'effetto contrario, presente in maniera massiccia nelle fotocamere e videocamere con obiettivo *fisheye* (letteralmente "occhio di pesce").



(a) distorsione a "cuscino" (b) distorsione a "botte"

Figura 2.1: I due diversi tipi di distorsione radiale.

2.1 Modellazione del sensore in presenza di distorsioni

Definendo r la distanza di ogni pixel dal centro reale dell'immagine, $\bar{x}_u = (x_u, y_u)$ le coordinate non distorte, $\bar{x}_d = (x_d, y_d)$ quelle distorte, $\bar{x}_c = (x_c, y_c)$ le coordinate del centro ottico (detto anche "punto principale"), O_n l'n-simo coefficiente dello sviluppo in serie della distorsione radiale e P_n l'equivalente per la distorsione tangenziale, si ha

la seguente espressione per le coordinate non distorte date dal modello di distorsione di *Brown* [4]:

$$\begin{aligned} x_u &= x_d + (x_d - x_c)(O_1 r^2 + O_2 r^4 + ...) \\ &\quad +(P_1(r^2 + 2(x_d - x_c)^2) \\ &\quad +2P_2(x_d - x_c)(y_d - y_c))(1 + P_3 r^2 + ...) \end{aligned} \tag{2.1.1}$$

$$\begin{aligned} y_u &= y_d + (y_d - y_c)(O_1 r^2 + O_2 r^4 + ...) \\ &\quad +(P_2(r^2 + 2(y_d - y_c)^2) \\ &\quad +2P_1(x_d - x_c)(y_d - y_c))(1 + P_3 r^2 + ...) \end{aligned} \tag{2.1.2}$$

Introducendo quindi 4 parametri kc_n, due per la distorsione radiale (kc_1, kc_2)e due per quella tangenziale (kc_3, kc_4), si può esprimere la (2.1.1) e la (2.1.2) nel seguente modo:

$$x_d = (1 + kc_1 r^2 + kc_2 r^4)x_u + 2kc_3 \cdot x_u \cdot y_u + kc_4(r^2 + 2x_u^2) \tag{2.1.3}$$

$$y_d = (1 + kc_1 r^2 + kc_2 r^4)x_u + kc_3(r^2 + 2y_u^2) + 2kc_4 \cdot x_u \cdot y_u \tag{2.1.4}$$

2.2 Calibrazione instrinseca ed estrinseca

Trovate le equazioni (2.1.3) (2.1.4) che legano ogni pixel dell'immagine senza distorsione all'immagine distorta, rimangono solamente da calcolare i coefficienti kc_n che, insieme ai fattori di scala fk_u e fk_v e alle coordinate del centro ottico (u_o, v_o) formano i parametri riferiti al modello interno della webcam, detti **instrinseci**, mentre le matrici di rotazione R e traslazione T (o, in alternativa, la matrice in coordinate omogenee G) contengono i parametri relativi alla posizione nello spazio, detti **estrinseci** (equazioni (1.5.3) a pagina 36).

Si vuole quindi ricavare questi 14 parametri (8 intrinseci, 6 estrinseci), e per farlo si è seguito un metodo diretto di taratura sviluppato da *Zhengyou Zhang*[5] ed implementato in un toolbox per Matlab che è poi stato usato per calcolare i parametri di entrambe le webcam. Questo metodo prevede di calcolare i parametri mediante metodi lineari di inversione della matrice di proiezione prospettica $\bar{P}$ ((1.5.4) a pagina 37) conoscendo le coordinate nello spazio e nell'immagine di un certo numero di punti. Per esplicitare questa relazione definiamo i seguenti vettori:

$$\begin{array}{rcc} q_1^T & = & fk_u r_1^T + u_o r_3^T \\ q_2^T & = & fk_v r_2^T + v_o r_3^T \\ q_3^T & = & r_3^T \\ q_{14} & = & fk_u t_1 + u_o t_3 \\ q_{24} & = & fk_v t_2 + v_o t_3 \\ q_{34} & = & t_3 \end{array}$$

per cui la matrice $\bar{P}$ diventa:

$$P = \begin{bmatrix} q_1^T & q_{14} \\ q_2^T & q_{24} \\ q_3^T & q_{34} \end{bmatrix}$$

Dall'equazione 1.5.2 abbiamo che:

$$\begin{bmatrix} ku \\ kv \\ k \end{bmatrix} = \bar{P}\bar{X} = \begin{bmatrix} q_1^T \bar{X} & q_{14} \\ q_2^T \bar{X} & q_{24} \\ q_3^T \bar{X} & q_{34} \end{bmatrix}$$

Quindi ogni punto i con note le coordinate nello spazio $\bar{X}_i$ e le coordinate nell'immagine (u_i, v_i) fornisce una coppia di equazioni:

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

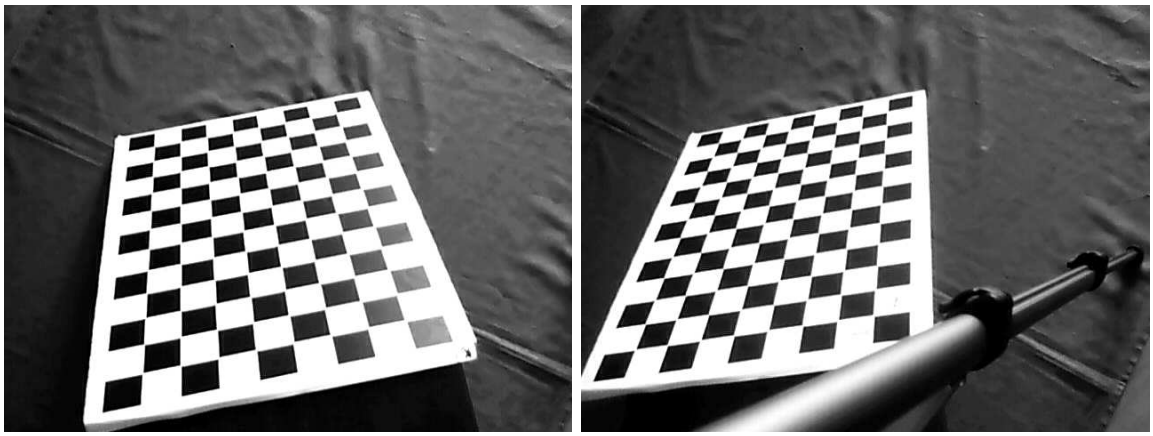
Sapendo che la $\bar{P}$ contiene 10 incognite, si può immaginare che basti conoscere 5 punti non complanari a stimarne i valori, ma nella realtà bisogna tener presente i 4 coefficienti di distorsione, gli errori di misura delle coordinate dei punti e gli errori di quantizzazione, così si preferisce acquisire le coordinate del maggior numero di punti possibile in modo da minimizzare l'errore. Per automatizzare questo processo

si è ricorso al toolbox di Matlab sviluppato da *Jean-Yves Bouguet*[1] della California Tech basato sul già citato metodo di *Zhang* che si basa su una serie di immagini che presentano punti complanari la cui posizione non è nota ma di cui si conosce la distanza tra di essi, per questo motivo sono state ricavate 30 coppie di immagini dalle due webcam ritraenti una scacchiera formata da quadrati (alternati bianchi e neri) di lato $L = 30mm$ posta in diverse posizioni (figura 2.3). Ripetuto il procedimento per le singole webcam è poi necessario passare alla taratura stereoscopica che riduce ulteriormente l'errore e calcola le matrici R e T della webcam destra prendendo come punto di riferimento la webcam sinistra secondo il sistema di riferimento mostrato in figura 2.5. Per far in modo che rimanga costante la posizione reciproca delle due webcam, è stato costruito il supporto mostrato in figura 2.2. Nella seguente tabella

Figura 2.2: Il supporto per le webcam utilizzato per questo lavoro.

sono presentati i parametri calcolati:

[1] http://www.vision.caltech.edu/bouguetj/calib_doc/



(a) immagine di sinistra (b) immagine di destra

Figura 2.3: Esempio di immagine destra e sinistra della scacchiera di riferimento.

webcam sinistra	webcam destra
$fk_u = 844.2543$	$fk_u = 844.5916$
$fk_v = 858.5060$	$fk_v = 858.6891$
$u_o = 343.9745$	$u_o = 320.2725$
$v_o = 232.0483$	$v_o = 212.9339$
$kc_1 = 0.1207$	$kc_1 = 0.1185$
$kc_2 = -0.5954$	$kc_2 = -0.5572$
$kc_3 = 5.505 \cdot 10^{-4}$	$kc_3 = 0.0012$
$kc_4 = -3.97 \cdot 10^{-4}$	$kc_4 = -7.78 \cdot 10^{-4}$
$R = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$	$R = \begin{bmatrix} 0.9753 & -0.0261 & 0.2193 \\ 0.0222 & 0.9995 & 0.0206 \\ -0.2197 & -0.0152 & 0.9754 \end{bmatrix}$
$T = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} -195.7568 \\ -2.7282 \\ 45.0868 \end{bmatrix}$

Di conseguenza le matrici di proiezione prospettica per la webcam sinistra e destra sono, rispettivamente:

$$\bar{P}_{sinistra} = \bar{P}_L = \begin{bmatrix} 844.2543 & 0 & 343.9745 & 0 \\ 0 & 858.5060 & 232.0483 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\bar{P}_{destra} = \bar{P}_R = \begin{bmatrix} 753.3554 & -26.9647 & 497.6325 & -150894.5 \\ -27.7608 & 855.0506 & 255.4010 & 72578 \\ -0.2197 & -0.0152 & 0.9754 & 45.0868 \end{bmatrix} \tag{2.2.1}$$

Da questi dati si deduce che le due webcam dello stesso modello presentano comunque valori leggermente diversi tra di loro, ciò a testimonianza del fatto che sono imperfezioni costruttive a determinare molte delle distorsioni presenti nell'immagine. Si può anche notare che la webcam destra è posizionata a circa 200mm (per la precisione a 200.8907mm, cioè il primo valore di $R^{-1} \cdot T$) dalla sinistra ed è rivolta leggermente verso sinistra come si può anche notare nella figura 2.5.

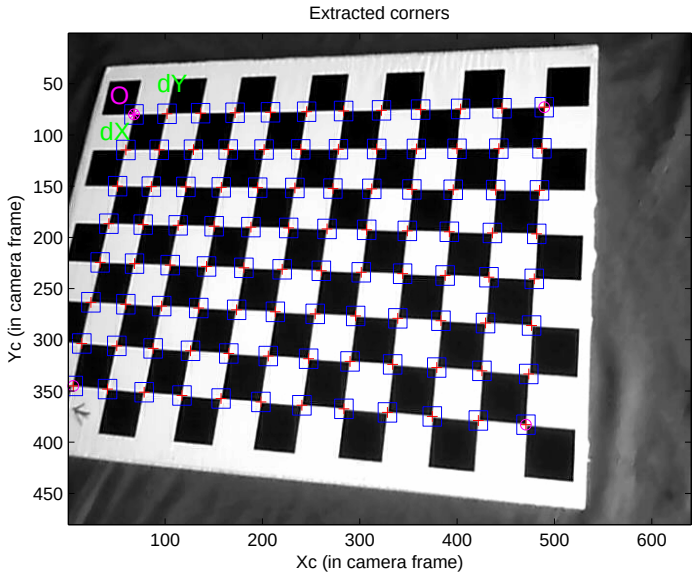


Figura 2.4: Punti di taratura rilevati dall'algoritmo su di un'immagine della scacchiera.

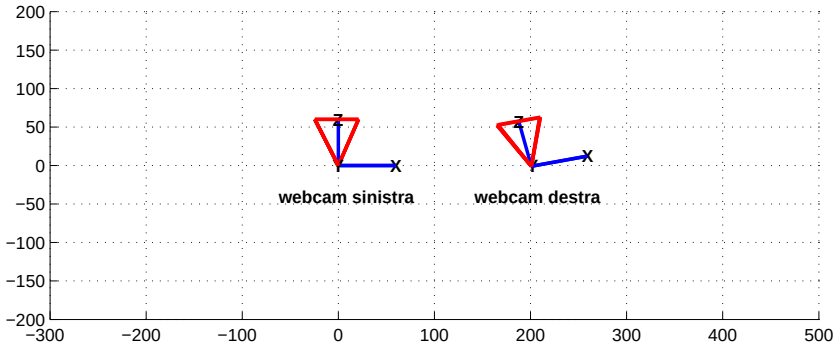


Figura 2.5: Posizione e direzione delle webcam viste dall'alto, valori in mm.

2.3 Geometria epipolare

La geometria epipolare è la geometria delle intersezioni tra i piani dell'immagine ed è analizzata per risolvere il problema della ricerca dei punti corrispondenti tra le due immagini, con l'obiettivo di poter ricavare in seguito le coordinate di punti nello spazio. Nella figura 2.6 due webcam O_L e O_R sono puntate verso un generico punto X dello spazio che viene proiettato nei punti X_L, X_R (secondo la proiezione prospettica trattata in precedenza), mentre le proiezioni (e_L, e_R) di ciascun punto focale delle due webcam sul piano immagine dell'altra sono chiamati **epipoli**. Possiamo notare come la retta O_L-X sia vista come un singolo punto dalla webcam sinistra, mentre nella webcam destra essa appaia come la **retta epipolare** l' che congiunge e_R con X_R, quindi si può affermare che:

- ogni punto X dello spazio genera una retta epipolare sulla webcam destra a partire dalla sua proiezione sul piano immagine e_L
- tutte le rette epipolari si intersecano nel punto epipolare

Si arriva così alla conclusione che ogni retta passante per il punto epipolare è una retta epipolare visto che può essere generata a partire da un certo punto nello spazio. Ovviamente lo stesso discorso vale invertendo le webcam, ossia tracciando la retta epipolare sul piano immagine della webcam sinistra partendo dalla proiezione di un punto nello spazio sulla webcam destra. Importante è comunque il fatto che, trovata la proiezione di un punto su un'immagine, per trovare la corrispondente posizione sull'altro piano immagine basti cercare nell'intorno di una retta nota, secondo una condizione detta **vincolo epipolare**, riducendo la ricerca sui pixel dell'immagine nelle vicinanze della retta epipolare. Si parla di intorno perché non è garantita la precisione a causa del rumore dato dal sensore, del rumore di quantizzazione e dei

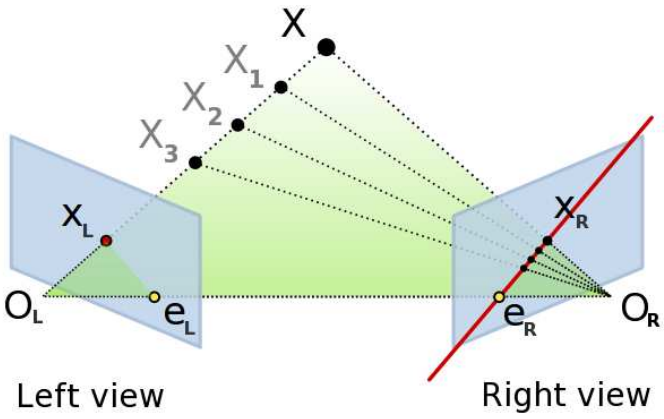


Figura 2.6: Geometria epipolare.

difetti della webcam non presi in considerazione nella modellizzazione.

Il vincolo epipolare che lega un certo punto proiettato X_L alla retta epipolare l' sulla seconda webcam può essere rappresentato attraverso una matrice non invertibile 3x3 F detta **matrice fondamentale**. La ricerca di questo vincolo viene diviso in due parti, dove nella prima ci si prepone l'obiettivo di trovare un punto X'_R sulla seconda immagine che può essere un potenziale punto corrispondente tramite una matrice di trasformazione prospettica tra i due piani immagine $\bar{P}'_R\bar{P}_L^{-1}$ (dove con $\bar{P}_L^{-1}$ si intende la pseudoinversa della matrice $\bar{P}_L$), quindi nella seconda parte si calcola la retta l' passante tra questo punto e l'epipolo e_R tramite $l' = e_R \times X'_R$, ottenendo quindi:

$$F \cdot X_L = (e_R \times X'_R)\bar{P}'_R\bar{P}_L^{-1}X_L$$

ricavabile in una forma computazionalmente più veloce, come specificato in [6], pag. 412:

$$F_{ij} = (-1)^{i+j} \det \begin{bmatrix} \sim \bar{P}_L^i \\ \sim \bar{P}_R^j \end{bmatrix}$$

dove $\sim \bar{P}_L^i$ indica la matrice di trasformazione prospettica della webcam sinistra in cui è stata omessa la riga i.

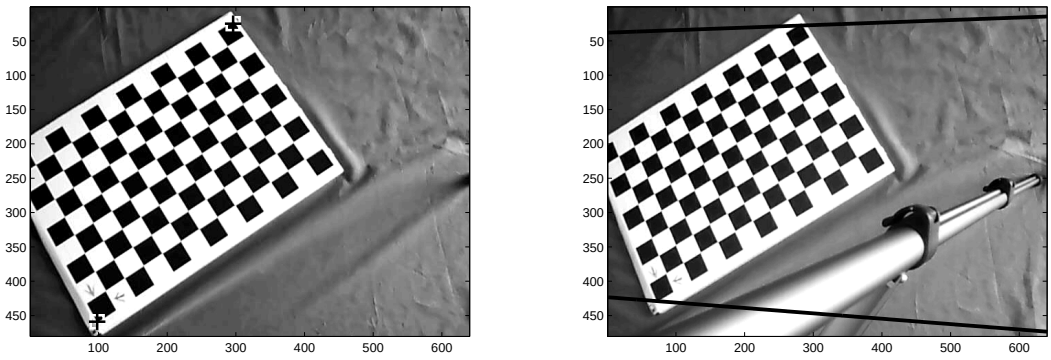
La matrice F presenta interessanti caratteristiche:

1. Dati X_L e X_R proiezioni nei piani immagine delle due webcam a partire dal punto nello spazio $\bar{X}$, vale la relazione $X_L F X_R = 0$

2. Se $l' = FX_L$, allora $l = F^T X_R$ cioè la retta epipolare sull'immagine sinistra è data dalla trasposta della matrice fondamentale F per il punto X_R sulla webcam destra.

3. La retta epipolare sulla webcam destra l' data dalla proiezione dell'epipolo sinistro e_L è nulla, difatti $Fe_L = 0$ e, di conseguenza, $F^T e_R = 0$.

Esempio Si genereranno ora due rette epipolari sulla webcam destra partendo da un punto sulla webcam sinistra. Grazie alle due matrici P_L e P_R (2.2.1) è stato possibile ricostruire la matrice F, quindi sono stati calcolati i parametri della retta l' moltiplicando F per i due punti proiettati in coordinate omogenee. I parametri, nella forma $l' = [a\, b\, c]^T$ con $ax + by + c = 0$, sono stati utilizzati per creare le due linee epipolari sulla seconda immagine. Il risultato è mostrato nella figura 2.7 mentre il codice Matlab è riportato nell'appendice B a pagina 78.



(a) immagine di sinistra con due punti evidenziati (b) immagine di destra con le rette epipolari dei due punti

Figura 2.7: Corrispondenza punto proiettato - retta epipolare nelle due immagini.

2.4 Triangolazione stereoscopica

Nella sezione 2.2 sono stati calcolati la matrice di rotazione R ed il vettore di traslazione T delle due webcam nel sistema di riferimento dato dal piano immagine della webcam sinistra, per cui la matrice di rotazione della webcam sinistra è la matrice identità ed il suo vettore di traslazione è il vettore nullo, quindi ogni volta che verranno presentati R e T si farà riferimento unicamente alla webcam destra.
Il significato di R e T è che, per ogni punto $\bar{X} = [XYZ]$ nello spazio, le sue coordinate nel sistema di riferimento dato dalla webcam sinistra è $\bar{X}_L = [X_L Y_L Z_L]$ mentre nel sistema di riferimento della webcam destra è $\bar{X}_R = [X_R Y_R Z_R]$, le quali coordinate sono in relazione tra loro secondo il seguente cambio di base:

$$\bar{X}_R = R \cdot \bar{X}_L + T \tag{2.4.1}$$

assumendo R e T noti senza errori.
Si normalizzano ora le coordinate delle proiezioni sui piani immagine, traslandole in modo da centrarle rispetto al centro ottico (u_o, v_o) e dividendole per il fattore di scala fk (figura 2.8) e rimuovendo la distorsione utilizzando i parametri kc ricavati dalla taratura. Le componenti dei vettori normalizzati variano in un intervallo $[-1, 1]$ con lo 0 posto nel centro ottico. Trasformati in coordinate omogenee, si considerino quindi $\bar{x}_L = \bar{X}_L/Z_L = [x_L y_L 1]^T$ e $\bar{x}_R = \bar{X}_R/Z_R = [x_R y_R 1]^T$ le coordinate normalizzate omogenee delle proiezioni del punto $\bar{X}$ su i due piani immagine. Il problema di triangolazione stereoscopica consiste nel ricavare le coordinate del punto $\bar{X}$ (quindi, a meno di un cambio di base, $\bar{X}_L$ o $\bar{X}_R$) dalla coppia di punti $\bar{x}_L$ e $\bar{x}_R$ conoscendo R e T e assumendo di conoscerne i valori con errori trascurabili.
L'equazione (2.4.1) può essere riscritta come:

$$Z_R \bar{x}_R = Z_L \bar{x}_L + T$$

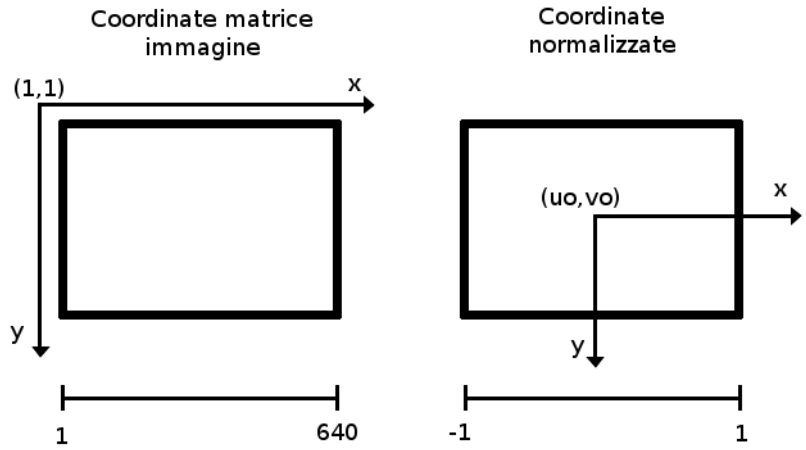


Figura 2.8: Sistema di coordinate immagine - coordinate normalizzate.

da cui:

$$Z_R \bar{x}_R - R Z_L \bar{x}_L = T$$

esprimibile anche in forma matriciale:

$$\begin{bmatrix} -R\bar{x}_L \\ \bar{x}_R \end{bmatrix}^T \cdot \begin{bmatrix} Z_L \\ Z_R \end{bmatrix} = A \cdot \begin{bmatrix} Z_L \\ Z_R \end{bmatrix} = T$$

È possibile quindi calcolare i valori Z_L, Z_R grazie alla pseudoinversa della matrice A:

$$\begin{bmatrix} Z_L \\ Z_R \end{bmatrix} = (A^T A)^{-1} A^T T$$

Definendo $\alpha_L = -R\bar{x}_L$ è possibile riscrivere la matrice A:

$$A = [\alpha_L \, \bar{x}_R]$$

Quindi, attraverso alcuni passaggi, si può arrivare alla formulazione per Z_L, Z_R in funzione dei vettori normalizzati:

$$A^T A = \begin{bmatrix} \|\alpha_L\|^2 & \langle \alpha_L, \bar{x}_R \rangle \\ \langle \bar{x}_R, \alpha_L \rangle & \|\bar{x}_R\|^2 \end{bmatrix}$$

$$(A^T A)^{-1} = \frac{1}{D} \begin{bmatrix} \|\bar{x}_R\|^2 & -\langle \alpha_L, \bar{x}_R \rangle \\ -\langle \bar{x}_R, \alpha_L \rangle & \|\alpha_L\|^2 \end{bmatrix}$$

dove il determinante D è uno scalare di valore:

$$D = \|\alpha_L\|^2\|\bar{x}_R\|^2 - \langle\alpha_L, \bar{x}_R\rangle^2$$

per cui

$$(A^TA)^{-1}A^T = \frac{1}{D}\begin{bmatrix} \|\bar{x}_R\|^2 & -\langle\alpha_L, \bar{x}_R\rangle \\ -\langle\bar{x}_R, \alpha_L\rangle & \|\alpha_L\|^2 \end{bmatrix} \cdot \begin{bmatrix} - & \alpha_L^T & - \\ - & \bar{x}_R^T & - \end{bmatrix} \cdot \begin{bmatrix} | \\ T \\ | \end{bmatrix}$$

con il risultato:

$$\begin{bmatrix} Z_L \\ Z_R \end{bmatrix} = \frac{1}{D}\begin{bmatrix} \|\bar{x}_R\|^2\langle\alpha_L, T\rangle - \langle\alpha_L, \bar{x}_R\rangle\langle\bar{x}_R, T\rangle \\ -\langle\alpha_L, \bar{x}_R\rangle\langle\alpha_L, T\rangle + \|\alpha_L\|^2\langle\bar{x}_R, T\rangle \end{bmatrix} \tag{2.4.2}$$

Ottenuto il valore di Z_L è possibile ricavare facilmente il valore delle coordinate del punto secondo il riferimento dato dalla webcam sinistra $\bar{X}_L$ moltiplicando ogni componente di $\bar{x}_L$ per Z_L:

$$\bar{X}_L = \bar{x}_L^T\begin{bmatrix} Z_L & 0 & 0 \\ 0 & Z_L & 0 \\ 0 & 0 & Z_L \end{bmatrix} \tag{2.4.3}$$

mentre se si vogliono ottenere le coordinate rispetto alla webcam destra basta utilizzare l'equazione (2.4.1).

Esiste una seconda tecnica per calcolare le coordinate di $\bar{X}$ data dal fatto che è possibile riproiettare le rette epipolari nello spazio a partire dai punti proiettati sui piani immagine per trovare il punto di incidenza, il quale corrisponde esattamente al punto $\bar{X}$. Un problema è dato dal fatto che, a causa della non perfetta conoscenza di R e T, non è garantito che queste due rette si intersechino in un punto, quindi è necessario approssimare la posizione del punto a quello la cui distanza da entrambe le rette è minima. Questa tecnica è stata implementata nel toolbox di taratura utilizzato nella sezione 2.2 e verrà quindi usato come riferimento per la verifica della correttezza dei dati data la sua maggior robustezza a fronte di errori presenti nei parametri instrinseci ed estrinseci.

Capitolo 3

Applicazioni

Nei capitoli precedenti sono stati analizzati i metodi di analisi delle immagini e si è creato un modello delle webcam con i parametri necessari alla triangolazione stereoscopica, verranno quindi presentate delle applicazioni pratiche che utilizzeranno questi metodi con lo scopo di verificare la validità degli algoritmi presentati in questo testo e la congruenza dei dati ricavati.

3.1 Verifica validità della triangolazione

Ci si propone ora di verificare la correttezza del modello matematico delle webcam e delle equazioni di triangolazione attraverso il calcolo delle distanze tra sei punti nello spazio di cui si presuppongono note le proiezioni sul piano immagine. In figura 3.1 sono presentate le due immagini con le proiezioni dei punti presi in esame e, essendo noto il lato di ogni quadrato della scacchiera (pari a 30mm), è nota la distanza nello spazio tra ognuno dei punti, quindi è possibile verificare direttamente la validità delle coordinate ottenute tramite triangolazione confrontando queste distanze con quelle date dalle distanze euclidee tra le coordinate ottenute nel sistema di riferimento della telecamera sinistra.

Come primo passo si caricano in memoria i parametri instrinseci ed estrinseci della

webcam sinistra e destra e si inseriscono le coordinate del punto nelle due immagini in due matrici 2x6 in cui la prima riga di ogni matrice contiene tutte le coordinate sull'asse x mentre la seconda riga contiene le coordinate y:

$$x_{webcam} = \begin{bmatrix} x_1 & x_2 & \cdots & x_6 \\ y_1 & y_2 & \cdots & y_6 \end{bmatrix}$$

In codice Matlab:

```
load parametri.mat
xL=[287 9  273 138 173 359
    59 231 307 395 264 214];
xR=[272 18 237 111 152 325
    60 219 297 373 253 210];
```

Il file "parametri.mat" contiene i parametri che serviranno alla triangolazione, cioè i fattori di scala *fc_left* e *fc_right* (rispettivamente per la webcam sinistra e destra), le posizioni dei centri ottici *cc_left* e *cc_right*, i parametri di distorsione *kc_left* e *kc_right* e le matrici R e T.

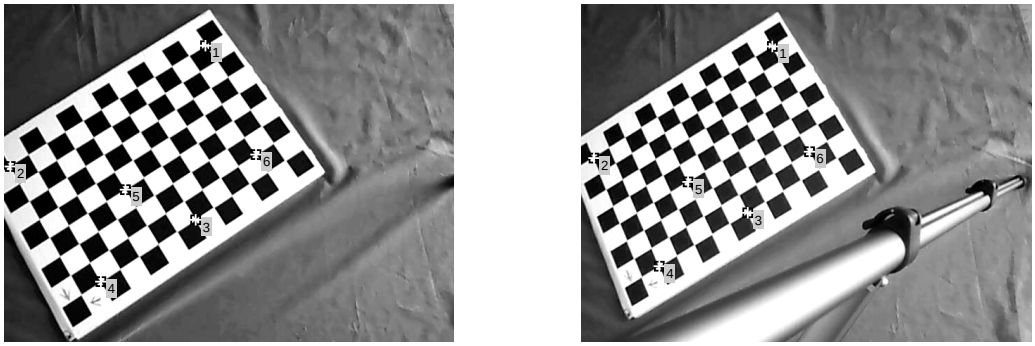
Si normalizzano quindi le coordinate xL e xR sottraendo la posizione del centro ottico e dividendo ogni coordinata per il fattore di scala, ottenendo così coordinate centrate nel centro ottico e di valore compreso tra -1 ed 1, come mostrato in figura 2.8. Nel passo successivo viene eliminata la distorsione utilizzando una funzione prelevata dal toolbox di Matlab per la taratura del sensore ("comp_distortion_oulu"), la quale prende in ingresso le coordinate dei punti e i parametri kc restituendo le coordinate prive di distorsione che vengono quindi convertite in coordinate omogenee:

```
% normalizzo sottraendo le coordinate del centro ottico e dividendo per fk
xL_d=[(xL(1,:) - cc_left(1))/fc_left(1);(xL(2,:) - cc_left(2))/fc_left(2)];
xR_d=[(xR(1,:) - cc_right(1))/fc_right(1);(xR(2,:) - cc_right(2))/fc_right(2)];
% elimino distorsione
xL_n = comp_distortion_oulu(xL_d,kc_left);
xR_n = comp_distortion_oulu(xR_d,kc_right);
% rendo le coordinate omogenee
xL_n = [xL_n;ones(1,size(xL,2))];
xR_n = [xR_n;ones(1,size(xL,2))];
```

Si passa quindi al calcolo della coordinata nello spazio del punto utilizzando le equazioni (2.4.2) e (2.4.3):

```
alpha_L=-R*xL_n;
N=dot(xR_n,xR_n).*dot(alpha_L,T)-dot(alpha_L,xR_n).*dot(xR_n,T);
D=dot(xR_n,xR_n).*dot(alpha_L,alpha_L)-dot(alpha_L,xR_n).*dot(alpha_L,xR_n);
Z=N/D;
X=xL_n.*(repmat(Z,[3,1]));
```

dove X è la matrice 2x6 di coordinate nello spazio dei sei punti. Nella seguente



(a) webcam di sinistra (b) webcam di destra

Figura 3.1: Le due immagini della scacchiera con evidenziate le proiezione dei punti presi in esame.

tabella vengono presentate le distanze reali e le distanze rilevate, più l'errore assoluto e l'errore percentuale sulla distanza (tutti i valori, escludendo le percentuali, sono espressi in millimetri).

	distanze reali	distanze calcolate	errori assoluti	errori percentuali
distanza P1-P2	330.0000	331.4216	1.4216	0.4308%
distanza P1-P3	258.0698	262.5078	4.4380	1.7197%
distanza P1-P4	366.1967	369.5281	3.3314	0.9097%
distanza P1-P5	241.8677	244.8301	2.9624	1.2248%
distanza P1-P6	182.4829	185.8194	3.3365	1.8284%
distanza P2-P3	276.5863	277.3019	0.7156	0.2587%
distanza P2-P4	212.1320	212.7143	0.5822	0.2745%
distanza P2-P5	169.7056	170.4948	0.7892	0.4650%
distanza P2-P6	349.8571	350.6484	0.7913	0.2262%
distanza P3-P4	150.0000	149.8093	0.1907	0.1271%
distanza P3-P5	108.1665	108.1488	0.0178	0.0164%
distanza P3-P6	123.6932	124.9747	1.2815	1.0360%
distanza P4-P5	127.2792	127.3584	0.0791	0.0622%
distanza P4-P6	271.6616	272.5842	0.9226	0.3396%
distanza P5-P6	189.7367	190.0455	0.3088	0.1627%

L'errore assoluto medio è di $1,4113mm$ (con deviazione standard $1,4036$), quindi ampiamente all'interno della tolleranza permettendo di affermare quindi che le coordinate calcolate sono corrette. Per completezza sono riportati i valori di errore assoluto medio, errore percentuale medio e deviazione standard dell'errore assoluto dati dal calcolo delle distanze delle coordinate restituite dall'algoritmo di triangolazione fornito con il toolbox di calibrazione:

errore assoluto medio	errore percentuale medio	deviazione standard
1,1161mm	0,4986%	1,2667

Come è possibile notare, questi valori medi non si discostano molto da quelli risultanti dall'algoritmo di triangolazione trattato nel precedente capitolo, quindi ciò conferma ulteriormente la validità di questo algoritmo.

Il codice Matlab completo di questa applicazione è disponibile nell'appendice B a pagina 79, mentre la funzione Matlab per la triangolazione delle coordinate è a pagina 80.

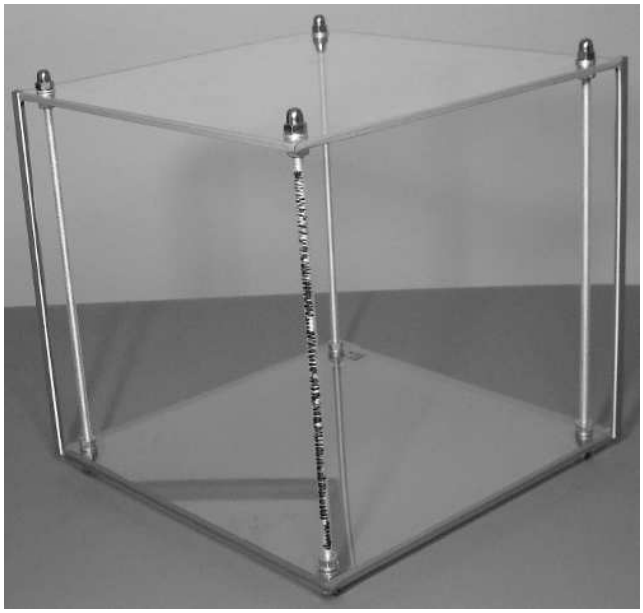


Figura 3.2: Immagine dell'oggetto.

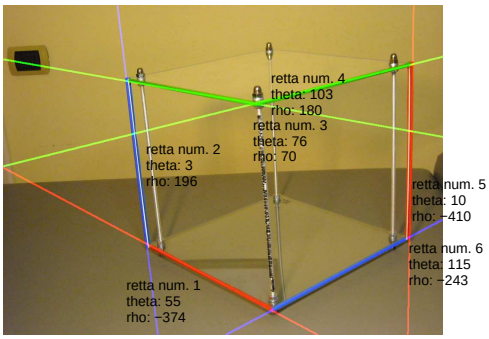
3.2 Localizzazione automatica di oggetti nello spazio

Si vuole ora automatizzare il processo di localizzazione della posizione di oggetti noti nello spazio e ricavare quindi le coordinate di punti corrispondenti tra le due immagini attraverso le tecniche analizzate e spiegate nel capitolo 1. In questa sezione ci si limiterà a fornire esempi di applicazioni pratiche date dalla combinazione di queste tecniche.

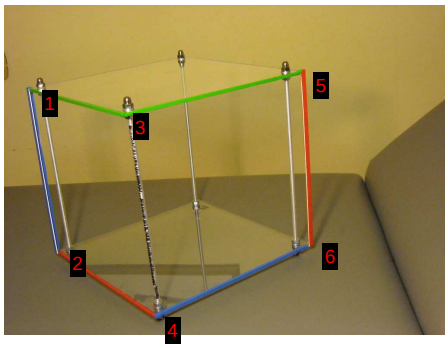
La maggior parte degli oggetti di cui potrebbe essere necessario ricavare la posizione nello spazio tramite visione stereoscopica presenta una forma delimitata da linee rette, come potrebbe essere ad esempio una casa, un'automobile, una strada, un pezzo semilavorato o un cavo elettrico, quindi viene presentato ora un esempio di rilevamento degli spigoli di un oggetto. Nella figura 3.2 è presentato un oggetto di forma approssimabile ad un cubo costruito con barre di acciaio e plexiglass i cui spigoli visibili

nell'immagine sono colorati in tre modi differenti, per cui una prima segmentazione dell'immagine avverrà per colori (come spiegato nella sezione 1.2 a pagina 9) scartando i pixel il cui valore di tono è nettamente diverso da quello desiderato, quindi si applica la trasformata di *Hough* (sezione 1.4, pagina 24) per rilevare la presenza di rette, corrispondenti agli spigoli. Queste si intersecano sui vertici dell'oggetto le cui coordinate sui piani immagine potranno poi essere processate per ricavare la posizione nello spazio dei vertici dell'oggetto. Nella figura 3.3(a) viene mostrata l'immagine data dalla webcam sinistra con evidenziate le rette rilevate dall'applicazione della trasformata di *Hough* ed i loro parametri, mentre nell'immagine della webcam destra in figura 3.3(b) vengono evidenziati e numerati i vertici visibili dell'oggetto.

Una forma geometrica a cui è possibile approssimare molti manufatti è la sfera, la quale proiettata in due dimensioni appare da qualsiasi punto d'osservazione come una circonferenza ed è molto utile nella visione stereoscopica perchè la triangolazione della posizione dei centri delle circonferenze proiettate sui piani immagine restituisce esattamente la posizione nello spazio del centro della sfera. Per questo motivo è presentato un esempio di applicazione pratica che consiste nel rilevare la presenza di circonferenze in una coppia di immagini raffiguranti tre sfere. Per facilitare la ricerca, inoltre, è possibile avvalersi delle linee epipolari (sull'immagine destra) ricavate dalla rilevazione dei centri delle circonferenze nell'immagine sinistra. Nella figura 3.4 è mostrato il risultato dell'applicazione della trasformata generalizzata di *Hough* (pagina 28) e della segmentazione per colore, evidenziando il centro della circonferenza e ricostruendo la circonferenza rilevata con il raggio trovato.

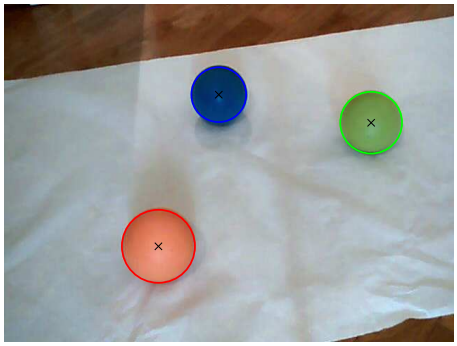


(a) webcam di sinistra, con evidenziate le rette rilevate

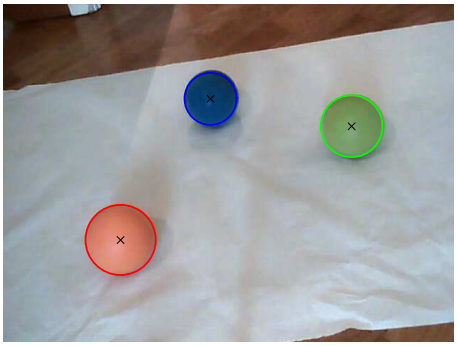


(b) webcam di destra, con evidenziati e numerati i vertici visibili

Figura 3.3: Spigoli e vertici rilevati dell'oggetto.



(a) webcam di sinistra



(b) webcam di destra

Figura 3.4: Centri delle circonferenze rilevati e circonferenze ricostruite.

3.3 Localizzazione stereoscopica di un oggetto in movimento

Come ultima applicazione si è scelto di localizzare nello spazio una sfera che rotola su di un piano tramite triangolazione stereoscopica utilizzando la coppia di webcam di cui sono stati calcolati i parametri di taratura. La sfera è inizialmente ferma ed è lasciata rotolare su di un piano inclinato fino ad incontrare un secondo piano inclinato su cui decelera, invertendo il moto per tornare sul primo piano inclinato, creando

un movimento oscillatorio smorzato dall'attrito volvente che si esaurisce quindi dopo pochi secondi.
L'acquisizione dei dati ha riportato però alcune irregolarità dovute ad alcune imperfezioni nel software di cattura e nell'ambiente di lavoro:

- Le webcam di acquisizione, al momento della redazione di questo testo, non vengono supportate da driver liberi quindi è stato necessario catturare il video con il software fornito dall'azienda che le ha prodotte, il quale però restituisce unicamente un flusso video compresso con codec *Windows Media Video* a basso bitrate quindi pesantemente compresso. I fotogrammi catturati presentano un alto livello di rumore ed altri difetti, tra cui una bassa gamma cromatica e sfocature di tipo *motion blur*.

- Le webcam sono costruite per poter catturare video ad una risoluzione di 640x480 pixel a 15 fotogrammi per secondo, quindi si avrà in generale una bassa frequenza di campioni per catturare il movimento dell'oggetto, risultando in incertezze in presenza di alte velocità.

- Per far risaltare l'oggetto si è posto uno sfondo di carta bianca la quale però presenta pieghe e zone d'ombra che possono influire sulla rilevazione, sarebbe quindi preferibile ricorrere a teli che presentino una certa rigidezza e che offrano anche una migliore riflessione della luce nel caso si volesse riproporre questa tecnica.

- L'illuminazione della scena è stata affidata a lampade fluorescenti mentre per una perfetta resa cromatica si dovrebbe ricorrere a lampade alogene per applicazioni fotografiche.

- La sfera è una palla di gomma spugnosa dalla superficie ruvida con irregolarità che quindi presenta un moto rotolatorio soggetto a discontinuità.

L'algoritmo di rilevazione della sfera sulle immagini riportate dalla webcam è stato quindi progettato per risultare il più robusto possibile, facendo ricorso a tutte le tecniche trattate nei capitoli precedenti, ed è stato diviso in 4 fasi:

1. Acquisizione del video ed estrazione dei singoli fotogrammi
2. Analisi del primo fotogramma per ricavare i valori iniziali
3. Analisi dei fotogrammi della telecamera sinistra e destra
4. Triangolazione delle coordinate

3.3.1 Acquisizione del video ed estrazione dei singoli fotogrammi

Il video è stato catturato tramite il software fornito dal produttore, quindi sono stati sincronizzati manualmente i due video e sono stati estratti i fotogrammi su cui si è voluto applicare l'algoritmo (in pratica, i fotogrammi in cui la sfera è in movimento) grazie al software *Avidemux* (disponibile su piattaforma *GNU/Linux*).

3.3.2 Analisi del primo fotogramma per ricavare i valori iniziali

Il primo fotogramma di ciascun video viene analizzato in maniera molto approfondita dall'algoritmo applicando la trasformata generalizzata di *Hough* (pagina 28) su un numero elevato di valori di raggi, conoscendo solo in modo qualitativo il valore del raggio della proiezione sui piani immagini del contorno dell'oggetto, per poter così ricavare le coordinate delle proiezioni della sfera. Completata l'analisi, si avranno in uscita il raggio e le coordinate delle circonferenze rilevate con cui proseguire l'analisi.

3.3.3 Analisi dei fotogrammi della telecamera sinistra e destra

Per ogni fotogramma successivo al primo viene applicato un algoritmo che prevede di creare una serie di "maschere" binarie che indicano le zone che soddisfano alcuni parametri e che quindi hanno più probabilità di appartenere alla proiezione della sfera:

1. Tono di colore: la sfera è di colore verde, quindi con un valore di tono compreso tra due valori precisi (figura 3.7(a))

2. Saturazione del colore: la sfera è di un colore molto saturo mentre lo sfondo, quasi bianco, presenta un colore poco saturo (figura 3.7(b))

3. Vicinanza: la posizione della sfera nel fotogramma deve essere simile a quella del fotogramma precedente, quindi viene creata una maschera che comprende un intorno di pixel, centrato nel centro della circonferenza nel fotogramma precedente, definito dal massimo scostamento che si potrebbe avere sommato ad un valore di tolleranza

Si applica quindi un AND logico sulle tre maschere binarie ricavate (figure 3.7(c), 3.7(d) e 3.7(e))formando un'ulteriore maschera su cui viene eseguita un'operazione morfologica di "Open" (trattata nell'appendice A, pagina 75) per rimuovere punti isolati (figura 3.7(f)). Quindi sul risultato è applicato l'algoritmo di ricerca dei contorni di *Canny* che restituisce un'immagine binaria dei bordi, la quale però subisce un'ultima operazione che consiste nell'AND logico con una matrice A binaria a diagonali alternate di valore 0 e 1 che ha lo scopo di dimezzare il numero di pixel con valore '1' per dimezzare artificiosamente la complessità di calcolo nei successivi passaggi.
Si passa quindi alla trasformata generalizzata di *Hough* la cui matrice di accumulazione nello spazio dei parametri presenta un picco che corrisponde al centro della

circonferenza rilevata nell'immagine. In figura 3.8 è evidenziato il contorno rilevato dall'algoritmo di *Canny* ed è mostrato il centro della circonferenza rilevato tramite trasformata di *Hough*. Questo passo è ripetuto per ogni fotogramma, sfruttando due variabili d'appoggio che memorizzano il raggio e la posizione del centro della circonferenza rilevata al passo precedente in modo da poter costruire la maschera di vicinanza e di poter applicare la trasformata di *Hough* su un numero limitato di raggi nell'intorno del raggio precedente che, per video dall'elevato numero di fotogrammi per secondo, può essere limitato al valore intero precedente e successivo al raggio della circonferenza nel fotogramma precedente.

3.3.4 Triangolazione delle coordinate

Ottenute le coordinate dei centri delle circonferenze nei fotogrammi di entrambe le webcam, si passa alla triangolazione ottenendo così le coordinate nello spazio della sfera nei singoli istanti catturati dai flussi video. Ad ogni coordinata è poi sottratta la prima coordinata, in modo da ottenere un sistema di riferimento solidale alla telecamera sinistra e con l'origine coincidente con la coordinata della sfera nel punto iniziale.

Gli algoritmi di localizzazione per i primi fotogrammi e per quelli successivi vengono presentati nell'appendice B a pagina 81.

3.3.5 Applicazione dell'algoritmo di localizzazione

Sono stati realizzati due set di fotogrammi corrispondenti a due video diversi e a ciascuno di essi è stato applicato l'algoritmo che ha portato ai risultati presentati nelle figure 3.9, 3.10, 3.11 e 3.12 (tutti i valori sono in millimetri) dove il punto rosso rappresenta il punto di partenza ed il verde rappresenta il punto finale. È stato anche

calcolato il modulo della velocità e la sua media mobile, mostrati nelle figure 3.13 e 3.14. I video analizzati sono presenti in allegato alla tesi e sono anche reperibili ai seguenti URL:

`http://www.jumbalaya.net/video/set1_sinistra.avi`
`http://www.jumbalaya.net/video/set1_destra.avi`
`http://www.jumbalaya.net/video/set2_sinistra.avi`
`http://www.jumbalaya.net/video/set2_destra.avi`

3.3.6 Valutazione dei risultati

Nei grafici dei risultati sono visibili chiaramente delle discontinuità dovute ad errori di acquisizione, i quali erano stati previsti dovuti alle già citate condizioni non perfette di acquisizione dei video. Il modulo della velocità è stato calcolato misurando la distanza tra una posizione e quella del fotogramma precedente e dividendola poi per l'intervallo di campionamento (in questo caso pari a 1/15 di secondo, ovvero circa 0.066 secondi), ma ovviamente ciò porta ad un'amplificazione del rumore che quindi è stato necessario filtrare, come è mostrato dalla linea nera nelle figure 3.13 e 3.14.
È interessante notare nel primo set il caso particolare in cui l'oggetto esce parzialmente dal campo visivo della telecamera sinistra (intorno al secondo 4) per cui l'algoritmo restituisce in quell'istante dei valori di posizione molto vicini tra loro che generano quindi un valore di velocità molto basso, anche se in realtà la sfera si muove con una velocità maggiore. Questo problema non si presenta nel secondo video, per cui il valore di velocità filtrato del secondo set presenta un'andamento sinusoidale esponenzialmente decrescente come è prevedibile nel caso di un oggetto fisico soggetto ad attrito volvente.

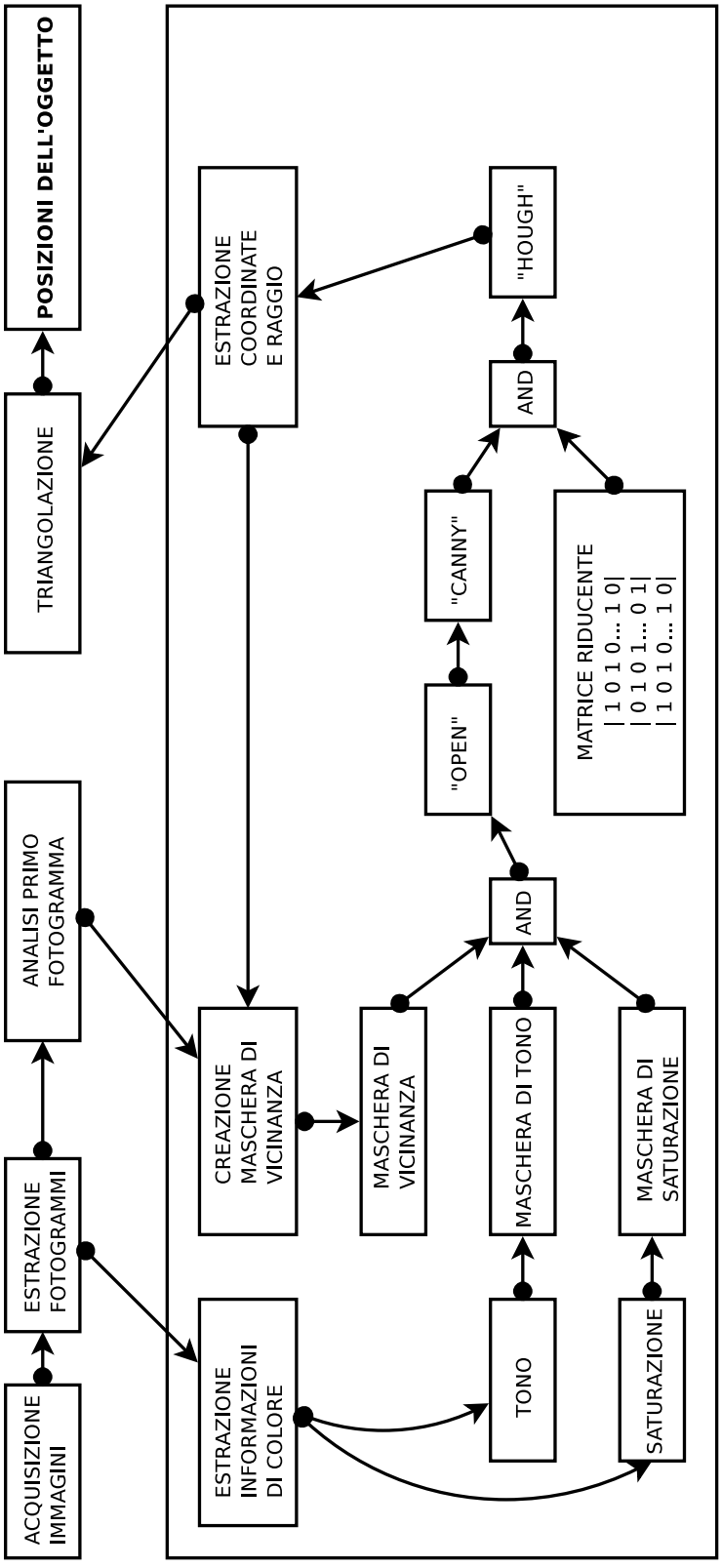
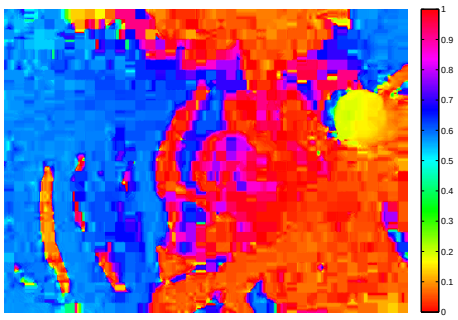
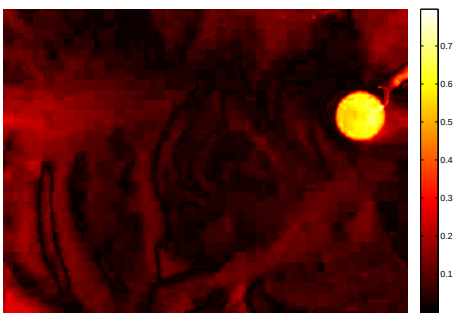


Figura 3.5: Schema dell'algoritmo di localizzazione della sfera.

Figura 3.6: Un fotogramma del video.



(a) mappa del tono di colore

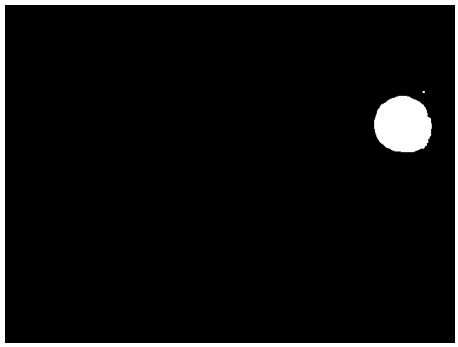


(b) mappa del livello di saturazione

(c) maschera ricavata dal tono di colore

(d) maschera ricavata dalla saturazione

(e) maschera di vicinanza



(f) maschera finale (dopo "open")

Figura 3.7: Passaggi dell'algoritmo di rilevazione della sfera.

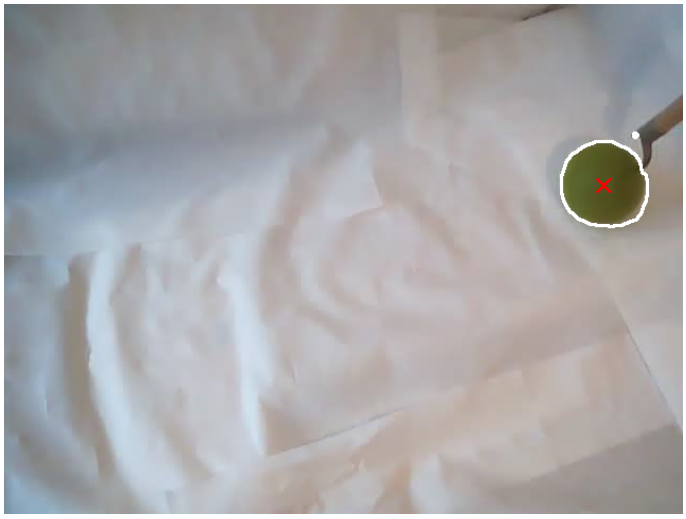


Figura 3.8: Contorno risultante e centro rilevato dall'algoritmo.

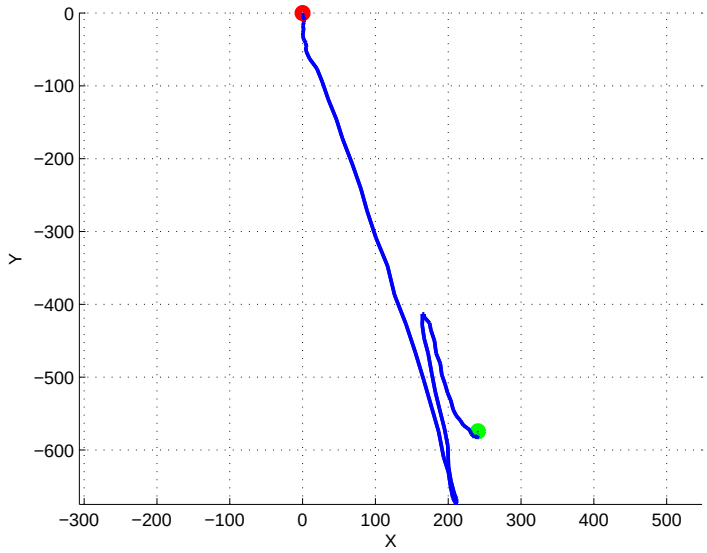


Figura 3.9: Traiettoria, visione dall'alto (assi X,Y) - primo set.

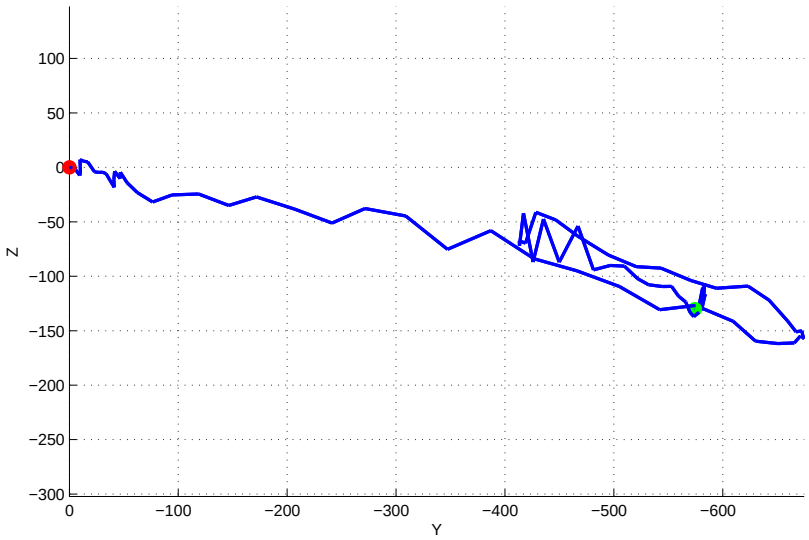


Figura 3.10: Traiettoria, visione laterale (assi Y,Z) - primo set.

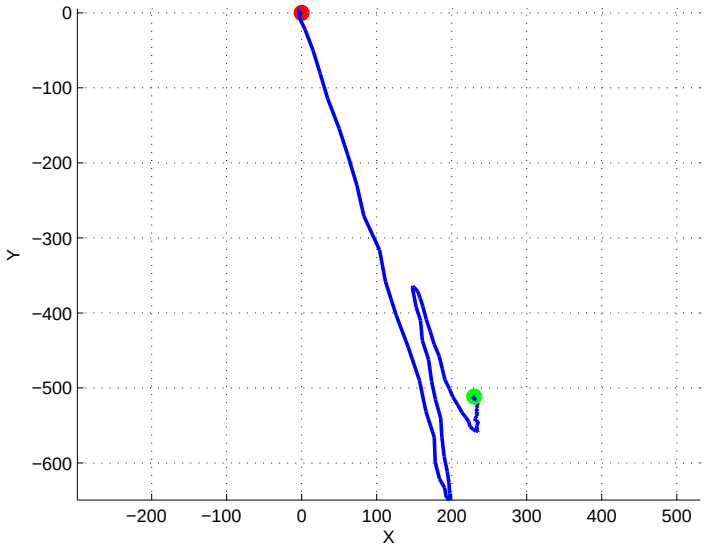


Figura 3.11: Traiettoria, visione dall'alto (assi X,Y) - secondo set.

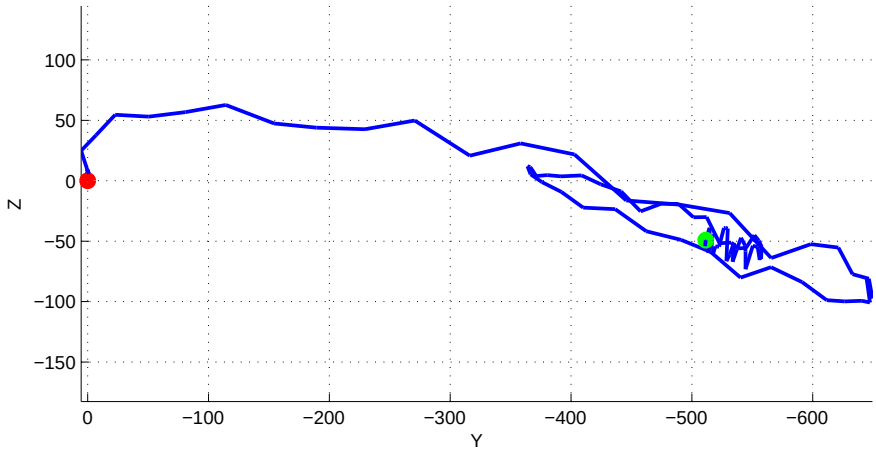


Figura 3.12: Traiettoria, visione laterale (assi Y,Z) - secondo set.

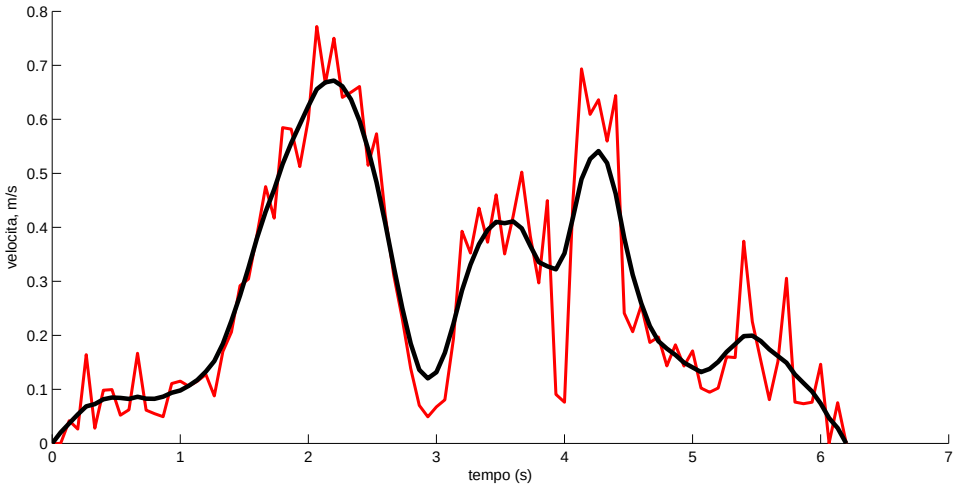


Figura 3.13: Modulo della velocità (rosso) e media mobile (nero) - primo set.

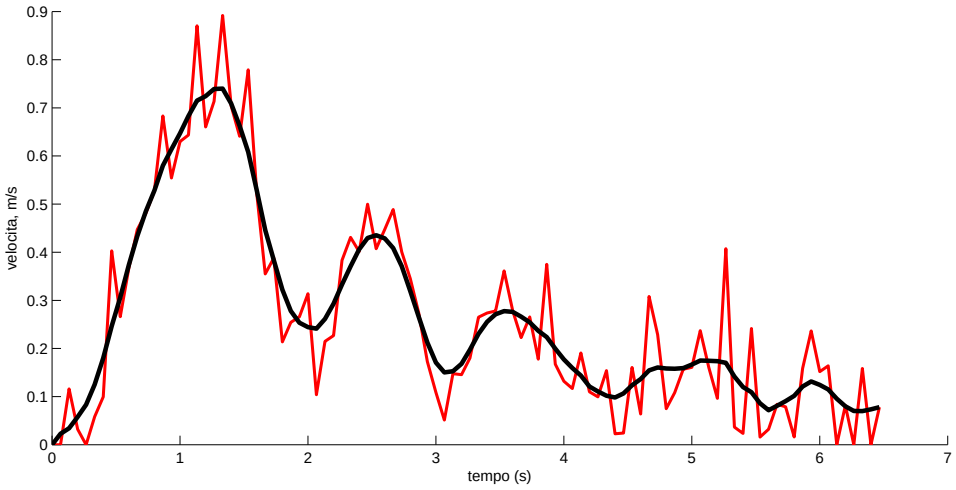


Figura 3.14: Modulo della velocità (rosso) e media mobile (nero) - secondo set.

Capitolo 4

Conclusioni e sviluppi futuri

Le applicazioni pratiche viste nell'ultimo capitolo sono solo un esempio tra le innumerevoli possibilità di un sistema di visione stereoscopica, la cui versatilità è data dal poter progettare un sistema di analisi che si basi sulle tecniche trattate nel capitolo 1 e su molte altre tecniche che non sono state trattate e che forniscono strumenti anche più potenti di quelli analizzati.

La localizzazione di oggetti, come si è visto, necessita di un buon algoritmo di riconoscimento che elabori le immagini date dai sensori estraendone le caratteristiche (dimensione, posizione) che verranno poi elaborate per restituire le posizioni nello spazio, difatti si è visto come le condizioni non ideali di cattura delle immagini porti ad errori non modesti nei risultati, per questo è molto importante anche concentrarsi sulle fasi precedenti all'acquisizione dei dati creando un ambiente controllato e conoscendo perfettamente tutte le condizioni operative.

L'estrema versatilità di questa tecnologia le permette di essere applicata in innumerevoli campi, ma ultimamente vede la più comune implementazione nei sistemi di guida assistita, in cui quindi si debba conoscere la distanza di un veicolo dai riferimenti posti sul manto stradale e dagli altri veicoli, favorendo inoltre la consapevolezza del

guidatore sulla presenza di cartelli stradali, pedoni sulla carreggiata od ostacoli sulla traiettoria. Inoltre permette di conoscere le caratteristiche e la posizione istantanea di oggetti a cui non possiamo avere accesso fisico, come volatili, pezzi semilavorati in una catena d'assemblaggio, persone, edifici.

Appendice A
Operazione morfologica di "open"

Un'operazione morfologica consiste nella modificazione della forma o struttura spaziale degli oggetti nell'immagine, modificandoli in funzione della loro topologia. Generalmente consiste nell'eseguire una funzione di matching tra ogni pixel dell'immagine di partenza con una maschera, definita come "elemento strutturale" cioè un matrice binaria di dimensioni 3x3. Un'operazione basica è detta *Erosione*, definita così:

$$D = F \ominus B = F(x, y) se B \subseteq F$$

Se ogni componente dell'elemento strutturale F, centrato nel valore del pixel sotto analisi dell'immagine B, non coincide esattamente con quello degli adiacenti, il pixel viene posto al valore complementare a quello dell'elemento strutturale, cioè della maschera di matching, nell'immagine d'uscita. In caso contrario si pone il pixel allo stesso valore delle componenti dell'elemento strutturale. Ciò può essere implementato tramite una matrice 3x3 con entrate '1' che viene centrata su tutti i pixel dell'immagine originale restituendo un valore binario che è uguale a 0 se tutti i valori del pixel nell'intorno hanno valore 0 (compreso il pixel stesso). In pratica si ha una vera erosione dell'immagine rimuovendo un pixel dal bordo. Una seconda operazione basica, opposta alla precedente, è detta *Dilatazione*:

$$D = F \oplus B$$

Se il pixel sotto esame o alcuni pixel adiacenti hanno un valore coincidente al pixel corrispondente dell'elemento strutturale, al pixel sotto esame viene assegnato il valore dell'elemento strutturale. Un'implementazione consiste nel creare una matrice 3x3 con entrate '1' che pone a 1 il pixel se anche solo uno dei pixel adiacenti ha valore 1, il che equivale ad aggiungere un pixel al bordo dell'immagine.

Esempio Viene ora fornita un'immagine binaria espressa come matrice:

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

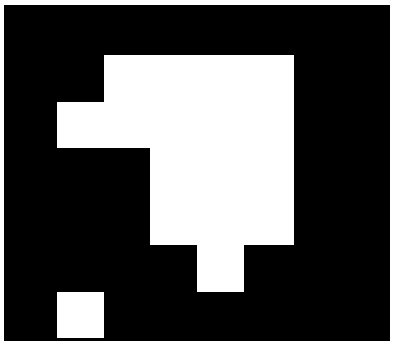
la quale è visibile come immagine nella figura 4.1(a).

Viene quindi creato l'elemento strutturale F:

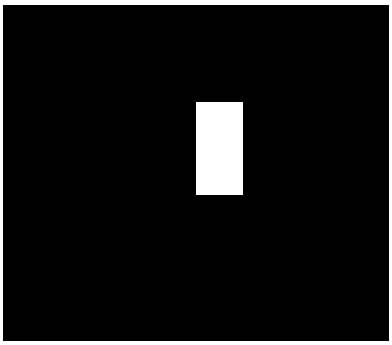
$$F = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

e viene eseguita l'operazione di erosione che porta al risultato mostrato in figura 4.1(b), mentre il risultato dell'operazione di dilatazione è mostrato in figura 4.1(c).

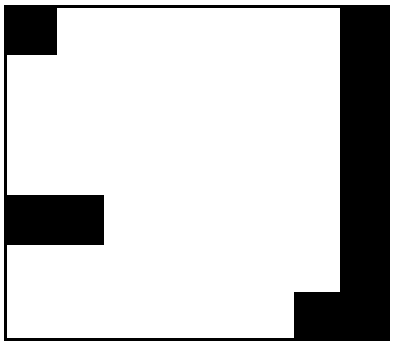
L'operazione di *Open* consiste nell'eseguire un'operazione di erosione seguita da una dilatazione, permettendo così di eliminare punti isolati con valore 1 dell'immagine binaria, come nel risultato mostrato in figura 4.1(d).



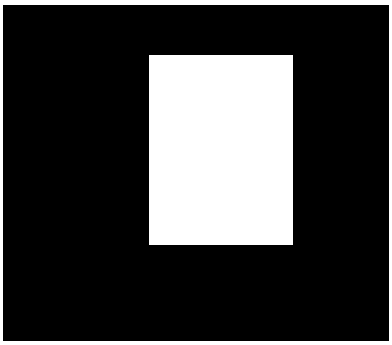
(a) matrice originale



(b) risultato operazione di erosione



(c) risultato operazione di dilatazione



(d) risultato operazione di 'open'

Figura 4.1: Operazioni morfologiche.

(a) immagine originale

(b) risultato operazione di 'open'

Figura 4.2: Risultato di un'operazione di 'open' su un'immagine con disturbi

Appendice B
Funzioni in codice Matlab

prospettiva_inversa.m

```
%prospettiva_inversa
%  INPUT:
%    img:   immagine di partenza, codifica uint8
%    ver:   coordinate dei 4 vertici, matrice 4x2
%    dim:   dimensioni dell'immagine risultante (valore a scelta)
% OUTPUT:
%      Y:   immagine risultante, codifica uint8

function [ Y ] = prospettiva_inversa(img, ver, dim)
%ordine dei punti da scegliere: alto-sx, senso antiorario
%in Xo Yo i punti nell'img originale
Xo=ver(:,1);
Yo=ver(:,2);
%creo i vettori con gli estremi della nuova immagine in cui disegnare la
%rettificata
sizex=dim(1);
sizey=dim(2);
Yp=[0;   0; sizey; sizey];
Xp=[0; sizex; sizex;   0];
%costruisco le matrici B e D con le coordinate
B = [ Xp Yp ones(4,1) zeros(4,3)        -Xp.*Xo -Yp.*Xo ...
      zeros(4,3)        Xp Yp ones(4,1) -Xp.*Yo -Yp.*Yo ];
B = reshape (B', 8 , 8 )';
X = [ Xo , Yo ];
X = reshape (X', 8 , 1 );
%calcolo lambda
l=B\X;
%calcolo le matrici A e C
A = reshape([l(1:8)'  1 ],3,3)';
C = [l(7:8)' 1];
%ora dato un punto della rettificata posso sapere a quale punto corrisponde
%dell'originale con: p'=A*p/C*p
%esempio: pp=A*[xp;yp;1]/(C*[xp;yp;1])
%una volta trovato pp, posso usare i metodi di interpolazione per ricavare
%il valore da piazzare nella img rettificata.
%per semplicità, uso il valore del pixel che meglio approssima le
%coordinate trovate
Y=zeros(sizex,sizey,3,'uint8');
for xp=1:sizex
    for yp=1:sizey
        pp=A*[xp;yp;1]/(C*[xp;yp;1]);
        ppr=round(pp);
        Y(xp,yp,:)=img(ppr(1),ppr(2),:);
    end
end
```

rette polari.m

```
%carico le due img
S=double(rgb2gray(imread('S_19.jpg')));
D=double(rgb2gray(imread('D_19.jpg')));
[m n] = size(S);
figure(1),imshow(S,[]);
axis image;
figure(2),imshow(D,[]);
axis image;
%inserico le matrici P
PL=[844.2543         0  343.9745         0;
    0  858.5060  232.0483         0;
    0         0    1.0000         0];
PR=[753.355387141362,-26.9647026673393,497.632464724751,-150894.492870716;
    -27.7608382310299,855.050588846018,225.400975512392,7257.82371057045;
```

```
    -0.219747493714791,-0.0152400881391118,0.975437736977394,45.0867652955666];
%% creo la matrice fondamentale F
X1 = PL([2 3],:);
X2 = PL([3 1],:);
X3 = PL([1 2],:);
Y1 = PR([2 3],:);
Y2 = PR([3 1],:);
Y3 = PR([1 2],:);
F = [det([X1; Y1]) det([X2; Y1]) det([X3; Y1])
     det([X1; Y2]) det([X2; Y2]) det([X3; Y2])
     det([X1; Y3]) det([X2; Y3]) det([X3; Y3])];

%% punto 1
s_x=98;
s_y=459;

figure(1);
hold on;
plot(s_x, s_y, 'ws','MarkerSize',13,'LineWidth',2);
plot(s_x, s_y, 'k+','MarkerSize',13,'LineWidth',2);

%linea epipolare
s_P = [s_x; s_y; 1];
d_P = F*s_P;
d_epipolo_x = 1:2*m;

%ax+by+c=0; y = (-c-ax)/b
d_epipolo_y = (-d_P(3)-d_P(1)*d_epipolo_x)/d_P(2);

figure(2); hold on;
plot(d_epipolo_x, d_epipolo_y, 'w','LineWidth',3);

%% punto 2
s_x=296;
s_y=25;

figure(1);
hold on;
plot(s_x, s_y, 'ws','MarkerSize',13,'LineWidth',2);
plot(s_x, s_y, 'k+','MarkerSize',13,'LineWidth',2);
s_P = [s_x; s_y; 1];
d_P = F*s_P;
d_epipolo_x = 1:2*m;
d_epipolo_y = (-d_P(3)-d_P(1)*d_epipolo_x)/d_P(2);
figure(2); hold on;
plot(d_epipolo_x, d_epipolo_y, 'w','LineWidth',3);
```

triangolazione.m

```
clear all
close all
%carico le due img
S=double(rgb2gray(imread('S_19.jpg')));
D=double(rgb2gray(imread('D_19.jpg')));
figure(1),imshow(S,[]);
axis image;
figure(2),imshow(D,[]);
axis image;

%% inserisco i valori dalla calibrazione delle webcam
load parametri.mat

%% inserimento coordinate
xL=[287 9 273 138 173 359
    59 231 307 395 264 214];
xR=[272 18 237 111 152 325
    60 219 297 373 253 210];

figure(1);
hold on;
for i=1:size(xL,2)
plot(xL(1,i), xL(2,i), 'ks','MarkerSize',13,'LineWidth',2);
plot(xL(1,i), xL(2,i), 'w+','MarkerSize',13,'LineWidth',2);
text(xL(1,i)+10,xL(2,i)+10,int2str(i),'FontSize',15,'BackgroundColor',[0.8 0.8 0.8]);
end

for i=1:size(xL,2)
figure(2);
hold on;
plot(xR(1,i), xR(2,i), 'ks','MarkerSize',13,'LineWidth',2);
plot(xR(1,i), xR(2,i), 'w+','MarkerSize',13,'LineWidth',2);
text(xR(1,i)+10,xR(2,i)+10,int2str(i),'FontSize',15,'BackgroundColor',[0.8 0.8 0.8]);
end

X=triangolazione_stereo(xL,xR,R,T,fc_left,cc_left,kc_left,fc_right,cc_right,kc_right);

%% coordinate sistema calib
```

```
xL_d=[(xL(1,:) - cc_left(1))/fc_left(1);(xL(2,:) - cc_left(2))/fc_left(2)];
xR_d=[(xR(1,:) - cc_right(1))/fc_right(1);(xR(2,:) - cc_right(2))/fc_right(2)];
xL_n = comp_distortion_oulu(xL_d,kc_left);
xR_n = comp_distortion_oulu(xR_d,kc_right);
xL_n = [xL_n;ones(1,size(xL,2))];
xR_n = [xR_n;ones(1,size(xL,2))];
xt=xL_n;
xtt=xR_n;
N = size(xt,2);
u = R * xt;
n_xt2 = dot(xt,xt);
n_xtt2 = dot(xtt,xtt);
T_vect = repmat(T, [1 N]);
DD = n_xt2 .* n_xtt2 - dot(u,xtt).^2;
dot_uT = dot(u,T_vect);
dot_xttT = dot(xtt,T_vect);
dot_xttu = dot(u,xtt);
NN1 = dot_xttu.*dot_xttT - n_xtt2 .* dot_uT;
NN2 = n_xt2.*dot_xttT - dot_uT.*dot_xttu;
Zt = NN1./DD;
Ztt = NN2./DD;
X_1 = xt .* repmat(Zt,[3 1]);
X_2 = R'*(xtt.*repmat(Ztt,[3,1])  - T_vect);
disp('sistema calib');
Xc = 1/2 * (X_1 + X_2);

%% verifico l'errore nei tre punti - singole distanze - mio sistema
% Distanze reali tra:
% 1. primo punto - secondo punto
% 2. primo punto - terzo punto
% 3. primo punto - quarto punto
% 4. primo punto - quinto punto
% 5. primo punto - sesto punto
% 6. secondo punto - terzo punto
% 7. secondo punto - quarto punto
% 8. secondo punto - quinto punto
% 9. secondo punto - sesto punto
%10. terzo punto - quarto punto
%11. terzo punto - quinto punto
%12. terzo punto - sesto punto
%13. quarto punto- quinto punto
%14. quarto punto- sesto punto
%15. quinto punto- sesto punto

distanze=30*[11,sqrt((7)^2+(5)^2),sqrt((7)^2+(10)^2),...
    sqrt((4)^2+(7)^2),sqrt((1)^2+(6)^2),sqrt((7)^2+(6)^2),sqrt((7)^2+(1)^2),...
    sqrt((4)^2+(4)^2),sqrt((6)^2+(10)^2),5,sqrt((2)^2+(3)^2),sqrt((1)^2+(4)^2),...
    sqrt((3)^2+(3)^2),sqrt((1)^2+(9)^2),sqrt((2)^2+(6)^2)]

% Calcolo le distanze tra i 6 vettori
distanze_calcolate=pdist(X')

% Errore assoluto
errore_assoluto=abs(distanze-distanze_calcolate)

% Errore relativo percentuale
errore_percentuale=errore_assoluto./distanze*100

%% verifico l'errore nei tre punti - singole distanze - sistema Calib
% Calcolo le distanze tra i 6 vettori
distanze_calcolate_c=pdist(Xc')

% Errore assoluto
errore_assoluto_c=abs(distanze-distanze_calcolate_c)
% Errore relativo percentuale
errore_percentuale_c=errore_assoluto_c./distanze*100

%% media errori

errore_medio=mean(errore_assoluto)
errore_medio_c=mean(errore_assoluto_c)

%% deviazione standard

deviazione_standard=std(errore_assoluto)
deviazione_standard_c=std(errore_assoluto_c)
```

triangolazione_stereo.m

```
%[X]=triangolazione_stereo(x,R,T,fc_left,cc_left,kc_left,fc_right,cc_right,kc_right)
%   INPUT
%   xJ: vettore [xJ1 xJ2 ...], 2xN
%   OUTPUT:
%   X: vettore [XL1 XL2 ...], 3xN
function [X]=triangolazione_stereo(xL,xR,R,T,fc_left,cc_left,kc_left,fc_right,cc_right,kc_right)

% normalizzo sottraendo le coords del centro ottico e dividendo per fk
```

```
xL_d=[(xL(1,:) - cc_left(1))/fc_left(1);(xL(2,:) - cc_left(2))/fc_left(2)];
xR_d=[(xR(1,:) - cc_right(1))/fc_right(1);(xR(2,:) - cc_right(2))/fc_right(2)];

% elimino distorsione
xL_n = comp_distortion_oulu(xL_d,kc_left);
xR_n = comp_distortion_oulu(xR_d,kc_right);

% rendo le coordinate omogenee
xL_n = [xL_n;ones(1,size(xL,2))];
xR_n = [xR_n;ones(1,size(xL,2))];

% replico T per tutte le coordinate in esame
T = repmat(T, [1 size(xL,2)]);

% calcolo N,D
alpha_L=-R*xL_n;
N=dot(xR_n,xR_n).*dot(alpha_L,T)-dot(alpha_L,xR_n).*dot(xR_n,T);
D=dot(xR_n,xR_n).*dot(alpha_L,alpha_L)-dot(alpha_L,xR_n).*dot(alpha_L,xR_n);

%calcolo Z
Z=N./D;

% coordinate X
X=xL_n.*(repmat(Z,[3,1]));
return
```

preparazione_ricerca2.m

```
function [raggio_s,raggio_d,picco_s,picco_d]=preparazione_ricerca2(raggio,hue_s,hue_d,sat_s,sat_d,file)

%% sinistra
img=imread([file 's1.jpg']);
figure(1)
imshow(img);
img=rgb2hsv(img);

% cerco il verde
maschera=img(:,:,1)>hue_s(1)&img(:,:,1)<hue_s(2);

%elimino le parti saturate (bianche)
maschera=maschera & img(:,:,2)>sat_s(1) & img(:,:,2)<sat_s(2);

%cerco i contorni
I=edge(bwmorph(maschera,'open'),'Canny');

% maschera riducente
A=[1 0;0 1];
A=repmat(A,size(img,1)/2,size(img,2)/2);
I=I&A;

figure(2)
imshow(I);
drawnow;

% applico Hough_cerchi alla prima immagine
% seleziono i raggi entro cui cercare
raggi=raggio-6:1:raggio+6;

cont=0;
while ~cont
    [picco_s,raggio_s]=hough_cerchi(I,raggi);
    %plotto il cerchio trovato
    figure(1);hold on;
    rectangle('Position',[picco_s(2)-raggio_s,picco_s(1)-raggio_s,raggio_s*2,raggio_s*2],...
        'Curvature',[1,1],'EdgeColor','red','LineWidth',2)
    hold off
    drawnow;
    % se il cerchio va bene, continuo, altrimenti riprovo con un raggio più
    % grande (o una maggiore risoluzione?)
    cont=input('La img è valida? - 1 si, 0 no (default:no): ');
    raggi=raggi+20;
end

%% destra
img=imread([file 'd1.jpg']);
figure(1)
imshow(img);
img=rgb2hsv(img);

% cerco il verde
maschera=img(:,:,1)>hue_d(1)&img(:,:,1)<hue_d(2);

%elimino le parti saturate (bianche)
maschera=maschera & img(:,:,2)>sat_d(1) & img(:,:,2)<sat_d(2);

%cerco i contorni
I=edge(bwmorph(maschera,'open'),'Canny');
```

```
% maschera riducente
A=[1 0;0 1];
A=repmat(A,size(img,1)/2,size(img,2)/2);
I=I&A;

figure(2)
imshow(I);
drawnow;

% applico Hough_cerchi alla prima immagine
%seleziono i raggi entro cui cercare
raggi=raggio-6:1:raggio+6;

cont=0;
while ~cont
    [picco_d,raggio_d]=hough_cerchi(I,raggi);
    %plotto il cerchio trovato
    figure(1);hold on;
    rectangle('Position',[picco_d(2)-raggio_d,picco_d(1)-raggio_d,raggio_d*2,raggio_d*2],...
        'Curvature',[1,1],'EdgeColor','red','LineWidth',2)
    hold off
    drawnow;
    % se il cerchio va bene, continuo, altrimenti riprovo con un raggio più
    % grande (o una maggiore risoluzione?)
    cont=input('La img è valida? - 1 si, 0 no (default:no): ');
    raggi=raggi+20;
end
close all
drawnow;
return
```

ricerca_cerchi2.m

```
function x=ricerca_cerchi2(num_immagini,raggio,hue,sat,picco,dx,stringa,file)

stringa=[file stringa];

dim=size(imread([stringa '1.jpg']));

%creo una matrice che dimezzi il numero di pixel del contorno
A=[1 0;0 1];
A=repmat(A,dim(1)/2,dim(2)/2);

%inizio iterazione tra tutte le immagini
x=zeros(2,num_immagini);

for i=1:num_immagini
    i
    raggio
    %maschera mobile per le successive iterazioni
    maschera_mobile=logical(false(dim(1),dim(2)));
    [a,b,c,d]=intorno(dim(1),dim(2),picco,raggio,dx);
    maschera_mobile(a:b,c:d)=true;

    %creo raggi
    raggi=raggio-1:raggio+1;

    img=rgb2hsv(imread([stringa int2str(i) '.jpg']));

    maschera=(img(:,:,1)>hue(1)) & (img(:,:,1)<hue(2)) & img(:,:,2)>sat(1) &...
        img(:,:,2)<sat(2) & maschera_mobile;
    maschera=edge(bwmorph(maschera,'open'),'Canny') & A;

    %applico hough_cerchi
    [picco,raggio]=hough_cerchi(maschera,raggi);
    save('debug.mat','stringa','i','hue','sat','maschera_mobile','maschera','A',...
        'raggio','picco','x');

    %salvo il valore del centro trovato
    x(:,i)=picco';
end
return
```

Elenco delle figure

Bibliografia

[1] Rafael C. Gonzalez - Richard E. Woods, *"Digital image processing"*, Pearson Prentice Hall, 2008.

[2] Simon Just - Kjeldgaard Pedersen, *"Circular Hough Transform"*, Aalborg University, Vision, Graphics, and Interactive Systems, 2007

[3] Davide Scaramuzza, *"Progetto e realizzazione di un sistema di visione stereoscopica per la robotica, con applicazione all'inseguimento di corpi in moto e all'autolocalizzazione"*, Tesi di laurea in Ingegneria Elettronica, Università di Bologna, 2004.

[4] Duane C. Brown, *"Close-Range Camera Calibration"*, DBA Systems Inc., 1971.

[5] Zhengyou Zhang, *"Flexible Camera Calibration By Viewing a Plane From Unknown Orientations"*, Microsoft Research, 1999.

[6] Richard Hartley - Andrew Zisserman, *"Multiple view geometry in computer vision"*, Cambridge University Press, 2003.