ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

SCUOLA DI INGEGNERIA E ARCHITETTURA
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

SEGMENTAZIONE DEL PAVIMENTO DI UNA SCENA BASATA SUL RICONOSCIMENTO DEL PATTERN

Tesi in
ELABORAZIONE DELLE IMMAGINI LM

Relatore:
Prof. Alessandro Bevilacqua

Presentata da:
Silvia Franci

Correlatore:
Dr. Ing. Alessandro Gherardi

Sessione III
Anno Accademico 2012–2013

Un'immagine vale più di mille parole.
(Anonimo)

Indice

Capitolo 1

Introduzione

Negli ultimi anni si è vista una grande crescita di software di Realtà Aumentata che consente di arricchire le percezioni dell'utente sull'ambiente circostante attraverso informazioni aggiuntive presentate su diversi supporti, ad esempio i display di smartphone e tablet, cruscotti delle automobili e particolari occhiali. Le applicazioni sono molteplici: si va dal riconoscimento dei QR code (QR Droid^{TM}per sistemi Android [1]) ai sistemi per auto che individuano, riconoscono e proiettano i cartelli stradali sul parabrezza (BMW Head-Up Display [2]) oppure aiutano nelle manovre di parcheggio, applicazioni di *geotagging* (Acrossair per iOS [3]), numerosi giochi, fino ad arrivare ai Google Glass [4]. Questi software sfruttano diverse tecniche di elaborazione delle immagini, ma hanno una componente comune: la segmentazione, che permette di individuare ed isolare gli oggetti o le zone di interesse presenti nei vari contesti, su cui verranno poi eseguite le successive elaborazioni.
Nel contesto della tesi è stata sviluppata un'applicazione di Realtà Aumentata per tablet Android nell'ambito dell'*interior design*. Dopo aver acquisito un'istantanea con la fotocamera del dispositivo di un ambiente interno, è possibile selezionare attraverso un "tocco" sul display il rivestimento (pavimento o parete) di cui si vuole simulare il cambiamento di colore o texture. Una volta individuata ed evidenziata l'area di interesse, in tempo reale è possibile cambiare interattivamente l'aspetto del rivestimento precedentemente selezionato. La tesi si focalizza sulla ricerca di un metodo che consenta di

avere una segmentazione accurata della superficie di interesse.
L'algoritmo di segmentazione studiato, utilizzato nell'applicazione di Realtà Aumentata, è sviluppato nel contesto della collaborazione tra il *Computer Vision Group (CVG)*[1], coordinato dal Prof. Alessandro Bevilacqua e Maticad S.r.l., un'azienda che opera nel settore dell'*Information Technology*, *Distributed Applications*, Internet e Computer Grafica, presso la quale ho effettuato un periodo di tirocinio. Maticad, oltre a software per pc desktop, sviluppa applicazioni per iOS e in questo contesto, durante il tirocinio, ho sviluppata un'applicazione demo per iOS 7 volta a studiare le prestazioni dei sensori (ottico, inerziali, magnetici), in vista di un futuro *porting* dell'applicazione su quel sistema operativo.

La tesi è così suddivisa. Nel capitolo 2 vengono analizzati diversi algoritmi per la segmentazione di immagini presenti in letteratura e viene effettuato un confronto su immagini inerenti l'ambito dell'applicazione. Nel capitolo 3 vengono analizzate le problematiche relative alla segmentazione di immagini di ambienti interni. Verrà inoltre proposto un metodo per migliorare l'efficacia e la precisione dell'algoritmo di segmentazione usato. Nel capitolo 4 si illustrano le varie tipologie di superfici individuate e in che modo queste possono essere riconosciute e classificate con tecniche di *machine learning*. Il capitolo 5 descrive quali parti dell'applicazione sono state sviluppate mentre nel capitolo 6 sono esposte le scelte effettuate per affrontare le problematiche relative al *tuning* dei vari parametri dell'algoritmo di segmentazione. Infine, nel capitolo 7 si discutono i risultati sperimentali ottenuti dall'applicazione del metodo di raffinamento della segmentazione proposto.

[1]Sito del *CVG*: `http://cvg.deis.unibo.it`

Capitolo 2

Stato dell'Arte

La segmentazione di immagini è un ambito di ricerca molto studiato. In letteratura sono presenti molte tecniche e algoritmi, ma non esiste un metodo valido per tutti gli scenari possibili. La segmentazione di immagini consiste nel suddividere un'immagine in regioni omogenee e significative che variano in base all'applicazione e al contesto. In generale, per segmentare un'immagine, si deve eseguire una classificazione dei pixel in modo da raggruppare quelli che hanno caratteristiche e proprietà comuni (ad esempio colore, intensità o texture) e che apparterranno quindi ad un medesimo oggetto. È molto difficile ottenere una segmentazione precisa e accurata, in quanto ogni tecnica ha i suoi pregi e difetti e prende in considerazione determinati aspetti.
In questa sezione ci occuperemo di analizzare, confrontare e suddividere i vari metodi presenti in letteratura.

2.1 *Snake*

Il metodo degli *snake*, appartiene alla famiglia degli *active contours*, in cui viene specificato un contorno grossolano dell'oggetto di interesse, si procede poi iterativamente muovendo il contorno verso la soluzione finale attraverso una combinazione di informazioni derivate dall'immagine stessa e l'intervento dell'utente. Nel caso degli *snake* si utilizza una curva che, da ogni punto di partenza, si deforma per raggiungere il contorno più marcato. La curva, o

snake è una *spline* bidimensionale che si "muove" cercando di minimizzare una funzione di energia composta da tre termini [5]:

- *Energia interna*: forza la *spline* ad essere piccola e omogenea. Per trovare questa energia si sfruttano le derivate della funzione che controlla la forma della *spline*. In particolare la derivata prima (la sua intensità è maggiore con l'aumentare della lunghezza dello *snake*) e la derivata seconda (la sua intensità è maggiore per curve più acute). Azzerando in un punto la componente relativa alla derivata seconda si possono creare degli angoli.

- *Elementi dell'immagine*: responsabili dell'avvicinamento della *spine* ai bordi di un oggetto e sono definiti sull'immagine. Si può utilizzare una combinazione di:

  - Intensità dell'immagine: in base ad un peso attrae la curva verso linee più chiare o più scure.
  - Funzione inversa del gradiente: associando bassa energia nelle aree in cui sono presenti i bordi e alta energia altrimenti la curva sarà attratta verso i contorni con valore del gradiente maggiore.

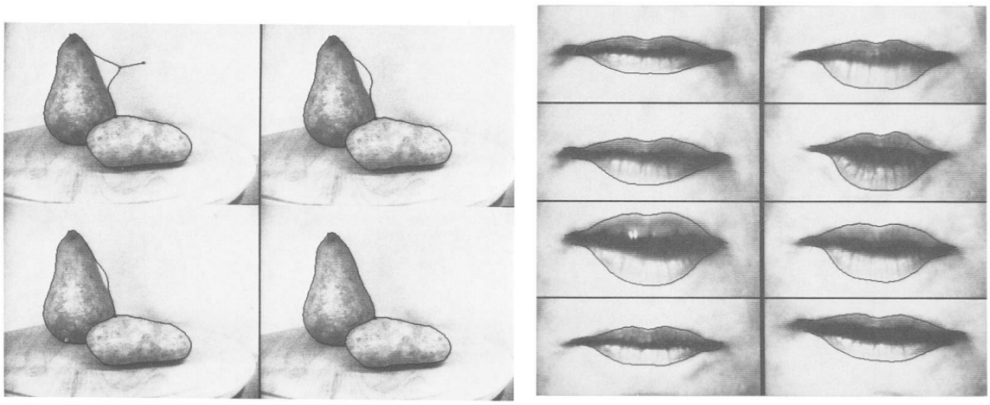
- *Energia esterna*: è data dall'interazione con l'utente che può definire dei punti che attraggono o respingono la curva verso determinate zone.

Per risolvere efficientemente il problema della minimizzazione dell'energia, modellabile come un sistema lineare sparso, si può usare uno *sparse direct solver* [6].

Una variante di questo metodo prevede l'uso delle *B-spline*, in modo da avere meno gradi di libertà e poter stimare una forma a priori in base alla distribuzione tipica dei *control point* dell'oggetto che si sta analizzando. Per allineare il modello di distribuzione dei punti, per ogni *control point* si esegue una ricerca, in una direzione normale al contorno, per trovare il miglior punto sul bordo dell'immagine. In applicazioni interattive è possibile inserire ulteriori vincoli definiti dall'utente, ad esempio l'inserimento di punti che

attraggono o respingono la curva in determinate zone [6].

Per applicazioni dinamiche, in cui un oggetto si deforma ed evolve in ogni frame, si possono sfruttare stime acquisite nei frame precedenti per predire e vincolare la nuova forma dell'oggetto. Per fare ciò si può utilizzare una tecnica chiamata *Kalman snake* in cui, all'inizio di ogni step di valutazione, la densità di probabilità viene aggiornata in base ad un modello dinamico lineare (le osservazioni precedenti cambiano in modo deterministico). Viene inoltre aggiunta una componente di perturbazione (modellata solitamente come una Gaussiana) in modo da avere una diffusione stocastica della densità di probabilità ed un aumento di entropia nel sistema (quindi di incertezza). Le misurazioni nel frame corrente introducono nuove informazioni che aiutano a rifinire la stima corrente e quindi ripristinano un certo grado di certezza [6]. In Figura 2.1a si può vedere l'evoluzione dello *snake* verso il bordo dopo che è stato spostato dall'utente. In Figura 2.1b un esempio di applicazione dinamica in cui si traccia una bocca.



(a) (b)

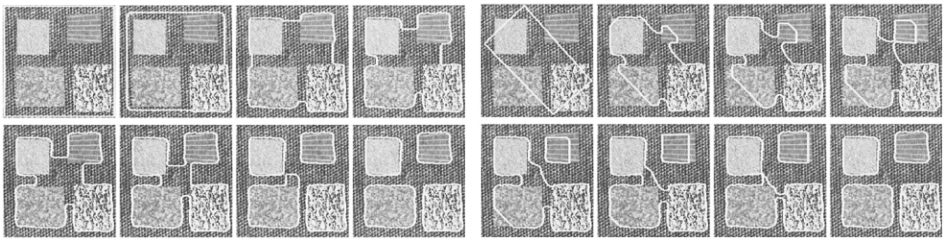
Figura 2.1: Applicazioni del metodo *snake* [5]. (a) Evoluzione di una *spline* verso il contorno dell'oggetto. (b) Tracciamento dei movimenti della bocca.

2.2 *Level set*

I metodi precedentemente descritti sono modellati attraverso delle curve parametriche (ad esempio le *B-spline*), il che rende difficile cambiarne la topologia mentre evolve. Inoltre, se la forma dell'oggetto cambia drasticamente, bisogna effettuare una riparametrizzazione della curva. Una approccio alternativo, il *level set*, permette di tracciare l'evoluzione di una curva definita in modo implicito, rappresentandola come l'insieme di punti in cui una determinata funzione vale zero [7]. La rappresentazione implicita della curva dà la possibilità di gestire i cambiamenti della curva (anche l'unione e la divisione di diversi segmenti) in modo automatico [8]. Inoltre, mentre si deforma la curva iniziale, si possono anche misurare determinate proprietà sia dentro che fuori alla regione definita dalla curva (*feature* del *background* e del *foreground*) [9].
Le definizioni della curva possono essere molteplici, a seconda dell'applicazione che si andrà a considerare, ad esempio:

- *Texture segmentation*: la funzione obiettivo consiste nel trovare la curva geodesica di lunghezza minima che corrisponde ai pixel con più alta probabilità di essere contorni. Per l'evoluzione della curva, ed il raggiungimento della segmentazione finale, si tiene conto dell'influenza delle caratteristiche statistiche della regione e dei bordi presenti nell'immagine. La curva viene inoltre vincolata dalle caratteristiche interne della regione (ad esempio la regolarità). [10]. Un esempio è dato dalla Figura 2.2.

- *geodesic active contour*: la funzione obiettivo è data dalla somma di due termini: uno che fa muovere la curva nella direzione di maggior curvatura del bordo dell'immagine e l'altro termine muove la curva verso intensità superiori del gradiente [6]. Questa definizione può essere usata anche per la stima del movimento e il tracciamento di un oggetto (Figura 2.3) [9].



(a) (b)

Figura 2.2: Applicazioni del metodo *level set* [5]. In (a) e (b) si può vedere come iniziando da curve diverse si possa ottenere lo stesso risultato.

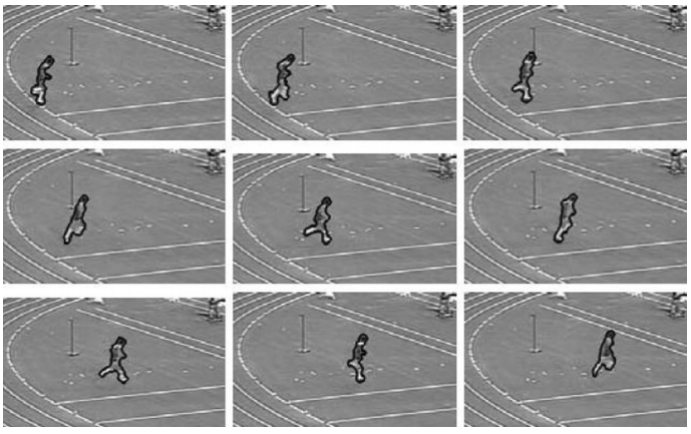


Figura 2.3: Applicazione del metodo *level set* per il tracciamento di oggetti.

2.3 *Region growing*

La tecnica del *region growing* permette di allargare una regione in base a determinati criteri di espansione. Nel *region growing* si parte da un insieme iniziale di punti, detti *seed* (selezionati in modo automatico o attraverso l'interazione con l'utente), si prosegue quindi con l'espansione della regione appendendo a ogni *seed* i pixel del vicinato con proprietà predefinite simili a quelle del *seed*. La scelta dei criteri di similarità per l'inclusione di un pixel nella regione, dipende sia dal problema che si sta analizzando che dal tipo di immagini che si hanno a disposizione. Per immagini a colori si possono utilizzare informazioni sul colore, sfruttando determinati spazi colorimetrici (come il *CIE L*a*b**) che permettono di ottenere una misura migliore della differenza tra pixel. Si possono utilizzare informazioni sulle texture, misurazioni statistiche o di energia, entropia ed omogeneità per confrontare regioni vicine. Per caratterizzare ulteriormente una regione si possono utilizzare dei meccanismi per controllare la crescita della regione. Ad esempio, in [11], per ogni passo di crescita si tiene traccia del numero di pixel che sono stati aggiunti alla regione, se il tasso di crescita è al di sotto di una determinata soglia allora la regione corrisponderà ad elementi dell'immagine *"flat"*, senza una particolare texture (Figura 2.4).
Un problema del *region growing* sta nella formulazione di una regola per fermare la crescita della regione, che dovrebbe avvenire quando non ci sono più pixel che soddisfano i criteri di inclusione. Vincoli come i valori dell'intensità, texture e colore sono di natura locale e non tengono conto del processo di crescita. Altri criteri che si possono utilizzare per potenziare l'algoritmo di *region growing* possono riguardare forma e dimensione della regione [12]. Un ulteriore limitazione consiste nel fatto che la qualità della segmentazione finale dipende molto dalla posizione iniziale dei *seed*. Per risolvere questo problema, in [11], i *seed* vengono scelti in modo automatico cercando le regioni del gradiente che non contengono bordi. Le regioni trovate formeranno i *seed* iniziali.
Il *region growing* si può utilizzare anche in applicazioni dinamiche. In [13], per tracciare un oggetto, il *region growing* è basato sulla differenza tra frame

successivi e cambiamenti locali dell'oggetto precedentemente individuato.

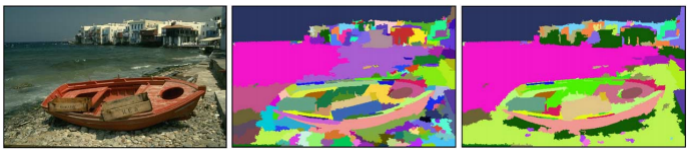


Figura 2.4: Applicazione del metodo *region growing* [11].

2.4 *Watershed*

La segmentazione tramite *watershed* si applica principalmente ad immagini in scala di grigio e consiste nel suddividere l'immagine in diversi "bacini di attrazione" ognuno contenente un minimo locale. Questo approccio si basa sulla visualizzazione dell'immagine (solitamente rappresentante l'intensità del gradiente) in tre dimensioni: le coordinate spaziali e l'intensità del pixel che corrisponde all'altezza (Figura 2.5) [12]. Con questa interpretazione si possono individuare tre tipi di punti:

- Punti appartenenti ad un minimo locale.
- Punti in cui una goccia d'acqua, posizionata in quel punto, cadrebbe sicuramente in un determinato minimo locale. Sono questi i punti che formano il bacino di attrazione.
- Punti in cui la goccia d'acqua avrebbe la stessa probabilità di cadere in più minimi locali. Questi punti formano le linee divisorie dei bacini di attrazione e corrispondono quindi al contorno (connesso) di un oggetto, il risultato della segmentazione.

Per la costruzione delle linee di separazione dei diversi bacini (o dighe) si usano immagini binarie, corrispondenti ai diversi livelli dell'immagine 3D, e la dilatazione morfologica.

Generalmente, questo metodo porta a problemi di "super-segmentazione" a causa del rumore e di altre irregolarità del gradiente [12] e per il fatto che

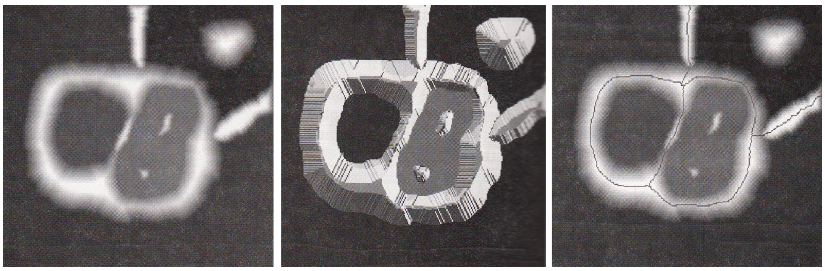


Figura 2.5: Rappresentazione dell'immagine bidimensionale in 3D e risultato ottenuto applicando la segmentazione con *watershed* [12].

ad ogni minimo locale viene assegnata solamente una regione [6]. Per risolvere questo problema si possono utilizzare del *marker*, componenti connesse che appartengono all'immagine [6]. Possono essere definiti interattivamente dall'utente (marcando con un click del mouse il centro approssimato dell'oggetto di interesse) oppure in una fase di preprocessamento è possibile definire un insieme di vincoli che i *marker* devono soddisfare (ed esempio avere una certa intensità del gradiente).

Figura 2.6: Applicazione del metodo *watershed*.

2.5 Tecniche di *clustering*

Con le tecniche di *clustering* il problema della segmentazione viene modellato come la ricerca di cluster (o mode) in una funzione di densità di probabilità sconosciuta, in cui ogni campione è un vettore di *feature* (ad esempio contenente posizione, colore, gradiente) corrispondente ad ogni pixel dell'immagine. Le principali tecniche di *clustering* utilizzate per la segmentazione

sono: *k-means*, *mixture of Gaussians* e *mean shift*.

Con il *k-means* ed il *mixture of Gaussians* la funzione di densità di probabilità viene modellata in modo parametrico. Nel *k-means*, dato un numero di cluster k che si vogliono ottenere, lo scopo è quello di minimizzare la varianza totale intra-cluster. Inizialmente vengono create k partizioni a cui vengono assegnati i punti in ingresso casualmente o secondo alcune informazioni euristiche. Iterativamente viene calcolato il centroide di ogni partizione e si riassegnano i punti in ingresso al cluster con il centroide più vicino ad esso, finché l'algoritmo non converge. Nelle applicazioni reali l'algoritmo spesso converge rapidamente, ma a volte può richiedere un grande numero di iterazioni, molto dipende anche dal numero di cluster scelto e dall'assegnamento iniziale dei punti. Inoltre non garantisce che il risultato finale sia una soluzione ottima, ma assicura la convergenza ad una soluzione.
Nel *mixture of Gaussians* la funzione di densità di probabilità viene rappresentata come una somma pesata di Gaussiane. Un campione può essere associato al cluster con il centro più vicino (assegnamento *hard* di appartenenza) oppure attraverso un assegnamento *soft* a più cluster presenti nelle vicinanze. Per associare i campioni ad un cluster viene utilizzata la distanza di Mahalanobis [6]. Questo modello è parametrizzato dai vettori delle medie, le matrici di covarianza ed i pesi relativi ad ogni Gaussiana presente. Solitamente per trovare l'insieme delle Gaussiane che modellano i dati si usa l'algoritmo *Expectation Maximization (EM)* che si alterna iterativamente in due fasi:

- Fase di *Expectation*: si stima la probabilità che un campione appartenga ad una determinata Gaussiana.

- Fase di *Maximization*: si aggiornano i valori dei parametri massimizzando la funzione di probabilità trovata nella fase precedente.

Alla fine di questo procedimento ad ogni Gaussiana trovata corrisponderà un oggetto segmentato.

A differenza dei due metodi precedentemente descritti il *mean shift* definisce la funzione di densità di probabilità in modo implicito, usando un modello non parametrico continuo. Una volta stabilita la funzione di densità, per ottenere la segmentazione, bisogna individuarne i picchi principali e identificare le aree dello spazio in input che "salgono" verso lo stesso picco: queste regioni si possono considerare appartenenti ad uno stesso oggetto. Per stimare la funzione di densità, si effettua uno *smoothing* dei campioni in ingresso convolvendoli con una funzione *kernel* [6]. Per trovare i picchi si utilizza una tecnica nota come *multiple restart gradient descent* in cui:

- Si parte da un minimo locale y scelto casualmente.
- Si procede calcolando il gradiente della funzione di densità in y e si esegue un passo in salita nella direzione del gradiente.
- Si prosegue finché non si raggiunge un massimo locale in cui l'algoritmo di *mean shift* converge.

Per trovare tutti i massimi locali si può applicare separatamente il *multiple restart gradient descent* in diversi punti dello spazio. Le regioni in cui la densità dei campioni in input è più bassa non caratterizzano lo spazio delle *feature*, quindi i passi dell'algoritmo di *mean shift* possono essere più grandi. Viceversa, nelle zone vicino ai massimi locali i passi dell'algoritmo diventano più piccoli e l'analisi dello spazio diventa più raffinata. La dimensione dei passi dell'algoritmo è cruciale per le performance complessive: se lo step è troppo grande l'algoritmo divergerà, mentre se è troppo piccolo la convergenza sarà molto lenta [14]. In Figura 2.7 si può vedere un esempio di applicazione del metodo.

2.6 Tecniche basate sui grafi

In letteratura sono presenti molte tecniche di segmentazione che fanno uso di grafi. I vantaggi di questa rappresentazione sono molteplici: i pixel e le relazioni di vicinato si possono descrivere in modo pressoché immediato

Figura 2.7: Segmentazione ottenuta attraverso il metodo di *mean shift*, i bordi sono stati disegnati sopra all'immagine originale [14].

attraverso i nodi e gli archi di un grafo. I grafi sono flessibili, computazionalmente efficienti, sono inoltre disponibili una grande varietà di metodi ed algoritmi di ottimizzazione e risoluzione derivanti dagli studi sulla teoria dei grafi provenienti da molteplici discipline diverse ed applicabili in *computer vision*. Nel seguito verranno illustrate le principali tecniche per ottenere la segmentazione di immagini.

Il *region splitting and merging* consiste nel suddividere o unire (anche in maniera gerarchica) ricorsivamente regioni di un'immagine in base a caratteristiche statistiche o alla soddisfazione di determinati vincoli.
Durante lo *splitting* un'immagine viene suddivisa in quadranti sempre più piccoli in modo che per ognuno di essi valga un certo predicato. Con il *merging* regioni adiacenti vengono unite in quanto soddisfano determinati criteri di similarità. I vincoli ed i criteri per stabilire se una regione debba essere suddivisa o unita ad un'altra sono molteplici e cambiano a seconda dell'applicazione. Ad esempio se l'intensità dei pixel in una regione supera una certa soglia (determinata magari attraverso l'analisi dell'istogramma) allora la regione verrà suddivisa, altrimenti se in due regioni l'intensità è simile allora verranno unite. Si possono osservare le caratteristiche statistiche di una regione altre misurazioni che permettono di verificare la similarità o le differenze tra le diverse regioni presenti [6, 12].

Nelle tecniche di segmentazione basate sui grafi l'immagine viene vista appunto come un grafo (orientato o non orientato) in cui ad ogni pixel (o gruppi di pixel) corrisponde un nodo, che viene poi collegato tramite degli archi ai nodi vicini. Il costo di ogni arco è determinato in base alle proprietà dei pixel connessi: differenza dell'intensità, colore, posizione o altri attributi locali. La segmentazione basata sui grafi consiste nel selezionare un sottoinsieme di archi in base al loro costo. Si otterrà in una partizione in regioni in modo che ognuna di esse corrisponda ad una componente connessa del grafo (si veda Figura 2.8) [15]. Queste tecniche sono molto usate in applicazioni pratiche essendo molto flessibili e computazionalmente efficienti.
In [15], per valutare se tra due componenti c'è un bordo, viene definito un predicato che si adatta ai cambiamenti locali dei dati e compara:

- La differenza interna di un componente: corrisponde all'arco con il maggior costo nel *minimum spanning tree* del componente.
- La differenza tra le due componenti: tra tutti i costi degli archi che collegano i due componenti si sceglie quello con costo minore. Se non ci sono archi tra i due componenti il costo viene settato a ∞.

Due regioni vengono unite se la loro differenza è minore di una certa soglia (determinata attraverso le differenze interne delle singole regioni). L'unione delle regioni avviene usando una variante dell'algoritmo di Kruskal. La complessità computazionale di questo algoritmo è *O(n log n)*, con n uguale al numero di pixel dell'immagine, il che lo rende applicabile anche per applicazioni video [15].

La tecnica di *intelligent scissors* ottimizza la definizione del contorno in real-time, man mano che l'utente traccia un profilo grezzo dell'oggetto di interesse. Ciò è possibile grazie al fatto che l'*intelligent scissors* permette all'utente di selezione interattivamente il contorno più adatto dall'insieme di *tutti* i contorni ottimi che si espandono da un *seed* [16]. La definizione del contorno viene formulata come un problema di ricerca su un grafo (sfruttando la programmazione dinamica) in cui si cerca di trovare il cammino ottimo tra

Figura 2.8: Esempio di segmentazione che è possibile ottenere utilizzando i grafi e il *region merging* [15].

un nodo iniziale ed un insieme di altri possibili nodi. Ogni pixel corrisponde ad un nodo e vengono creati degli archi (a cui viene assegnato un costo) che lo collegano a tutti gli otto pixel del vicinato. Il cammino ottimo viene definito come il cammino con il costo cumulativo minimo.

Per il calcolo della curva ottima, chiamata anche *live-wire*, si esegue una fase di preprocessamento, in cui per determinare i costi corrispondenti ad ogni arco (archi che fanno parte di bordi marcati avranno costi minori) viene usata una somma pesata delle *feature* che solitamente si usano per caratterizzare i bordi:

- *Zero-crossing* del Laplaciano: è una *feature* binaria usata per la localizzazione del bordo, corrisponde ai punti di intensità massima o minima del gradiente.

- Intensità del gradiente: usato per correlare l'intensità del bordo e il costo visto che lo *zero-crossing* non fa distinzione tra valori massimi e minimi del gradiente. L'intensità del gradiente viene scalata ed invertita in modo che un gradiente elevato risulti in un costo basso e viceversa.

- Orientamento del gradiente: aggiunge una componente di uniformità, associando costi alti a cambiamenti improvvisi del contorno.

Una volta determinati i costi, si procede con la fase di programmazione dinamica eseguendo una ricerca sul grafo per trovare tutti i cammini con costi

cumulativi minimi che si espandono dal *seed*; viene utilizzato un algoritmo simile a quello di Dijstra. I cammini ottimi vengono calcolati a partire dal *seed* fino a tutti gli altri punti perché la ricerca sul grafo produce uno *spanning tree* con costo minimo su tutta l'immagine [16]. Al termine dell'algoritmo si avrà una mappa contenente dei puntatori che indicano i cammini minimi trovati. A questo punto, man mano che l'utente muove il mouse, il contorno si adatterà ai cammini a costo minimo seguendo i percorsi ottimi definiti dai puntatori precedentemente trovati.
Per rendere il contorno più stabile ed evitare che salti verso bordi marcati presenti nelle vicinanze (si veda Figura 2.9), si possono utilizzare due strategie:

- *Path cooling*: vengono generati automaticamente dei nuovi *seed* alla fine di segmenti di contorno stabili. La stabilità viene misurata attraverso il tempo (segmenti che per un certo numero di millisecondi non hanno subito modifiche) ed alla *coalescenza* [1] (numero di volte che un segmento è stato ridisegnato in seguito a differenti posizioni del mouse).

- *Interactive dynamic learning*: favorisce i cammini che hanno caratteristiche simili a quelle dei segmenti già trovati e che sono considerati "buoni". Le caratteristiche vengono aggiornate interattivamente man mano che viene definito il contorno. L'algoritmo di training si basa solamente sulla porzione più recente del percorso, in modo da adattarsi ai cambiamenti che si possono verificare lungo il contorno.

Una limitazione di questo metodo consiste nel fatto che ogni pixel dell'immagine viene mappato in un nodo connesso ai suoi otto vicini, quindi il costo computazionale dell'algoritmo è fortemente dipendente dalla risoluzione dell'immagine [17].
Per ovviare a questo problema è stato proposto un metodo in cui viene usato uno *slimmed graph* (un grafo "ridotto") durante la fase di programmazione

[1] Un singolo cammino minimo è definito da ogni pixel ad un determinato *seed*. Diversi percorsi si uniscono e condividono porzioni del loro cammino ottimo con i cammini di altri pixel.

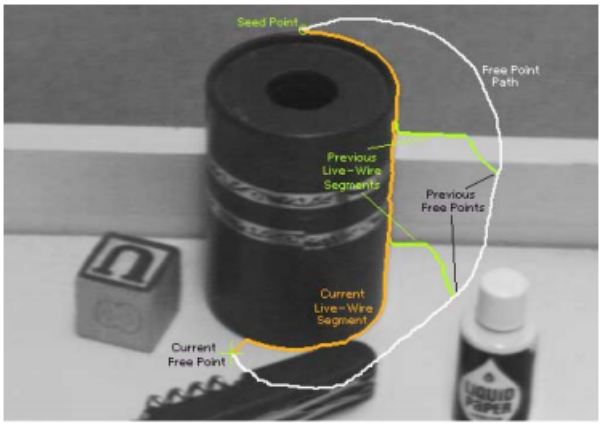


Figura 2.9: Contorno individuabile con il metodo di *intelligent scissors*. La linea bianca corrisponde alle posizioni del mouse, quella arancione al contorno individuato dall'algoritmo (corrispondente alla posizione attuale del mouse), mentre le linee verdi rappresentano i contorni individuati da posizioni precedenti del mouse [16].

dinamica, permettendo di avere una diminuzione del tempo di ricerca proporzionale al numero totale di nodi presenti. Il grafo ridotto viene ottenuto potando i nodi corrispondenti a regioni che non contengono bordi, perché è stato osservato che questi non venivano coinvolti nella ricerca del cammino ottimo [17]. Lo *slimmed graph* viene generato solamente una volta nella fase di preprocessamento dell'immagine. In questo grafo i nodi non rappresentano più un singolo pixel, ma regioni più estese (ad esempio un nodo può corrispondere ad un'area in cui il gradiente ha un valore basso, mentre diversi nodi vengono associati a regioni con valori alti del gradiente).

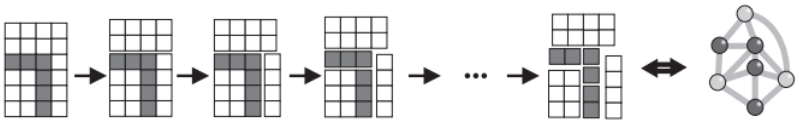


Figura 2.10: Costruzione dello *slimmed graph* [17].

La suddivisione dell'immagine viene effettuata con la *Binary Space Partition (BSP)* con cui si possono ottenere regioni rettangolari ed il numero di figli per ogni blocco può essere variabile (si veda figura 2.10). La *BSP* viene applicata ad un'immagine contenente l'intensità del gradiente normalizzato.

L'immagine viene suddivisa in due blocchi se la somma totale dell'intensità del gradiente normalizzato supera una determinata soglia (decisa dall'utente). Ricorsivamente si suddividerà ogni blocco (non necessariamente i blocchi figli devono avere le stesse dimensioni) finché la somma del gradiente normalizzato non sarà al di sotto della soglia.
Con l'uso dello *slimmed graph* cambia anche la definizione della funzione costo in quanto i nodi non rappresentano più un singolo pixel ma regioni dell'immagine. Il costo dell'arco tra due nodi vicini è una funzione che tiene conto della distanza dei centri di massa dei due blocchi e la media dell'intensità del gradiente [17]. Una volta ottenuto lo *slimmed graph* ed aver calcolato i costi degli archi, per il calcolo del cammino ottimo si procede come nell'algoritmo precedentemente descritto. In Figura 2.11 si possono vedere delle segmentazioni ottenute utilizzando lo *slimmed graph*.

(a) (b)

Figura 2.11: Altri esempi di segmentazioni che è possibile ottenere con l'*intelligent scissors* [17]. (a) Immagine con contorni difficilmente distinguibili. (b) Immagine contenente rumore.

In [18] viene proposto un metodo in cui la segmentazione risulta dall'integrazione di tecniche di *merging* e di tecniche basate sui grafi. Partendo da un'immagine si esegue una sequenza di aggregazioni *bottom-up*, in cui i pixel vengono gradualmente uniti per ottenere regioni sempre più grandi.
Ad ogni passo vengono considerate coppie di regioni adiacenti e si calcolano delle misure di probabilità per decidere se debbano essere unite oppure no.

Per il calcolo delle probabilità si tiene conto della distribuzione dell'intensità dei pixel e delle texture nell'intorno di ogni regione che si sta considerando. Le probabilità così ottenute andranno a costituire i pesi degli archi del grafo non orientato costruito sull'immagine. All'inizio dell'algoritmo ad ogni pixel viene fatto corrispondere un nodo. Per effettuare il *merging* dei nodi si una un metodo chiamato *Segmentation by Weighted Aggregation (SWA)*, grazie al quale, passo dopo passo, si ottiene una gerarchia di grafi più piccoli (*coarse*) collegati ai livelli successivi. I costi degli archi dei nuovi grafi si determinano per eredità dai livelli precedenti e si modificano in base alle proprietà della regione sottostante (calcolate ricorsivamente durante la procedura di *merging*) [18].
Dopo che è stata individuata una segmentazione nel livello più *coarse*, l'esatta appartenenza di ogni pixel viene calcolata propagando, attraverso la gerarchia di grafi, l'assegnamento ottenuto fino al primo livello, in cui ad ogni nodo corrisponde un pixel. La complessità computazionale di questo algoritmo è $O(n)$, con n uguale al numero di pixel dell'immagine e non richiede il settaggio dei parametri da parte dell'utente. Un esempio di questo metodo di può vedere in Figura 2.12.

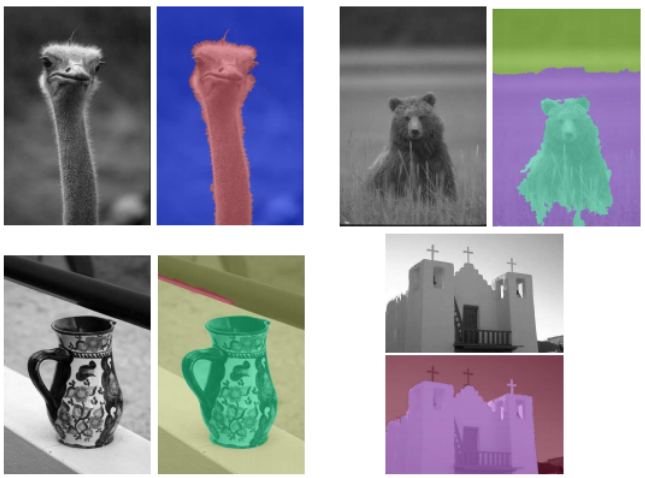


Figura 2.12: Applicazione del metodo di aggregazione probabilistica [18].

Con la tecnica di *normalized cut* si esaminano le similarità tra pixel vicini

e si cerca di separare i gruppi che sono connessi da affinità deboli. Un'immagine viene rappresentata attraverso un grafo pesato e non orientato: ogni nodo del grafo corrisponde al vettore di *feature* di un pixel, esiste un arco per ogni coppia di nodi ed il peso di ogni arco è una funzione della similarità dei nodi connessi [19]. La segmentazione si ottiene cercando di partizionare il grafo in un insieme di gruppi disgiunti, in cui all'interno di uno stesso gruppo la misura di similarità è alta, mentre tra gruppi diversi la similarità è bassa (un esempio in Figura 2.13). Per partizionare il grafo in due gruppi bisogna effettuare un taglio, che corrisponde alla somma dei pesi di tutti gli archi che vengono rimossi per dividerli. La bipartizione ottimale sarebbe quella che minimizza il costo del taglio, ma si è visto che questo metodo tende ad isolare singoli nodi del grafo [6]. Per questo motivo in [19] viene definito il taglio normalizzato, tra i gruppi A e B, definito nel modo seguente:

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)}$$

Dove *cut(A,B)* è il taglio classico e *assoc(A,V)* è la somma di tutte le connessione che partono dai nodi in A e arrivano a tutti gli altri nodi del grafo, lo stesso vale per *assoc(B,V)*. La minimizzazione del taglio normalizzato è un problema NP-completo, anche nel caso semplice di un grafo a griglia, ma in [19] viene dimostrato che applicando il taglio normalizzato a valori reali si riesce a trovare efficientemente una soluzione approssimata e discreta. Il procedimento è il seguente: dopo aver associato il grafo all'immagine si costruisce la matrice $\mathbf{W} = [w_{ij}]$ contenente i pesi relativi a tutte le coppie di nodi, si calcola poi la somma $\mathbf{d}(i) = \sum_j w(i,j)$ di ogni riga di $\boldsymbol{W}$ e questi valori vanno a costruire la matrice diagonale $\boldsymbol{D}$. A questo punto il problema della minimizzazione del taglio normalizzato equivale a trovare l'autovettore con i più piccoli autovalori nel sistema $(\mathbf{D}-\mathbf{W})\mathbf{x} = \lambda\mathbf{D}\mathbf{x}$. Trovato l'autovettore, le variabili corrispondenti ai valori positivi e negativi vengono associate rispettivamente alle due partizioni del taglio. Questo processo può essere ulteriormente ripetuto per suddividere gerarchicamente l'immagine. Il calcolo del taglio normalizzato richiede la risoluzione di un sistema sparso di autovalori, che potrebbe risultare lento. Sono state però proposte delle tecniche,

che sfruttano approcci basati sull'*algebraic multigrid* per accelerarlo [6].

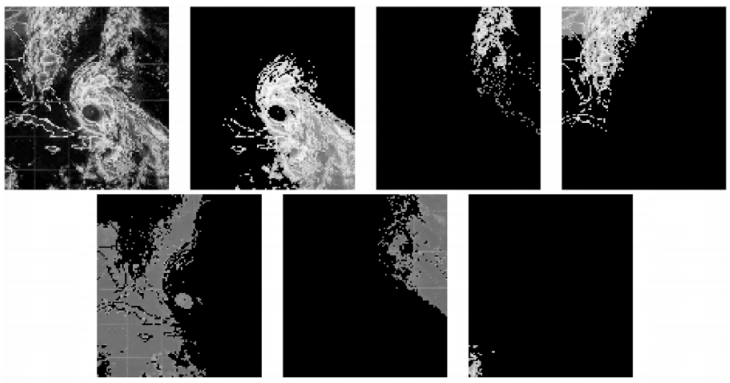


Figura 2.13: Estrazione delle componenti di un'immagine satellitare con il metodo *normalized cut* [19].

L'*interactive graph cut* sfrutta la conoscenza dell'utente sul dominio applicativo per ottenere un risultato ottimale della segmentazione finale. Si applica ad immagini in scala di grigi e si ottiene una segmentazione binaria. L'utente deve indicare alcune zone (*seed*) dell'immagine che sicuramente fanno parte dell'oggetto di interesse ed altre zone che sono sicuramente parte del background. Queste informazioni sono utilizzate come vincoli durante il processo di segmentazione e i pixel corrispondenti ai *seed* vengono utilizzati per costruire gli istogrammi che caratterizzano le distribuzioni di intensità del *background* e del *foreground*. La segmentazione si ottiene minimizzando una funzione di energia composta da due termini che rappresentano:

- Le proprietà della regione: per ogni pixel si indica qual è la "penalità" nell'associarlo a background oppure al *foreground*, comparando ad esempio l'intensità del pixel agli istogrammi calcolati dai *seed*.

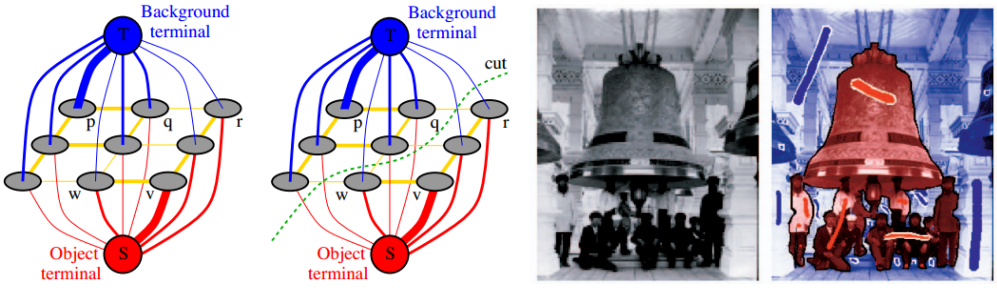
- Le proprietà del contorno: può essere interpretato come una "penalità" per la discontinuità di due pixel vicini. Solitamente assume un valore alto per nodi simili, si avvicina invece a zero se i due nodi sono molto differenti, in questo modo il taglio lungo i bordi di un oggetto dovrebbe costare poco [20].

Il metodo presentato in [20] si basa su un algoritmo di *graph cut* per trovare la migliore segmentazione globale. Data un'immagine viene creato un grafo non orientato nel modo seguente (per una rappresentazione schematica si veda Figura 2.14a):

- Per ogni pixel viene creato un nodo.
- Ogni nodo è collegato al proprio vicinato tramite archi con costi non negativi corrispondenti al termine che caratterizza il contorno nella funzione di energia. Per la determinazione del costo si possono utilizzare diversi criteri: intensità e direzione del gradiente, *zero-crossing* del Laplaciano, distanza dai *seed*.
- Sono presenti due nodi aggiuntivi, l'*object terminal (S)* ed il *background terminal (T)*. I nodi terminali solitamente corrispondono alle possibili label che possono essere assegnate ai pixel [21].
- Ogni nodo ha quindi due archi aggiuntivi, ognuno dei quali lo collega ad un terminale. Il costo di questi archi aggiuntivi, corrispondente al termine che descrive la regione nella funzione di energia, cambia in base alle caratteristiche dell'area di appartenenza: se il pixel appartiene ai *seed* si assegnerà un valore costante, altrimenti il costo varia a seconda che il pixel appartenga ad un bordo o un'altra regione dell'immagine.

Una volta costruito il grafo, lo scopo è trovare il taglio globale ottimo che separi i due terminali. Il taglio minimo può essere calcolato trovando il *flusso massimo* che parte dal nodo S ed arriva al nodo T. Il *flusso massimo* può essere visto come la "quantità massima di acqua" che scorre dal nodo S al nodo T, interpretando gli archi del grafo come tubi con capacità uguale al loro costo. Si può dimostrare che il *flusso massimo* da S a T satura un insieme di archi del grafo, dividendo i nodi in due partizioni disgiunte che corrispondono al taglio minimo [21]. Il taglio minimo ha complessità computazionale polinomiale. Con l'*interactive graph cut*, inoltre, si può efficientemente raffinare la segmentazione incorporando eventuali nuovi *seed* definiti interattivamente dall'utente. In Figura 2.14b si può vedere il risultato della segmentazione ed

i *seed* indicati dall'utente. Si può inoltre utilizzare per la segmentazione di video, che vengono trattati come un unico volume: i *seed* necessari vengono inseriti attraverso un'interfaccia che permette di selezionare i frame. Le informazioni riguardati i *seed*, la regione ed il contorno vengono propagate automaticamente attraverso i frame, in quanto la computazione del taglio ottimo globale avviene su un grafo 3D [20].



(a) (b)

Figura 2.14: Segmentazione attraverso il metodo di *graph cut* (a) Esempio schematico di un grafo: il costo degli archi è rappresentato dallo spessore. (b) Applicazione del *graph cut* ad un'immagine [20].

Il metodo precedentemente descritto può essere esteso in diversi modi. Con il *GrabCut* si potenzia il *graph cut* applicandolo iterativamente all'immagine, stimando ed "imparando" di volta in volta nuove caratteristiche della regione per raffinare la segmentazione. Questo metodo si applica ad immagini a colori in cui il *background* e il *foreground* sono rappresentati ognuno da un *Gaussian Mixture Model (GMM)* che è un insieme di K Gaussiane con *full-covariance*. Nel *framework* di ottimizzazione, per ragioni computazionali viene introdotto un vettore $\mathbf{k} = \{k_1, ..., k_n, ...k_N\}$ con $k_n \in \{1...K\}$ che assegna ad ogni pixel n una componente del *GMM*. Incorporando il *GMM* nella funzione di energia i parametri del modello diventano: i pesi π, le medie μ e le covarianze Σ alle diverse Gaussiane presenti. L'algoritmo iterativo di *GrabCut* è il seguente:

0. Inizializzazione: l'utente attraverso un *bounding box* definisce il *background* dell'oggetto che vuole segmentare.

1. Assegnazione delle componenti del *GMM* ai pixel in base alle caratteristiche del contorno e della regione di appartenenza.

2. Stima dei parametri del *GMM*: ad esempio si definisce l'insieme di pixel *F(k)* che appartengono al *foreground*, si stimano pesi, media e covarianza in base al valore dei pixel in *F(k)*.

3. Ottimizzazione globale utilizzando l'algoritmo del taglio minimo descritto per l'*interactive graph cut*.

4. Ripetizione dal passo 1 fino alla convergenza.

La struttura dell'algoritmo garantisce la convergenza, in quanto si può dimostrare che ad ogni ciclo si minimizza iterativamente la funzione di energia rispetto a tre insiemi di variabili, $\mathbf{k}$, $\theta = \{\pi, \mu, \Sigma\}$, $\alpha = (\alpha_1, ..., \alpha_N)$ con $\alpha_n \in \{0, 1\}$ che rappresenta l'"opacità" del pixel. La funzione decresce quindi monotonicamente, garantendo almeno il raggiungimento di un minimo locale. Con questo algoritmo iterativo l'interazione con l'utente è più facile visto che deve disegnare solamente un *bounding box* e si riduce il grado di interazione con per il raffinamento della regione (si veda Figura 2.15).

Figura 2.15: Esempi di applicazione del metodo *grab cut* [22].

Il metodo di *random walk* consente di eseguire una segmentazione interattiva dell'immagine una volta che sono stati specificati dei *seed* per gli oggetti che si vogliono estrarre. L'algoritmo di *random walker* richiede la risoluzione di un sistema sparso di equazioni lineari che può essere risolto velocemente grazie a diversi metodi [23]. Ogni *seed* specificato dall'utente indica le posizioni dei pixel che hanno la stessa *label*. L'algoritmo richiederebbe che ad ogni pixel non selezionato venga associata una *tupla* (con K dimensioni, K=numero dei *seed*) che specifica la probabilità che un *random walker*, partendo dalla posizione del pixel, raggiunga ognuno dei K *seed*. Le

computazioni richieste da questo algoritmo sarebbero troppo pesanti ed inefficienti. Fortunatamente il problema del *random walker* è stato dimostrato essere uguale alla risoluzione del problema di Dirichlet in cui nelle condizioni al contorno le posizioni dei *seed* e il valore del *seed* stesso vengono imposti uguali a zero. La soluzione del problema di Dirichlet su un grafo arbitrario è data esattamente dalla distribuzione di potenziale sui nodi di un circuito elettrico, in cui le resistenze rappresentano l'inverso dei costi degli archi. In questo modo è possibili calcolare analiticamente le probabilità richieste.
L'immagine viene rappresentata da un grafo non orientato. Ad ogni arco viene associato un costo (in [23] viene usata una funzione di peso Gaussiana) che corrisponde alla probabilità che il *random walker* lo attraversi. I passi dell'algoritmo sono i seguenti:

1. Calcolare i pesi di tutti gli archi del grafo in base alla funzione di peso Gaussiana.

2. Ottenere l'insieme dei pixel corrispondenti ai K *seed*, interattivamente o automaticamente.

3. Risolvere il sistema sparso di equazioni lineari (si veda [23]) e definire quindi la *tupla* delle probabilità per ogni nodo. La risoluzione di questi sistemi si può ottenere utilizzando dei risolutori iterativi, attraverso metodi di *algebraic multigrid* e si possono inoltre efficientemente parallelizzare (per essere eseguiti in GPU ad esempio).

4. Assegnare ad ogni nodo il valore massimo della *tupla* in modo da ottenere la segmentazione finale.

In Figura 2.16 si può vedere un esempio di applicazione di questo metodo.

Figura 2.16: Esempi di applicazione del metodo *random walk* [23] in immagini naturali.

2.7 Confronto dei metodi descritti

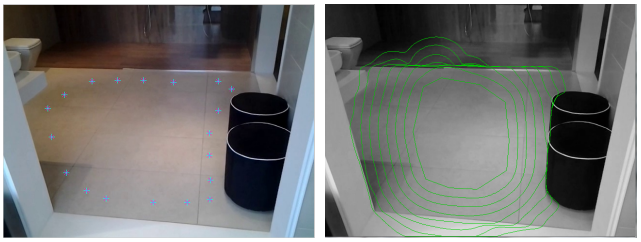
Alcuni metodi di segmentazione mostrati in precedenza sono stati applicate ad immagini, che saranno poi utilizzate nel contesto dell'applicazione, per verificarne l'efficienza e la precisione. Sono state utilizzate le seguenti immagini (Figura 2.17) che presentano diverse caratteristiche tipiche di ambienti interni: ombre *soft* e *hard*, elementi di arredo con bordi non rettilinei, linee di fuga, differenti condizioni luminose.

(a) (b) (c)

Figura 2.17: Immagini utilizzate per il confronto dei vari metodi presenti in letteratura. (a) Pavimenti uniformi. (b) Pavimenti con texture non uniforme. (c) Pavimenti con linee di fuga evidenti.

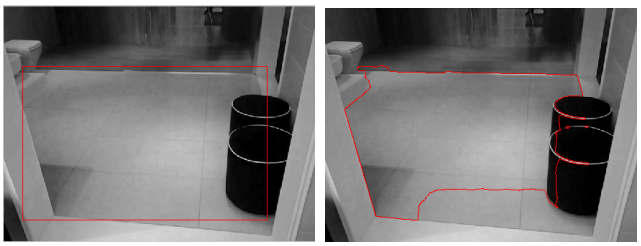
All'immagine con pavimento uniforme sono state applicate le seguenti tecniche. In Figura 2.18 è stato applicato il metodo *snake*: si può notare (Figura 2.18b) che dopo diverse iterazioni il metodo non converge verso la segmentazione ottimale, ma le curve si espandono in aree che non fanno parte della superficie di interesse. In Figura 2.19 è stato applicato il metodo *level set*: anche in questo caso si può notare (Figura 2.19b) che la segmentazione risultante diverge notevolmente dal bordo effettivo dell'area di interesse. In Figura 2.20 è stato applicato il metodo *watershed* che consente di ottenere una segmentazione pressoché ottimale della superficie di interesse (Figura 2.20b). In Figura 2.21 è stato applicato il *k-means* con una particolare implementazione dell'algoritmo proposta dal professore Alessandro Bevilacqua (descritta nel capitolo 5). La segmentazione risultante dipende molto dal numero di *cluster* scelti inizialmente. In Figura 2.22 sviluppato per l'esame di Elaborazione delle Immagini LM: la segmentazione ottenuta risulta abbastanza accurata tranne nelle zone in cui sono presenti ombre. In Fi-

gura 2.23 è stato applicato il metodo di *mean shift*: anche in questo caso la segmentazione risulta molto accurata a pare la piccola zona d'ombra (in basso a destra) che non viene riconosciuta come appartenente alla superficie di interesse. In Figura 2.24 è stato applicato il metodo *grab cut*: dopo tre iterazioni dell'algoritmo la segmentazione è abbastanza accurata ma non viene riconosciuta correttamente l'area in basso a destra della superficie di interesse (Figura 2.24b). In Figura 2.25 è stato applicato il metodo *intelligent scissor*: la segmentazione finale è perfetta ma può essere ottenuta solo grazie ad una elevata interazione con l'utente, che deve indicare la maggior parte dei punti che fanno parte del bordo della superficie di interesse. In Figura 2.26 è stato applicato il metodo *random walker*: la segmentazione risultate diverge notevolmente dal bordo effettivo dell'area di interesse.



(a) (b)

Figura 2.18: Applicazione del metodo *snake* ad una scena con pavimento uniforme. (a) Punti che definiscono lo snake iniziale. (b) Snake ottenuti con diverse iterazioni.

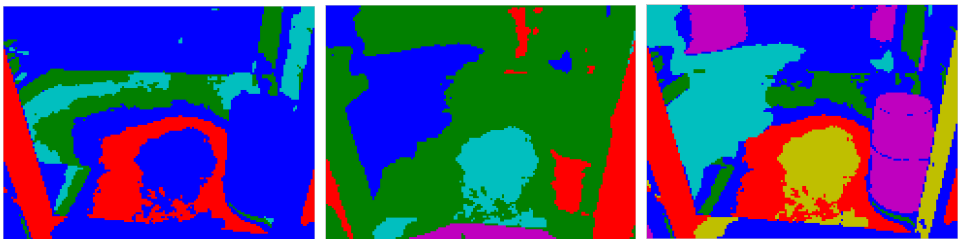


(a) (b)

Figura 2.19: Applicazione del metodo *level set* ad una scena con pavimento uniforme. (a) Rettangolo iniziale. (b) Segmentazione risultante.

(a) (b)

Figura 2.20: Applicazione del metodo *watershed* ad una scena con pavimento uniforme. (a) Seed iniziali. (b) Segmentazione risultante.



(a) (b) (c)

Figura 2.21: Applicazione del metodo *kmeans* ad una scena con pavimento uniforme. Segmentazione ottenuta imponendo (a) 4 cluster (b) 5 cluster (c) 6 cluster.

Figura 2.22: Applicazione del metodo di *region growing* sviluppato per l'esame di Elaborazione delle Immagini LM.

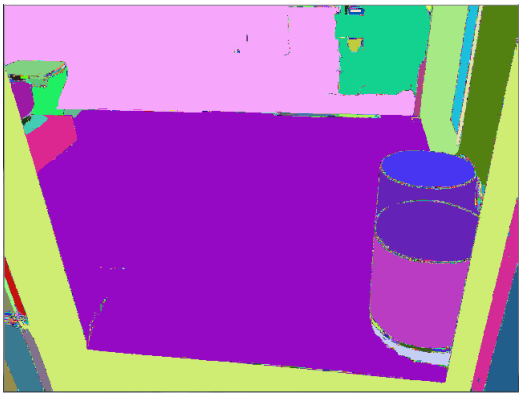


Figura 2.23: Applicazione del metodo *mean shift*.

(a) (b)

Figura 2.24: Applicazione del metodo *grabcut* ad una scena con pavimento uniforme. (a) Definizione del rettangolo e dei *seed* iniziali per la definizione delle zone di *foreground* e *background*. (b) Segmentazione risultante dopo tre iterazioni.

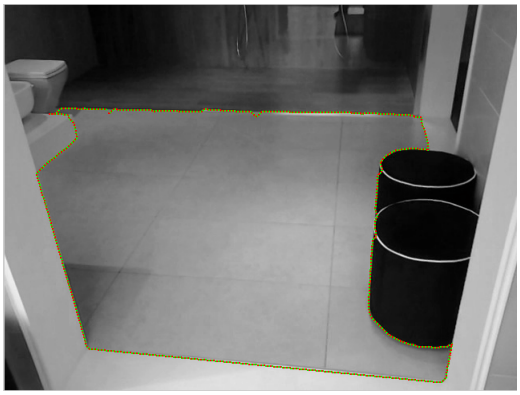


Figura 2.25: Applicazione del metodo *intelligent scissor*.

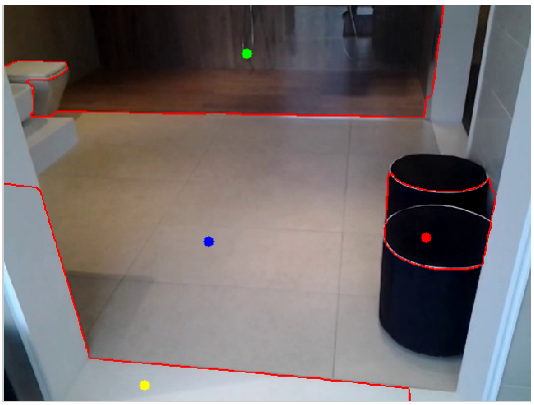
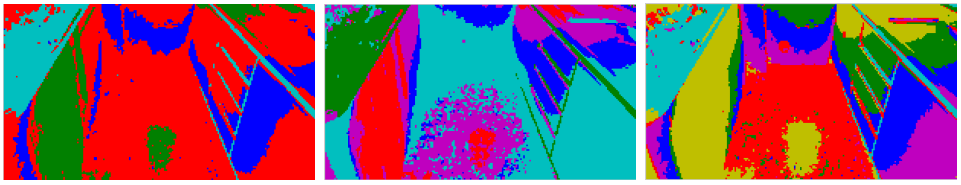


Figura 2.26: Applicazione del metodo *random walker*, i punti colorati rappresentano i *seed* iniziali.

In presenza di pavimenti con texture non uniforme sono state applicate le seguenti tecniche. In Figura 2.27 è stato applicato il metodo *watershed*: data la grande variabilità della texture del pavimento la segmentazione non risulta accurata come nel caso del pavimento uniforme. In Figura 2.28 è stato applicato il metodo *k-means*: la segmentazione risultante dipende molto dal numero di *cluster* scelti inizialmente. In Figura 2.29 è stato applicato il metodo *mean shift*: applicando questo algoritmo all'intera immagine non è possibile ottenere una segmentazione accurata dell'area di interesse. In Figura 2.30 è stato applicato il metodo *grab cut*: la parte in alto a sinistra della superficie di interesse non viene riconosciuta correttamente, inoltre è presente una porzione di muro erroneamente inclusa nella segmentazione finale.

(a) (b)

Figura 2.27: Applicazione del metodo *watershed* ad una scena con pavimenti non uniformi. (a) Seed iniziali. (b) Segmentazione risultante.



(a) (b) (c)

Figura 2.28: Applicazione del metodo *kmeans* ad una scena con pavimento non uniforme. Segmentazione ottenuta imponendo (a) 4 cluster (b) 5 cluster (c) 6 cluster.

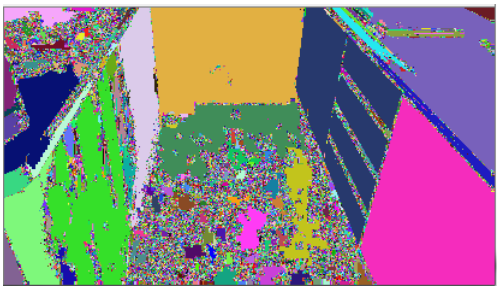
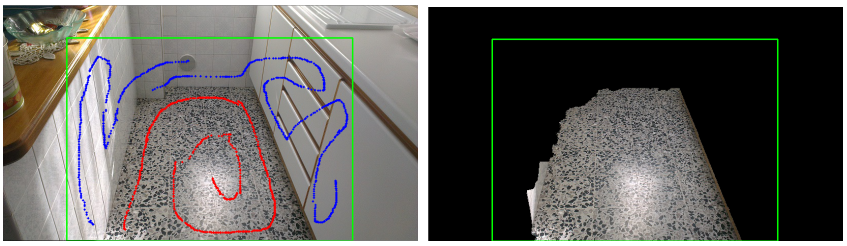


Figura 2.29: Applicazione del metodo *mean shift*.



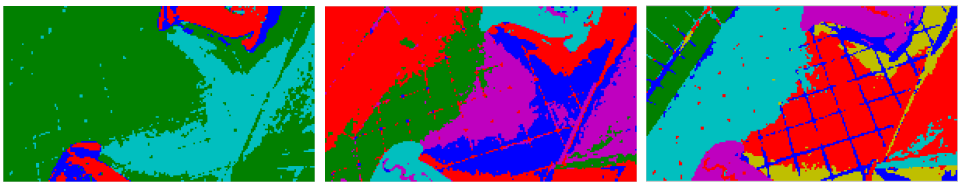
(a) (b)

Figura 2.30: Applicazione del metodo *grabcut* ad una scena con pavimento non uniforme. (a) Definizione del rettangolo iniziale. (b) Segmentazione risultante dopo quattro iterazioni.

In presenza di pavimenti con linee di fuga evidenti sono state applicate le seguenti tecniche. In Figura 2.31 è stato applicato il metodo *watershed*: come si può vedere in Figura 2.31b non tutte le piastrelle vengono riconosciute come appartenenti alla stessa regione. In Figura 2.32 è stato applicato il *k-means* anche in questo ultimo caso la segmentazione risultante dipende molto dal numero di *cluster* scelti inizialmente. In Figura 2.31 è stato applicato il metodo *grab cut*: la segmentazione risulta abbastanza precisa anche se alcune piastrelle non vengono incluse nella superficie finale.

(a) (b)

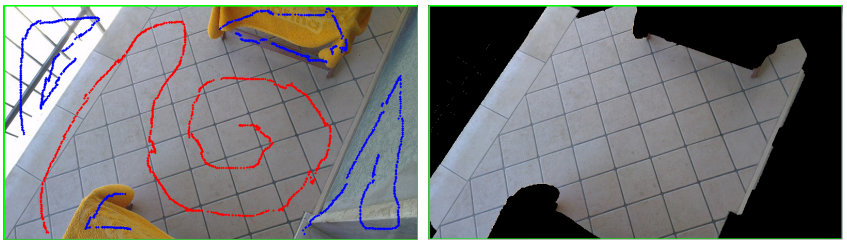
Figura 2.31: Applicazione del metodo *watershed* ad una scena con pavimenti in cui si distinguono chiaramente le linee di fuga. (a) Seed iniziali. (b) Segmentazione risultante.



(a) (b) (c)

Figura 2.32: Applicazione del metodo *kmeans* ad una scena con pavimento in cui si distinguono chiaramente le linee di fuga. Segmentazione ottenuta imponendo (a) 4 cluster (b) 5 cluster (c) 6 cluster.

Come si può vedere dalle immagini precedenti, a parte pochi casi, non è possibile ottenere una segmentazione accurata applicando una delle tecniche descritte all'intera immagine. Per questo nel seguito illustreremo un metodo per migliorare la segmentazione di un pavimento applicando localmente una combinazione di alcune tecniche esposte prima.



(a) (b)

Figura 2.33: Applicazione del metodo *grabcut* ad una scena con pavimento in cui di distinguono chiaramente le linee di fuga. (a) Definizione del rettangolo iniziale. (b) Segmentazione risultante dopo 4 iterazioni.

Capitolo 3

Analisi e Progetto

Nel contesto della tesi è stata sviluppata un'applicazione di Realtà Aumentata per dispositivi *mobile* nell'ambito dell'*interior design*. L'obiettivo dell'applicazione è quello di dare all'utente la possibilità di cambiare virtualmente la texture dei rivestimenti (pavimenti o pareti) presenti nell'ambiente che sta inquadrando. Il flusso operativo generale è il seguente: dopo aver preso un'istantanea con la fotocamera del dispositivo di un ambiente interno, è possibile selezionare, attraverso un tocco sul display, la superficie di cui si vuole simulare un cambiamento d'aspetto. Parte quindi la fase di segmentazione automatica per individuare la superficie appartenente al rivestimento selezionato. Trovata l'area di interesse viene sovrapposto all'immagine originale un *layer* che permette di cambiare interattivamente e visualizzare con la giusta prospettiva la texure desiderata.
Vista la grande varietà di ambienti e rivestimenti che si possono trovare non è possibile estrarre la regione di interesse in modo automatico e preciso con un solo algoritmo di segmentazione. Per questo la tesi si focalizza sulla ricerca di un metodo per la segmentazione ed il raffinamento dell'area di interesse in base alle caratteristiche di quest'ultima. L'estrazione della regione di interesse avviene in due fasi: ad una segmentazione "grezza" preliminare per identificare la maggior parte della superficie segue una fase per perfezionare il bordo dell'area trovata. La rifinitura avviene solamente nelle zone che più probabilmente produrranno un elevato fastidio visivo (avendo uno scosta-

mento sostanziale dalla segmentazione ottimale) e con tecniche dipendenti dalla struttura dell'area individuata.

Nel seguito vengono descritte le scelte progettuali che hanno portato alla realizzazione dell'applicazione.

3.1 Linee guida allo sviluppo di applicazioni *mobile*

Lo sviluppo di applicazioni per dispositivi mobili, grazie alla diffusione sempre maggiore di smartphone e tablet, ha avuto una crescita esponenziale negli ultimi anni. Per ognuno dei maggiori sistemi operativi mobili (Android, Apple iOS, Windows Phone, BlackBarry e altre) sono presenti degli ambienti di sviluppo che permettono la creazione di applicazioni in modo facile e veloce, ad esempio il *plug-in Android Development Tool* per Eclipse, *Xcode* per applicazioni Apple, *Microsoft Visual Studio*. Questi strumenti permettono la realizzazione di applicazioni semplici in pochi passi e nello stesso tempo spingono lo sviluppatore ad utilizzare principi di astrazione e modularità integrati nell'architettura della piattaforma utilizzata. Per applicazioni più complesse è necessario rispettare criteri di ingegneria del software per assicurare uno sviluppo sicuro e di alta qualità. Le problematiche che sorgono nella progettazione di applicazione mobile sono molteplici [24]:

- Integrazione con l'hardware del dispositivo e gestione dei sensori presenti: accelerometri, giroscopi, GPS, fotocamere sono costruiti da diversi produttori il che rende la qualità e il livello di accuratezza non omogeneo [24]. Inoltre per dispositivi con schermi piccoli si hanno delle limitazioni sul numero di informazioni che possono essere presentate di volta in volta [25].

- Sicurezza e possibili interazioni con altre applicazioni: i diversi sistemi operativi essendo "aperti" permettono l'installazione di potenziali *ma-*

laware che possono interferire con le normali operazioni del dispositivo [24].

- Supporto per le diverse piattaforme e versioni di un sistema operativo. Una decisione importante che deve fare lo sviluppatore riguarda la scelta delle piattaforme per cui sviluppare l'applicazione. È piuttosto costoso supportare piattaforme multiple specialmente quando ci sono più versioni della stessa [24]. Un'applicazione, se sviluppata su più sistemi operativi, deve essere consistente con le altre versioni [25].

- Progettazione dell'interfaccia grafica: il comportamento degli elementi grafici presenti deve essere coerente con quello delle altre applicazioni e deve aderire alle linee guida definite esternamente nel *Software Development Kit (SDK)* della piattaforma [24].

- Il fattore tempo è critico per gli utenti di dispositivi mobili e deve essere tenuto in considerazione fin dall'inizio della progettazione. Gli utenti non desiderano aspettare alcuni minuti per l'avvio di un'applicazione ed in seguito potrebbero avere la necessita di cambiare velocemente contesto per accedere ad altre funzionalità del dispositivo [25].

- Complessità del testing vista la grande varietà di dispositivi in circolazione [24].

- Rispetto di qualità non funzionali: uso efficiente delle risorse del dispositivo, controllo del consumo di energia, reattività, scalabilità, affidabilità (robustezza, connettività, stabilità) e usabilità [24].

Bisogna quindi trovare delle tecniche appropriate per la gestione di progetti sempre più complessi e tenere conto che il mondo dei *device* mobili cambia rapidamente e ci vogliono dei metodi per il supporto e la manutenzione [24].

3.2 Piattaforma hardware e software

L'applicazione è stata sviluppata per dispositivi *mobile*. I requisiti hardware minimi sono:

- Fotocamera posteriore: per la cattura delle immagini dell'ambiente.
- Sensori inerziali: per determinare la giusta prospettiva della nuova texture del rivestimento.

Visto che si tratta di un'applicazione di elaborazione delle immagini si è pensato di utilizzare la libreria grafica *open source* OpenCV (*Open source Computer Vision*) che include diverse centinaia di algoritmi di *computer vision* e *machine learning*, è rilasciata sotto licenza *BSD*. Supporta i linguaggi C/C++, Phyton, Java e MATLAB e i sistemi operativi Windows, Linux, Mac OS, iOS and Android. La libreria OpenCV è stata progettata per ottenere un'elevata efficienza computazionale con particolare riguardo alle applicazioni real-time. È scritta in linguaggio C/C++ ottimizzato e può sfruttare l'elaborazione multi-core; usando OpenCV in combinazione con OpenCL o CUDA si possono ottenere i vantaggi dell'accelerazione hardware delle piattaforme eterogenee sottostanti [26]. In particolare, è disponibile una versione di OpenCV che NVIDIA ha ottimizzato specificatamente per i processori Tegra 3 e Tegra 4 (montati principalmente su dispositivi Android) che permette incrementare notevolmente la velocità con cui vengono eseguite le funzioni di elaborazione delle immagini.

Per decidere su quali dispositivi e su che sistema operativo effettuare un primo sviluppo dell'applicazione è stata effettuata la seguente analisi di mercato. Verrà poi descritto il processo di calibrazione per rimuovere eventuali distorsioni dovute alla struttura della fotocamera e in che modo avviene la gestione dei sensori inerziali nei due sistemi operativi *mobile* principali: Android e iOS.

3.2.1 Analisi di mercato

Negli ultimi anni il mercato dei tablet è stato in costante crescita, superando addirittura le vendite di pc [27]. Secondo gli ultimi dati forniti dall'*International Data Corporation* (*IDC*, una società che si occupa di ricerche di mercato) il mercato globale dei tablet è così suddiviso: In Figura

Vendor	4Q13 Unit Shipments	4Q13 Market Share	4Q12 Unit Shipments	4Q12 Market Share	Year-over-Year Growth
Apple	26.0	33.8%	22.9	38.2%	13.5%
Samsung	14.5	18.8%	7.8	13.0%	85.9%
Amazon.com Inc.	5.8	7.6%	5.9	9.9%	-1.7%
Asus	3.9	5.1%	3.1	5.1%	25.8%
Lenovo	3.4	4.4%	0.8	1.3%	325.0%
Others	23.3	30.3%	19.5	32.5%	19.5%
Total	**76.9**	**100.0%**	**60**	**100.0%**	**28.2%**

Figura 3.1: Principali venditori, spedizioni e quote di mercato nel quarto trimestre del 2013, *IDC Worldwide Tablet Tracker*, 29 gennaio 2014 [28].

3.1, la voce *Others* rappresenta una combinazione di venditori importanti (come Microsoft, HP e Dell) e venditori poco conosciuti che solitamente distribuiscono dispositivi Android di fascia bassa [29]. In Figura 3.2, sempre secondo *IDC* si può vedere una previsione sulle quote di mercato dei vari sistemi operativi più diffusi sui tablet nel periodo 2012-2017 [30]. Vista la grande espansione di Android e la vasta gamma di dispositivi che ne fanno uso, nel seguito analizzeremo più nel dettaglio la distribuzione dei venditori e dei vari modelli di tablet con questo sistema operativo.
Un'altra compagnia che si occupa di ricerche di mercato, specializzata in dispositivi mobili e applicazioni web è *Localytics* [31] che fornisce la seguente analisi del mercato globale dei dispositivi Android. In Figura 3.3 si può vedere come il mercato sia suddiviso tra i vari produttori, con Samsung in testa, mentre in Figura 3.4 si può vedere una classifica dei modelli di tablet più venduti (l'unico tablet della famiglia Kindle Fire che ha la fotocamera posteriore è il Kindle Fire HDX da 8,9 pollici).

Per quanto riguarda le diverse versioni del sistema operativo (Figura 3.5) e le dimensioni dello schermo più comuni (definite come combinazione di dimensione/diagonale e densità/dpi, figura 3.6) possiamo vedere le statistiche

Tablet OS	2012 Market Share	2013 Market Share*	2017 Market Share*
Android	52.0%	60.8%	58.8%
iOS	45.6%	35.0%	30.6%
Windows	0.9%	3.4%	10.2%
Other	1.4%	0.8%	0.4%
Grand Total	**100.0%**	**100.0%**	**100.0%**

Figura 3.2: Previsioni sulle quote di mercato dei vari sistemi operativi più diffusi sui tablet nel periodo 2012-2017 (in tabella * indica una previsione), *IDC Worldwide Tablet Tracker*, dicembre 2013 [30].

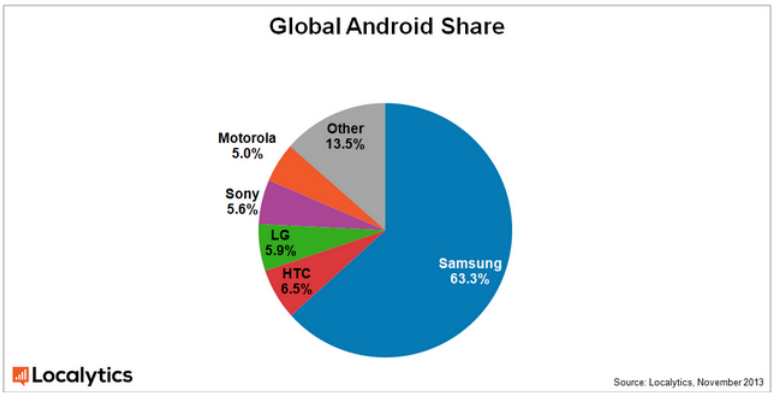


Figura 3.3: Suddivisione del mercato Android in base ai vari produttori [32].

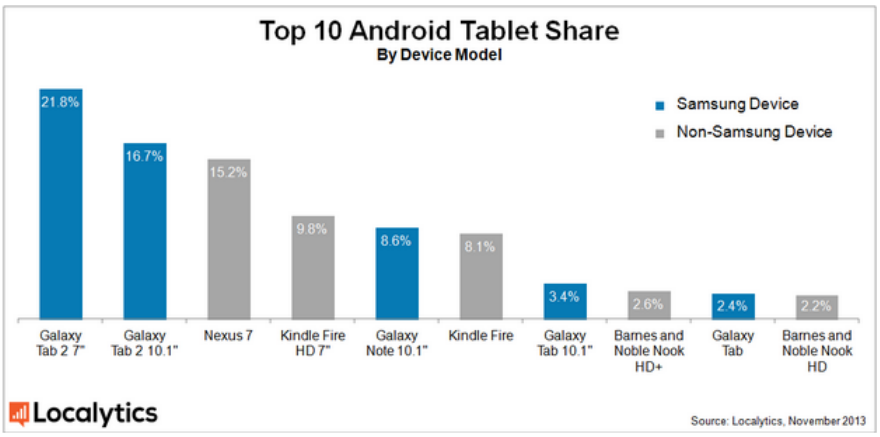


Figura 3.4: Classifica dei 10 tablet più venduti [32].

ufficiali di Android (i dati si riferiscono ai dispositivi che hanno installata l'ultima versione del *Google Play Store*) [33].

Version	Codename	API	Distribution
2.2	Froyo	8	1.2%
2.3.3 - 2.3.7	Gingerbread	10	19.0%
3.2	Honeycomb	13	0.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	15.2%
4.1.x	Jelly Bean	16	35.3%
4.2.x		17	17.1%
4.3		18	9.6%
4.4	KitKat	19	2.5%

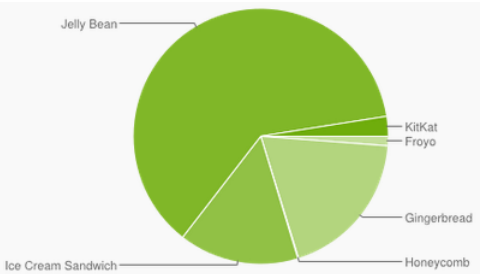
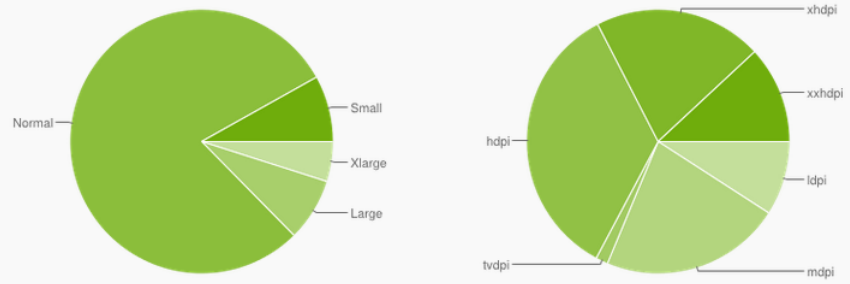


Figura 3.5: Distribuzione delle varie versioni del sistema operativo Android [33].

In Figura 3.6, le righe si riferiscono alla diagonale dello schermo misurata in pollici:

- *small*: 2" - 3.5"
- *normal*: 3.5" - 4.5"
- *large*: 4.5" - 7"
- *xlarge*: 7" - 10" o più

	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	8.1%						8.1%
Normal	0.2%	13.6%		33.7%	19.9%	11.9%	79.3%
Large	0.7%	4.3%	1.5%	0.6%	0.6%		7.7%
Xlarge	0.1%	4.3%		0.3%	0.2%		4.9%
Total	9.1%	22.2%	1.5%	34.6%	20.7%	11.9%	



Data collected during a 7-day period ending on March 3, 2014.
Any screen configurations with less than 0.1% distribution are not shown.

Figura 3.6: Distribuzione delle varie configurazioni del display dei dispositivi Android [33].

Le colonne si riferiscono alla densità del display misurata in pixel per pollice (dpi):

- *ldpi*: 100 dpi - 125 dpi
- *mdpi*: 125 dpi - 190 dpi
- *hdpi*: 190 dpi - 280 dpi
- *xhdpi*: 280 dpi - 350 dpi o più

Inoltre, sempre grazie a *IDC* è stata recuperato il seguente grafico (Figura 3.7) in cui è stata fatta una previsione sulle dimensioni dei display dei tablet nel periodo 2012-2017.

3.2.2 Gestione della fotocamera

La cattura di un'istantanea con la fotocamera è la prima fase del flusso operativo dell'applicazione. Di seguito è illustrato il funzionamento del processo di acquisizione su iOS e Android, i due sistemi operativi *mobile*

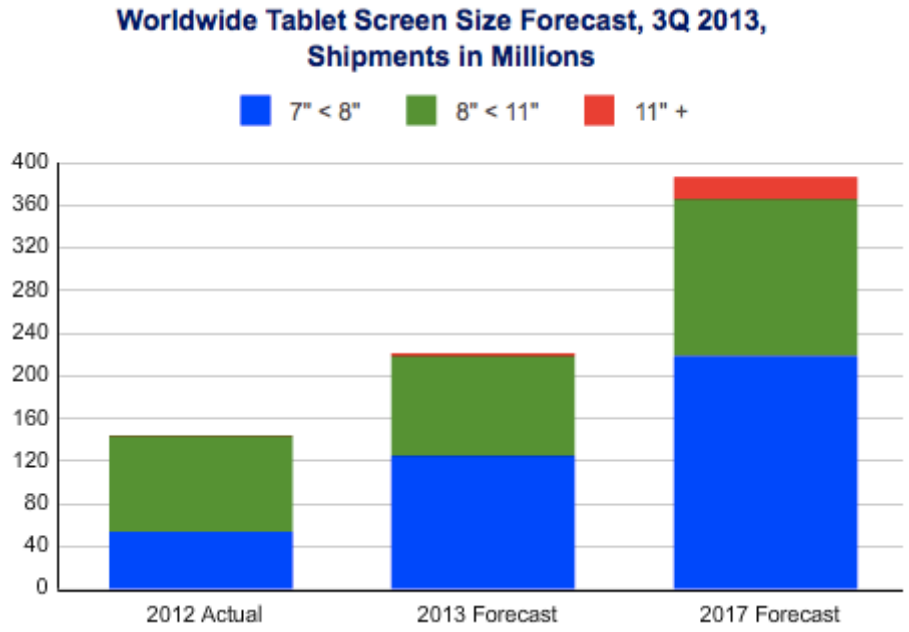


Figura 3.7: Previsione riguardante la dimensione dei display dei dispositivi nel periodo 2012-2017, *IDC Worldwide Quarterly Tablet Tracker*, dicembre 2013 [30]

principali. La libreria OpenCV gestisce ogni immagine usando la classe `Mat`, sostanzialmente una matrice in cui vengono memorizzati i valori dei singoli pixel. Per questo ogni frame che deve essere elaborato è memorizzato in questa forma.
Durante il periodo di tirocinio presso Maticad S.r.l. ho studiato in che modo è possibile interfacciarsi ai dispositivi Apple (in particolare su un *iPad 2* fornito dall'azienda) per ottenere i frame dalla fotocamera e i dati relativi ai sensori inerziali. L'accesso ai frame provenienti dalla fotocamera dei dispositivi Apple avviene utilizzando il *framework AV Foundation* che fornisce un'interfaccia Objective-C per ottenere dati audiovisivi. Gli step per accedere ai frame della fotocamera sono i seguenti:

- Creare l'oggetto `AVCaptureSession` che si occupa di coordinare il flusso di dati provenienti dai dispositivi in input e di presentarli ai dispositivi in output. Permette di impostare la risoluzione di cattura.

- Creare l'oggetto `AVCaptureDeviceInput` che rappresenta un *device* vi-

deo in input e aggiungerlo alla sessione.

- Creare l'oggetto `AVCaptureVideoDataOutput` che rappresenta l'output per la sessione di cattura. I frame possono essere ottenuti nello spazio colorimetri YUV (formato nativo dei pixel) oppure nello spazio RGBA.

- Implementare il metodo `(void)captureOutput:(AVCaptureOutput *) captureOutput didOutputSampleBuffer: (CMSampleBufferRef) sampBuff fromConnection:(AVCaptureConnection*) connection` per elaborare i frame in arrivo, chiamato ogni volta che è disponibile un nuovo frame. `CMSampleBufferRef` è il tipo del frame proveniente dalla fotocamera e per poter essere elaborato deve essere convertito prima in `CVImageBufferRef` e poi in `Mat`.

Un approccio alternativo è quello di utilizzare un *wrapper* fornito dal *framework* di OpenCV, in cui bisogna creare l'oggetto `CvVideoCamera` ed inizializzare le impostazioni di cattura come ad esempio la risoluzione, i frame per secondo, l'orientamento di cattura. La disponibilità di un nuovo frame viene notificata attraverso il metodo `(void) processImage:(cv::Mat &)image`. Come si può vedere dalla dichiarazione del metodo, si ottiene subito il frame di tipo `Mat`, pronto per le successive elaborazioni.

In Android la cattura dei frame avviene in maniera simile, acquisendoli direttamente come oggetto `Mat`. Alla creazione dell'*Activity* principale viene caricata la libreria OpenCV e inizializzato l'oggetto `CameraBridgeViewBase` che rappresenta la fotocamera. Tramite questo oggetto è possibile avviare ed interrompere la cattura e modificare alcune impostazioni come la risoluzione. La disponibilità e la visualizzazione di un nuovo frame avviene attraverso il metodo `public Mat onCameraFrame(Mat inputFrame)`: la matrice *inputFrame* è il frame catturato, mentre la matrice ritornata dal metodo contiene le eventuali elaborazioni effettuate.

3.2.3 Calibrazione della fotocamera

Il processo di acquisizione delle immagini inizia con la rilevazione dei raggi luminosi che, emanati da una sorgente, vengono poi riflessi da un oggetto, permettendo la percezione del colore. La luce riflessa che colpisce la camera viene memorizzata dal sensore fotosensibile e contribuisce alla formazione dell'immagine. Un semplice modello per descrivere questo sistema consiste nella *camera a pinhole* in cui l'immagine è formata solamente dai raggi riflessi dalla scena che passano attraverso un piccolo foro (il *pinhole*). In Figura 3.8 è rappresentato il modello appena descritto in cui f è la lunghezza focale della camera, Z è la distanza dell'oggetto dalla camera, X è la dimensione dell'oggetto e x la dimensione dell'oggetto proiettato sul piano immagine. Per la formazioni di immagini adatte all'elaborazione questo modello non è molto adatto in quanto, per tempi di esposizione brevi, non riesce a catturare abbastanza luce. Per questo tutte le fotocamere moderne usano un sistema di lenti che da un lato permette di accumulare una maggiore quantità di luce, avendo però lo svantaggio di introdurre delle distorsioni (esempi in Figura 3.9 e Figura 3.10). La calibrazione ha, in senso lato, lo scopo di correggere le principali deviazioni dal modello della camera a *pinhole* ed anche di determinare la relazione tra l'unità di misura della fotocamera (pixel) e l'unità di misura del mondo reale (millimetri ad esempio).

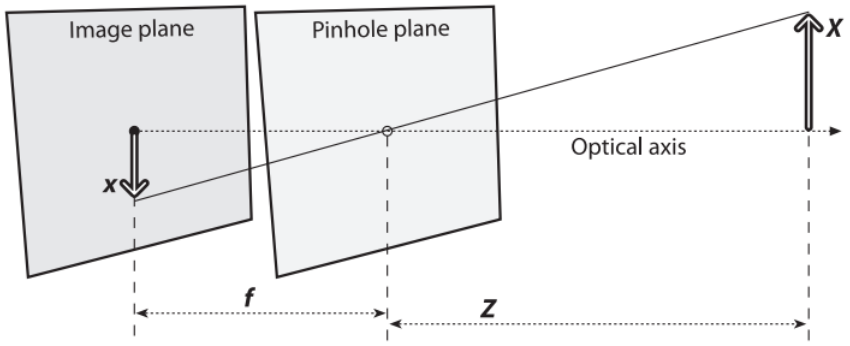


Figura 3.8: Modello della camera a *pinhole*: il foro lascia passare solo i raggi luminosi provenienti da un particolare punto dello spazio [34].

Le distorsioni dovute alle lenti possono essere di natura:

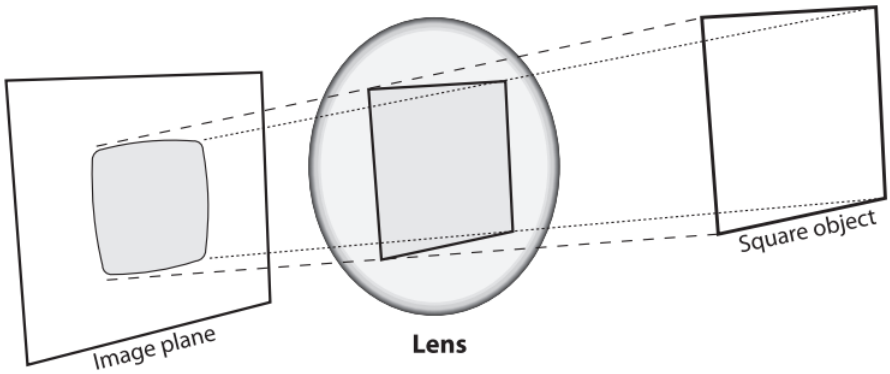


Figura 3.9: Esempio di distorsione radiale introdotta dalle lenti [34].

- *Radiale*: dovuta alla forma della lente, la posizione dei pixel vicini al bordo dell'immagine viene alterata producendo effetti come il *fish-eye* o il *barrel*. Solitamente la distorsione è 0 nel centro della lente ed aumenta muovendosi verso il bordo. Per correggere questo fenomeno si usano le seguenti equazioni [34]:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.1}$$

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{3.2}$$

- *Tangenziale*: dovuta a difetti nel processo di assemblaggio dei vari componenti della camera. La lente risulta non perfettamente parallela al piano immagine e si ottiene l'effetto mostrato in Figura 3.10. Per correggere questo fenomeno di usano le seguenti equazioni [34]:

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{3.3}$$

$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{3.4}$$

Nelle equazioni 3.1, 3.2, 3.3, 3.4, (x,y) sono le coordinate del pixel nell'immagine originale. Attraverso la calibrazione è possibile ottenere la matrice

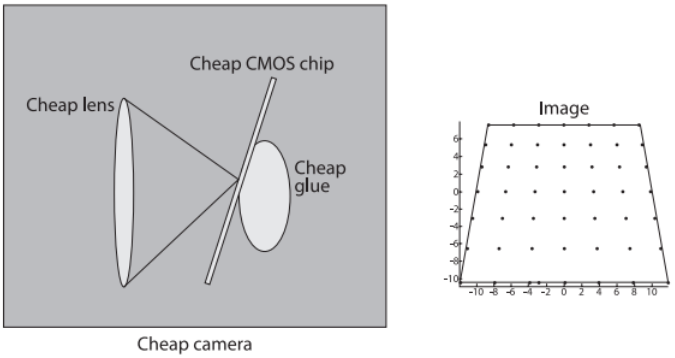


Figura 3.10: Esempio di distorsione tangenziale introdotta dalle lenti [34].

3.5 contenente parametri che caratterizzano le distorsioni dovute alle lenti.

$$Distortion_{coefficients} = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3) \tag{3.5}$$

Oltre alla distorsione dovuta alle lenti bisogna considerare anche un eventuale scostamento dovuto ad un non perfetto allineamento tra l'asse ottico ed il chip del sensore ottico. La calibrazione consiste nel trovare quattro parametri:

- f_x e f_y: tengono conto dell'*aspect ratio* del sensore fisico. La lunghezza focale f_x (misurata in pixel) è data dal prodotto della lunghezza focale fisica della camera (F, misurata in millimetri) e la dimensione s_x di un singolo elemento del sensore (misurata in pixel per millimetro). Lo stesso vale per f_y.

- c_x e c_y: rappresentano un possibile scostamento sul piano immagine del sensore, delle coordinate del centro del piano di proiezione dell'asse ottico.

I parametri sopra descritti vanno a formare la matrice 3.6 degli intrinseci che caratterizza una fotocamera.

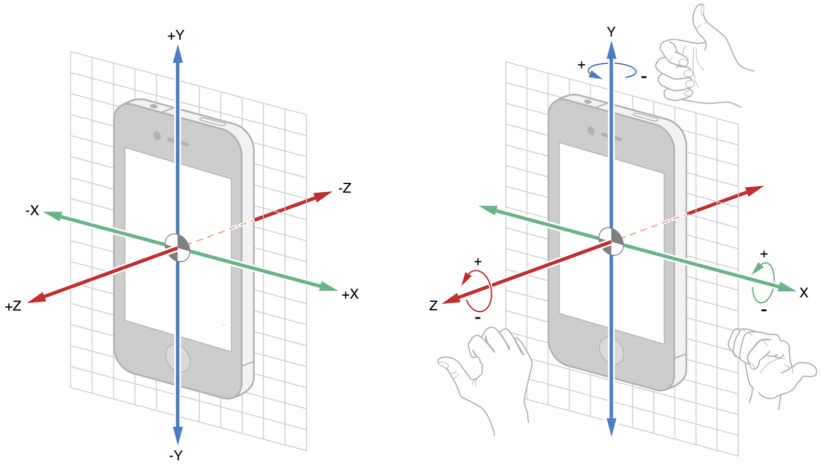
$$Intrinsic_{coefficients} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

Il processo di calibrazione è indipendente dal dispositivo in quanto, come vedremo in seguito, basta prendere diverse istantanee di un pattern ben definito (solitamente si fa uso di una scacchiera in bianco e nero) e grazie a strumenti presenti in MATLAB o a funzioni di OpenCV è possibile ottenere i valori della matrice degli interiseci.

3.2.4 Gestione dei sensori inerziali

Nell'ambito dell'applicazione i sensori inerziali vengono utilizzati per capire in che modo è orientato il dispositivo rispetto all'ambiente in cui si trova. Conoscendo l'orientamento possiamo proiettare correttamente il *layer* virtuale che permette di cambiare interattivamente l'aspetto della superficie selezionata. Solitamente i sensori inerziali (accelerometro e giroscopio) consentono di monitorare i movimenti come rotazioni, oscillazioni, cambi di inclinazione, rispetto al sistema di riferimento del dispositivo. Per determinare la posizione del dispositivo rispetto al sistema di riferimento del mondo esterno occorre utilizzare i sensori inerziali in combinazione a sensori geomagnetici (magnetometro). Nel seguito vedremo come sono gestiti questi sensori sia in iOS che in Android.

Sui dispositivi Apple, a partire dall'iPad 2, sono presenti tre tipi di sensori: accelerometro, giroscopio e magnetometro. L'accelerometro, composto attualmente da tre accelerometri (uno per ogni asse, x, y, z), misura i cambiamenti di velocità nel tempo lungo un percorso lineare. Combinando i tre accelerometri si possono rilevare i movimenti in tutte le direzioni ed ottenere l'orientamento del dispositivo. Il giroscopio misura il grado di rotazione attorno ai tre assi in radianti al secondo. Il magnetometro (che deve essere calibrato periodicamente) misura il campo magnetico totale osservabile al dispositivo che è uguale al campo magnetico terrestre più un errore introdotto dal dispositivo stesso e dall'ambiente circostante, viene misurato in microtesla [35]. In Figura 3.11 una rappresentazione dei sistemi di riferimento dei sensori. La gestione dei sensori avviene attraverso il *framework Core Motion* che mette a disposizione diversi metodi per recuperare ed elaborare i dati



(a) (b)

Figura 3.11: Sistemi di riferimento del (a) magnetometro e del (b) giroscopio [35]

ottenuti. Il punto di accesso centrale è la classe *CMMotionManager* che, una volta settato l'intervallo di aggiornamento, si occupa di raccogliere i dati provenienti dai sensori e passarli all'applicazione per le successive elaborazioni. Il recupero dei dati può avvenire con due modalità:

- Approccio *pull*: l'applicazione attiva il sensore e periodicamente campiona la misura più recente a disposizione.
- Approccio *push*: l'applicazione specifica un intervallo di aggiornamento (in numero di aggiornamenti al secondo, Hz) ed implementa un blocco per la gestione dei dati in arrivo.

Con il *CMMotionManager* è possibile accedere ai dati grezzi provenienti dai sensori (attraverso gli oggetti *CMAccelerometerData*, *CMGyroData* e *CMMagnetometerData*) oppure ottenere delle misure aggregate attraverso l'oggetto *CMDeviceMotion*. Quest'ultimo permette di ottenere diverse misurazioni raffinate (attraverso algoritmi interni) come la matrice di rotazione del dispositivo, il vettore di accelerazione di gravità, l'accelerazione applicata dall'utente, il campo magnetico (senza il rumore dovuto al dispositivo stesso) e misure filtrate di accelerometro e giroscopio. I dati ottenuti dal *CMDevi-*

ceMotion sono sempre misurati in relazione ad un frame di riferimento che può essere settato nei modi seguenti:

- *CMAttitudeReferenceFrameXArbitraryZVertical*: è il frame di riferimento di default, settato nel momento in cui viene lanciata l'applicazione. L'asse z è diretto verticalmente con direzione opposta alla gravità, l'asse x ha una direzione arbitraria sul piano orizzontale.

- *CMAttitudeReferenceFrameXArbitraryCorrectedZVertical*: il frame di riferimento è uguale al precedente eccetto per il magnetometro, che se presente e calibrato, viene utilizzato per migliorare l'accuratezza delle misurazioni dell'inclinazione (*yaw*) nel tempo.

- *CMAttitudeReferenceFrameXMagneticNorthZVertical*: l'asse z è verticale e l'asse x punta verso il nord magnetico.

- *CMAttitudeReferenceFrameXTrueNorthZVertical*: l'asse z è verticale e l'asse x punta verso il nord geografico. È l'opzione che richiede il maggior uso di CPU in quanto, oltre all'uso del magnetometro, deve essere disponibili la posizione dell'utente per calcolare la differenza tra il nord magnetico e geografico.

Per l'applicazione è stato usato il frame di riferimento *CMAttitudeReferenceFrameXMagneticNorthZVertical* in quanto è quello che meglio risponde ai reguisiti dell'applicazione, permettendo di ottenere la matrice di rotazione che consente alla griglia di rimanere allineata con il sistema di riferimento mondo. Per la visualizzazione della griglia sono state utilizzate le librerie grafiche OpenGL presenti nativamente in iOS.

Con il sistema operativo Android, data la grande varietà di dispositivi che lo usano, la gestione dei sensori è più complicata. Non tutti i dispositivi hanno lo stesso insieme di sensori e la qualità e l'accuratezza possono variare significativamente da un produttore all'altro. Per questo, specificando determinati elementi nel *Manifest* (`<uses-feature>`), per le applicazioni pubblicate su *Google Play* solamente i dispositivi che hanno tutti i sensori

necessari al funzionamento dell'applicazione la possono scaricare. È possibile accedere ai sensori disponibili sul dispositivo ed acquisire dati grezzi attraverso l'*Android sensor framework*. Questo strumento fornisce diverse classi ed interfacce per eseguire le diverse operazioni che coinvolgono i sensori. Prima di tutta bisogna determinare quali sensori sono presenti sul dispositivo e le capacità di ognuno (accuratezza, massimo range, consumo di energia). Se le caratteristiche sono soddisfacenti per l'applicazione si può proseguire con l'acquisizione dei dati provenienti dai sensori. Con la classe `SensorManager` si occupa della gestione dei sensori e tra le altre cose è possibile accedere alla lista dei sensori presenti sul dispositivo, effettuare la calibrazione, impostare la velocità di acquisizione. Con la classe `Sensor` è possibile creare una specifica istanza di un sensore ed ottenere informazioni sulle sue capacità. Il monitoraggio dei dati provenienti dai sensori può avvenire in due modi:

- Implementando la *callback* `onAccuracyChanged()`: l'accuratezza del sensore è cambiata e viene fornito il riferimento al sensore in questione.

- Implementando la *callback* `onSensorChanged()`: un sensore ha un nuovo dato disponibile; oltre al valore del nuovo dato è possibile recuperare ulteriori informazioni come l'accuratezza, il *timestamp* in cui è stato generato.

La frequenza con cui leggere i dati provenienti da un sensore può essere specificata con i seguenti valori: `SENSOR_DELAY_NORMAL` (ogni 200.000 ms), `SENSOR_DELAY_GAME` (ogni 20.000 ms), `SENSOR_DELAY_UI` (ogni 60.000) oppure `SENSOR_DELAY_FASTEST` (ogni 0 ms) [36]. Non ci sono metodi pubblici per sapere la frequenza di campionamento del sensore (può essere derivata dai *timestamp*).

3.2.5 Scelta della piattaforma di sviluppo

In base alle analisi e ai test effettuati è stato scelto di eseguire un primo sviluppo dell'applicazione per dispositivi *mobile* con sistema operativo Android. La ragione principale di questa scelta è che Android consente una

migliore integrazione con la libreria OpenCV, sfruttando maggiormente l'accelerazione hardware, ottenuta tramite elaborazioni in GPU, fondamentale per applicazioni di elaborazione delle immagini (per iOS non è ancora disponibile una versione di questo tipo). Inoltre vista la crescente diffusione sul mercato sarà possibile raggiungere un maggior numero di utenti (a scapito però di una maggiore frammentazione dei dispositivi).

Come obiettivo futuro, molto probabilmente, ci sarà il *porting* dell'applicazione su iOS in quanto si è visto che le misure provenienti dai sensori risultano più stabili (pensando anche ad una versione real-time dell'applicazione) ed è inoltre il secondo sistema operativo *mobile* più diffuso.

3.3 Progettazione dell'applicazione

La funzionalità base dell'applicazione è quella di segmentare in modo preciso ed efficiente la superficie selezionata dall'utente dopo aver preso un'istantanea dell'ambiente in cui si trova. Il problema in questione è molto complesso, in quanto le scene che si andranno ad analizzare possono avere caratteristiche molto diverse le une dalle altre sotto molteplici aspetti: forma, colore e dimensione dell'area di interesse, condizioni luminose differenti anche all'interno della stessa superficie, presenza di ombre e di oggetti che occludono parti dell'area. Per questo, per ottenere la segmentazione ottimale, è stato pensato di procedere in modo incrementale, suddividendo il problema in diverse fasi (Figura 3.12):

1. Individuazione e classificazione del tipo di superficie selezionata attraverso tecniche di *machine learning*.

2. Segmentazione preliminare per trovare la maggior parte della regione di interesse.

3. Classificazione delle possibili imperfezioni, anomalie e condizioni che possono portare ad una segmentazione non perfetta. È una fase "pre-

liminare" da effettuare inizialmente solo in fase di progetto per individuare i diversi casi. La classificazione verrà effettuata in base al metodo di risoluzione ed in base ai diversi livelli di disturbo visivo percepito dall'utente.

4. *Detection* delle eventuali zone lungo il bordo che sono incongruenti. Verrà effettuato un campionamento del bordo dell'immagine segmentata alla ricerca di anomalie individuate dalla fase precedente.

5. Trattamento delle discontinuità. Questa fase consiste nel trovare dei metodi per correggere le anomalie trovate ed ottenere una segmentazione corretta.

6. Rifinitura della regione. Una volta risolte le discontinuità serve un ulteriore fase di *detection* per verificare che l'anomalia sia stata definitivamente risolta.

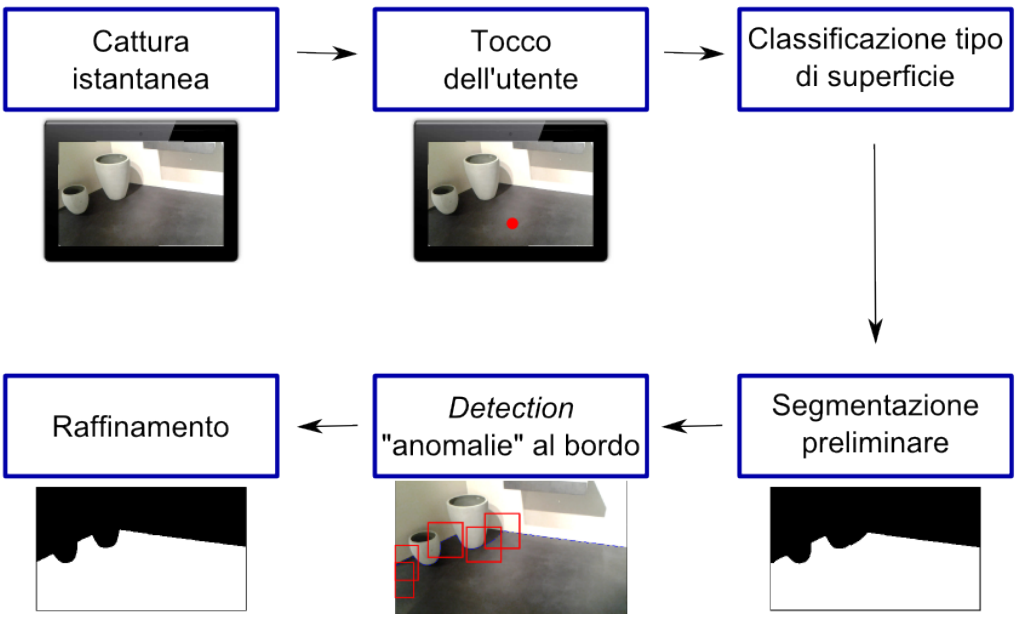


Figura 3.12: Flusso operativo del metodo proposto per la segmentazione di pavimenti.

La classificazione del tipo di superficie viene trattata nel prossimo capitolo. Nel seguito vengono descritte le altre fasi che costituiscono il metodo proposto.

3.3.1 Segmentazione preliminare

La segmentazione preliminare consiste nel trovare la maggior parte della regione di interesse per poter poi facilitare la fase successiva di rifinitura. Data la grande varietà di scenari che si possono incontrare, non è possibile effettuare la segmentazione iniziale con un singolo metodo. Per questo, in base alla tipologia di superficie, sono stati individuati i seguenti metodi:

- Per l'individuazione di superfici con texture omogenee e uniformi viene utilizzato l'algoritmo di *region growing* sviluppato per l'esame di Elaborazione delle Immagini LM (descritto nel Capitolo 5).

- Per individuare superfici con texture irregolari non uniformi viene utilizzata una particolare implementazione dell'algoritmo di *k-means* proposta dal professore Alessandro Bevilacqua (descritto nel Capitolo 5).

In entrambi i casi viene utilizzato lo spazio colorimetrico $L^*a^*b^*$ che è stato studiato per approssimare la visione umana. Fa parte della famiglia degli spazi colorimetrici basati sulla teoria degli opponenti che suggerisce che i colori vengano percepiti su tre differenti canali, nel caso dello spazio $L^*a^*b^*$ sono L^* per la luminanza (bianco-nero), a^* (verde-magenta) e b^* (blu-giallo) per la dimensione dei colori opponenti. Questa rappresentazione include tutti i colori percepibili ed è indipendente dal dispositivo che li visualizza.

3.3.2 Classificazione delle anomalie al bordo

Per l'individuazione delle possibili anomalie e imperfezioni che si possono avere durante la fase segmentazione sono state analizzate delle immagini contenenti ambienti reali. Le immagini presentano scene di interni e si riferiscono a diverse ambientazioni. Trattandosi di ambienti interni e dovendo segmentare essenzialmente elementi come pavimenti e pareti, si osserva che il

loro contorno è formato principalmente da segmenti rettilinei ed angoli netti. Scostamenti da questa conformazione sono dovuti per lo più ad elementi di arredo oppure a particolari scelte architettoniche.
Per trovare una classificazione adeguata, durante l'analisi delle immagini si è cercato di partire dai casi più semplici ed ideali (quelli in cui si ha una segmentazione perfetta) fino ad arrivare ai casi più complicati, identificando le zone dell'immagine che più probabilmente avrebbero portato ad una errata segmentazione, basandosi sull'aspetto visivo dell'immagine e su una prima applicazione dell'operatore di Sobel (per la *detection* dei contorni). I gradi di disturbo visivo associabili ad un determinato tratto del bordo segmentato possono essere:

- Nessuno: la segmentazione è stata effettuata correttamente ed il bordo trovato coincide perfettamente con quello dell'area di interesse.
- Accettabile: il bordo trovato non coincide perfettamente ma lo scostamento è visibile solamente dopo un'attenta analisi del risultato.
- Elevato: il bordo trovato si discosta notevolmente da quello effettivo dell'area di interesse.

Le diverse immagini in Figura 3.13 illustrano schematicamente la situazione in cui un potenziale utente potrebbe trovarsi dopo la prima fase di segmentazione "grezza": il puntino nero rappresenta la zona dell'immagine selezionata dall'utente, l'area bianca è il *foreground*, la superficie individuata dalla segmentazione preliminare e che deve essere raffinata, l'area grigia è il *background*, indica le zone restanti dell'immagine che non fanno parte della superficie di interesse.
Le classi individuate sono le seguenti:

- *Classe A*: se assumiamo inizialmente che il bordo dell'area di interesse sia costituito solamente da linee rette, la prima classe individuata presenta un bordo netto e rettilineo e coincide con quello dell'immagine, come si può vedere in Figura 3.13. Se dopo aver calcolato le caratteristiche di similarità (come colore, texture, gradiente, ecc.) di *background*

e *foreground*, queste risultano significativamente diverse, allora il bordo trovato è corretto. Questo primo caso non necessita di rifiniture e non provoca alcun disturbo visivo.

- *Classe B*: la geometria di questa classe è simile alla precedente, ma paragonando le caratteristiche di similarità si scopre una certa affinità tra *background* e *foreground* (Figura 3.13) il che può portare anche ad un elevato disturbo visivo. Questo potrebbe essere dovuto a diversi fattori come ad esempio presenza di ombre, di linee di fuga marcate, texture di elementi presenti sulla scena simili a quelli dell'area di interesse. Per risolvere questo problema bisogna effettuare un'analisi più approfondita dell'area applicando tecniche come il *k-means*, *CLAHE* o la *Gray Level Co-occurrence Matrix (GLCM)*, come vedremo in seguito.

- *Classe C*: il caso individuato da questa classe presenta un bordo formato da segmenti di rette (Figura 3.13). Si potrebbe procedere separando ogni singolo segmento, per ottenere una conformazione geometrica simile alla prima classe trovata. Ogni segmento, ricadrà nella *classe A* se le caratteristiche di similarità tra *background* e *foreground* calcolate sono diverse, nella *classe B* altrimenti.

- *Classe D*: a causa di particolari condizioni ambientali, come luci intense o rumore dovuto alla cattura dell'immagine, il contorno della zona segmentata potrebbe essere discontinuo e contenere quindi dei buchi (Figura 3.13) provocando un certo disturbo visivo. Per risolvere questa situazione bisognerebbe trovare la retta che collega i due segmenti e calcolare poi le caratteristiche di similarità come nei casi precedenti.

- *Classe E*: un semplice caso si ha per i bordi che coincidono con i contorni dell'immagine (Figura 3.13). In questo caso il bordo può essere considerato corretto, non sono necessarie correzioni e non ci sono disturbi visivi.

- *Classe F*: un altro caso che può verificarsi è quello in cui il bordo trovato non sia esattamente sovrapposto al bordo effettivo dell'area di interesse

(Figura 3.13). Una situazione del genere può provocare un elevato disturbo visivo e corrisponde ad una errata segmentazione. Per risolvere questa condizione si deve cercare di spostare il bordo trovato verso un bordo con intensità più "forte" presente nelle vicinanze. Dopo questa operazione, calcolando e comparando le caratteristiche di similarità si procederà a seconda che si ricada nel caso della *classe A* o della *classe B*.

- *Classe G*: Una variante della classe precedente si ha se sono presenti diversi bordi nelle vicinanze del contorno trovato (Figura 3.13), anche in questo caso il disturbo visivo potrebbe essere elevato. In questa condizione bisogna analizzare approfonditamente le aree tra i diversi segmenti (con tecniche come *k-means*, *CLAHE* o *GLCM*) e decidere in base ai risultati quale bordo è il migliore.

- *Classe I*: Se nell'ambiente sono presenti elementi di arredo, persone o altri ostacoli è possibile che l'area di interesse sia suddivisa in più parti (Figura 3.13). Se l'area mancante occupa una grande proporzione dell'immagine il disturbo visivo può essere elevato, mentre se l'area è piccola può passare inosservata rendendolo accettabile. In questi casi si potrebbero campionare dei punti oltre il bordo trovato e fare una *texture analysis* per vedere se ci sono aree dello stesso tipo, altrimenti chiedere l'intervento dell'utente per evidenziare ulteriori aree.

- *Classe J*: la seguente classe tiene in considerazione anche della presenza di eventuali elementi di arredo con forme arbitrarie, con bordi non rettilinei (Figura 3.13). Per verificare la correttezza della segmentazione si potrebbe suddividere il bordo trovato in segmenti in modo che ognuno approssimi un tratto rettilineo. Ogni segmento, analizzate le caratteristiche di similarità, si può far ricadere nella *classe A* o nella *classe B*.

- *Classe K*: un caso piuttosto frequente è quello in cui il bordo trovato non è regolare, ma frastagliato e non corrisponde al bordo effettivo dell'area di interesse (Figura 3.13) procurando un elevato disturbo visivo.

Questa situazione è dovuta a diversi fattori, come il rumore dell'immagine (riconducibile al processo di acquisizione ed alla presenza di condizioni luminose non ottimali) e la qualità dell'algoritmo di segmentazione preliminare. L'assunzione di bordo rettilineo è data da un'approssimazione dei punti del contorno secondo un modello parametrico lineare: ad esempio la trasformata di Hough (descritta in seguito) dei punti di segmentazione che permette di identificare le eventuali spezzate presenti. Una volta trovata la retta che meglio approssima il bordo trovato si può procedere con il calcolo ed il confronto dei parametri di similarità per far ricadere il caso nella *classe A* o nella *classe B*, altrimenti si deve eseguire un'ulteriore scomposizione come nella classe precedente.

3.3.3 *Detection* e risoluzione delle anomalie

La *detection* delle anomalie consiste nel capire se e in che misura il bordo trovato dalla prima fase di segmentazione si avvicina alla segmentazione ottimale o *ground truth*, che indica esattamente qual è l'area di interesse da individuare, ottenuta attraverso una segmentazione manuale dell'immagine. Visto che durante la fase di validazione del sistema non si avrà a disposizione la *ground truth*, il bordo trovato viene confrontato localmente, eseguendo un campionamento ed analizzando poi l'area nell'intorno del punto individuato, con il risultato ottenuto da altre tecniche (descritte nel seguito) che consentono di identificare ed isolare ulteriori caratteristiche, non ancora considerate, per migliorare la qualità della segmentazione. Le tecniche da applicare per il confronto del bordo individuato, sono state scelte in base ai risultati ottenuti nella fase di ricerca preliminare, in cui sono stati applicati vari metodi allo stato dell'arte ad immagini del dominio applicativo in questione. In base alle diverse tipologie di pavimenti che si potranno incontrare, sono state individuate le seguenti tecniche:

- Per superfici con texture omogenee e uniformi si applica il *Canny edge detector*, in quanto consente di individuare e localizzare accuratamente gli *edge* "forti" presenti nell'immagine. Inoltre per verificare la presenza di rette viene utilizzata la trasformata di Hough.

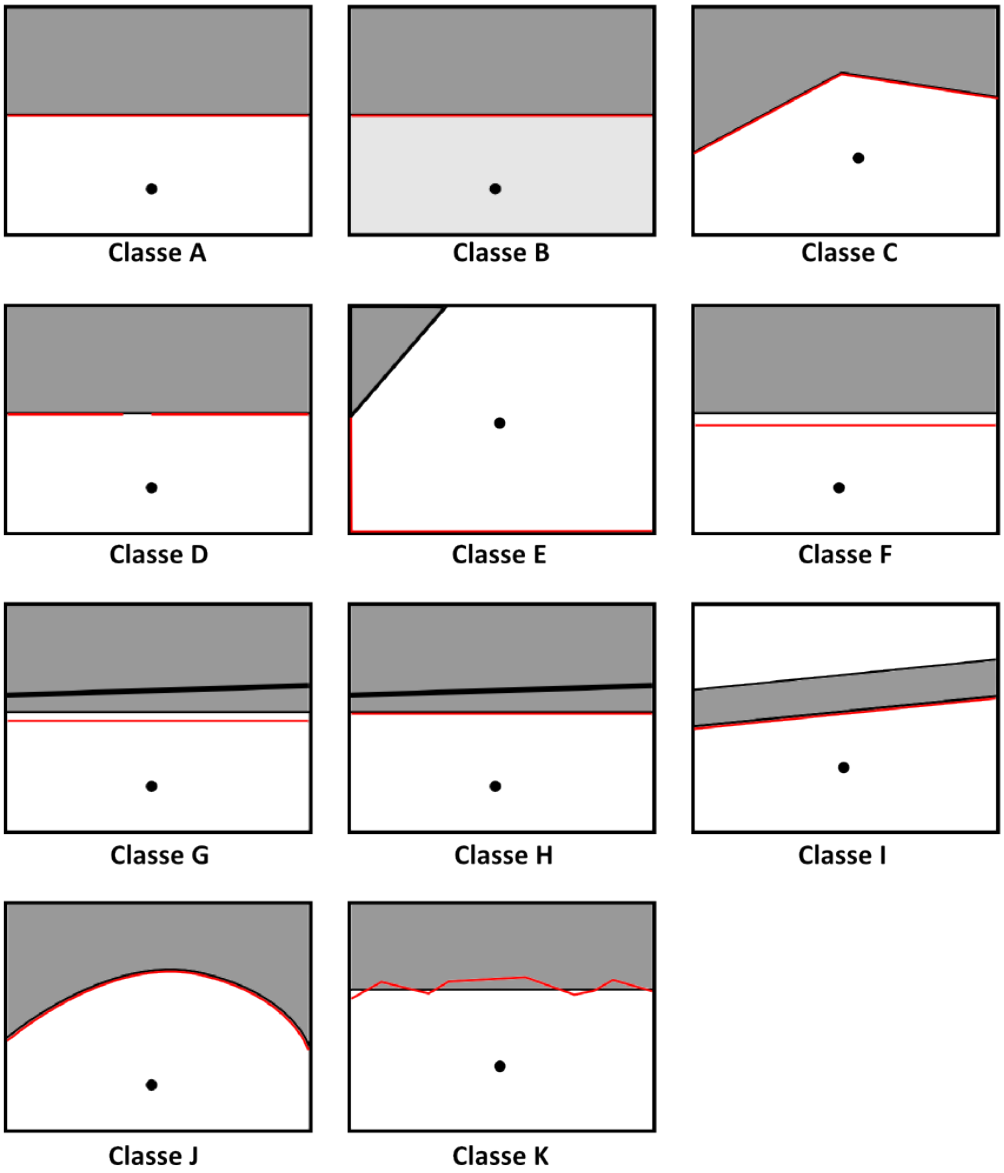
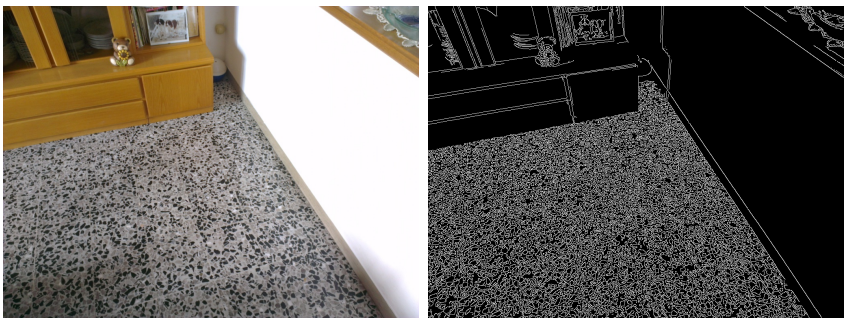


Figura 3.13: Disegni schematici delle classi di anomalie al bordo individuate.

- Per superfici con texture irregolari non uniformi viene applicato prima la tecnica di *mean shift* per cercare di uniformare l'area di interesse e poi il *watershed* per trovare un possibile bordo.

Il *Canny edge detector* [37] è un algoritmo consolidato nell'ambito dell'elaborazione delle immagini per il riconoscimento e la localizzazione dei contorni in un'immagine. L'algoritmo prevede diversi passi: viene applicato prima un filtro gaussiano per eliminare il rumore. Viene poi calcolato il gradiente e nelle tre direzioni (orizzontale, verticale e diagonale) per ciascun punto dell'immagine. Per decidere quali punti appartengono effettivamente ad un contorno vengono considerati solamente i massimi locali che corrispondono ai punti in cui la derivata del gradiente si annulla. L'estrazione dei massimi locali avviene attraverso un processo di sogliatura con isteresi: si definiscono due soglie (una bassa e una alta) che vengono poi confrontate con il gradiente in ciascun punto. Se il valore del gradiente è inferiore alla soglia bassa, il punto viene scartato. Se il valore del gradiente è superiore alla soglia alta, il punto viene accettato come contorno. Infine se il valore del gradiente cade tra le due soglie, il punto viene accettato solamente se contiguo ad un punto già identificato come bordo. Al termine di questo processo si ottiene un'immagine binaria in cui sono evidenziati i contorni (Figura 3.14).



(a) (b)

Figura 3.14: Esempio di applicazione del *Canny edge detector* con soglie $th_{low} = 50, th_{high} = 100$.

Per verificare la presenza di rette si può utilizzare la trasformata di Hough [38, 34]. Con questa tecnica, dopo aver parametrizzato i punti del bordo

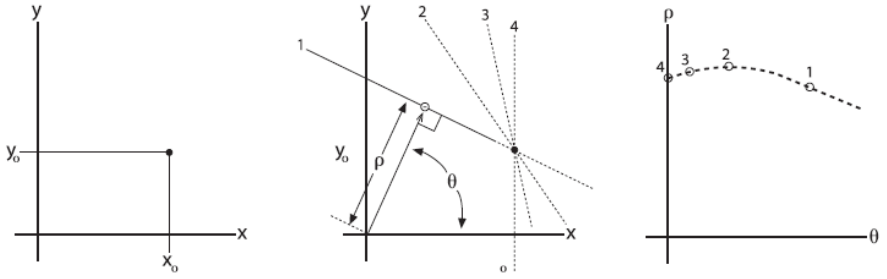
trovato, utilizzando un sistema di *voting* si deciderà se questi appartengono ad una retta o meno. Ogni punto (x_0, y_0) corrispondente ad un *edge* dell'immagine, viene parametrizzato attraverso il fascio di rette che gli passano attraverso. Ogni retta viene rappresentata in coordinate polari (ρ, θ):

- $\rho \geq 0$: rappresenta la distanza tra la retta e l'origine del sistema di riferimento, misurata in pixel.
- $\theta \in [0, 2\pi)$: è l'angolo che il vettore perpendicolare alla retta forma con l'asse x, misurata in radianti.

con la seguente equazione (eq. 3.7):

$$\rho(\theta) = x_0 \cos\theta + y_0 \sin\theta \tag{3.7}$$

Se per un dato punto (x_0, y_0) (Figura 3.15a) calcoliamo il fascio di rette passante per quel punto (Figura 3.15b) e per ogni retta plottiamo le coordinate corrispondenti nel piano polare (Figura 3.15c), otteniamo una sinusoide che è unica per quel punto. Se le sinusoidi relative a due punti distinti si intersecano, allora il punto di intersezione nel piano polare, corrisponde ad una retta passante per entrambi i punti nell'immagine originale (Figura 3.16).



(a) (b) (c)

Figura 3.15: (a) Punto (x_0, y_0) nell'immagine. (b) Famiglia di rette passanti per quel punto e parametrizzate da ρ e θ. (c) Sinusoide risultante.

A questo punto, per decidere l'effettiva presenza di rette, viene usato un *piano accumulatore* (con dimensioni ρ e θ) per tenere traccia, durante l'elaborazione di ogni punto, del numero di volte in cui si presenta una determinata coppia di valori (ρ, θ). Il piano accumulatore può essere visto come

un istogramma bidimensionale in cui le celle sono di dimensione ρ pixel per θ radianti. Una volta che sono stati processati tutti i punti, nel piano accumulatore viene fatta una ricerca per determinare i massimi locali che indicano i parametri che più probabilmente rappresenteranno delle rette nell'immagine originale [38]. Ciò significa che una retta può essere trovata individuando il numero di intersezioni tra le diverse sinusoidi. Se molte sinusoidi si intersecano in un punto, significa che la retta corrispondente sarà formata da più punti.

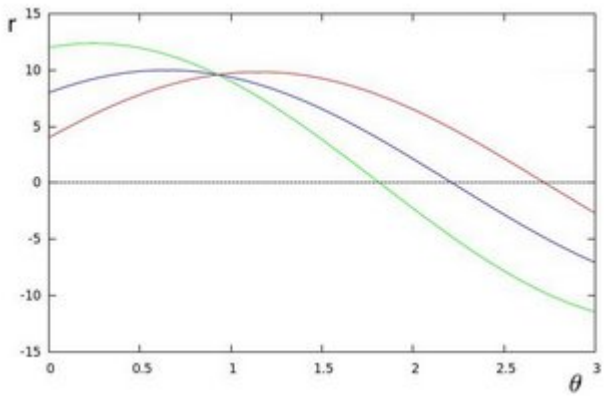


Figura 3.16: Sinusoidi risultati dall'applicazione della trasformazione a tre punti del bordo.

Nella trasformata di Hough è possibile specificare una soglia che determina il numero minimo di intersezioni richieste per l'individuazione di una retta. È importante anche decidere la risoluzione angolare di θ che limita l'inclinazione che le rette individuate possono avere nell'immagine originale. Questi parametri influiscono sia sulla qualità delle rette trovate che sul carico computazionale.

In presenza di superfici con texture non uniformi viene applicato il *mean shift* [14] una tecnica di *clustering* che consente di raggruppare aree dell'immagine trovando i picchi della funzione di distribuzione del colore su una certa area. Dato un insieme multidimensionale di punti, nel nostro caso le coordinate *(x,y)* del pixel che si sta considerando e la terna dei valori *(R,G,B)*, il *mean shift* è in grado di trovare blocchi di dati con caratteristiche

simili all'interno di una finestra che scorre su tutta l'immagine. Sfruttando l'abilità del *mean shift* di trovare le mode (quindi i picchi) della distribuzione, tutti i punti all'interno della finestra che convergono verso un picco diverranno appartenenti a quel picco. Questa relazione di appartenenza definisce i raggruppamenti finali. Per ogni punto (X,Y) dell'immagine originale vengono applicate un certo numero di iterazioni dell'algoritmo di *mean shift* considerando ogni punto del vicinato *(x,y)* tale che:

$$(x, y) : X - sp \leq x \leq X + sp, Y - sp \leq y \leq Y - sp, \|(R, G, B) - (r, g, b)\| \leq sr \tag{3.8}$$

In cui `sp` e `sr` definiscono rispettivamente le dimensioni della finestra spaziale e la distanza massima tra i vettori che rappresentano due colori. Da notare che l'algoritmo non dipende dallo spazio colorimetrico usato, quindi si può utilizzare un qualsiasi spazio che definisca i colori con tre componenti. Ad ogni iterazione viene trovato il valore medio delle coordinate spaziali *(X',Y')* e del vettore colore *(R',G',B')* che diventano il centro del vicinato per la prossima iterazione. Terminato il processo, il colore del pixel iniziale diventa il colore trovato nell'ultima iterazione [39]. Viene utilizzata un'ulteriore estensione di questo algoritmo in cui il *mean shift* viene applicato ad una piramide Gaussiana derivata dall'immagine originale. In questo modo è possibile rifinire i cluster di colori presenti nei livelli più alti della piramide man mano che si scende, fino ad arrivare all'immagine originale. La piramide Gaussiana (Figura 3.17) viene ottenuta sottocampionando successivamente l'immagine iniziale fino ad arrivare al livello desiderato. Per ottenere il livello *i-esimo* si convolve l'immagine al livello *i-1* con un kernel Gaussiano e si rimuovo tutte le righe e le colonne pari. Così facendo ogni immagine ha esattamente un quarto dell'area dell'immagine precedente, riducendo notevolmente il tempo computazionale richiesto per le elaborazioni.

L'algoritmo di *watershed* utilizzato corrisponde alla descrizione fornita nel Capitolo 2.

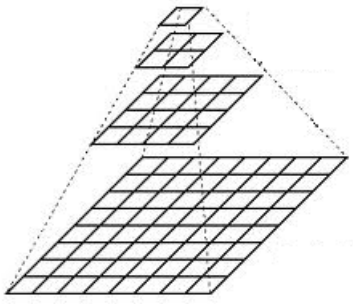


Figura 3.17: Illustrazione della piramide Gaussiana [39].

3.4 Misurazione dei risultati

La precisione della segmentazione ottenuta viene misurata nelle diverse fasi del progetto con i seguenti metodi:

- Visivamente: il risultato della segmentazione viene sovrapposto all'immagine originale valutandone l'accuratezza e l'eventuale disturbo visivo.

- Numericamente: nella fase di segmentazione preliminare per effettuare il *tuning* dei parametri degli algoritmi utilizzati avendo a disposizione la *ground truth* viene effettuata una differenza tra le maschere di segmentazione individuate. In particolare vengono utilizzate due metriche:

  – *Sum of Absolute Difference (SAD)*, che rappresenta la differenza *pixel based* tra la maschera di *ground truth* G e il risultato della segmentazione I.

  $$errore = \sum_{r=1}^{R} \sum_{c=1}^{C} |I_{r,c} - G_{r,c}| \tag{3.9}$$

  – Distanza di Hausdorff nella fase di raffinamento del bordo della segmentazione (Appendice A).

  Maggiore sarà la differenza, minore sarà l'accuratezza della segmentazione ottenuta.

Capitolo 4

Classificazione

Come detto precedentemente l'applicazione consiste nel segmentare il pavimento presente nella scena in base al tocco dell'utente. Essendoci una grande varietà di possibili pavimenti che si possono incontrare non è possibile ottenere una segmentazione accurata solamente con una determinata tecnica, ma per ogni tipologia di superficie bisogna individuare il metodo che permette di identificarla in modo più efficace ed efficiente possibile.
Il primo passo consiste quindi nel determinare il tipo di superficie in base al tocco dell'utente. Per fare ciò si può seguire la seguente metodologia:

1. Definire le classi iniziali in cui suddividere i pavimenti.

2. Definire l'insieme di *feature* da utilizzare per discriminare tra le varie classi, ad esempio le *feature* di Haralick [40], informazioni sulla geometria e sul colore.

3. Ottenere diversi campioni della tipologia di superfici che si vogliono riconoscere. Durante questa fase, se necessario, occorre applicare l'omografia inversa per evitare distorsioni prospettiche.

4. Utilizzare delle tecniche di *machine learning* per la classificazione delle superfici individuate.

4.1 Definizione delle classi delle superfici

L'applicazione che è stata sviluppata è rivolta prevalentemente ad aziende del settore ceramico ed arredamento di interni. Le scene che si andranno ad analizzare consistono principalmente in ambienti chiusi e potranno avere caratteristiche molto diverse le une dalle altre sotto molteplici aspetti: forma, colore e dimensione dell'area di interesse e delle eventuali piastrelle che la compongono, condizioni luminose differenti anche all'interno della stessa superficie, presenza di ombre e di oggetti che occludono parti dell'area. Quindi, vista la grande varietà di possibili superfici che si possono trovare, è stata stabilita la seguente classificazione per facilitarne il riconoscimento.
Una prima discriminazione può essere fatta in base alle proprietà geometriche della superficie di interesse. Come si può vedere in Figura 4.1 ho individuato due raggruppamenti principali:

- *LFV*: pavimenti che presentano linee di fuga[1] visibili e nette. Le possibili sottoclassi sono:
  - *RPSA*: superfici composte da elementi di forma regolare e della stessa dimensione ed allineati su una griglia (posa rettangolare), quindi con linee di fuga perpendicolari e parallele tra loro (formando quindi piastrelle quadrate o rettangolari).
  - *RPSN*: superfici composte da elementi uguali ai precedenti ma non allineati su una griglia, quindi con una posa sfalsata di una certa quantità.
  - *RPD*: superfici composte da elementi di forma regolare e con linee di fuga perpendicolari e parallele tra loro ma sono presenti elementi di dimensione diversa (posa mista).
  - *RGE*: superfici composte da elementi di forma regolare e della stessa dimensione, ma con linee di fuga in una disposizione diversa dalla precedente formando quindi differenti forme geometriche.

[1] Per linea di fuga si intende lo spazio presente tra una piastrella e l'altra, solitamente di colore diverso dalle piastrelle stesse in modo da delinearne chiaramente la forma.

- *LFI*: pavimenti in cui le linee di fuga non sono visibili e non sono individuabili singoli elementi. Le possibili sottoclassi sono:
  - *UNI*: superfici uniformi in cui non è possibile individuare alcun pattern o elemento distintivo.
  - *ALT*: superfici formate da disegni complessi in cui non è possibile individuare nessun pattern significativo.
  - *PREG*: superfici che non presentano linee di fuga marcate ma in cui è possibile individuare un pattern semplice che si ripete per tutta l'area di interesse, come ad esempio pavimenti in marmo o parquet.
  - *PCOMP*: superfici che presentano pattern particolari, complessi e ripetuti.
  - *PIRR*: superfici composte da elementi di forma irregolare, come ad esempio nei pavimenti a mosaico o porfido.

4.2 Definizione delle *feature*

Nel determinare quali siano le *feature* migliori per descrivere i vari tipi di superfici si può pensare a quali siano le caratteristiche su cui gli esseri umani si focalizzano per interpretare le informazioni contenute nelle immagini. Le tre caratteristiche fondamentali su cui si basa la percezione umana sono [40]:

- L'aspetto spettrale, che definisce la variazione media dei toni dell'immagine nelle varie bande dello spettro elettromagnetico visibile o a infrarosso.
- La texture, che contiene informazioni sulla distribuzione spaziale della variazione del colore in una certa banda.
- Informazioni sul contesto, che sono derivate da osservazioni nell'intorno dell'area che si sta analizzando.

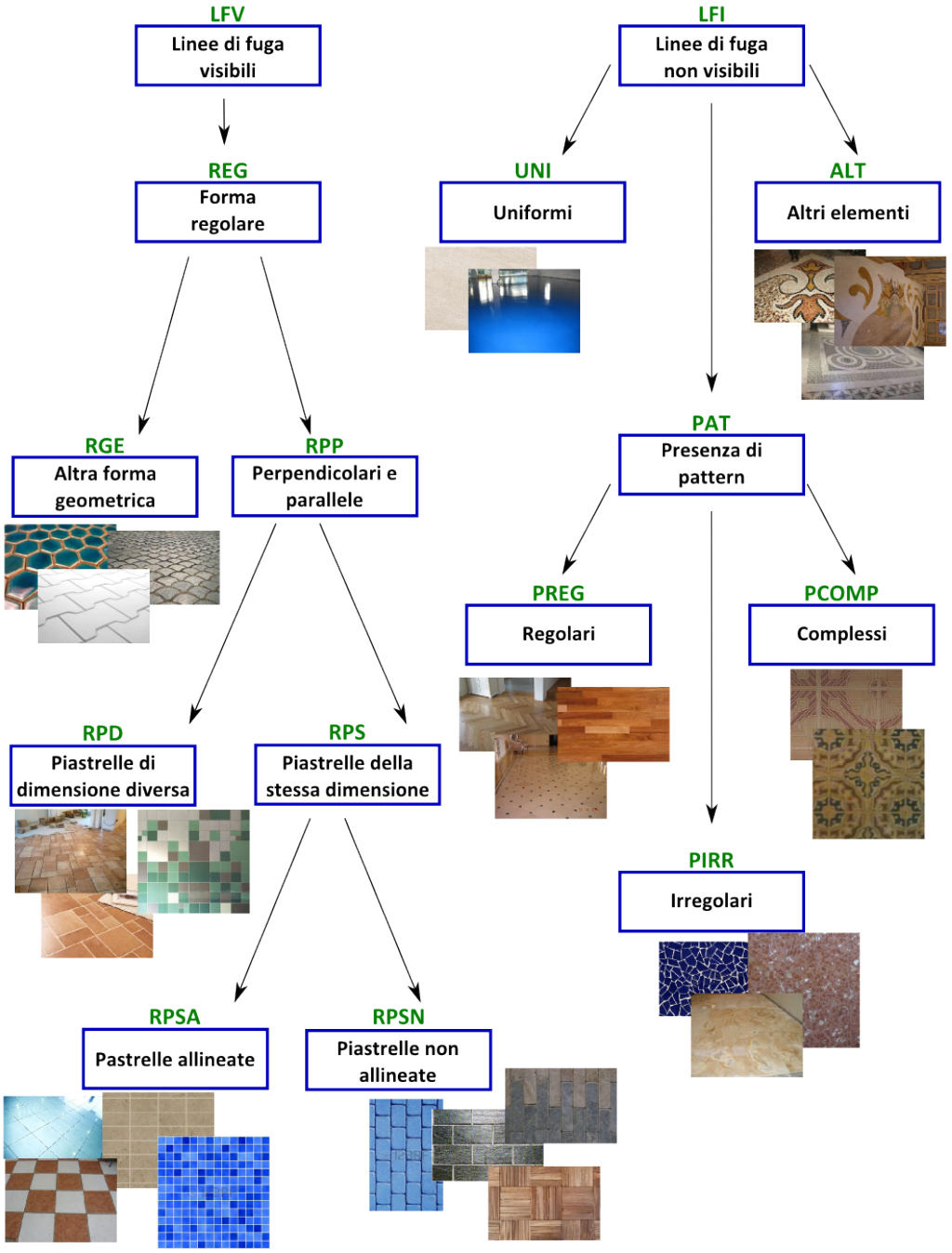


Figura 4.1: Suddivisione dei pavimenti in base alla geometria.

Quando diverse aree di un'immagine vengono processate indipendentemente da un computer, allora gli aspetti più importanti sono la texture ed il colore. La texture è una proprietà naturale di tutte le superfici. Contiene informazioni importanti sulla disposizione strutturale delle superfici e le loro relazioni con l'ambiente circostante. Anche se è abbastanza facile per gli esseri umani osservare, riconoscere e descrivere una texture in termini empirici, è stato molto difficile ottenere una definizione e un'analisi precisa al computer [40]. Una procedura generale per estrarre le proprietà delle texture da aree di un'immagine consiste nel calcolo delle *feature* di Haralick, che tengono conto sia dell'aspetto statistico che strutturale di una superficie.

4.2.1 *Feature* di Haralick

Le *feature* di Haralick [40] sono calcolate nel dominio spaziale e si basano sull'assunzione che le informazioni relative ad una texture sono contenute in tutta l'area dell'immagine che si sta analizzando. Le informazioni che è possibile recuperare riguardano caratteristiche come l'omogeneità, contrasto, numero e natura dei bordi presenti, complessità dell'immagine, presenza di strutture lineari o meno.
Le informazioni sulla texture sono ricavate dalla *Gray Level Co-occurrence Matrix (GLCM)*. Si tratta di una matrice $\mathbf{G}$ che indica il numero di volte in cui vale una determinata relazione spaziale Q tra le intensità di coppie di pixel (z_i, z_j) nell'immagine. La posizione relativa che devono avere due pixel viene stabilita da un certo operatore Q, ad esempio *"p2 è il pixel immediatamente a destra di p1"* (un esempio è mostrato in Figura 4.2), *"p2 è posizionato un pixel a destra ed un pixel sopra a p1"*. La scelta dell'operatore è fondamentale nel rilevare una texture particolare. Più precisamente le caratteristiche di una texture sono ricavate dalla matrice delle frequenze relative, ottenuta dividendo ogni elemento di $\mathbf{G}$ per il numero totale di pixel che soddisfano la relazione stabilita dell'operatore (uguale alla somma degli elementi in $\mathbf{G}$). Ogni elemento $p_{ij} = g_{ij}/n$ stima la probabilità che una coppia di punti che soddisfa l'operatore Q avrà valori (z_i, z_j).
Una volta calcolata la matrice delle frequenze relative e definite le seguenti

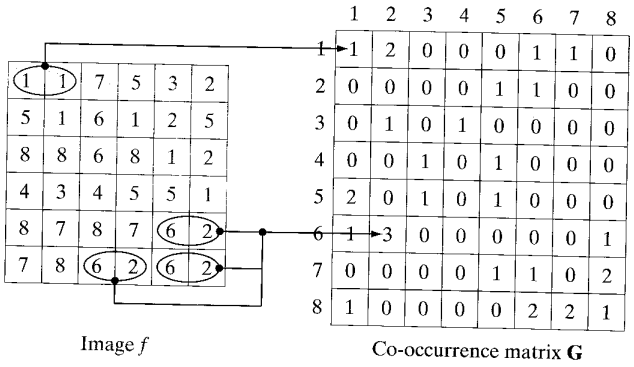


Figura 4.2: Calcolo della matrice di co-occorrenza di un'immagine con 8 livelli di grigio [12].

quantità:

- N_g: numero di livelli di grigio.
- Somma della riga *i-esima* e della della colonna *j-esima*:

$$p_x(i) = \sum_{j=1}^{N_g} p(i,j); \qquad p_y(j) = \sum_{i=1} N_g p(i,j) \tag{4.1}$$

- $$p_{x+y}(k) = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g}{}_{i+j=k} p(i,j) \qquad k = 2,3,...,2N_g \tag{4.2}$$

- $$p_{x-y}(k) = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g}{}_{|i-j|=k} p(i,j) \qquad k = 0,1,...,N_g - 1 \tag{4.3}$$

si possono calcolare le *feature* che descrivono la texture [40]:

- *Angular Second Moment*: misura l'uniformità dell'immagine. In un'immagine omogenea ci sono poche transizioni tra livelli di grigio.

$$f_1 = \sum_{i=1}^{N_g} \sum_{j=1}^{N_g} p(i,j)^2 \tag{4.4}$$

- *Contrast*: è una misura delle variazioni locali dei livelli di grigio presenti nell'immagine.

$$f_2 = \sum_{n=0}^{N_g-1} n^2 p_{x-y}(n) \tag{4.5}$$

- *Correlation*: misura quanto i livelli di grigio sono linearmente dipendenti.

$$f_3 = \frac{\sum_i \sum_j (ij) p(i,j) - \mu_x \mu_y}{\sigma_x \sigma_y} \tag{4.6}$$

  in cui $\mu_x, \mu_y, \sigma_x, \sigma_y$ sono le medie e le deviazioni standard di p_x e p_y

- *Sum of Squares*: varianza .

$$f_4 = \sum_i \sum_j (i-\mu)^2 p(i,j) \tag{4.7}$$

- *Inverse Difference Moment*: misura la vicinanza spaziale della distribuzione degli elementi di **G** alla diagonale.

$$f_5 = \sum_i \sum_j \frac{1}{1+(i-j)^2} p(i,j) \tag{4.8}$$

- *Sum Average*:

$$f_6 = \sum_{i=2}^{2N_g} i p_{x+y}(i) \tag{4.9}$$

- *Sum Variance*:

$$f_7 = \sum_{i=2}^{2N_g} (i - f_8)^2 p_{x+y}(i) \tag{4.10}$$

- *Sum Entropy*:

$$f_8 = -\sum_{i=2}^{2N_g} p_{x+y}(i) \log p_{x+y}(i) \tag{4.11}$$

- *Entropy*: misura la casualità degli elementi di **G**.

$$f_9 = -\sum_i \sum_j p(i,j) \log p_{x+y}(i) \tag{4.12}$$

- *Difference Variance*:

$$f_{10} = \sum_{i=1}^{N_g} (i - \mu_{p_{x-y}})^2 p_{x-y}(i) \tag{4.13}$$

- *Difference Entropy*:

$$f_{11} = -\sum_{i=0}^{N_g-1} p_{x-y}(i) \log(p_{x-y}(i)) \tag{4.14}$$

- *Information Measure of Correlation*:

$$f_{12} = \frac{f_9 - HXY1}{\max\{HX, HY\}} \tag{4.15}$$

$$f_{13} = (1 - exp[-2.0(HXY2 - f_9)])^{1/2} \tag{4.16}$$

in cui HX e HY sono le entropie di p_x e p_y.

$$HXY1 = -\sum_i \sum_j p(i,j) \log(p_x(i) p_y(j)) \tag{4.17}$$

$$HXY2 = -\sum_i \sum_j p_x(i) p_y(j) \log(p_x(i) p_y(j)) \tag{4.18}$$

- *Maximal Correlation Coefficient*:

$$f_{14} = (getSecondMaxEigenvalue(Q))^{1/2} \tag{4.19}$$

in cui

$$Q(i,j) = \sum_k \frac{p(i,k)p(j,k)}{p_x(i)p_y(k)} \tag{4.20}$$

Per quanto riguarda il carico computazionale, il numero di operazioni richieste per il calcolo della *GLCM* è direttamente proporzionale alla risoluzione dell'immagine [40], mentre la dimensione è determinata dal numero di possibili livelli di grigio. Ad esempio per un'immagine a 8 bit, la matrice ha

dimensione 256×256. In ambito mobile quello che più influisce sulle prestazioni è la dimensione dell'area dell'immagine su cui verrà calcolata la matrice di co-occorrenza: la finestra su cui effettuare l'elaborazione deve essere abbastanza grande da catturare la struttura della texture da analizzare e nello stesso tempo non deve influire eccessivamente sulle prestazioni.

4.3 *Machine Learning*

Il *machine learning* è una branca dell'intelligenza artificiale che studia metodologie e modelli di apprendimento automatico, al fine di riconoscere, classificare o raggruppare entità e situazioni di interesse a partire dalle osservazioni sui dati in ingresso. Uno degli obiettivi principali del *machine learning* è quello di imparare a riconoscere automaticamente modelli complessi e prendere decisioni intelligenti basate su dati; la difficoltà sta nel fatto che l'insieme di tutte le possibili configurazioni di un fenomeno è troppo grande per essere coperto dall'insieme degli esempi osservati. Da qui è necessario l'utilizzo di tecniche per generalizzare gli esempi forniti, in modo da essere in grado di produrre risultati utili per casi mai visti in precedenza. Le tecniche di *machine learning* possono essere suddivise in tre categorie principali:

- Apprendimento *supervisionato*: ad ogni campione in ingresso è associata un'etichetta corrispondente alla classe rappresentata. Un sistema di questo tipo genera una funzione di inferenza che associa (*classification*) ad ogni dato in ingresso il suo valore corretto nell'insieme di uscita (l'etichetta). Quando saranno presentati campioni sconosciuti, sprovvisti di etichetta, il classificatore dovrebbe essere in grado di predire la classe corrispondente.

- Apprendimento *non supervisionato*: ai dati in ingresso non è associata un'etichetta e si cerca di determinare automaticamente le classi (*clustering*) in cui sono organizzati i dati in quanto non sono note a priori. Lo scopo è quello di confrontare i dati e ricercare similarità o differenze, attribuendo alla stessa classe i dati che sono "vicini" sulla base di qualche metrica predeterminata.

- Apprendimento *con rinforzo*: si basa sul presupposto di poter ricevere degli stimoli (ricompense o punizioni) dall'esterno a seconda delle scelte dell'algoritmo. In base agli stimoli ricevuti si cercherà di inferire un modello per classificare i dati futuri. Ai dati inizialmente non è associata esplicitamente un'etichetta, ma viene determinata in base agli stimoli esterni.

L'apprendimento generalmente avviene analizzando le *feature* che caratterizzano l'insieme di campioni forniti nella fase preliminare di *training* (*training set*). Successivamente, nella fase di *test* si verifica se e con che precisione il sistema è in grado di riconoscere e classificare correttamente esempi mai visti in precedenza (*test set*). Per misurare l'accuratezza di un determinato metodo di apprendimento automatico vengono usate principalmente due tecniche [34]:

- *Cross-validazione*: consiste nel dividere i campioni a disposizione in K sottoinsiemi differenti; si effettua il training utilizzando *K-1* sottoinsiemi e l'insieme rimanente viene utilizzato per il test. Si ripete questa operazione K volte scegliendo per il test un diverso sottoinsieme. L'errore finale si ottiene come media dei K errori trovati.

- *Boostrapping*: è simile alla *cross-validazione* ma per il test set vengono scelti a random dei campioni dal training set, ciò significa che molti dati verranno riusati in differenti test set e che altri potrebbero non venire mai selezionati. I dati del test set così ottenuto non vengono utilizzati per il training.

Nel contesto della tesi, vista la grande varietà di pavimenti e rivestimenti che si possono presentare, l'apprendimento automatico viene utilizzato per determinare la classe della superficie in seguito al tocco dell'utente. Visto che conosciamo quali sono le classi in cui sono suddivisi i pavimenti, verrà utilizzato un algoritmo di apprendimento automatico supervisionato: i *Random Trees* [41].

4.3.1 *Random Trees*

Il metodo dei *Random Trees* o *Random Forests* [41] si basa sulla costruzione di molteplici alberi decisionali. Per la classificazione di un nuovo dato, si fa analizzare il vettore di *feature* che lo rappresenta da ogni albero della foresta ed alla fine del processo, ciascuno di essi restituisce sui nodi terminali (o foglie) un "voto" che indica la classe inferita per quel dato. La classificazione finale corrisponde alla classe che ha ricevuto più "voti" da parte di tutti gli alberi della foresta.
Un singolo albero decisionale è un albero binario che ha lo scopo di predire la classe di un dato in base al vettore di *feature* che lo rappresenta. Il processo di predizione parte dal nodo radice e durante l'analisi si deciderà se scendere verso il nodo figlio di destra o di sinistra in base al valore assunto dalla variabile del vettore delle *feature* associata a quel nodo. Il valore della variabile viene confrontato con una soglia (anche questa associata al nodo): se il valore è minore della soglia si scenderà verso sinistra, altrimenti verso destra. La classe risultante sarà sulle foglie dell'albero. Questa suddivisione viene ripetuta ricorsivamente fino alle foglie dell'albero.
Avendo a disposizione N campioni nel *training set* e M variabili per ogni vettore delle *feature*, ciascun albero di una *Random Forest* viene fatto espandere nel modo seguente [41, 39]:

- Il training set di ogni albero è composto da N campioni scelti a random con rimpiazzo. Alcuni dati potrebbero essere presenti più volte ed altri potrebbero essere assenti.

- In ogni nodo vengono scelte a random $m << M$ variabili. La soglia che permette di ottenere la massima separazione tra le variabili selezionate viene utilizzata per suddividere il nodo. Il valore di m è mantenuto costante durante la crescita dell'albero e solitamente è uguale a $\sqrt{M}$.

- Ogni albero viene fatto crescere fino alla massima altezza consentita, non viene effettuato nessun *pruning*.

Con i *Random Tree* non c'è la necessità di procedure per la stima dell'accuratezza, come ad esempio la *cross-validazione*, *bootstrapping* o un test set

separato per avere una misura dell'errore di classificazione. L'errore viene stimato internamente durante il training. Quando vengono scelti a random i campioni che dovranno formare il training set di un albero, circa un terzo dei dati (*Out-Of-Bag (OOB) data*) non vengono usati per il training ma per ottenere una stima obiettiva dell'errore di classificazione man mano che gli alberi vengono aggiunti alla foresta. Dopo che ogni vettore appartenente all'insieme *OOB* è stato classificato, si prende come riferimento la classe che ha ottenuto il maggior numero di voti. Il numero di volte in cui la classe considerata è diversa dalla vera classe dei vettori appartenenti all'insieme *OOB* rappresenta la valutazione dell'errore.

La struttura dei *Random Trees* consente all'utente di ottenere molte informazioni sui dati che hanno permesso una predizione accurata. La maggior parte di queste informazioni deriva dall'uso dei dati *OOB* che sono stati lasciati fuori dalla fase di training. Le informazioni consistono in:

- *Importanza delle variabili*: utile per ridurre il numero di variabili che il classificatore deve considerare aumentando di conseguenza la velocità computazionale (non dovendo più calcolare determinate *feature*). Per stimare l'importanza della variabile *m-esima*, per ogni vettore di *feature*, si esegue una permutazione random dei valori assunti dalla variabile nell'insieme dei vettori *OOB* (questo assicura che la distribuzione della variabile rimane la stessa dei dati originali). Si esegue una classificazione con i vettori alterati e si procede calcolando il nuovo errore. La quantità in cui il nuovo errore supera l'errore originale definisce l'importanza della *m-esima* variabile [44].

- *Prossimità (somiglianza) tra coppie di dati*: viene calcolata utilizzando una matrice *prox NxN*. Tutti i dati a disposizione (sia del training set che nell'insieme *OOB*) vengono classificati e se i dati k e n finiscono nello stesso nodo terminale si incrementano le relative prossimità di uno (*prox(n,k)++, prox(k,n)++*). La matrice viene poi normalizzata dividendola per il numero totale di alberi. La prossimità tra un dato e se stesso è uno.

- *Rilevamento degli outlier*: sono i dati che si scostano maggiormente da tutti gli altri dati appartenenti alla stessa classe (il valore della prossimità è piccolo). Per un certo vettore n appartenente alla classe C lo scostamento si calcola con l'equazione 4.21. Si trova poi la media (μ_{outC}) e la deviazione standard (σ_{outC}) degli scostamenti all'interno della stessa classe. La misura normalizzata di un *outlier* è data dall'equazione 4.22.

$$out(n) = nSample/\sum_{k \in C} prox(n,k)^2 \tag{4.21}$$

$$OUT(n) = \frac{out(n) - \mu_{outC}}{\sigma_{outC}} \tag{4.22}$$

Una caratteristica importante dei *Random Trees* consiste nel non essere soggetto ad *overfitting* all'aumentare del numero degli alberi [41]. L'*overfitting* è un fenomeno che si verifica quando il classificatore modella perfettamente i dati presenti nel training set ma non è in grado di generalizzare e quindi riconoscere correttamente dati mai visti in precedenza.
Supponendo che il *training set* sia composto da n campioni, ognuno caratterizzato da m *feature* e nella foresta siano presenti M alberi, la complessità computazionale della costruzione della foresta è data da $O(M(nm \log n))$ [42]. Dovendo fare assunzioni sulla dimensione di un albero, il tasso standard di crescita di un albero con n foglie è $O(\log n)$. Inoltre ad ogni possibile profondità devono essere considerate tutti gli n campioni che fanno parte del *training set* per cui otteniamo $O(n \log n)$. Visto che ad ogni nodo vengono presi in considerazione tutte le m *feature* otterremo $O(mn \log n)$. Il training viene effettuato nello stesso modo su tutti gli M alberi presenti nella foresta, quindi il costo computazionale finale sarà $O(M(mn \log n))$.

Capitolo 5

Implementazione

Nei seguenti paragrafi vengono descritte le parti dell'applicazione che sono state implementate: i risultati della calibrazione effettuata su dispositivi iOS e Android, la fase di classificazione del pavimento, gli algoritmi utilizzati per la segmentazione preliminare e poi per il raffinamento.
Durante lo sviluppo è stata utilizzata l'ultima versione delle librerie OpenCV, la 2.4.8.

5.1 Risultato della calibrazione

Per effettuare la calibrazione della fotocamera occorre prendere diverse istantanee di un pattern ben definito, di cui si conoscono la forma e le dimensioni (solitamente si fa uso di una scacchiera in bianco e nero). Disponendo il pattern in diverse direzioni (si veda Figura 5.1) è possibile calcolare, attraverso delle operazioni messe a disposizione dalla libreria OpenCV, la posizione e l'orientamento (relativo) della camera per ogni immagine acquisita, così come il valore degli intrinseci.

La calibrazione è stata effettuata su due dispositivi: un Apple iPad 2 ed un Samsung Galaxy Tab II. Ciò ha permesso di calcolare i seguenti parametri (Tabella 5.1) della matrice degli intrinseci (la matrice delle distorsioni non è stata calcolata dato che le distorsioni ottiche dovute alle lenti sono già

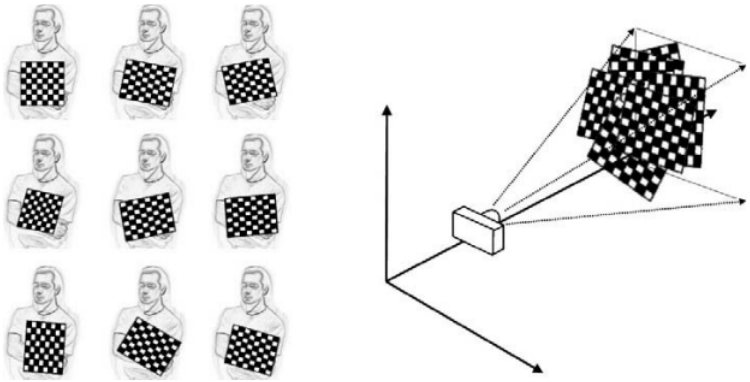


Figura 5.1: Diversi orientamenti di una scacchiera per ottenere una corretta calibrazione [34].

	Apple iPad 2	Samsung Galaxy Tab II
f_x	1278	1072
f_y	1274	1060
c_x	525	670
c_y	374	350

Tabella 5.1: Parametri di calibrazione.

calibrate e l'immagine proveniente dal sensore non è affetta da alterazioni geometriche significative). Le immagini utilizzate nella fase di calibrazione su iPad 2 hanno risoluzione 960×720, mentre sul dispositivo Android 1280×720

Questi parametri fanno si che la griglia virtuale venga visualizzata correttamente, con la giusta prospettiva e senza distorsioni.

5.2 Reperimento dei campioni

Le immagini utilizzate nella successiva fase di classificazione sono state ottenute sia fotografando i diversi ambienti interni in cui sono stata durante il periodo della tesi, sia attraverso *Google Immagini* [43]. Per evitare problemi di distorsione prospettica e di classificazione data la non invarianza di scala delle *feature* di Haralick, ad alcune immagini occorre applicare l'omografia inversa (Appendice B). Questa trasformazione permette di riproiettare l'immagine su un piano, eliminando gli effetti dovuti alla prospettiva (Figura

5.2): un esempio evidente si può vedere nelle linee di fuga che sono rette parallele nella realtà ma che nell'immagine sembrano andare ad incontrarsi.

(a) (b)

Figura 5.2: Esempio di correzione prospettica. (a) Immagine prima della trasformazione. (b) Immagine senza distorsioni prospettiche.

5.3 Classificazione della superficie

Per determinare il tipo della superficie viene eseguita una classificazione con i *Random Tree* dell'area nell'intorno del punto toccato dall'utente. L'area in questione deve essere abbastanza grande da avere sufficiente contenuto informativo su texture, colore e struttura della superficie considerata. Analizzando diverse immagini raffiguranti scene simili a quelle che si andranno ad incontrare nella realtà, è risultato che la dimensione ottimale della finestra (*patch*) rappresentate la superficie deve avere un lato compreso tra il 20% e il 25% della larghezza dell'immagine. Ad esempio in immagini con risoluzione 1024×768 la finestra avrà un lato compreso tra 205 pixel e 255 pixel.
Per l'addestramento dei *Random Trees* sono state estratte delle *patch* di dimensione 220×220 sia dalle immagini a disposizione che da altre immagini reperite tramite *Google Immagini* [43]. Da notare che prima dell'estrazione delle *patch*, se necessario, bisogna effettuare la correzione prospettica per rimuovere gli artefatti dovuti appunto alla prospettiva.
Viste le numerose classi di pavimenti individuate in Figura 4.1, in un primo momento abbiamo deciso di concentrarci sulle classi che identificano la maggior parte delle superfici che possono capitare: *UNI*, *PCOMP*, *REG* e *PIRR*.

Per ognuna di queste classi sono state raccolte delle immagini rappresentative e sono state estratte diverse *patch*. Tutte le *patch* raccolte sono state analizzate per calcolare le *feature* di Haralick, usando le formule indicate in [40] (e imponendo distanza pari a uno per il calcolo delle *GLCM*), da impiegare poi per la classificazione con i *Random Tree*. Le *patch* sono state suddivise in *training set* e *test set* avendo cura che i campioni in un insieme non fossero presenti anche nell'altro. Dal momento che i *Random Tree* sono un algoritmo di apprendimento automatico supervisionato ad ogni *patch* è stata associata un'etichetta numerica (per ragioni computazionali) che rappresenta la relativa classe. È stata utilizzata l'implementazione dei *Random Tree* presente in OpenCV. L'algoritmo di *machine learning* è implementato dalla classe `CVRTrees` che oltre ai metodi per l'addestramento e la predizione della classe per nuovi campioni, mette a disposizione varie funzione per ottenere informazioni aggiuntive, come l'importanza delle variabili, la prossimità, il numero di alberi usati e l'errore di classificazione. Per il training i parametri principali sono:

- `max_depth`: è la profondità massima dell'albero. Un valore piccolo porterà ad avere poca accuratezza, mentre un valore elevato può portare ad *overfitting*.

- `max_trees_in_forest`: numero massimo di alberi presenti nella foresta. Tipicamente più alberi ci sono e maggiore sarà l'accuratezza. Tuttavia il miglioramento dell'accuratezza generalmente diminuisce e diventa asintotico quando il numero di alberi oltrepassa una certa soglia. Inoltre il numero di alberi fa crescere linearmente il tempo richiesto per la predizione.

- `priors`: è un vettore di dimensione pari al numero delle classi che indica qual è il peso relativo associato ad un errore di classificazione per ognuna delle classi presenti. Ad esempio se il peso di una classe è 1 e il peso di una seconda classe è 10 allora ogni errore nella predizione della seconda classe è equivalente a 10 predizioni errate della prima classe. Nel nostro caso ad ogni classe è stato associato un peso pari a uno.

Per la predizione della classe di una nuova immagine basta fornire al classificatore il vettore delle feature corrispondente. Il risultato sarà un numero che identifica la classe che meglio rappresenta il campione fornito.

5.4 Algoritmi

In questa sezione vengono descritti gli algoritmi che sono stati utilizzati per ottenere la segmentazione preliminare (*region growing* e *k-means*) e per il raffinamento finale.

5.4.1 *Region growing*

L'algoritmo di *region growing* è stato sviluppato come progetto per l'esame di Elaborazione delle Immagini LM e viene utilizzato per ottenere la segmentazione preliminare di immagini con pavimenti uniformi. Una volta identificato il tipo di superficie, per la segmentazione dell'area di interesse si procede appunto con il *region growing*, una tecnica che permette di allargare una regione in base a determinati criteri di espansione. Nel *region growing* si parte da un insieme iniziale di punti, detti *"seed"*, si prosegue quindi con l'espansione della regione appendendo a ogni *seed* i pixel del vicinato con proprietà predefinite simili a quelle del *seed* (ad esempio intensità di colore che cadono in un certo intervallo) [12]. Per l'espansione della regione vengono utilizzate tre matrici di supporto:

- `gradImg`: matrice del gradiente ottenuta applicando l'operatore Scharr (simile al Sobel) per evidenziare gli *edge* dell'immagine.

- `thickEdgeImg`: matrice binaria che si ottiene applicando prima il *Canny edge detector* (con le soglie impostate a th_1=`50` e th_2=`70`, individuate in seguito ad analisi sperimentali) all'immagine trasformata in LAB, poi un'erosione (con un kernel 3x3) per rendere i bordi più spessi, in modo da eliminare eventuale rumore.

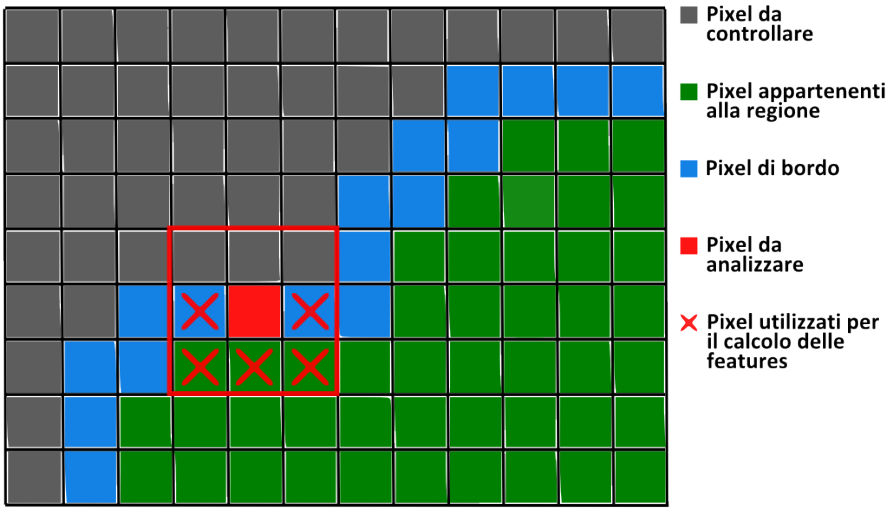


Figura 5.3: Vari tipologie di pixel presenti nella matrice `rgBorder`.

- `rgBorder`: matrice binaria che rappresenta l'area che di volta in volta viene espansa. In fase di inizializzazione l'unico pixel a 1 è quello corrispondente al tocco dell'utente.

. L'espansione della regione avviene in due fasi. Prima si effettua un'espansione grossolana trovando la componente connessa al *seed* nella matrice `thickEdgeImg`. Poi si procede con il *region growing* vero e proprio. Iterativamente si dilata la regione trovata nella fase precedente e si analizzano i pixel del bordo candidati a entrare nella regione (pixel blu in Figura 5.3). Per ognuno di essi e per ogni canale dell'immagine vengono calcolate:

- La media e la deviazione standard dell'intensità del colore nello spazio LAB nell'intorno del pixel considerato.

- La media dell'intensità del gradiente nell'intorno del pixel considerato.

Confrontando queste misure con quelle dei pixel che fanno già parte della regione (pixel contrassegnati da una x in Figura 5.3) si decide la tipologia del punto esaminato: se il punto è *interno* nella matrice `rgBorder` il pixel corrispondente viene settato a 1, se è *esterno* la matrice `rgBorder` rimane invariata, se fa parte del *bordo* anche in questo caso il pixel corrispondente

nella matrice `rgBorder` viene settato a 1. Il processo iterativo continua finché non è più possibile espandere la regione in quanto tutti i pixel sul bordo sono esterni alla regione. In Figura 5.4 si può vedere un'applicazione del metodo proposto.

(a) (b)

Figura 5.4: Applicazione del metodo *region growing* sviluppato per il progetto di Elaborazione delle Immagini LM. (a) Immagine originale. (b) Contorno trovato

5.4.2 *K-means*

L'algoritmo di *k-means*, utilizzato per la segmentazione preliminare di superfici con pavimenti non uniformi, è stato proposto dal Prof. Alessandro Bevilacqua. In questa versione i punti che si vanno a considerare sono rappresentati dagli istogrammi locali calcolati in determinate aree dell'immagine, a formare una griglia che copre l'intera immagine. È possibile specificare la dimensione (`dimW`) della finestra su cui sarà calcolato l'istogramma, il numero di *bin* di ogni istogramma (`nbin`) e il numero di cluster che si vogliono ottenere (`nCluster`). La dimensione della finestra deve essere scelta in modo da contenere informazioni significative sulla texture e la struttura dell'area da analizzare. Anche in questo caso viene utilizzato lo spazio colorimetrico LAB. L'immagine viene scomposta separando i tre piani del colore e per ognuno di essi vengono estratti gli istogrammi locali. I vettori risultanti vengono poi analizzati con il *k-means* (utilizzando l'implementazione disponibile in OpenCV) per trovare i cluster. In Figura 5.5 si può vedere un risultato dell'algoritmo.

(a) (b) (c)

Figura 5.5: Applicazione del metodo *k-means* proposto dal professore Alessandro Bevilacqua. (a) Immagine originale. (b) Cluster trovati. (c) Visualizzazione dei cluster sopra all'immagine originale

5.4.3 Raffinamento del bordo trovato

Una volta ottenuta la segmentazione preliminare in base al tipo della superficie, attraverso uno dei metodi descritti sopra, si può procedere con la rifinitura del bordo trovato. Per fare ciò viene eseguito un campionamento del bordo, effettuando poi un'analisi dell'area (*patch*) nell'intorno del punto individuato.
Nel caso di superfici con texture uniforme viene applicato come primo metodo per il confronto il *Canny edge detector*, in quanto questo algoritmo dà una buona localizzazione dei bordi. Se con il *Canny* si trovato un solo contorno nella *patch* sotto esame (Figura 5.6e) significa che è presente un bordo "forte". In questo caso per confrontare i due risultati si calcola la distanza di Hausdorff che permette di misurare l'entità dello scostamento. Se la distanza di Hausdorff è al di sotto di una certa soglia il contorno finale è costituito dal contorno individuato dal *Canny* (Figura 5.6f), in quanto ritenuto più preciso del nostro algoritmo di segmentazione preliminare. Altrimenti, se con il *Canny* vengono individuati più contorni si procede applicando la trasformata di Hough per verificare la presenza di rette: se viene trovata almeno una retta, al contorno finale viene aggiunto il bordo del *Canny* che la contiene, se non vengono trovate rette si procede applicando l'algoritmo di *watershed*.

Nel caso di superfici con texture non uniformi ad ogni *patch* viene prima applicato il *mean shift* (Figura 5.8d) per cercare di "livellare" il pattern e rimuovere quindi un po' di variabilità. Viene applicato anche in questo caso il *Canny* (Figura 5.8e) per verificare l'entità del livellamento del metodo prece-

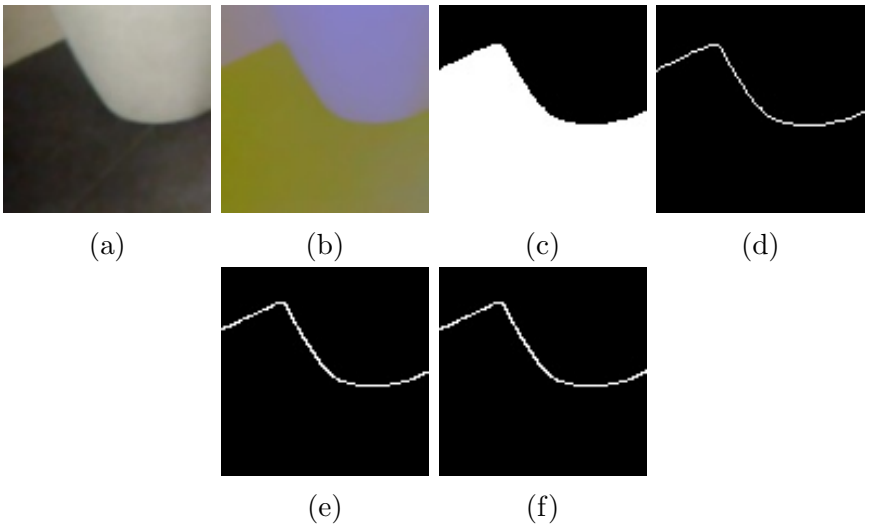


Figura 5.6: Esempio di raffinamento del contorno in presenza di un *edge* individuato dal *Canny*. (a) Patch originale. (b) *Patch* nello spazio colorimetrico LAB. (c) Maschera trovata dopo la segmentazione preliminare. (d) Contorno trovato dopo la segmentazione preliminare. (e) Applicazione del *Canny* all'immagine in LAB. (f) Contorno finale.

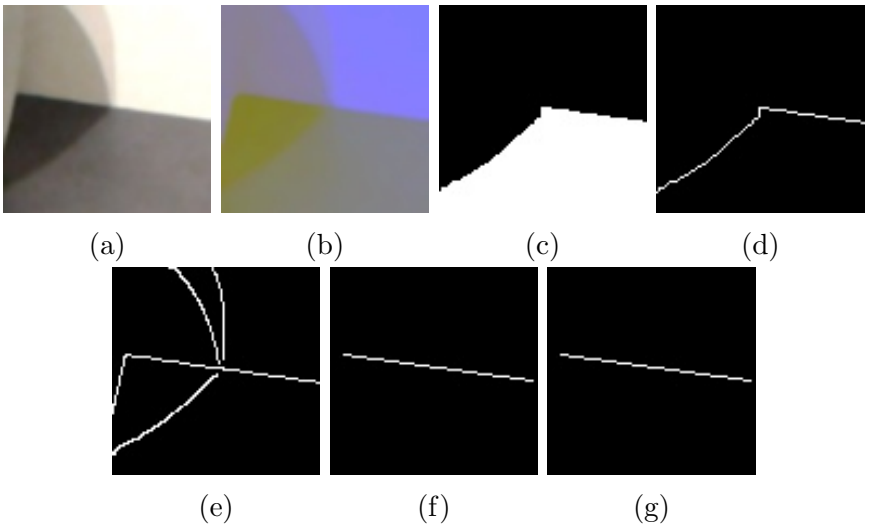


Figura 5.7: Esempio di raffinamento del contorno in presenza di più *edge* individuati dal *Canny*. (a) *Patch* originale. (b) *Patch* nello spazio colorimetrico LAB. (c) Maschera trovata dopo la segmentazione preliminare. (d) Contorno trovato dopo la segmentazione preliminare. (e) Applicazione del *Canny* all'immagine in LAB. (f) Retta individuata dalla trasformata di Hough. (g) Contorno finale.

dente: se il numero di contorni è elevato significa che la superficie analizzata è ancora altamente disomogenea, in questo caso si procede con l'algoritmo di *watershed* (Figura 5.8f). Se il numero di contorni è sotto ad una certa soglia si può procedere, come nel caso di superfici omogenee, ricercando la presenza rette con la trasformata di Hough.
In entrambi i casi, se la *patch* si trova sul bordo dell'immagine e il *Canny* non rivela la presenza di *edge*, non si eseguono ulteriori elaborazioni ed al contorno finale viene aggiunto il bordo corrispondente.

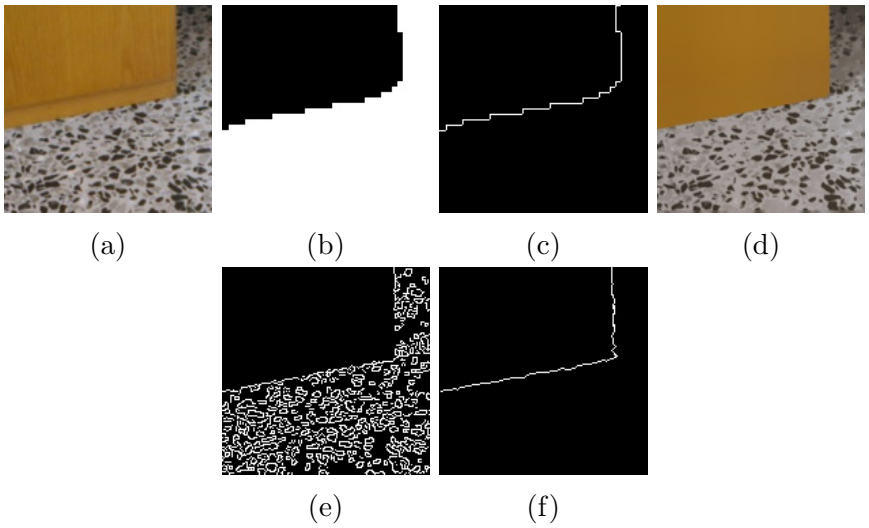


Figura 5.8: Esempio di raffinamento di una superficie con texture non uniforme. (a) *Patch* originale. (b) Maschera trovata dopo la segmentazione preliminare. (c) Contorno trovato dopo la segmentazione preliminare. (d) Applicazione del *mean shift*. (e) Applicazione del *Canny*. (f) Contorno trovato con il metodo *watershed*, corrisponde al contorno finale raffinato.

Capitolo 6

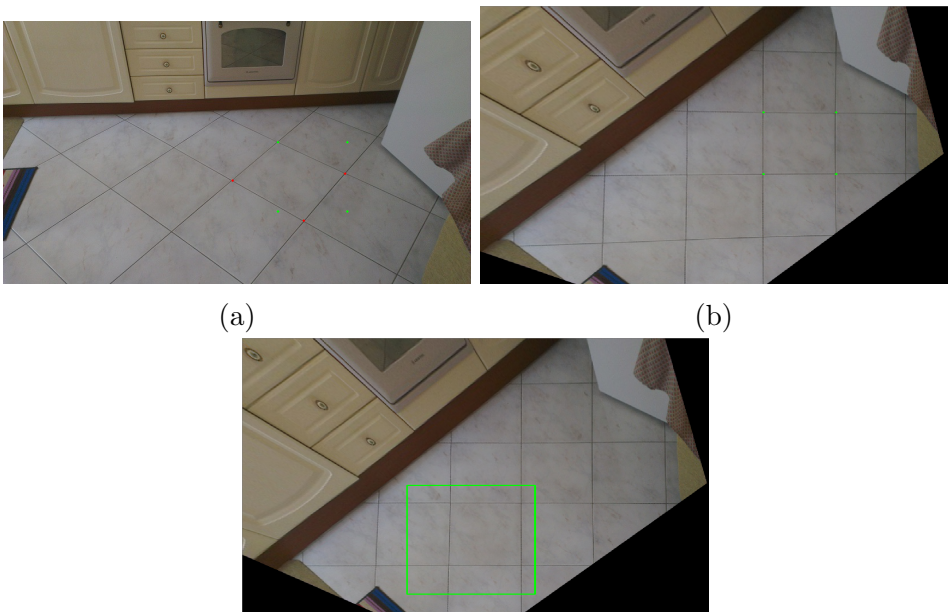
Sperimentazione

In questo capitolo viene descritto in che modo è stata effettuata la correzione prospettica per l'estrazione dei campioni per l'addestramento del classificatore e come sono stati determinati i parametri degli algoritmi utilizzati.

6.1 Correzione prospettica

Per estrarre delle *patch* che non avessero distorsioni prospettiche è stato realizzato un semplice programma in C++ che consente di caricare un'immagine, applicare la correzione prospettica al piano che contiene il pavimento ed estrarre le *patch* della dimensione desiderata. Per l'applicazione della correzione prospettica devono essere selezionati quattro punti coplanari (durante l'inserimento se vengono commessi errori è possibile eliminare l'ultimo punto inserito con il tasto "-") preferibilmente a formare un quadrato o un rettangolo. Se i punti selezionati corrispondono ai vertici di un quadrato (tasto "q", puntini rossi in Figura 6.1a), le coordinate dei punti nell'immagine trasformata corrisponderanno ai vertici di un quadrato con lato pari alla media dei lati del quadrato iniziale e orientato parallelamente agli assi (puntini verdi in Figura 6.1a). Se i punti selezionati corrispondono ad un rettangolo (tasto "r"), le coordinate dei punti nell'immagine trasformata corrisponderanno ai vertici di un rettangolo con lato maggiore uguale alla media dei lati maggiori del rettangolo originale (lato minore calcolato nello stesso modo) sempre

orientato parallelamente agli assi.
Dopo aver selezionato i punti e averne indicato la geometria, viene prima ottenuta la matrice 3×3 della trasformazione con la funzione OpenCV `Mat trasfMatrix = getPerspectiveTransform(inputPoint, projPoint)`, che viene poi applicata all'immagine con la funzione `void warpPerspective (imgDistorted, imgOk, trasfMatrix)` per rimuovere la distorsione prospettica (Figura 6.1b). A questo punto è possibile selezionare (attraverso una sagoma che compare sull'immagine, Figura 6.1c) e salvare le aree della superficie di interesse da utilizzare per l'addestramento semplicemente con un click del mouse.



(a) (b)

(c)

Figura 6.1: Screenshot del programma C++ realizzato. (a) I punti rossi rappresentano un elemento con distorsione prospettica, quelli verdi indicano come devono essere trasformati i punti rossi. (b) Immagine ottenuta rimuovendo la distorsione prospettica, sono evidenziati i punti trasformati. (c) Finestra che permette l'individuazione e il salvataggio delle patch desiderate.

6.2 *Tuning* dei parametri del classificatore

Per il training sono stati usati 285 campioni (94 della classe *UNI*, 101 della classe *PIRR*, 25 della classe *PCOMP*, 65 della classe *REG*). Per il test sono stati utilizzati 60 campioni (13 della classe *UNI*, 23 della classe *PIRR*, 10 della classe *PCOMP*, 4 della classe *REG*). Per valutare quali siano i valori migliori per i parametri `max_depth` e `max_trees_in_forest` è stata effettuata un'analisi granulometrica che ha prodotto i seguenti risultati. Mantenendo fisso `max_trees_in_forest` = 100 è stato fatto variare il parametro `max_depth` da 1 a 50 ottenendo i seguenti errori nella classificazione dei campioni del *test set* (Figura 6.2).

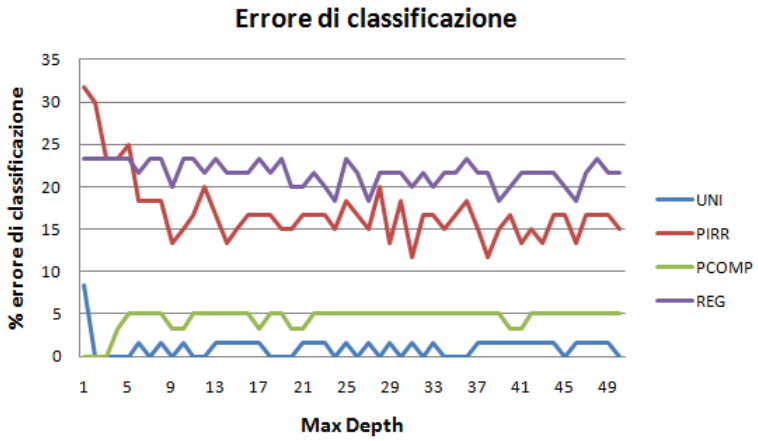


Figura 6.2: Il grafico mostra l'andamento dell'errore di classificazione al variare dell'altezza massima consentita ad ogni albero per ogni clase considerata.

Mantenendo fisso `max_depth` = 25 e variando `max_trees_in_forest` tra 1 e 150 si ottengono i seguenti errori di classificazione (Figura 6.3).

Come si può vedere dai grafici riportati solamente i campioni della classe *UNI* vengono riconosciuti correttamente nella quasi totalità dei casi. Analizzando i risultati della classificazione per i singoli campioni delle altre classi, si è notato che molti campioni delle classi *PIRR* e *REG* venivano attribuite erroneamente alla classe *PCOMP*. Presumibilmente questo è dovuto al fatto che le *feature* di Haralick non sono in grado di discriminare sufficientemente tra le varie texture considerate. Visto che per la rifinitura del bordo del bordo

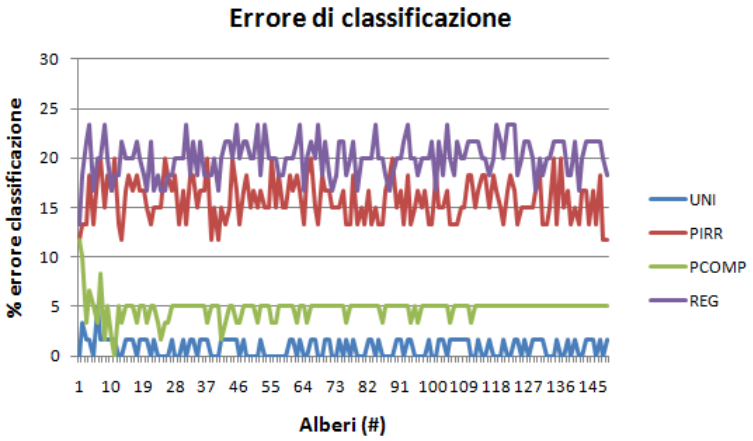


Figura 6.3: Il grafico mostra l'andamento dell'errore di classificazione al variare del numero di alberi presenti nella foresta.

delle classi *PIRR*, *PCOMP* e *REG* verrà trattata con lo stesso metodo si è deciso di inglobare queste classi in una più generale che chiameremo *NONUNI*. Per ulteriori sviluppi futuri, per catturare meglio le texture delle classi *PIRR*, *PCOMP* e *REG* occorrerà sicuramente utilizzare per la classificazione delle *feature* diverse, come ad esempio gli *edge* strutturati, i coefficienti *wavelet* o i filtri di Gabor orientati.
È stata quindi effettuata l'analisi granulometrica utilizzando solamente le classi *UNI* e *NONUNI*. I risultati sono i seguenti (Figura 6.5 e Figura 6.6). Si può notare che gli errori di classificazione sono diminuiti significativamente e la quasi totalità dei campioni viene riconosciuta correttamente. Analizzando i campioni erroneamente classificati si è visto che corrispondono a *patch* che sono difficili da classificare anche a occhio nudo, in quando presentano lievi variazioni della texture (Figura 6.4).

6.3 Granulometria dei parametri del *region growing*

Per verificare il comportamento dell'algoritmo di *region growing* al variare dei parametri principali è stata effettuata un'anasili granulometrica. I parametri che regolano la crescita della regione sono:

Figura 6.4: Esempio di *patch* che viene erroneamente classificata come appartenente alla classe *UNI*: le variazioni sono quasi impercettibili.

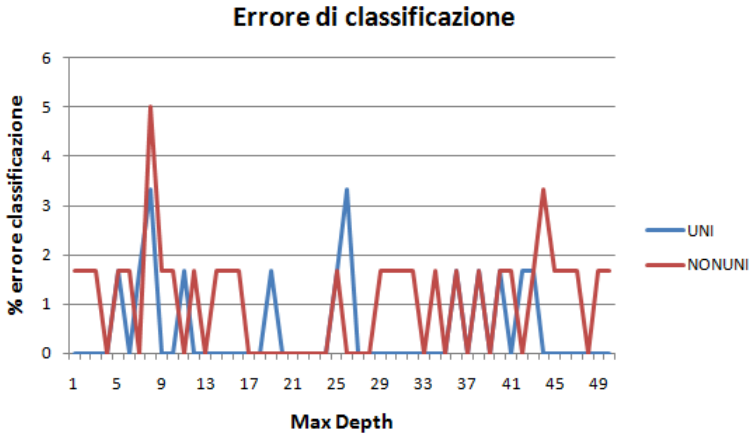


Figura 6.5: Il grafico mostra l'andamento dell'errore di classificazione al variare dell'altezza massima consentita ad ogni albero per ogni clase considerata.

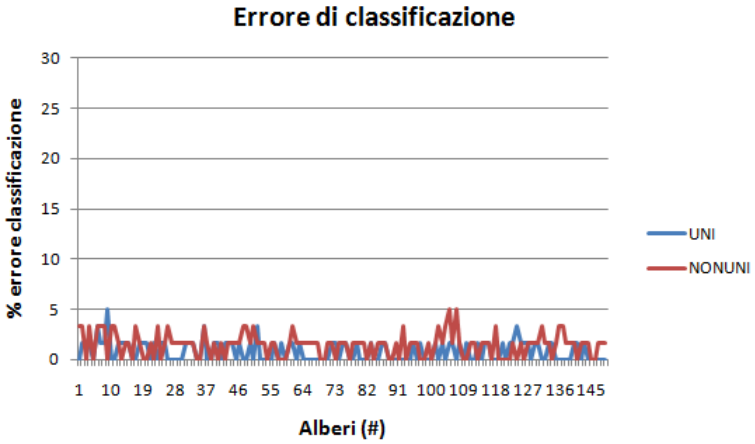
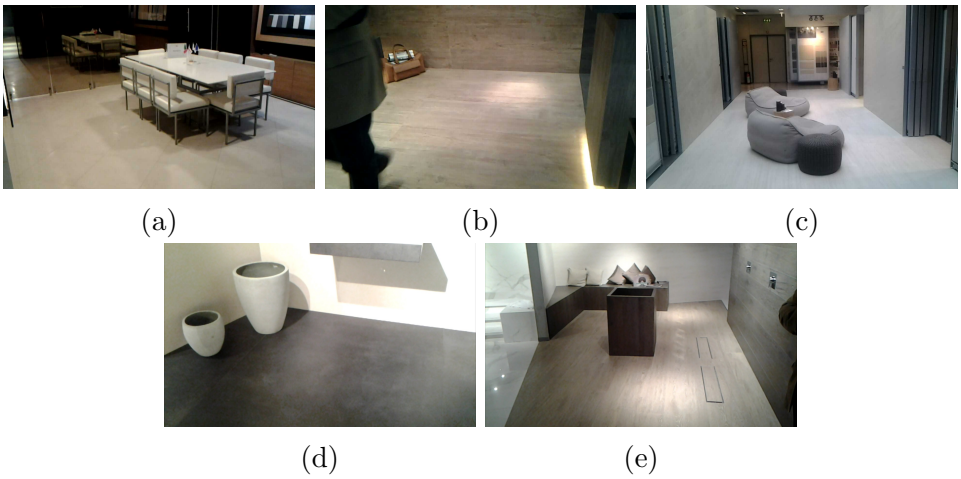


Figura 6.6: Il grafico mostra l'andamento dell'errore di classificazione al variare del numero di alberi presenti nella foresta.

- Differenza tra la media dell'intensità del colore nell'intorno del pixel considerato e la media dei pixel già appartenenti alla regione.
- Differenza tra la deviazione standard dell'intensità del colore nell'intorno del pixel considerato e la deviazione standard dei pixel già appartenenti alla regione.
- Differenza tra la media dell'intensità del gradiente del pixel considerato e il gradiente dei pixel già appartenenti alla regione.

Queste differenze vengono poi calcolate in percentuale rispetto allo scostamento massimo possibile sui tre canali (ad esempio 765 per la differenza della media, visto che su ogni canale lo scostamento massimo è 255).



(a) (b) (c)

(d) (e)

Figura 6.7: Immagini utilizzate per effettuare l'analisi granulometrica. (a) f3. (b) f5. (c) f8. (d) f9. (e) f12.

Per l'analisi granulometrica sono state usate le immagini in Figura 6.7, ottenendo i seguenti risultati. Mantenendo costanti i valori delle soglie della deviazione standard (3.5%) e del gradiente (1%), in Figura 6.8 si può vedere come vari l'errore di segmentazione all'aumentare della soglia della media. Un errore maggiore di 100 significa che la regione si è espansa troppo e ha superato la dimensione della *ground truth*. Con una soglia inferiore all'1.9% dello scostamento massimo è possibile ottenere un'errore ridotto per la maggior parte delle immagini.

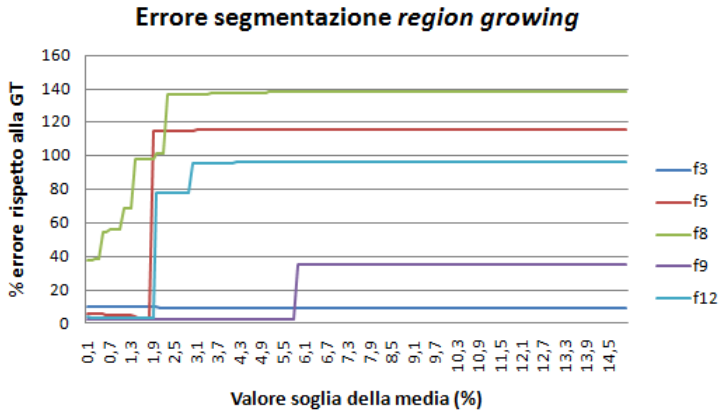


Figura 6.8: Il grafico mostra l'andamento dell'errore di segmentazione del *region growing* al variare della soglia della media.

Mantenendo costanti i valori della soglia della media (1%) e del gradiente (1%), in Figura 6.9 si può vedere l'andamento dell'errore di segmentazione al variare della soglia della deviazione standard. Anche in questo caso con un soglia inferiore all'1.9% dello scostamento massimo è possibile ottenere un'errore ridotto per la maggior parte delle immagini.

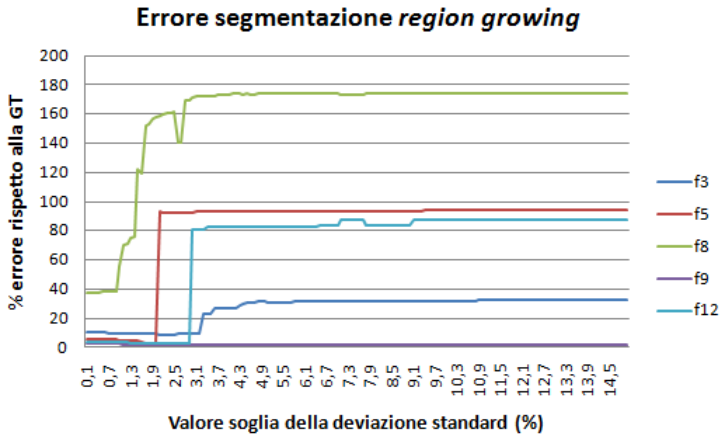


Figura 6.9: Il grafico mostra l'andamento dell'errore di segmentazione del *region growing* al variare della soglia della deviazione standard.

Mantenendo costanti i valori della soglia della media (1%) e della deviazione standard (3.5%) si può vedere come vari l'errore di segmentazione all'aumentare della soglia del gradiente (Figura 6.10). Come risultato si ha

che l'errore di segmentazione è pressoché costante al variare della soglia del gradiente.

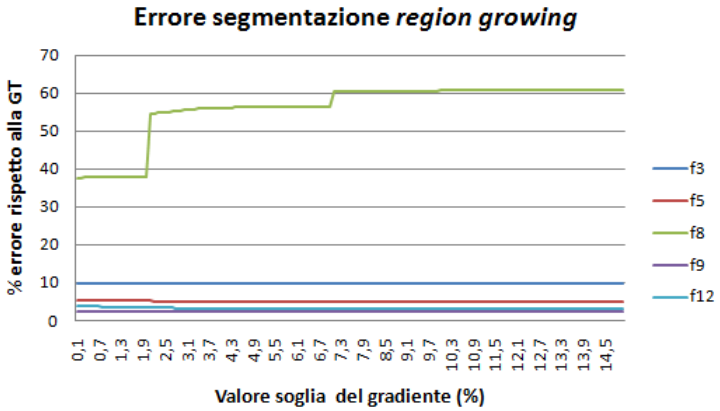


Figura 6.10: Il grafico mostra l'andamento dell'errore di segmentazione del *region growing* al variare della soglia del gradiente.

Come si può notare dai grafici l'immagine *f8* è quella che si discosta maggiormente dalle altre e presenta l'errore di segmentazione più elevato. Analizzando nel dettaglio l'immagine, l'errore è dovuto al fatto che i divani presenti nella scena hanno caratteristiche molto simili al pavimento. In particolare nella zona superiore la discontinuità tra il pavimento e il divano è minima, per cui l'algoritmo di *region growing* si espande all'interno dei divani non riuscendo a identificare la zona di confine. Questo si verifica anche per valori bassi delle soglie dei parametri utilizzati. Per ovviare a questo fenomeno occorrerebbe analizzare più nel dettaglio la variazione del parametro deviazione standard sul confine.

Capitolo 7

Risultati Sperimentali

In questo capitolo vengono descritti i risultati ottenuti dall'applicazione del metodo di segmentazione proposto su immagini contenenti diverse tipologie di pavimento.

7.1 Feature di Haralick

L'estrazione delle feature di Haralick e la classificazione del pavimento sono state testate anche su Android. Come piattaforma di test ho utilizzato il mio smartphone HTC One X con processore quad-core NVIDIA Tegra 3 da 1.5 GHz, 1 GB di RAM, fotocamera posteriore da 8 megapixel e sensori inerziali Panasonic. Lo sviluppo degli algoritmi di segmentazione è avvenuto principalmente su PC. Oltre alla libreria OpenCV è stato utilizzato il *framework* JNI che permette di incorporare nell'applicazione (scritta in Java) codice nativo, nel nostro caso C++. Per l'uso delle librerie OpenCV su un dispositivo Android è necessario installare l'applicazione *OpenCV Manager* presente nel *Play Store*.
Per l'estrazione delle *GLCM* è il calcolo delle feature di Haralick sono risultati necessari circa 3 secondi. Per caricare il classificatore con i Random Tree sono necessari 680 millisecondi. Per la predizione della classe del pavimento selezionato sono necessari 0,5 millisecondi.

7.2 Classificatore

Dalla fase di sperimentazione risulta che ci sono molte coppie di variabili (`max_depth`,`max_tree_in_forest`) che permettono di avere il 100% di campioni classificati correttamente. La coppia che permette di avere minor costo computazionale, in termini di memoria occupata e tempo impiegato, è quella con il minor numero di alberi nella foresta e la minima profondità massima che è risultata essere (`max_depth`=3,`max_tree_in_forest`=25). L'importanza delle variabili ottenuta con questi valori è mostrata nel seguente grafico (Figura 7.1). A differenza di altri tipi di classificatori i Random Tree utiliz-

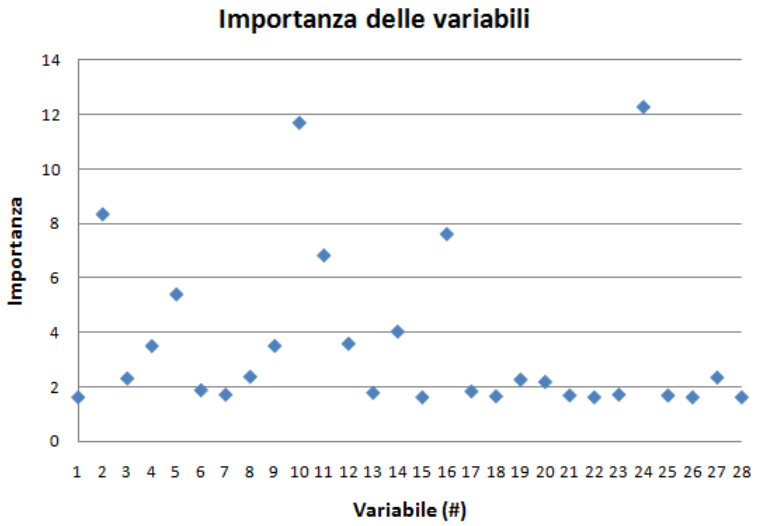


Figura 7.1: Importanza delle variabili con `max_depth`=3 e `max_tree_in_forest`=25

zano una *cross-validazione* interna per la scelta delle variabili durante la fase di *training*. Per questo, come viene spiegato in [44] da Breiman, ideatore dei Random Tree, non c'è bisogno di effettuare una selezione preliminare delle *feature* utilizzate.

7.3 Raffinamento

Il test del metodo di segmentazione proposto è stato effettuato su un PC con processore Intel Core i7 a 1.60GHz, 4 GB di RAM e sistema operativo

Windows 7 a 64 bit. Nel seguito vengono mostrati i risultati ottenuti su diverse immagini di interni.

In Figura 7.2 si può vedere l'applicazione del metodo proposto ad una immagine con pavimento uniforme. Il risultato della segmentazione preliminare con il *region growing* è in Figura 7.2c: da notare come l'ombra prodotta dal vaso non venga riconosciuta correttamente come appartenente alla regione di interesse, fatto evidenziato anche in Figura 7.2e facendo la differenza con la *groud truth* (Figura 7.2b). Grazie al processo di raffinamento è stato possibile rimediare all'errore della fase preliminare e ottenere la maschera in Figura 7.2f. Sovrapponendola all'immagine originale (Figura 7.2h) la segmentazione risulta visivamente accurata e precisa: i pixel non riconosciuti correttamente (evidenziati facendo la differenza tra la segmentazione finale e la *ground truth*, Figura 7.2g) non sono percebili. Per l'individuazione della segmentazione preliminare sono stati necessari 3278 millisecondi, per il raffinamento 242 millisecondi su un'immagine di 1280×752 pixel.

In Figura 7.3 si può vedere un altro esempio di applicazione del metodo proposto su un pavimento uniforme. In questo caso con il *region growing* (Figura 7.3c) non viene riconosciuta correttamente la zona dell'immagine in basso a destra in cui le condizioni luminose cambiano drasticamente (evidenziata in Figura 7.3e). Con il processo di raffinazione è stato possibile risolvere questo errore e definire in modo più preciso il resto del contorno, come si può vedere dalla segmentazione finale (Figura 7.3f) e dalla differenza di questa con la *ground truth* (Figura 7.3g). La sovrapposizione della maschera finale all'immagine originale (Figura 7.3h) risulta precisa anche nella zona in basso a sinistra in cui c'era del rumore dovuto al movimento della persona presente sulla scena. Per l'individuazione della segmentazione preliminare sono stati necessari 3587.11 millisecondi, per il raffinamento 733 millisecondi su un'immagine di 1280×752 pixel.

In Figura 7.4 il metodo proposto è stato applicato ad un'immagine con pavimento uniforme. In questo caso la segmentazione preliminare con il *region*

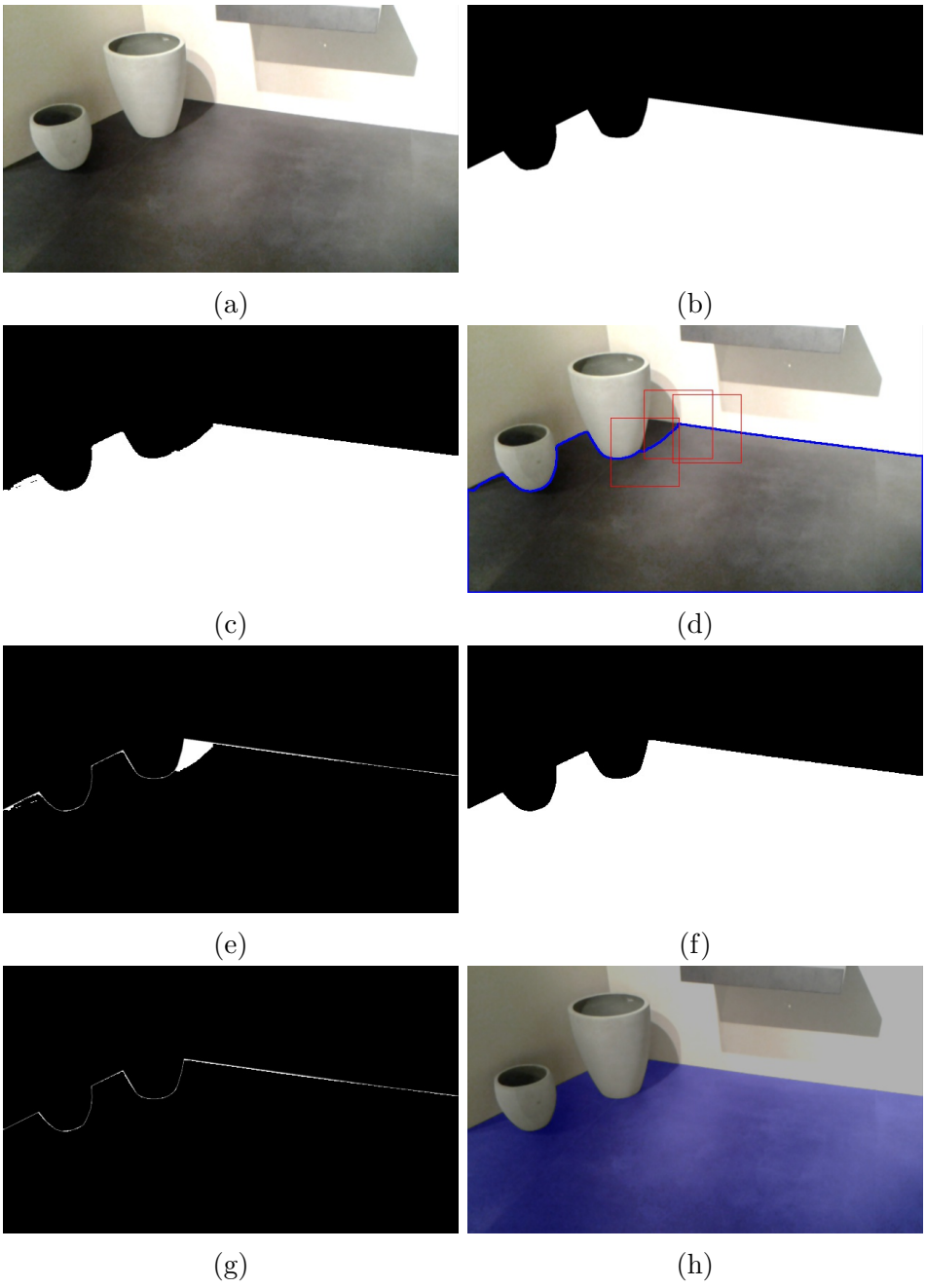


Figura 7.2: Applicazione del metodo proposto. (a) Immagine originale (*f9*). (b) *Ground truth*. (c) Segmentazione preliminare. (d) Visualizzazione del contorno preliminare (in blu) e rilevamento delle anomalie al bordo (quadrati rossi). (e) Differenza tra la segmentazione preliminare e la *ground truth*. (f) Maschera ottenuta con il processo di raffinamento. (g) Differenza tra la segmentazione finale e la *ground truth*. (d) Sovrapposizione della segmentazione ottenuta all'immagine originale.

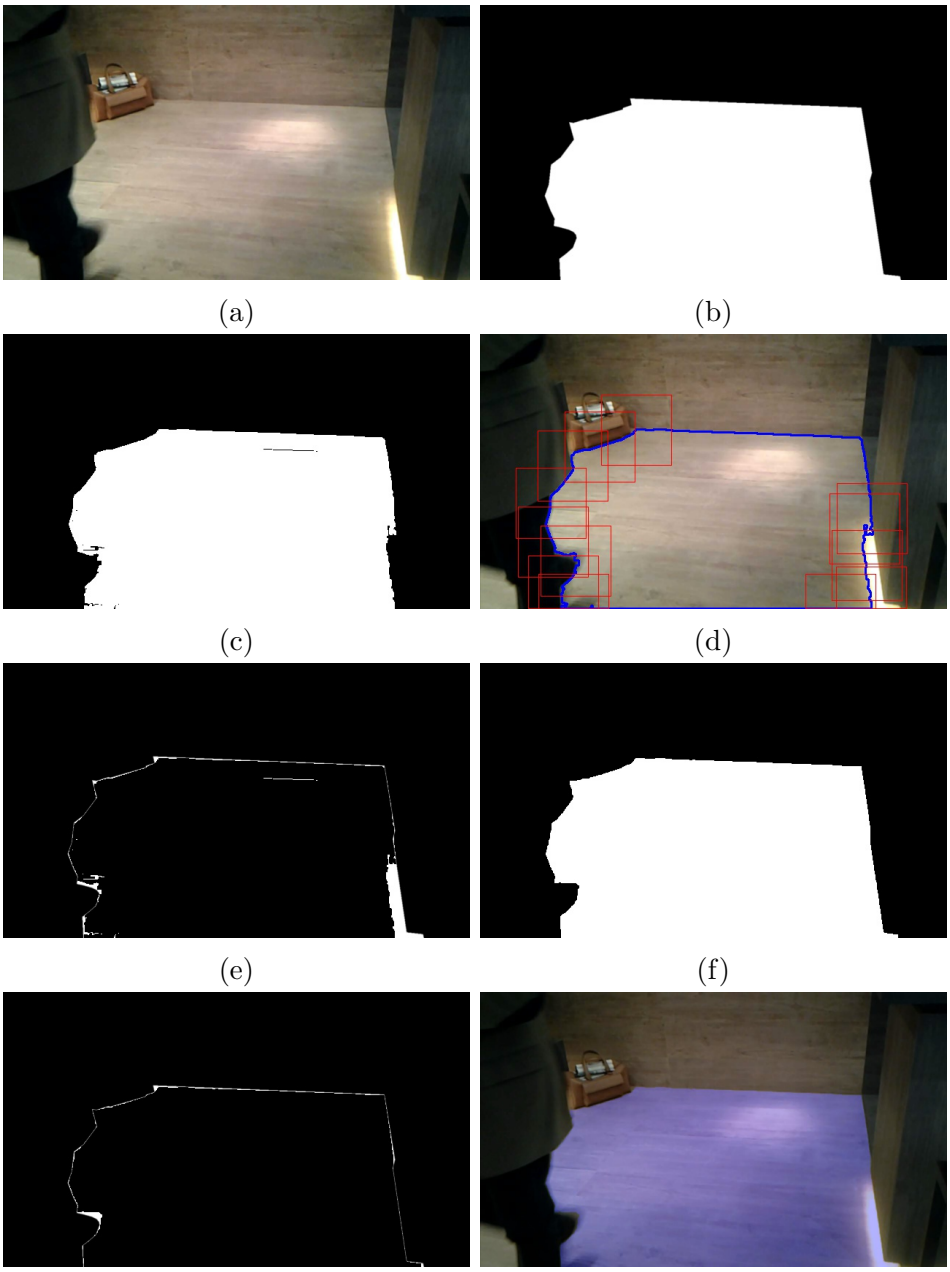
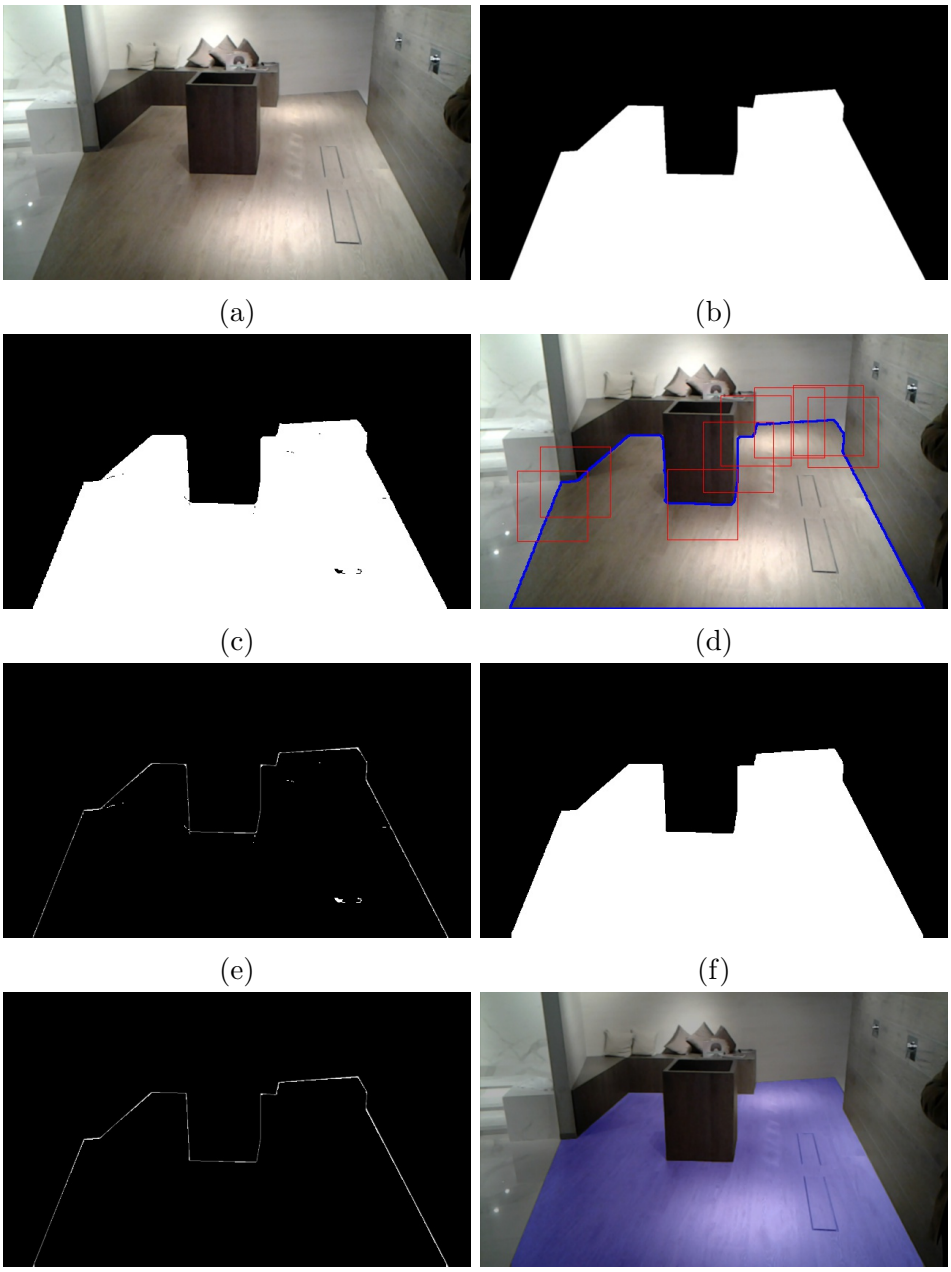


Figura 7.3: Applicazione del metodo proposto. (a) Immagine originale (*f5*). (b) *Ground truth*. (c) Segmentazione preliminare. (d) Risultato della segmentazione preliminare (contorno blu) e rilevamento delle anomalie al bordo (quadrati rossi). (e) Differenza tra la segmentazione preliminare e la *ground truth*. (f) Maschera ottenuta con il processo di raffinamento. (g) Differenza tra la segmentazione finale e la *ground truth*. (d) Sovrapposizione della segmentazione ottenuta all'immagine originale.



(a) (b) (c) (d) (e) (f) (g) (h)

Figura 7.4: Applicazione del metodo proposto. (a) Immagine originale (*f12*). (b) *Ground truth*. (c) Segmentazione preliminare. (d) Risultato della segmentazione preliminare (contorno blu) e rilevamento delle anomalie al bordo (quadrati rossi). (e) Differenza tra la segmentazione preliminare e la *ground truth*. (f) Maschera ottenuta con il processo di raffinamento. (g) Differenza tra la segmentazione finale e la *ground truth*. (d) Sovrapposizione della segmentazione ottenuta all'immagine originale.

growing (Figura 7.4c) risulta piuttosto precisa senza evidenti errori (Figura 7.4e). Con il processo di raffinazione viene "livellato" ed eliminato il rumore presente nel bordo trovato dalla fase precedente (Figura 7.4f). Dalla sovrapposizione con l'immagine originale (Figura 7.4h) la segmentazione risulta accurata e precisa. Per l'individuazione della segmentazione preliminare sono stati necessari 3626 millisecondi, per il raffinamento 565 millisecondi su un'immagine di 1280×752 pixel.

In Figura 7.5 il metodo proposto è stato applicato ad un'immagine con pavimento uniforme in cui però si possono notare diversi tipi di ombre: nella parte centrale della superficie sono presenti ombre *soft*, mentre sotto al tavolo ombre *hard*. Le ombre *soft* vengono riconosciute correttamente fin dalla segmentazione preliminare (Figura 7.5d). Per quanto riguarda le ombre *hard*, anche con il processo di raffinamento non è stato possibile ottenere ulteriori miglioramenti (la segmentazione è resa più difficile anche dalla presenza delle numerose sedie). Visivamente il risultato finale è comunque abbastanza accettabile. Per l'individuazione della segmentazione preliminare sono stati necessari 3629 millisecondi, per il raffinamento 1165.81 millisecondi su un'immagine di 1280×752 pixel.

In Figura 7.6 si può vedere come l'errore di segmentazione diminuisca dopo la fase di raffinamento. Nello specifico si può osservare come le figure *f9* e *f5* presentino il miglioramento maggiore dovuto al fatto che l'algoritmo di raffinamento è riuscito ad individuare ombre e condizioni di illuminazione non rilevate dalla segmentazione preliminare. La segmentazione preliminare dell'immagine *f12* presenta un contorno aderente alla *ground truth* ma con delle irregolarità, successivamente appianate dalla fase di rifinitura. Infine l'applicazione dell'algoritmo di raffinazione nell'immagine (*f3*) non ha prodotto ulteriori miglioramenti in quanto non è riuscito ad individuare correttamente le ombre *hard* presenti sotto al tavolo.

In Figura 7.7 il metodo proposto è stato applicato ad un'immagine con pavimento non uniforme. La segmentazione preliminare con il *k-means* ha

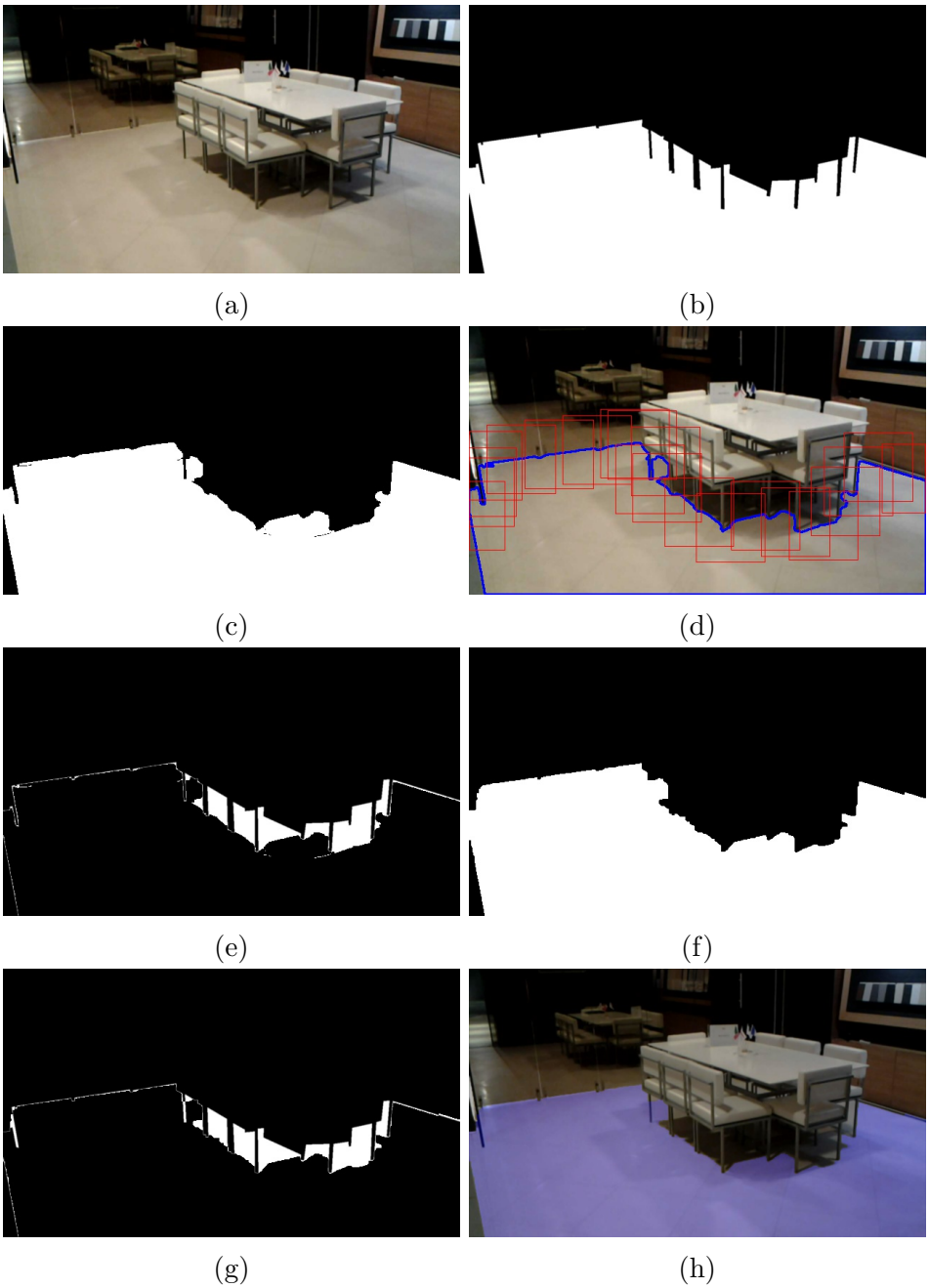


Figura 7.5: Applicazione del metodo proposto. (a) Immagine originale (*f3*). (b) *Ground truth*. (c) Segmentazione preliminare. (d) Risultato della segmentazione preliminare (contorno blu) e rilevamento delle anomalie al bordo (quadrati rossi). (e) Differenza tra la segmentazione preliminare e la *ground truth*. (f) Maschera ottenuta con il processo di raffinamento. (g) Differenza tra la segmentazione finale e la *ground truth*. (d) Sovrapposizione della segmentazione ottenuta all'immagine originale.

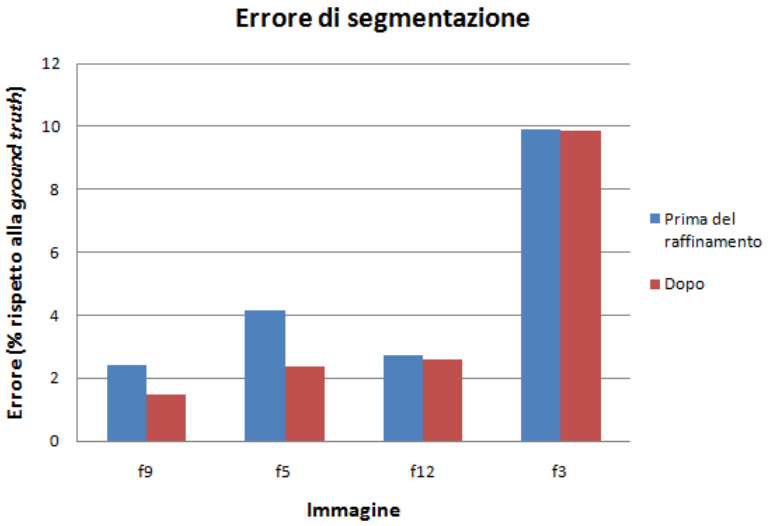


Figura 7.6: Il grafico mostra come in seguito al processo di rifinitura sia diminuito l'errore di segmetazione rispetto alla *ground truth* per 4 immagini con pavimenti classificati come uniformi.

prodotto il risultato in Figura 7.7c. Con il processo di raffinazione viene ottenuta la maschera in Figura 7.7f: come si può vedere facendo la differenza con la *ground truth* (Figura 7.7g) viene riconosciuto erroneamente come appartenente alla regione anche il battiscopa. Questo è dovuto al fatto che per l'analisi delle *patch* lungo quel bordo, data l'alta disomogeneità del pavimento, viene utilizzato il metodo *watershed* che dipende fortemente dalla scelta dei *seed* iniziali (settati in modo automatico in base alla geometria della maschera trovata nella fase preliminare). La sovrapposizione della segmentazione individuata con l'immagine originale (Figura 7.7h) appare però piuttosto naturale. Per l'individuazione della segmentazione preliminare sono stati necessari 11061 millisecondi: l'elevato costo computazionale è dovuto all'attuale implementazione del *k-means* in cui ogni livello del colore dell'immagine viene elaborato in modo seriale. Si potrebbe diminuire il costo computazionale attraverso un'implementazione parallela dell'algoritmo. Il tempo necessario per il raffinamento è di 18770 millisecondi: in questo caso il costo computazionale è più elevato rispetto ai casi con pavimento uniforme in quanto la porzione di contorno da analizzare è maggiore (numero di quadrati rossi in Figura 7.7d) su un'immagine di 2048×1536 pixel.

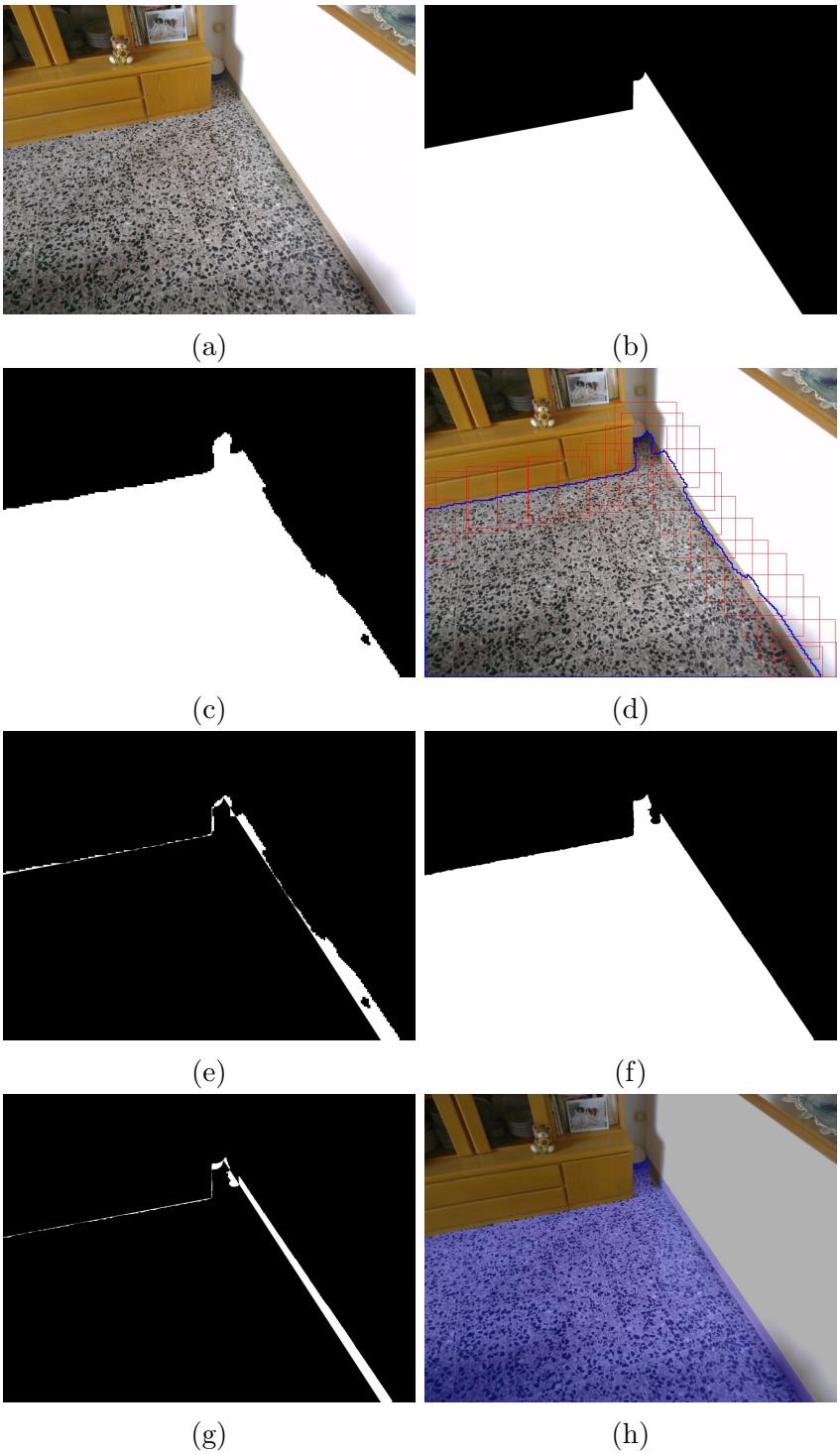


Figura 7.7: Applicazione del metodo proposto. (a) Immagine originale (*p27*). (b) *Ground truth*. (c) Segmentazione preliminare. (d) Risultato della segmentazione preliminare (contorno blu) e rilevamento delle anomalie al bordo (quadrati rossi). (e) Differenza tra la segmentazione preliminare e la *ground truth*. (f) Maschera ottenuta con il processo di raffinamento. (g) Differenza tra la segmentazione finale e la *ground truth*. (d) Sovrapposizione della segmentazione ottenuta all'immagine originale.

In Figura 7.8 è stata analizzata un'altra immagine con pavimento non uniforme. Applicando il *k-means* per la segmentazione preliminare si ottiene la maschera in Figura 7.8c. Come si può notare dalla differenza con la *ground truth* la zona più difficile da riconoscere correttamente è lo spazio presente tra i divani: essendo un'area piccola ed in ombra le caratteristiche del colore cambiano in modo significativo rispetto al resto del pavimento. Anche attraverso il processo di raffinazione non è stato possibile segmentare in modo più preciso quella zona (Figura 7.8f). Per l'individuazione della segmentazione preliminare sono stati necessari 11100 millisecondi, per il raffinamento 8059 millisecondi su un'immagine di 2048×1536 pixel.

In Figura 7.9 il metodo di segmentazione è stato applicato ad una immagine con pavimento non uniforme. La scena è particolarmente complessa in quanto ci sono vari elementi (porte, appendiabiti) che hanno un colore simile al pavimento. Come si può vedere in Figura 7.9d, con la segmentazione preliminare non è stata riconosciuta correttamente la parte superiore centrale del pavimento e anche con il raffinamento non è stato possibile ottenere dei miglioramenti, in quanto non sono ancora stati implementati meccanismi opportuni per includere nella regione di interesse aree simili. Da notare come sia stato però raffinato il bordo destro del pavimento, in particolare nell'area del termosifone (Figura 7.9h).

Infine in Figura 7.10 il metodo proposto è stato applicato ad una immagine con pavimento non uniforme. Come si può vedere in Figura 7.10e, la segmentazione preliminare nella parte sinistra del pavimento è molto minore della *ground truth*, il che porta degli errori anche in fase di rifinitura: infatti non si riesce a riconoscere correttamente la zona del pavimento sulla sinistra del comodino. Inoltre il bordo finale (Figura 7.10f) rimane aperto in diversi punti il che porta ad avere una maschera finale grande come tutta l'immagine (Figura 7.10g). Anche se non ancora implementati si stanno studiando meccanismi per identificare e chiudere le zone in cui il bordo è aperto.

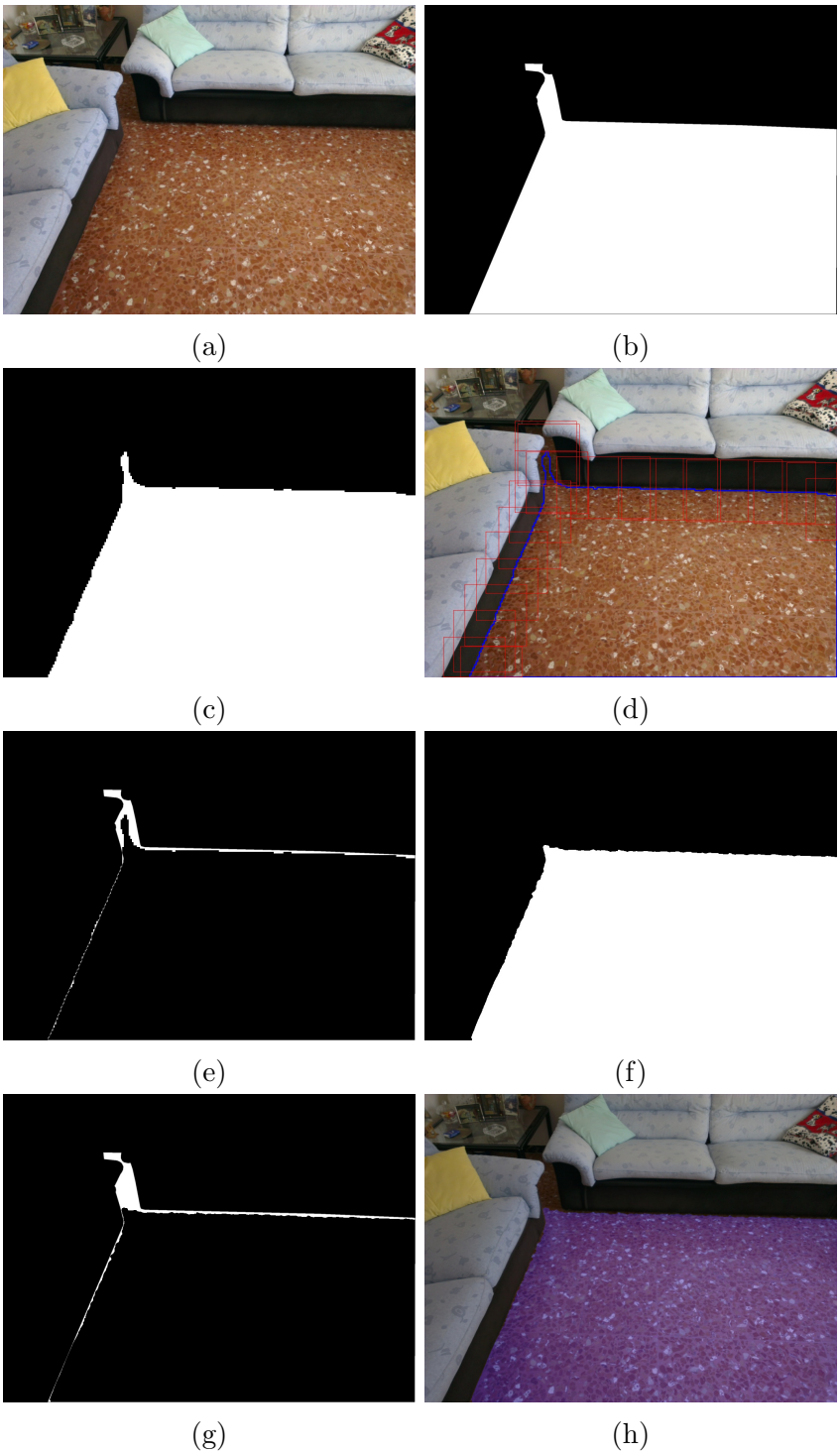


Figura 7.8: Applicazione del metodo proposto. (a) Immagine originale (*p34*). (b) *Ground truth*. (c) Segmentazione preliminare. (d) Risultato della segmentazione preliminare (contorno blu) e rilevamento delle anomalie al bordo (quadrati rossi). (e) Differenza tra la segmentazione preliminare e la *ground truth*. (f) Maschera ottenuta con il processo di raffinamento. (g) Differenza tra la segmentazione finale e la *ground truth*. (d) Sovrapposizione della segmentazione ottenuta all'immagine originale.

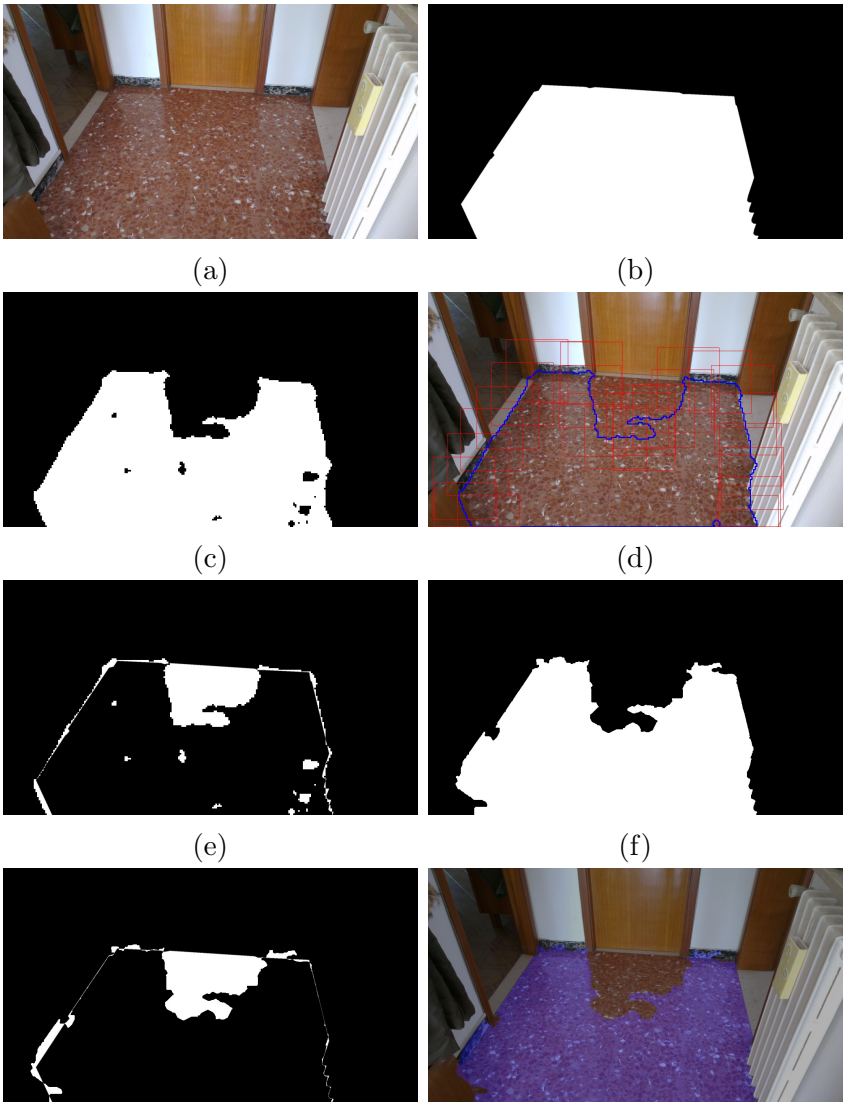


Figura 7.9: Applicazione del metodo proposto. (a) Immagine originale (*p28*). (b) *Ground truth*. (c) Segmentazione preliminare. (d) Risultato della segmentazione preliminare (contorno blu) e rilevamento delle anomalie al bordo (quadrati rossi). (e) Differenza tra la segmentazione preliminare e la *ground truth*. (f) Maschera ottenuta con il processo di raffinamento. (g) Differenza tra la segmentazione finale e la *ground truth*. (d) Sovrapposizione della segmentazione ottenuta all'immagine originale.

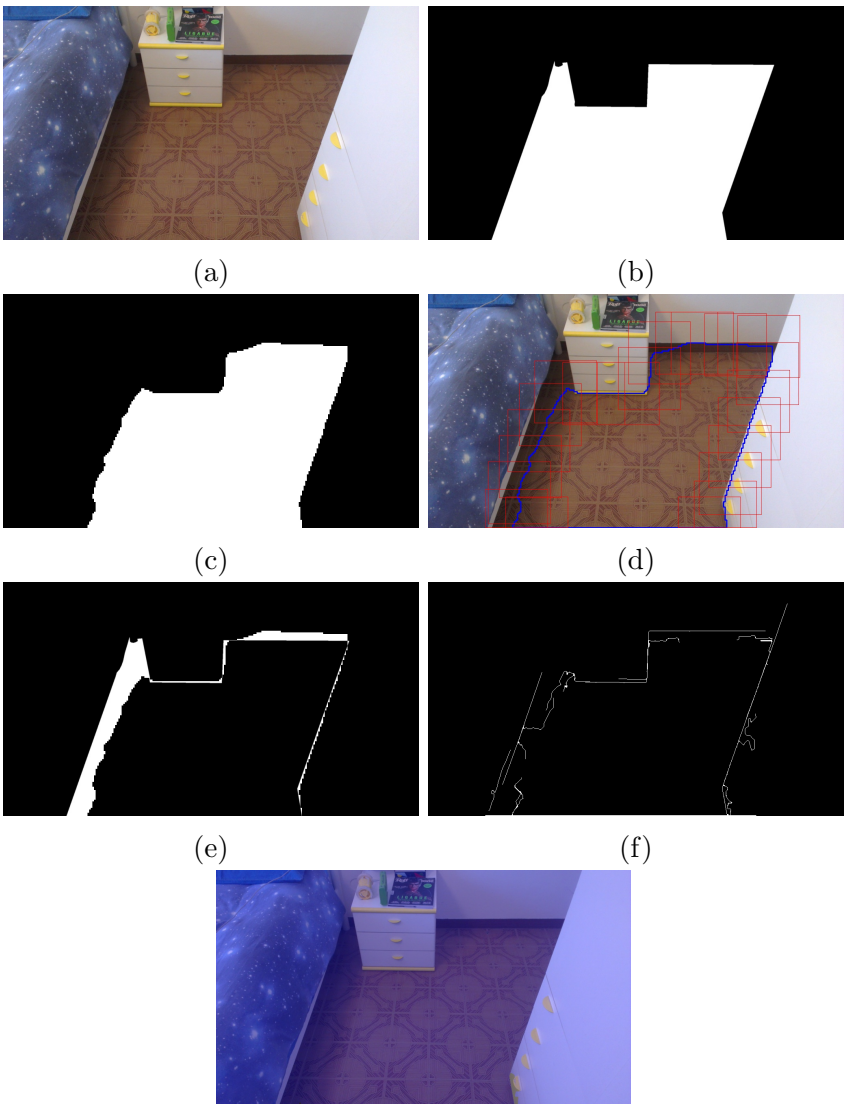


Figura 7.10: Applicazione del metodo proposto. (a) Immagine originale (*p29*). (b) *Ground truth*. (c) Segmentazione preliminare. (d) Risultato della segmentazione preliminare (contorno blu) e rilevamento delle anomalie al bordo (quadrati rossi). (e) Differenza tra la segmentazione preliminare e la *ground truth*. (f) Bordo finale ottenuto con il raffinamento (g) Sovrapposizione della segmentazione ottenuta all'immagine originale.

Nel grafico in Figura 7.11 si può vedere, come rispetto al caso di pavimenti uniformi, sia molto più difficile migliorare il risultato della segmentazione preliminare. Questo è dovuto alla maggiore complessità delle scene che si vanno ad analizzare. Nello specifico l'errore di segmentazione nell'immagine *p27* è dovuto principalmente al riconoscimento del battiscopa come appartente al pavimento, è possibile comunque ottenere un lieve miglioramento rispetto alla segmentazione ottimale. Nell'immagine *p34* l'errore è dovuto principalmente alla zona di pavimento che si intravede tra i due divani, anche attraverso il processo di raffinamento non è possibile migliorare la segmentazione di questa area. Anche nell'immagine *p28*, anche se attraverso il processo di raffinamento vengono migliorate alcune zone della segmentazione, la differenza dalla *ground truth* è ancora elevata.

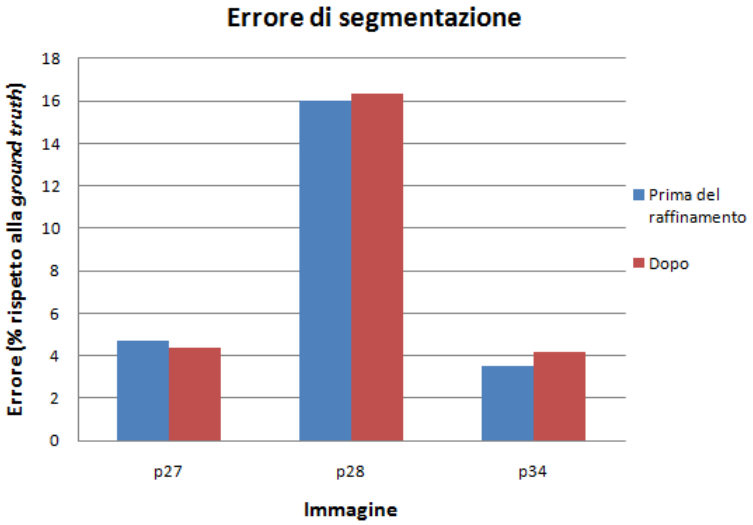


Figura 7.11: Il grafico mostra come varia l'errore di segmentazione rispetto alla *ground truth* prima e dopo il processo di raffinamento per 3 immagini con pavimenti classificati come non uniformi.

Capitolo 8

Conclusioni

L'algoritmo di segmentazione studiato in questa Tesi è stato sviluppato nel contesto della collaborazione tra il *Computer Vision Group* (*CVG*) e Maticad S.r.l., un'azienda che opera nel settore dell'*Information Technology*, Internet e Computer Grafica, presso la quale ho effettuato un periodo di tirocinio. Oggetto del lavoro è stato lo sviluppo di un'applicazione di Realtà Aumentata su dispositivo mobile, in grado di cambiare in tempo reale l'aspetto del pavimento di una scena statica.
A questo scopo, si è affrontato il tema della segmentazione semi-automatica di immagini con lo scopo di definire in modo accurato e preciso la superficie del pavimento presente nella scena, in seguito alla pre-identificazione dell'utente del pavimento, tramite un *tap* ("tocco") su *touch screen*. Data la grande varietà di ambienti e rivestimenti che si possono trovare, non è possibile estrarre la regione di interesse in modo automatico e preciso con un solo algoritmo di segmentazione. Per questa ragione il lavoro di Tesi si è focalizzato sulla ricerca di metodi per la segmentazione ed il raffinamento dell'area di interesse, in base alle caratteristiche di quest'ultima, caratteristiche che devono essere identificate automaticamente. Prima di trarre alcune considerazioni conclusive ritengo opportuno riassumere i passaggi più caratterizzanti del mio lavoro:

- Classificazione del pavimento: attualmente viene seguita una classificazione binaria (pavimenti uniformi e non uniformi) con i Random

Tree che permette di avere la totalità dei campioni di test riconosciuti correttamente.

- Segmentazione preliminare: viene applicato un differente algoritmo di segmentazione in base al tipo di superficie individuato. Per pavimenti uniformi viene utilizzato il *region growing* che consente di ottenere buoni risultati per scene in cui sono presenti elementi d'arredo con linee nette e caratteristiche diverse da quelle della superficie, altrimenti l'algoritmo si può espandere troppo ed influire negativamente anche sulla fase successiva di rifinitura. Per pavimenti non uniformi viene utilizzato il *k-means* che riesce a trovare nella maggior parte dei casi una segmentazione preliminare accettabile.

- Rifinitura: anche questa fase dipende dal tipo di superficie individuata. Per rifinire il bordo trovato dalla segmentazione preliminare vengono utilizzati i seguenti algoritmi. In scene con pavimenti uniformi: *Canny edge detector* e trasformata di Hough per l'individuazione dei bordi presenti e delle possibili rette; in caso questi non abbiano successo viene applicato l'algoritmo di *watershed*. Come si può vedere dalle immagini presentate nei risultati sperimentali questi algoritmi permettono di ottenere dei risultati più che accettabili. In scene con pavimenti non uniformi: viene applicata un'ulteriore fase di elaborazione per cercare "livellare" (attraverso il *mean-shift*) la superficie di interesse, procedendo poi come nel caso precedente. Data la grande varietà di superfici che rientrano in questa classe, non è sempre possibile ottenere dei buoni risultati con i metodi utilizzati.

Per valutare l'accuratezza del metodo proposto, la segmentazione ottenuta è stata confrontata con la *ground truth*, ottenuta segmentando manualmente delle immagini d'esempio. I risultati sperimentali dimostrano che in presenza di pavimenti uniformi è possibile ottenere una segmentazione accurata e precisa della superficie di interesse. Data la grande variabilità della classe dei pavimenti non uniformi è molto più difficile ottenere risultati perfetti. Come si può vedere dai risultati sperimentali la differenza tra la

segmentazione ottenuta e la *ground truth* è elevata, ma visivamente appare accettabile.

La complessità del problema trattato è derivata dal fatto di utilizzare un dispositivo mobile per acquisire la scena, che la scena sia reale, senza alcun vincolo sul tipo di illuminazione né sulle eventuali occlusioni del pavimento da parte di oggetti presente nell'ambiente, ed infine che vi sia un vincolo sul tempo di elaborazione, che deve essere compatibile con le modalità di utilizzo e la fruibilità da parte di un utente. Nonostante queste difficoltà, sono riuscita a definire una strategia di soluzione e un framework, che ha portato ai tre risultati principali:

- Ottimo riconoscimento e classificazione delle superfici selezionate dall'utente.
- Ottima segmentazione finale per superfici con pavimenti uniformi.
- Segmentazione finale per superfici con pavimenti non uniformi molto buona.

Grazie al lavoro di progettazione svolto, il metodo proposto può essere esteso e migliorato senza modificare l'architettura di sistema. In particolare, alcuni aspetti del metodo proposto che dovranno sicuramente essere considerati nei futuri sviluppi dell'applicazione sono i seguenti:

- La classificazione iniziale, la quale si dovrebbe estendere ad altre classi individuate (*PIRR*, *PCOMP*, *REG*), analizzando le *patch* con un diverso insieme di *feature* (ad esempio gli *edge* strutturati, i coefficienti *wavelet* o i filtri di Gabor orientati). In questo modo sarebbe possibile agire più miratamente nelle successive fasi di elaborazione.
- La segmentazione preliminare con il *region growing*, in cui si dovrebbero adattare localmente i parametri di espansione in base alle caratteristiche dell'area che si sta considerando. Invece per quanto riguarda il *k-means* si potrebbero regolare i parametri (dimensione dell'area da esaminare, numero di *bin* degli istogrammi locali e numero di *cluster*)

in base alle caratteristiche globali della scena, ad esempio determinando il numero di *cluster* partendo da un'analisi preliminare dell'istogramma dell'immagine.

- Infine, il raffinamento di superfici il quale, con pavimenti non uniformi dovrebbe includere ulteriori meccanismi per verificare e correggere l'eventuale presenza di buchi nel bordo individuato.

Appendice A

Distanza di Hausdorff

La distanza di Hausdorff permette di determinare il grado in cui due figure differiscono tra loro. Per fare ciò vengono usate immagini binarie che rappresentano i contorni degli oggetti da misurare e le coordinate dei pixel del bordo andranno a formare i punti dello spazio metrico da analizzare.
Nella definizione base, la distanza di Hausdorff misura la diversità di due insiemi di uno spazio metrico i cui punti hanno posizioni fisse le une rispetto alle altre. Dati due insiemi finiti di punti $A = \{a_1, ..., a_p\}$ e $B = \{b_1, ..., b_q\}$, la distanza di Hausdorff è definita come:

$$H(A, B) = max(h(A, B), h(B, A)) \tag{A.1}$$

dove

$$h(A, B) = \max_{a \in A} \max_{b \in B} \|a - b\| \tag{A.2}$$

in cui $\|.\|$ definisce una certa distanza tra A e B (ad esempio la distanza euclidea o la distanza di Manhattan) [45].
La funzione A.2 viene chiamata *distanza di Hausdorff diretta* da A fino a B. Identifica il punto $a \in A$ che è più lontano da qualsiasi punto di B e ne misura la distanza fino al punto di B più vicino. Ad esempio se $h(A,B) = d$, allora ogni punto di A sarà entro una distanza d da un certo punto di B e ci sarà qualche punto di A esattamente ad una distanza d dal punto più vicino di B.

La distanza di Hausdorff (eq. A.1) è il massimo tra *h(A,B)* e *h(B,A)*, misura quindi il grado di diversità tra i due insiemi, considerando la distanza del punto di A che è maggiormente distante da ogni punto di B e viceversa. Non c'è un accoppiamento esplicito tra i punti di A e di B, nel senso che diversi punti di A possono essere vicini allo stesso punto di B. La funzione *H(A,B)* può essere computata in tempo *O(pq)* per due insiemi di punti di dimensione p e q rispettivamente.
La distanza di Hausdorff è una funziona $d : P(X) \times P(X) \to \mathbb{R}_0^+$ e soddisfa le seguenti proprietà degli spazi metrici [45]:

- $d(A, B) >= 0$: la funzione è positiva ovunque.
- Se $A = B$ allora $d(A, B) = 0$: identità, una figura è identica solo a se stessa.
- $d(A, B) = d(B, A)$: simmetria, l'ordine in cui vengono comparate le figure non ha importanza.
- $d(A, B) \leq d(A, C) + d(C, B)$: disuguaglianza triangolare, se due figure sono molto diverse tra loro non possono essere entrambe simili ad una tersa figura.

Appendice B

Trasformazioni Omografiche

Una trasformazione planare omografica o proiettiva permette di determinare una trasformazione in grado di mappare punti di un piano, x_i, in punti di un altro piano, $x_i^{'}$ (espressi in coordinate omogenee). Sono trasformazioni lineari e sono rappresentate da una matrice non singolare 3×3 [46]:

$$\begin{bmatrix} x_1^{'} \\ x_2^{'} \\ x_3^{'} \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{B.1}$$

o in forma sintetica $x^{'} = Hx$. H può essere moltiplicata per un scalare diverso da zero senza alterare la trasformazione proiettiva e visto che i punti sono rappresentati in coordinate omogenee, solamente il rapporto tra gli elementi di H ha significato e non il valore in sé. Ci sono otto rapporti indipendenti tra i nove elementi di H quindi una trasformazione proiettiva ha otto gradi di libertà.
Come si può vedere in Figura B.1, una trasformazione prospettica, mappando punto per punto, preserva le rette, nel senso che una retta in un piano viene mappata in una retta nel secondo piano.

Le trasformazioni omografiche possono essere usate per rimuovere gli artefatti di un'immagine dovuti alla prospettiva. In generale linee parallele nella scena inquadrata non rimangono parallele nell'immagine ma convergono in un determinato punto. L'immagine catturata può essere vista come

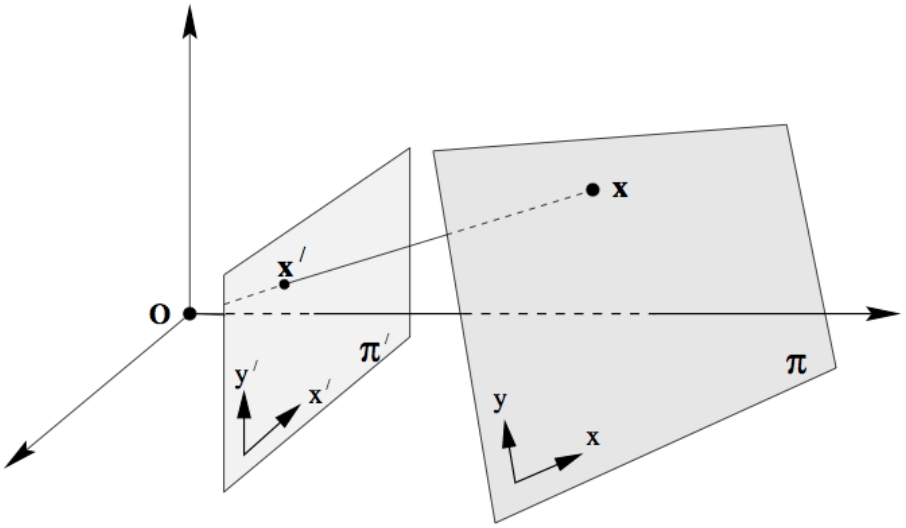


Figura B.1: Esempio di trasformazione prospettica [46].

una distorsione prospettica della scena reale. È possibile rimuovere le trasformazioni prospettiche trovando la trasformazione inversa e applicandola all'immagine. Come risultato si avrà una nuova immagine in cui gli oggetti sul piano considerato vengono visualizzati con la loro forma geometrica corretta (Figura B.2). Da notare che avranno la giusta prospettiva solamente gli oggetti posti sul piano che si sta trasformando, mentre gli elementi su altri piani saranno riproiettati in una posizione errata.
Si inizia selezionando dei punti dell'immagine (con coordinate *(x',y')*) che appartengono ad una sezione planare della scena reale (con coordinate *(x,y)*). La trasformazione proiettiva dell'equazione B.1 può essere scritta come [46]:

$$x' = \frac{x_1^{'}}{x_3^1} = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}} \tag{B.2}$$

$$y' = \frac{x_2^{'}}{x_3^1} = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \tag{B.3}$$

La proiezione di ogni punto genera due equazioni che coinvolgono gli elementi

di H, che dopo essere state moltiplicate diventano [46]:

$$x'(h_{31}x + h_{32}y + h_{33}) = h_{11}x + h_{12}y + h_{13} \tag{B.4}$$

$$y'(h_{31}x + h_{32}y + h_{33}) = h_{21}x + h_{22}y + h_{23} \tag{B.5}$$

Queste equazioni sono lineari rispetto agli elementi di H. Con l'individuazione di quattro punti si ottengono le otto equazioni che sono sufficienti per trovare H a meno di un fattore moltiplicativo insignificante. L'unica restrizione è che i quattro punti non devono essere collineari tre a tre (solitamente vengono scelti i vertici di un quadrato o un rettangolo) [46].
Da osservare che il calcolo di H non richiede nessuna conoscenza dei parametri della fotocamera o della posizione del piano che si vuole riproiettare [46].

(a) (b)

Figura B.2: Esempio di rimozione della distorsione prospettica. (a) Immagine originale, i punti rossi sono i punti utilizzati per la trasformazione. (b) Immagine trasformata, solamente il pavimento è proiettato correttamente, mentre il resto della stanza è distorto.

Bibliografia

[1] Qr droid ™. *https://play.google.com/store/apps/details?id=la.droid.qr&hl=it*, consultato in data 04/03/2014.

[2] Bmw head-up display. *http://www.bmw.it/it/footer/q-and-a/glossario/bmw-head-up-display.html*, consultato in data 04/03/2014.

[3] Acrossair augmented reality browser. *https://itunes.apple.com/it/app/acrossair-augmented-reality/id348209004?mt=8*, consultato in data 04/03/2014.

[4] Google glass. *http://www.google.com/glass/start/*, consultato in data 04/03/2014.

[5] M. Kass, A. Witkin, and D. Terzopoulos. Snake: Active contour models. *International Journal of Computer Vision*, pages 321–331, 1988.

[6] R. Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010.

[7] S. Osher and N. Paragios. *Geometric Level Set Method in Imaging, Vision, and Graphics*. Springer, 2003.

[8] M. Burger and S. J. Osher. A survey on level set methods for inverse problems and optimal design. *European Journal of Applied Mathematics*, (2):263–301, 2005.

[9] N. Paragios and R. Deriche. Geodesic active regions and level set methods for motion estimation and tracking. *Computer Vision And Image Understanding, Elsevier*, 97(3):259–282, 2005.

[10] N. Paragios and R. Deriche. Geodesic active regions and level set methods for supervised texture segmentation. *International Journal of Computer Vision*, 46(3):223–247, 2002.

[11] L. G. Ugarriza, E. Saber, V. Amuso S. R. Vantaram, M. Shaw, and R. Bhaskar. Automatic image segmentation by dynamic region growth and multiresolution merging. *IEEE Transaction on Image Processing*, 18(10):2275–2288, 2009.

[12] R. C. Gonzalez and R. E. Wood. *Digital Image Processing, Third Edition*. Pearson International Edition, 2008.

[13] I. Grinias and G. Tziritas. A semi-automatic seeded region growing algorithm for video object localization and tracking. *Signal Processing: Image Communication, Elsevier*, 16(10):977–986, 2001.

[14] D. Comaniciu and P. Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 24(5):603–619, 2002.

[15] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, 2004.

[16] E. N. Mortensen and W. A. Barrett. Intelligent scissors for image composition. *In Computer Graphics, SIGGRAPH Proceedings*, pages 191–198, 1995.

[17] K. Chun-Ho Wong, Pheng-Ann Heng, and Tien-Tsin Wong. Accelerating intelligent scissors using slimmed graph. *Journal of Graphics Tools*, 5(2):1–13, 2000.

[18] S. Alpert, M. Galun, R. Basri, and A. Brandt. Image segmentation by probabilistic bottom-up aggregation and cue integration. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2007.

[19] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[20] Y. Boykov and M. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in n-d images. *Proceeding of International Conference of Computer Vision*, 1:105–112, 2001.

[21] Y. Boykov and V. Kolmogorow. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

[22] C. Rother, V. Kolmogorov, and A. Blake. 'grabcut" - interactive foreground extraction using iterated graph cuts. *ACM Transactions on Graphics*, pages 309–314, 2004.

[23] L. Grady. Random walks for image segmentation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 28(11):1768–1783, 2006.

[24] A. I. Wasserman. Software engineering issues for mobile application development. *FoSER*, pages 397–400, 2010.

[25] J. Gong and P. Tarasewich. Guidelines for handheld mobile device interface design. *In Proceedings of the 2004 DSI Annual Meeting*, 2004.

[26] Descrizione libreria opencv. *http://opencv.org/about.html*.

[27] G. Rusconi. Tablet, i colossi non sono più soli. vicino il sorpasso sui pc. *Il Sole 24 Ore*, 23 ottobre 2013.

[28] A strong holiday quarter for the worldwide tablet market, but signs of slower growth are clear, according to idc. *IDC Press Release*, 20 gennaio 2014.

[29] Android growth drives another strong quarter for the worldwide tablet market, according to idc. *IDC Press Release*, 30 ottobre 2013.

[30] Worldwide tablet shipments forecast to slow to single-digit growth rates by 2017, according to idc. *IDC Press Release*, 03 dicembre 2013.

[31] Localytics home page. *http://www.localytics.com/*, consultato in data 03/01/2014.

[32] D. Hoch. Across fonblets and phablets samsung has 63% share of all android mobile devices. *Localytics Blog*, 7 novembre 2013.

[33] Statistiche ufficiali android. *http://developer.android.com/about/dashboards/index.html*, consultato in data 09/03/2014.

[34] G. Bradski and A. Kaehler. *Learning OpenCV, Computer Vision With the OpenCV Library*. O'Reilly, 2008.

[35] ios developer libray. *https://developer.apple.com/*, consultato in data 16/12/2013.

[36] Android developers. *http://developer.android.com/design/index.html*, consultato in data 02/01/2014.

[37] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[38] L. Shapiro and G. Stockman. *Computer Vision*. Prentice-Hall, 2000.

[39] Documentazione opencv. *http://docs.opencv.org/*, consultato in data 18/02/2014.

[40] R. M. Haralick, K. Shanmugam, and I. Dinstein. Textural feature for image classification. *IEEE Transaction on System, Man, and Cybernetics*, SMC-3(6):610–621, 1973.

[41] L. Breidman. Random forests. *Machine Learning*, 2001.

[42] I. H. Witten, E. Frank, and M. A. Hall. *Data Mining, Pratical Machine Learning Tools and Techniques*. Elsevier, 2011.

[43] Google immagini. *https://www.google.it/imghp?hl=it&tab=wi&ei=_oELU_vTIa72ygPckoA4&ved=0CAQQqi4oAg*, consultato in data 17/02/2014.

[44] L. Breiman. Looking inside the black box. *Wald Lecture II*, 2002.

[45] D. P. Huttenlocher, G. A. Klanderman, and W. J. Rucklidge. Comparing images using the hausdorff distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(9):850–863, 1993.

[46] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision, Second Edition*. Cambridge University Press, 2003.