

UNIVERSITÀ DEGLI STUDI DI PARMA

Dipartimento di Ingegneria dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

SEGMENTAZIONE E CLASSIFICAZIONE DI IMMAGINI DI
PROFONDITÀ MEDIANTE ESTRAZIONE DI
CARATTERISTICHE E MARKOV RANDOM FIELD

DEPTH IMAGE SEGMENTATION AND CLASSIFICATION USING FEATURE
EXTRACTION AND MARKOV RANDOM FIELDS

Relatore:
Chiar.mo Prof. Stefano Caselli

Correlatore:
Ing. Dario Lodi Rizzini

Tesi di Laurea di:
Andrea Atti

ANNO ACCADEMICO 2011-2012

Alla mia famiglia.

*“È già trascendenza riuscire a intendersi con un altro.
Non c’è bisogno di parlare col Dio ignoto, dove a parlare sono solo io, perché tanto
lui non risponde, se non le parole che io penso che lui dica.
Che sono poi gli esaudimenti dei miei desideri.”*

Umberto Galimberti, Le Storie

Ringraziamenti

Giunto al termine di questo percorso, desidero ringraziare chi, in maniera più o meno diretta e più o meno consapevole, ha contribuito alla sua realizzazione. Rivolgo innanzitutto i miei ringraziamenti al Prof. Stefano Caselli, che mi ha aperto le porte del laboratorio di robotica e mi ha permesso lo svolgimento di questa tesi.

Grazie a Dario sempre prodigo di consigli, per la costante disponibilità, per aver sempre trovato il tempo di aiutarmi, darmi supporto e che si è rivelato una guida fondamentale nello sviluppo del progetto.

Ancora grazie a Jacopo per le sue osservazioni preziose e costruttive, a Fabio per la cortesia avuta nei miei confronti, le giuste indicazioni e l'estrema passione che è riuscito a trasmettermi in tutte le occasioni in cui ci siamo incontrati in laboratorio.

Infine, ho desiderio di ringraziare con affetto i miei genitori ed i miei parenti tutti, senza i quali tutto questo non sarebbe stato possibile, per il sostegno ed il grande aiuto che mi hanno dato; in particolare a Giulia per essermi stata vicino ed avermi sopportato in ogni momento durante questi duri mesi di lavoro.

Grazie anche a tutti gli altri ragazzi del laboratorio con cui ho passato gli ultimi mesi, ai miei ex-colleghi di corso che mi hanno regalato momenti impagabili ed agli amici d'infanzia più cari che mi hanno sempre sostenuto tutto questo tempo.

Indice

1	Introduzione	1
2	Stato dell'arte	6
2.1	Percezione tridimensionale	7
2.1.1	Metodi ottici	8
2.1.2	Immagini di range	11
2.2	Estrazione delle caratteristiche	12
2.3	Il processo di segmentazione	14
2.3.1	Utilizzo di <i>Random Fields</i>	16
3	Descrizione del problema	20
3.1	Teoria ed impiego dei Markov Random Field in applicazioni di segmentazione	22
3.1.1	Il potenziale unario	26
3.1.2	Il potenziale di coppia	27
3.2	Il sistema d'acquisizione	30
3.3	Software e librerie utilizzate	33
3.3.1	Point Cloud Library	33
3.3.2	OpenCV	35
3.3.3	FANN: Fast Artificial Neural Network	36
3.3.4	Tiny SVM	36
3.3.5	GCO: Graph-Cut Optimization	36
3.3.6	Riepilogo dell'architettura applicativa	37

4 L'algoritmo realizzato	39
4.1 Overview dell'algoritmo	39
4.2 Descrizione dell'algoritmo	41
4.2.1 Acquisizione	41
4.2.2 Il preprocessing dei dati	45
4.2.2.1 Organizzazione della nuvola tramite octree	45
4.2.2.2 Calcolo delle normali	47
4.2.3 Estrazione delle features	49
4.2.3.1 Features di colore	49
4.2.3.2 Features di tessitura	51
4.2.3.3 Features di forma	54
4.2.4 La classificazione	58
4.2.4.1 Apprendimento supervisionato	59
4.2.4.2 Il ruolo del ground truth	61
Reti neurali	63
Support vector machines	65
4.2.4.3 Apprendimento non supervisionato	68
Algoritmo k-Means <i>base</i>	69
Algoritmo k-Means ottimizzato	72
4.2.5 Segmentazione mediante Markov Random Field (MRF)	76
4.2.5.1 Platt scaling	77
4.2.5.2 Distance scaling	78
4.2.5.3 Definizione del grafo e funzione energia associata . .	79
4.2.5.4 Processo d'inferenza e minimizzazione dell'energia .	81
Partizioni e spazio delle mosse	83
L'algoritmo di $\alpha\beta$ -Swap	84
L'algoritmo di α -Expansion	87
5 Risultati sperimentali	93
5.1 Setup sperimentale	93
5.2 Sequenza acquisita	94
5.2.1 Training set e test set	96

5.2.2	Risultati di classificazione supervisionata	98
5.2.2.1	Reti Neurali	100
5.2.2.2	Support Vector Machines	107
5.2.2.3	Tempi di addestramento e classificazione supervisionata	113
5.2.3	Risultati di classificazione non supervisionata	117
5.2.3.1	Tempi di addestramento e classificazione non supervisionata	127
5.2.4	Tempi d'esecuzione dell'algoritmo	129
6	Conclusioni e sviluppi futuri	133
6.1	Sviluppi futuri	135
Bibliografia		136

Elenco delle figure

2.1	Tassonomia dei sistemi di acquisizione	10
2.2	Esempio di immagine di range	11
2.3	Esempi d'utilizzo dei Markov Random Fields	17
2.4	Scomposizione della scena in superpixel	18
2.5	Esempio di segmentazione attraverso MRF	19
3.1	Esempio di Markov Random Field	23
3.2	Proprietà di Markovianità	24
3.3	Modello di Potts	27
3.4	Modello lineare troncato	28
3.5	Modello quadratico troncato	29
3.6	Il sensore Microsoft Kinect	30
3.7	I componenti principali del sensore	31
3.8	Architettura hardware del Microsoft Kinect	31
3.9	La struttura dati octree	34
3.10	Architettura dei componenti software	38
4.1	Algoritmo	42
4.2	Esempio di dati acquisiti	44
4.3	Scomposizione di una point cloud mediante octree	46
4.4	Calcolo della normali ad una nuvola di punti	48
4.5	Colore Vs. Texture	50
4.6	Proprietà di invarianza all'illuminazione dell'HSV	50
4.7	Spazi di colore a confronto	51

4.8	Il filtro di Gabor	52
4.9	Banco di filtri di Gabor per l'analisi della tessitura	53
4.10	Autovalori normalizzati	55
4.11	Coordinate sferiche	56
4.12	Step del processo di apprendimento supervisionato	60
4.13	Il fenomeno dell'overfitting	61
4.14	Esempio di rete neurale artificiale	64
4.15	Separazione lineare di dati tramite SVM	67
4.16	Esempio di kernel trick	68
4.17	Step del processo di apprendimento non supervisionato	69
4.18	Limiti dell'algoritmo k-Means base	71
4.19	Platt scaling	78
4.20	Sistemi di vicinato tridimensionale	79
4.21	26-vicinato variabile	80
4.22	MRF associato ad $E(\mathcal{V})$	81
4.23	Taglio a costo minimo	83
4.24	Grafo associato all'algoritmo di $\alpha\beta$ -Swap	86
4.25	$\alpha\beta$ -Swap: alcuni esempi di taglio	87
4.26	Grafo associato all'algoritmo di α -Expansion	90
4.27	α -Expansion: alcuni esempi di taglio	91
5.1	Immagini RGB della scena acquisita	94
5.2	Ground truth	95
5.3	Precision - Reti neurali	103
5.4	Recall - Reti neurali	104
5.5	Risultati di segmentazione - Reti neurali	106
5.6	Precision - Support Vector Machines	110
5.7	Recall - Support Vector Machines	111
5.8	Indici di performance a confronto	115
5.9	Risultati di segmentazione - Support Vector Machines	116
5.10	Numero ottimo di classi - DBI	119
5.11	Risultati di segmentazione non supervisionata senza MRF	120

5.12 Risultati di segmentazione non supervisionata con MRF	125
5.13 Risultati di segmentazione supervisionata Vs segmentazione non supervisionata	126
5.14 Tempi di classificazione a confronto	128
5.15 Tempi d'esecuzione dell'algoritmo - Caso supervisionato (ANN)	130
5.16 Tempi d'esecuzione dell'algoritmo - Caso supervisionato (SVM)	130
5.17 Tempi d'esecuzione dell'algoritmo - Caso non supervisionato	131

Elenco delle tabelle

4.1	Assegnazione dei pesi: $\alpha\beta$ -Swap	86
4.2	Assegnazione dei pesi: α -Expansion	91
5.1	Specifiche tecniche hardware	94
5.2	Numero di voxel - Training set e test set	96
5.3	Train e test set - Numero di voxel	97
5.4	Voxel nel train set e test set - Numero medio e deviazione standard	97
5.5	Parametri di training: reti neurali	101
5.6	Matrice di confusione relativa al processo di classificazione supervisionata attraverso reti neurali senza l'ausilio di MRF	101
5.7	Matrice di confusione relativa al processo di classificazione supervisionata attraverso l'impiego di reti neurali e MRF	101
5.8	Indici di Precision, Recall ed Accuracy - Reti Neurali	102
5.9	Parametri di training: Support Vector Machines	107
5.10	Matrice di confusione relativa al processo di classificazione supervisionata attraverso SVM senza l'ausilio di MRF	108
5.11	Matrice di confusione relativa al processo di classificazione supervisionata attraverso l'impiego di SVM e MRF	108
5.12	Indici di Precision, Recall ed Accuracy - Support Vector Machines . . .	109
5.13	Tempi d'esecuzione	112
5.14	Tempi di classificazione a confronto - caso supervisionato	113
5.15	Tempi di addestramento a confronto	114
5.16	Ripartizione delle classi generate sulle macro-classi del ground-truth . .	124

5.17 Tempi di classificazione a confronto - caso non supervisionato	127
---	-----

Capitolo 1

Introduzione

L'uomo è costantemente immerso all'interno di una realtà con la quale si deve continuamente relazionare e che deve comprendere al fine di poter interagire efficacemente con essa; uno degli aspetti strabilianti della mente umana è senz'altro l'abilità di riuscire a percepire, e conseguentemente interpretare con naturalezza ed estrema rapidità, immagini, oggetti ed eventi anche osservandoli per la prima volta: in pochi istanti possiamo comprendere le relazioni semantiche dell'ambiente all'interno del quale ci troviamo, la sua organizzazione, identificare le “parti” che lo compongono e/o i vari oggetti presenti al suo interno riuscendo così a comprendere la scena in tutta la sua interezza.

La possibilità di emulare queste capacità di interpretazione dell'osservato, tramite la realizzazione e l'impiego di sistemi autonomi intelligenti, è un compito molto impegnativo che ha interessato in maniera trasversale diversi campi della ricerca scientifica quali la robotica, la visione artificiale, il campo dell'apprendimento automatico, la psicologia cognitiva e le neuroscienze.

Nello specifico, per quanto riguarda la robotica, questa attitudine di comprensione della scena ricopre un ruolo di fondamentale importanza in svariati ambiti come, per esempio, applicazioni di navigazione mobile, manipolazione di oggetti, object-detection ed interazione uomo-robot: in tutti questi scenari l'attività di comprensione della scena si traduce nell'identificare e localizzare oggetti o particolari zone di interesse. Effettuare questo compito in maniera automatica, attraverso l'acquisizione di immagini di profondità, e riuscire a separare correttamente le entità osservate in classi di appartenenza,

rappresentano le principali capacità base di un sistema robotico la cui necessità è quella di attuare un determinato *behaviour* conseguentemente al task prefissato e grazie a ciò che è in grado di percepire attraverso i propri sensori.

In questa tesi è affrontato il problema della segmentazione automatica di scenari mediante l'utilizzo di dispositivi di acquisizione tridimensionale. L'obbiettivo è quindi quello di realizzare un sistema autonomo in grado riconoscere determinate entità all'interno della scena e di associarle correttamente alle rispettive classi d'appartenenza (es. muro, porta, ecc...). Per perseguire tale scopo ci si è serviti di un particolare tipo di sensore (nello specifico il *Microsoft Kinect* [1]) in grado di restituire un'immagine di prossimità che codifica informazioni di colore, profondità e la forma di ciò che viene osservato; questo tipo di sensore è in grado di restituire allo stesso istante due diversi tipi di immagine 2D: una di colore (come se si trattasse di una normale fotografia) e l'altra di profondità rappresentante il diverso grado di distanza degli oggetti dal punto di osservazione.

La principale differenza rispetto a sensori come le “normali” telecamere è ovvia: in questo caso il dispositivo è in grado di restituire maggiore informazione, mettendo di fatto a disposizione dati aggiuntivi derivanti dalla dimensione in più che è in grado di percepire. Rispetto ai sensori di profondità, come per esempio i *laser scanner*, i vantaggi sono diversi: innanzitutto il dato di profondità restituito è relativo a tutto l'osservato e non ad un unico piano di scansione; questo permette di poter acquisire tutta la scena nella sua interezza mantenendo fermo il sensore, evitando così la pianificazione ed esecuzione di traiettorie durante la fase di acquisizione, che sicuramente rappresentano ulteriore lavoro richiesto in fase di raccolta dati. Altro vantaggio è rappresentato dalla capacità di restituire all'utilizzatore anche le informazioni di colore associate alla scena osservata.

In conclusione, un dispositivo di questo tipo fa confluire in sé tutte le caratteristiche tipiche di una telecamera e quelle di un sensore di profondità creando di fatto un sistema di acquisizione completo che sopperisce alle lacune di entrambi i due tipi di sensori citati in precedenza. Ovviamente un sensore di questo tipo presenta anche dei limiti, questi sono dovuti principalmente alla scarsa precisione, un campo visivo piuttosto limitato,

l'impossibilità di utilizzo in particolari condizioni (nel caso del Kinect, il suo impiego in ambienti esterni risulta proibitivo) e l'elaborazione dei dati restituiti è complessa e computazionalmente pesante da trattare.

Dall'utilizzo congiunto di entrambi i tipi di informazione (colore e profondità) e grazie alla libreria software *Point Cloud Library* [2] (PCL) è possibile definire una rappresentazione 3D densa della scena grazie all'utilizzo di una struttura composta da un fitto insieme di punti, ognuno dei quali è caratterizzato da informazioni di tipo spaziale e di apparenza; tali tipi di strutture tridimensionali prendono il nome di *nuvole di punti* o equivalentemente *point cloud*.

Relativamente al grado di risoluzione adottato durante la procedura di acquisizione, può capitare che la nuvola restituita contenga diverse centinaia di migliaia di punti, ognuno dei quali deve essere acceduto numerose volte durante il ciclo di vita del sistema. Inoltre, per determinate operazioni può capitare di dover ottenere solamente un ben preciso sottoinsieme di questi punti ed estrarlo dalla totalità della cloud, o ancora vi è la necessità di dover ricavare rapidamente le relazioni di vicinanza tra più punti: per questo si necessita di un modello dati che funga da infrastruttura non solo efficace ma anche efficiente durante l'accesso ed il recupero dell'informazione.

Una volta ottenuta la nuvola nella sua interezza è possibile passare alla parte successiva dell'algoritmo che prevede l'organizzazione logica della rappresentazione spaziale in tante entità elementari chiamate *volumetric pixel* o più brevemente *voxel*: questi oggetti rappresentano l'unità minima di volume rappresentabile e circoscrivono al loro interno poche centinaia di punti della cloud. Questa suddivisione permette il campionamento selettivo di insiemi di punti limitati, la successiva estrazione di caratteristiche distintive di colore, texture, forma e la generazione di descrittori multidimensionali che rappresenteranno da un lato le caratteristiche principali di quel determinato volume di spazio, dall'altro la base di partenza per la costituzione di un *training set* necessario all'addestramento del modulo di classificazione; il compito principale di questo componente, una volta “allenato” a dovere, sarà quello di assegnare ogni singola porzione di cloud (voxel) ad una specifica categoria; quest'ultima potrà essere definita a priori in caso di apprendimento *supervisionato* o ricavata mediante tecniche di *clustering* per il caso *non*

supervisionato.

Terminata la fase di classificazione è già possibile apprezzare un primo esito di etichettatura della scena il quale però potrebbe risultare piuttosto impreciso e poco corretto se confrontato con i risultati attesi. Al fine di migliorare l'output finale del sistema si applica quindi un ultimo step correttivo mediante l'impiego di un particolare modello probabilistico, basato su una struttura a grafo non orientato, chiamato *Markov Random Field* (MRF) [3]. Modelli di questo tipo hanno riscosso parecchio successo in diversi campi della ricerca scientifica come la visione artificiale, la robotica, l'apprendimento automatico e l'intelligenza artificiale dove i MRF si sono dimostrati uno strumento estremamente potente (ed al tempo stesso elegante), a causa della loro capacità di considerare, ai fini dell'assegnazione della categoria d'appartenenza, non solo il singolo nodo del grafo ma anche le caratteristiche degli elementi a lui confinati e che compongono il suo “vicinato”.

La caratteristica dei MRF di poter gestire anche le informazioni riguardanti le relazioni di adiacenza tra gli elementi che lo compongono, permette di ottenere buoni risultati di segmentazione e un buon *trade-off* tra accuratezza e tempi d'esecuzione dell'algoritmo. Tutto questo grazie al lavoro svolto negli ultimi anni nel campo della teoria dei grafi: sebbene inizialmente i problemi d'inferenza fossero stati considerati intrattabili, la nascita di diversi algoritmi di minimizzazione dell'energia [4] (tra i quali i più popolari sono senz'altro il *Loopy Belief Propagation* (LBP) [5] e gli algoritmi di taglio [6]) hanno reso questo task computazionalmente trattabile in tempi accettabili.

In questa tesi, nel secondo capitolo si presenta un riepilogo dello stato dell'arte per ciò che riguarda i problemi di *segmentation* e come questi siano stati risolti mediante l'utilizzo di modelli probabilistici a grafo, nello specifico MRF e relative varianti.

Il terzo capitolo inizierà presentando un'introduzione sulla teoria alla base dei MRF per poi passare alla discussione della scelta degli strumenti hardware e software impiegati e le scelte architettoniche seguite per la realizzazione del sistema.

Il quarto capitolo verterà sulla presentazione ed una descrizione approfondita dell'algoritmo di segmentazione automatica sviluppato nella tesi.

A lavoro ultimato è stato possibile ricavare diversi indici di performance, al fine di mi-

surare quantitativamente la bontà complessiva dell'apparato realizzato nello svolgere il compito di classificazione; questi dati verranno discussi nel quinto capitolo. La tesi presenta, infine, le conclusioni ed i possibili sviluppi futuri.

Capitolo 2

Stato dell'arte

La sempre maggiore diffusione e disponibilità di sensori in grado di acquisire dati tridimensionali da una scena reale ha incrementato l'interesse per l'elaborazione e la rappresentazione di informazioni nelle tre dimensioni. Di particolare interesse sono i problemi di identificazione e riconoscimento di oggetti osservati e/o la classificazione semantica di scene, cioè la ripartizione in più categorie delle parti principali costituenti l'ambiente all'interno del quale ci si trova. La possibilità di ricavare questo tipo d'informazione è di fondamentale importanza nello sviluppo di sistemi orientati all'interazione tra uomo, robot e l'ambiente circostante.

Al fine di comprendere la struttura e la conformazione di ciò che viene osservato sono state sviluppate diverse tecniche e algoritmi; altrettanto numerosi sono gli ambiti scientifici nei quali si è sviluppato l'interesse in merito a questo tema, tra cui: la visione artificiale, la robotica, l'intelligenza artificiale e la computer graphics.

Nell'ambito delle attività di *scene understanding* si è soliti distinguere i concetti di *detection* e *recognition*. Detection significa individuare la porzione della scena che corrisponde ad una classe di oggetti o entità. Per esempio, nel caso di una scena costituita da un piano ed un insieme di oggetti posti su di esso, un corretto processo di detection permette di isolare indistintamente tali oggetti. È da notare come nonostante siamo di fronte ad un evento nel quale è stata rilevata la presenza di oggetti, di fatto conosciamo ancora poco su cosa effettivamente questi siano; siamo solo al corrente del fatto che questi possano essere presenti. La recognition invece si traduce nell'eventuale corrispondenza

trovata tra “l’evento” che si è verificato (presenza di oggetti) ed una categoria di pattern ben noti al sistema d’analisi (per esempio l’oggetto ha una forma sferica). Per tornare al caso del sistema di riconoscimento, il passo di recognition si traduce nell’effettiva identificazione di questi in base a caratteristiche distintive, come ad esempio forma e/o colore; proseguendo nell’esempio, riconoscendo un oggetto dalla forma sferica si potrà raffinare ulteriormente il risultato finale in: “presenza di oggetti: una palla”.

Detto questo, è da osservare come l’identificazione di qualcosa non sia possibile senza aver prima individuato zone d’indagine preferenziali, ma anche come la scelta di queste siano strettamente legate al tipo di entità di cui si vuole individuare. Alla luce di quanto appena detto, si capisce come non sempre detection e recognition sono distinguibili con chiarezza all’interno di una sistema di interpretazione della scena; ma come queste attività siano strettamente accoppiate e dipendano reciprocamente l’una dall’altra.

In questo capitolo viene presentata una sintetica tassonomia dei sensori adottati per l’acquisizione di scenari tridimensionali. Successivamente si riporta una breve classificazione delle principali tecniche d’estrazione/elaborazione di caratteristiche 2D/3D maggiormente utilizzate per l’ottenimento del risultato di segmentazione finale della scena osservata.

2.1 Percezione tridimensionale

La capacità di ricavare le informazioni di conformazione tridimensionale della scena è la prerogativa principale di un sistema che si occupa di segmentare la scena osservata. La possibilità di riuscire ad acquisire correttamente i dati tridimensionali relativi alla struttura che sta inquadrandolo è un task di fondamentale importanza che può essere assolto mediante l’utilizzo di differenti tipi di sensore. L’interesse della comunità scientifica ai problemi di detection e recognition è andato via via aumentando parallelamente all’evoluzione tecnologica sia dei sensori che dell’hardware di elaborazione. Esiste, pertanto, un’estesa letteratura che comprende ambiti di ricerca diversi, quali computer

vision, shape retrieval nella computer graphics, intelligenza artificiale e robotica.

I metodi per l’acquisizione automatica dei volumi di volumi all’interno di una scena sono svariati. Una prima classificazione può dividere sistemi riflessivi e trasmissivi (es. raggi X). Per le finalità e gli scopi di questa tesi verranno presi in esame solo i sistemi di tipo riflessivo, in particolare quelli ottici, quelli che cioè operano con la luce riflessa dagli oggetti, allo stesso modo del nostro sistema visivo. Il principio base dell’*image-based modeling* è molto semplice: le entità inquadrate irradiano luce visibile e questa può essere catturata attraverso l’uso di una convenzionale telecamera. Le caratteristiche della luce dipendono da diversi fattori quali: illuminazione della scena, geometria delle superfici e grado di riflessione delle superfici stesse. L’analisi al calcolatore permette poi di stimare la natura 3D degli oggetti. Le diverse tecniche vengono classificate soprattutto in base all’impiego (o meno) di diverse fonti di illuminazione esterne. Distinguiamo dunque tra due importanti categorie:

- *metodi attivi*: sistemi che irradiano la scena di interesse con opportune radiazioni elettromagnetiche (anche a frequenze non visibili all’occhio umano come gli infrarossi). In questo caso si ricorre all’uso di pattern luminosi, luce laser, radiazioni IR, etc.
- *metodi passivi*: si basano esclusivamente sull’analisi di immagini di colore della scena così com’è, sfruttando una o più riprese della telecamera, unico strumento di analisi.

I primi hanno il vantaggio di raggiungere risoluzioni elevate, con precisione adatta anche ad applicazioni industriali, richiedendo però un costo notevole e permettendo un’applicabilità ristretta a determinati ambiti. I secondi, pur avendo prestazioni inferiori, presentano una maggiore duttilità, ampia versatilità ed applicabilità in diversi contesti, oltre ad un costo ridotto.

2.1.1 Metodi ottici

Pur non potendo ricreare fedelmente i processi alla base della ricostruzione tridimensionale del sistema visivo umano, la visione artificiale deve analizzare tutti gli aspetti che

in un'immagine sono legati alla percezione della profondità di una scena: sfocamento, disparità, chiaroscuro, tessiture. Tutte le tecniche computazionali, attive o passive che siano, devono fare riferimento a questi “indizi” e ad altri fenomeni ottici, nel processo di modellazione. In figura 2.1 si può osservare come tra la famiglia dei metodi ottici passivi troviamo, ad esempio:

- depth from focus/defocus;
- shape from texture;
- shape from shading;
- stereo fotometrico;
- stereopsi;
- shape from silhouette;
- shape from photo-consistency.

Tra quella dei metodi ottici attivi invece:

- active defocus;
- stereo attivo;
- triangolazione attiva;
- interferometria;
- tempo di volo (TOF).

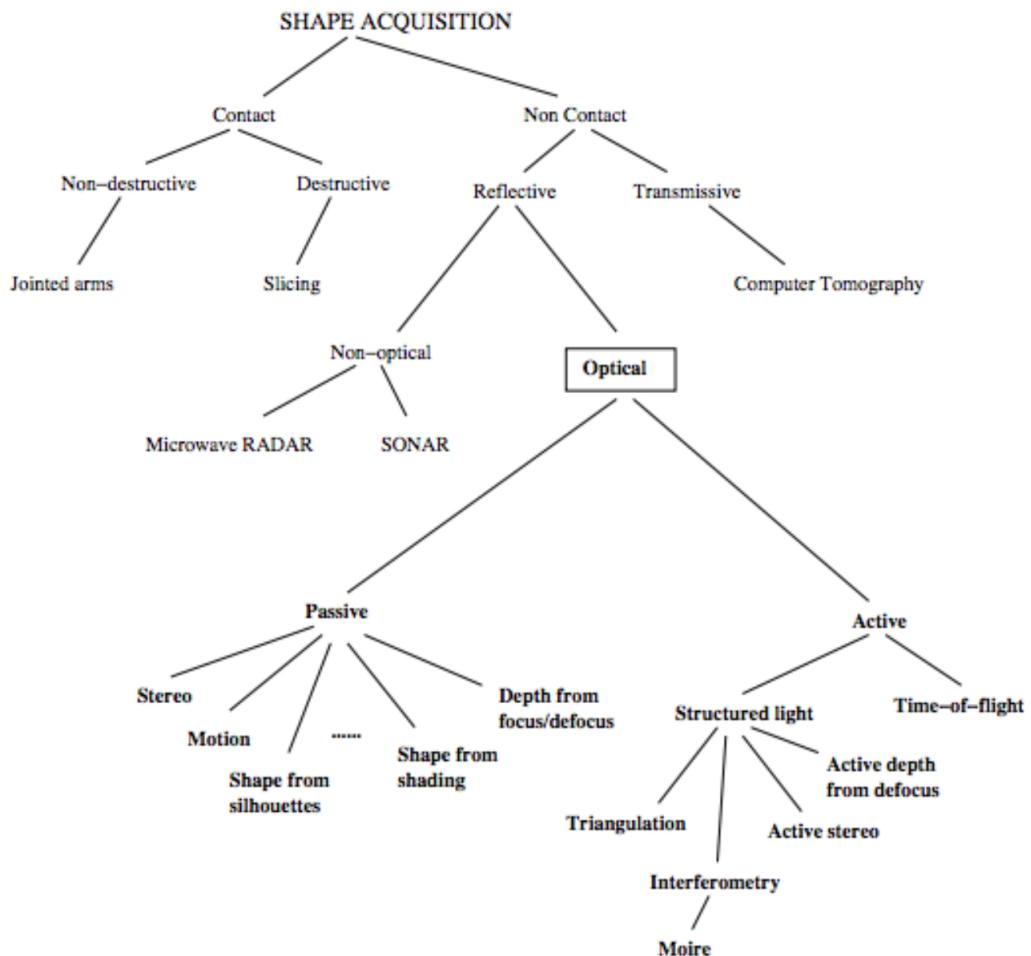


Figura 2.1: Tassonomia dei sistemi di acquisizione [7]



Figura 2.2: Esempio di immagine range: (a) immagine originale e (b) immagine range associata, più intensa è la luminosità più l'oggetto osservato è vicino

2.1.2 Immagini di range

Molti dei dispositivi ottici per l’acquisizione 3D di una scena o della superficie di un oggetto restituiscono un’immagine range, cioè un’immagine nella quale a ciascuna coordinata bidimensionale (x, y) è associata la relativa distanza dal sensore al primo punto visibile della scena.

Un’immagine range è costituita da misure discrete di una superficie 3D, rispetto ad un piano di riferimento (il piano della telecamera). Questa di solito viene anche chiamata immagine 2.5D. Un sensore range è un dispositivo (apparecchiature e software) che produce per l’appunto un’immagine range. La qualità di tale immagine si misura secondo i seguenti parametri:

- *risoluzione*: la più piccola variazione di profondità alla quale è sensibile il sensore;
- *accuratezza*: errore tra valore rilevato e valore esatto;
- *precisione*: deviazione standard su una serie di misure ripetute sullo stesso valore;
- *velocità*: quantità di misure ottenibili nell’unità di tempo.

Tra i tipi di sensori più utilizzati possiamo citarne alcuni:

- *laser scanner 3D*: i sistemi di misura basati sul laser sfruttano una lama di luce (laser) per digitalizzare le parti. Questi strumenti sono utilizzabili in maniera fissa, ancorati ad un cavalletto, oppure possono essere utilizzati in maniera “dinamica” attraverso il compimento di traiettorie.
- *luce strutturata*: gli scanner a luce strutturata sono composti generalmente da una testa ottica montata su un treppiede; addirittura alcune applicazioni particolari prevedono il montaggio della testa ottica su un robot antropomorfo con diversi gradi di libertà. Questo dispositivo di misura 3D proietta una serie di linee parallele di luce su un oggetto, da un punto di osservazione diverso da quello del proiettore viene acquisita l’immagine delle linee: queste risulteranno deformate rispetto a quelle inizialmente proiettate. Attraverso l’elaborazione delle linee deformate è possibile risalire alla forma dell’oggetto colpito dalla luce.
- *time of flight*: una telecamera time of flight, o equivalentemente “ToF camera”, è un sistema per l’acquisizione di immagini range che cerca di risolvere le distanze basandosi sulla conoscenza della velocità alla quale viaggia la luce, misurando con accuratezza il tempo di volo tra la telecamera ed il soggetto di un segnale luminoso, per ciascun punto dell’immagine.

2.2 Estrazione delle caratteristiche

Il compito di riconoscimento ed identificazione di parti o oggetti all’interno della scena implica la necessità di disporre di un metodo che permetta di estrarre e rappresentare i “tratti” distintivi delle entità osservate. Per quanto riguarda i dati tridimensionali acquisibili, questi vengono solitamente aggregati in nuvole di punti che ben si prestano per riportare il collocamento nelle tre dimensioni di ogni punto. Congiuntamente alle informazioni di colore che possono essere ricavate per la stessa scena raffigurata nelle immagini di range, queste sono le informazioni che possono essere sfruttate per ricavare i tratti distintivi di ogni zona presente nel quadro osservato. Nel corso degli anni sono state presentate diverse tecniche che permettono il calcolo di questo tipo d’informazione: tra queste troviamo gli ormai consolidati descrittori *Scale-Invariant Feature*

Transform (SIFT) [8] e *Speeded Up Robust Features* (SURF) [9] (che trovano il loro principale impiego nell’analisi d’immagini bidimensionali anche se ultimamente stanno avendo qualche applicazione in campo 3D [10]), fino ai più recenti *ORB* [11], che a causa della loro particolare implementazione basata su *FAST* [12] e *BRIEF* [13] combinata all’utilizzo di stringhe binarie di bit, hanno tempi d’esecuzione inferiori ai metodi precedenti, o ancora il recente *Fast Retina Keypoint* (FREAK) [14] il quale trae ispirazione dalla conformazione del *ganglio ciliare* e della *saccade*, elementi tipici dell’apparato visivo umano.

Tutti questi approcci permettono di ricavare un descrittore (con determinate caratteristiche di invarianza a rotazione, scala, ecc.) che distingua univocamente dei punti chiave (*keypoints*) rilevati preventivamente all’interno della scena; questi punti sono rilevabili solamente in determinate zone dell’osservato particolarmente distintive. Zone con “poca informazione”, cioè povere o completamente prive di texture, solitamente non permettono l’estrazione di punti chiave al loro interno. Questo limite giustifica l’impiego tali tecniche in compiti di riconoscimento oggetti e matching di corrispondenze tra differenti viste della stessa scena.

Altri descrittori utilizzati esclusivamente nella lavorazione di cloud tridimensionali sono i *Fast Point Feature Histogram* (FPFH) [15] e le *spin-images* (immagini di spin) [16]; i primi forniscono una caratterizzazione locale dei punti in base alla conformazione delle normali relative al loro vicinato, mentre i secondi permettono di ricavare informazioni di forma locali relative ad un punto, in base alla proiezione sul piano (definito dalla normale del punto stesso) degli elementi a lui vicini. Questi tipi di descrittori sono in grado di restituire una firma relativa a qualsiasi zona della cloud, grazie al fatto che non necessitano di una ricerca preliminare di zone d’interesse caratteristiche ma ricavano le informazioni direttamente dal vicinato del punto al quale vengono applicati.

Un altro metodo ampiamente utilizzato per la classificazione d’immagini è l’approccio di tipo *Bag of Words* (BoW) [17] nel quale un’immagine viene trattata alla stregua di un documento. Secondo questa tecnica le “parole” presenti all’interno dell’immagine sono definite tramite l’individuazione e l’estrazione di caratteristiche: una volta ricavate, queste vengono poi tradotte in descrittori numerici utilizzati per comporre un vocabolario di termini talvolta chiamato *code-book*. Ottenuta questa lista, ogni immagine viene

categorizzata in base a quanti e quali tipi di descrittori contiene: tale operazione avviene attraverso la ripartizione su istogramma dei termini presenti al suo interno. Conseguentemente a questo, la definizione di un modello BoW consiste in una rappresentazione di immagini mediante istogrammi basati su features indipendenti.

2.3 Il processo di segmentazione

Un altro tema molto importante nell’ambito della comprensione della scena è quello *segmentazione*. Nel caso, più semplice, di immagini bidimensionali. Segmentare significa partizionare un’immagine in regioni significative. Tale tecnica è spesso il prerequisito principale nei compiti di detection/recognition e viene utilizzata per ottenere una rappresentazione più compatta dell’osservato, estrarre degli oggetti o comunque come strumento per l’analisi delle immagini e permette di partizionare le immagini digitali in insiemi di pixel. Lo scopo della segmentazione è semplificare e/o cambiare la rappresentazione delle immagini in qualcosa che sia più significativo e facile da analizzare successivamente. La segmentazione è di solito utilizzata per localizzare oggetti e bordi (come linee, curve, ecc.); più precisamente, la segmentazione è il processo con il quale si classificano i pixel dell’immagine che hanno caratteristiche comuni, pertanto ciascun pixel in una regione è simile agli altri della stessa regione per una qualche proprietà o caratteristica (per esempio colore, intensità o tessitura). Conseguentemente a quanto appena detto, regioni differenti adiacenti tra loro sono significativamente differenti rispetto ad almeno una di queste caratteristiche. Il risultato di un’immagine segmentata è un insieme di partizioni che, collettivamente, coprono l’intera immagine. Al fine di segmentare in maniera efficace la scena è necessaria la scelta di strumenti adeguati per la raccolta dei dati e di tecniche idonee sia per l’estrazione delle caratteristiche distinctive degli elementi osservati che per la loro rielaborazione e ripartizione nelle categorie finali individuabili.

Una tecnica spesso usata per la segmentazione d’immagini è quella del *region growing* [18] (traducibile in italiano come “accrescimento delle regioni”) che consiste in un semplice metodo di segmentazione basato su regioni. Questo metodo ha una approccio basato su singolo pixel poiché comporta inizialmente la selezione dei punti seme ed

opera a livello di punti immagine. Questo approccio alla segmentazione esamina i pixel adiacenti di punti iniziali, i semi, e determina se i vicini possono essere aggiunti (in base ad opportune relazioni di similarità) alla regione corrispondente. Il processo viene così iterato fino a che non è più possibile portare avanti il processo di accrescimento.

Tale metodo porta con sé diversi aspetti, alcuni positivi ed altri negativi. Tra i principali vantaggi di questa tecnica ricordiamo:

- possono separare correttamente le regioni che hanno le stesse proprietà definite a priori;
- i metodi di region growing sono in grado di segmentare le immagini raggiungendo buoni livelli di precisione;
- il concetto alla base della procedura di accrescimento è semplice. Si richiede un numero ridotto di punti seme per rappresentare la proprietà che si desidera e per poi crescere la regione;
- è possibile determinare liberamente i punti seme ed i criteri che si vogliono seguire;
- si possono scegliere contemporaneamente criteri multipli.

Mentre i principali svantaggi del region growing sono i seguenti:

- il calcolo richiesto è computazionalmente dispendioso e può richiedere diverso dal tempo;
- la presenza di rumore o variazioni di intensità possono provocare buchi nell'immagine finale segmentata;
- questo metodo potrebbe non distinguere le ombreggiature nelle immagini reali.

In realtà il problema della presenza di rumore si può facilmente risolvere utilizzando apposite maschere per filtrare i buchi e/o i valori anomali. Pertanto, il problema del rumore passa in secondo piano e, se trattato adeguatamente, può anche essere ignorato. Concludendo, risulta evidente che il problema più grave nei metodi di region growing consiste nel tempo che essi impiegano.

2.3.1 Utilizzo di *Random Fields*

Nel corso degli anni i Markov Random Field (e varianti annesse) sono stati applicati in svariati ambiti e hanno trovato applicazione in differenti contesti, questo grazie all'elevato grado di astrazione che un modello a grafo comporta. Alcune delle principali applicazioni dei MRF sono: rimozione del rumore e ricostruzione di immagini [19], generazione di informazioni di disparità per la visione stereo [20], riconoscimento di oggetti, classificazione di texture e altri task legati al campo della visione artificiale [21]. Si riporta in figura 2.3 alcuni esempi di risultati di *image restoration* e generazione di mappa di disparità. Tutti questi tipi di applicazioni (stereo, segmentazione e image restoration) sono di fatto problemi di etichettatura dove si cerca di attribuire l'etichetta corretta ad ogni elemento base della scena (pixel nel caso di immagini 2D, voxel nel caso 3D). Normalmente per questo tipo di problemi i vincoli adottati, durante il processo di etichettatura, sono fortemente legati alla conformazione e le caratteristiche delle entità appartenenti al vicinato dell'elemento corrente che si desidera etichettare. In questi processi di assegnazione delle etichette questi tenderanno a variare gradualmente all'interno di regioni simili, mentre presenteranno forti discontinuità in corrispondenza di regioni di bordo; trovare l'etichettatura più probabile per tutti gli elementi presenti, tenendo conto delle loro relazioni di vicinato e della possibile presenza di rumore nei dati, è un compito che può essere ricondotto ad un problema di ottimizzazione in cui l'obiettivo è la minimizzazione di una determinata funzione energia.

Il framework messo a disposizione dai Markov Random Fields permette esattamente di fare quanto appena esposto: attraverso la definizione di un modello a grafo è possibile rappresentare tutti gli elementi da etichettare (nodi), gestire efficacemente le relazioni di vicinato che intercorrono tra questi (archi), ed associare una funzione energia che verrà conseguentemente minimizzata attraverso l'impiego di algoritmi di taglio a costo minimo applicati al grafo creato. A seguire verranno presentate le principali tecniche adottate negli ultimi anni per la segmentazione della scena attraverso MRF.

Modelli come i Markov Random Field hanno una lunga storia nel campo della segmentazione d'immagini [22, 23, 24] e l'attività di “etichettatura” automatica della scena è un argomento ampiamente studiato nell'ultimo decennio. Molto di questo lavoro è stato

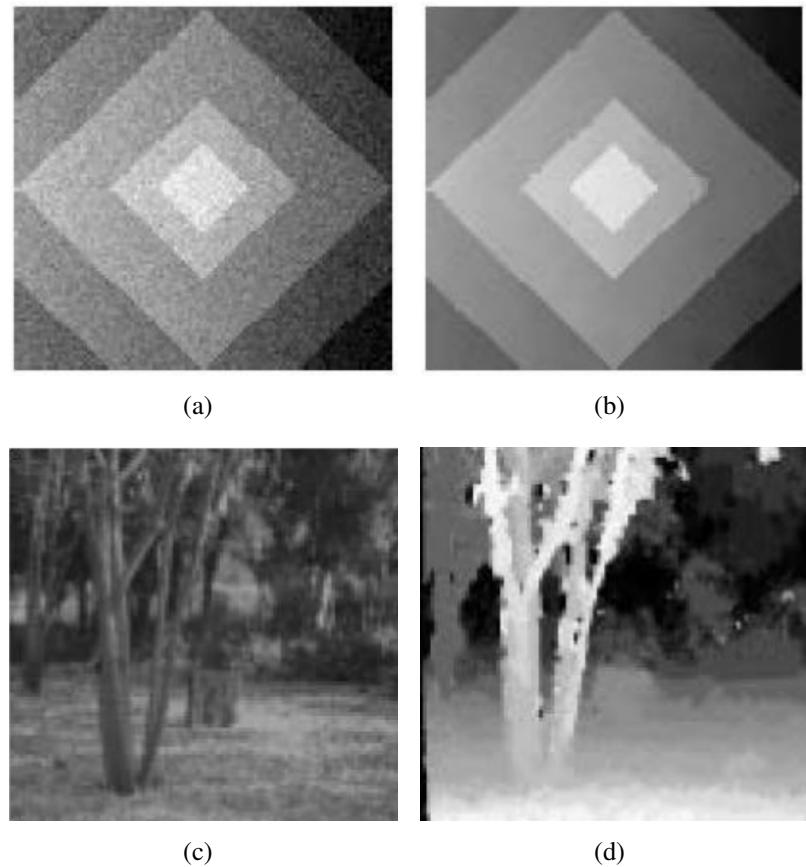


Figura 2.3: Esempi d'utilizzo dei Markov Random Fields: (a, b) applicazioni nel campo della ricostruzione d'immagini *image restoration* e (c, d) nella generazione di mappe di disparità per applicazioni stereo

svolto al fine di trovare un modello che fosse in grado di rappresentare efficacemente il contesto da trattare: l'utilizzo dei MRF [25, 26, 27, 28] e delle sue varie declinazioni come i *Conditional Random Field* (CRF) [29, 30, 31, 32, 33, 34] ed i *Discriminative Random Field* (DRF) [35, 36]: in questo tipo di approccio tipicamente si cerca di derivare una distribuzione di probabilità relativa alle etichette da assegnare o ai singoli pixel o, nel caso di una rappresentazione per porzioni dell'immagine (*patch*), ai cosiddetti *superpixel* (figura 2.4).

He et al. propongono un CRF multi-scala al fine di generare un sistema che possa imparare la disposizione delle varie etichette all'interno dell'immagine [31]. Gould et al.



Figura 2.4: Scomposizione della scena in superpixel: (a) immagine di partenza e (b) superpixel ricavati dall’immagine originale

hanno ricavato un sistema per codificare le posizioni relative tra le etichette [30] e sviluppato tecniche di inferenza per i MRF con potenziali di coppia [25] (si veda la sezione 3.1.2). Ladicky et al. hanno sfruttato una versione gerarchica dei MRF combinando pixel e regioni [27]. Anche in [37, 38] gli autori propongono un modello gerarchico multiscala. Tighe e Lazebnik [28] hanno combinato aspetti di scene recognition con MRF basati sul matching di superpixel su ampi datasets.

In diversi casi entità come i superpixel vengono largamente utilizzati sia per la loro efficienza [39] che per la loro capacità di aggregare informazioni a livello locale [40]; per migliorare ulteriormente il risultato finale di segmentazione sono stati proposti anche approcci basati su segmentazioni multiple della stessa scena [41, 26] o su tecniche di organizzazione gerarchica delle regioni che la compongono [42].

Altri tipi di applicazione di random field finalizzati alla segmentazione sono quelli esposti in [43, 44, 45], dove in questo caso vengono sfruttate le informazioni ricavate da più tipi di sensore al fine di risolvere efficacemente un problema di navigazione automatica su strada di veicoli intelligenti. In [43] sfruttano principalmente le informazioni relative alla morfologia del terreno (ricavate tramite lidar) per determinarne o meno la percorribilità mentre negli ultimi due articoli citati vengono considerati anche altri tipi di dato come, per esempio, quelli derivanti da proiezioni omografiche congiunte di ciò che si sta osservando.

L’arrivo sul mercato del *Microsoft Kinect* [1] e altri sensori a basso costo [46] stanno influenzando il panorama della ricerca scientifica in ambienti quali la robotica e la visio-

ne artificiale; il lavoro originariamente svolto sul Kinect ha riguardato esclusivamente il riconoscimento di pose utilizzando unicamente dati di profondità [47]. Il punto di contatto tra la visione artificiale e la robotica lo troviamo in applicazioni di percezione tramite dati RGB-D che combinano colore e range per compiti di mapping e modellazione 3D [48], riconoscimento d'oggetti [49] e segmentazione di nuvole di punti [50]. Per quanto riguarda il problema dell'etichettatura di ambienti interni, Silberman e Fergus [51] hanno effettuato studi intensivi su imponenti dataset come lo *NYU Depth Dataset V2* [52] sfruttando congiuntamente features SIFT ed i MRF.



Figura 2.5: Esempio di segmentazione delle scene attraverso l'utilizzo di Markov Random Fields: (a) immagine di partenza e (b) risultato finale di segmentazione, immagine tratta da [53]

Capitolo 3

Descrizione del problema

L’obiettivo di questa tesi è realizzare un algoritmo per la segmentazione di scene tridimensionali, basato su estrazione di features e Markov Random Fields. Il risultato della segmentazione è una point cloud suddivisa in parti, a ciascuna delle quali corrisponde un’etichetta. L’etichettatura permette di aggregare le parti elementari in cui la scena è suddivisa in sottoinsiemi aventi la stessa etichetta. L’interpretazione di questi sottoinsiemi, per esempio la corrispondenza tra etichette ed oggetti, elementi architettonici ed altre entità, non rientra tra i compiti del sistema. La segmentazione realizzata è di più alto livello e comunque consistente con compiti più specifici.

Il sistema realizzato riceve contemporaneamente in input immagini RGB e di profondità relative all’ambiente osservato; durante questa fase di acquisizione sarà restituita un’abbondante quantità dati che verranno successivamente rielaborati ed organizzati in nuvole di punti tridimensionali. Una volta ottenuto questo particolare tipo dato sensoriale, è necessaria una fase di indicizzazione e suddivisione dei dati ottenuti al fine di poter rendere efficiente la gestione della cloud durante l’esecuzione del processo di segmentazione. Avendo a che fare con dati definiti in uno spazio tridimensionale la suddivisione più sensata in questo caso è la ripartizione della nuvola in una struttura regolare di *voxel* i quali conterranno al loro interno un sottoinsieme finito di punti appartenenti alla cloud. L’esigenza di dover riconoscere ed etichettare adeguatamente tutte le parti “principali” della scena, rende necessaria la definizione di uno step intermedio nel svolgere un’atti-

vità d'estrazione delle caratteristiche maggiormente distintive per ogni voxel ricavato. Una volta calcolati questi *descrittori* sarà possibile passare ad una fase di classificazione nella quale si cercherà di assegnare ognuno di questi volumi elementari ad una determinata categoria d'appartenenza. Per questo passo di classificazione sono state adottate due differenti strategie: una di tipo supervisionato e una di tipo non supervisionato. La prima permette la ripartizione di ogni elemento in una delle possibili classi d'appartenenza (definite a priori dall'operatore); questo tipo d'approccio permette l'ottenimento di una segmentazione finale dell'ambiente osservato in macro-classi come, ad esempio, muri, porte e pavimento. Nel nel secondo caso invece è possibile ottenere la suddivisione dei dati raccolti in classi lasciando che sia lo stesso sistema a ricavarle in autonomia. Questo tipo di operare consente di ottenere, invece di una segmentazione definitiva in macro-classi, un risultato intermedio, utile all'interno di una sistema più grande, che possa essere utilizzato come dato d'ingresso per un processo di analisi e/o aggregazione delle parti ricavate.

Va osservato che l'associazione di ognuno di questi elementi ad una specifica classe non sia indipendente per ogni voxel, ma è lecito supporre che, almeno localmente, elementi a cui corrispondono caratteristiche simili dovranno essere ripartiti all'interno della stessa categoria. Il rumore presente nei dati rilevati durante il passo d'acquisizione e l'inevitabile grado d'incertezza (per quanto piccolo questo possa essere) presente nei classificatori ottenuti, fa sì che la classe di appartenenza d'un voxel non possa essere definita esclusivamente dal risultato di classificazione (in quanto questo potrebbe risultare errato), ma debba dipendere anche dalle categorie scelte per gli elementi "vicini" a quello considerato. Il modello definito dai Markov Random Fields permette la considerazione di questo aspetto al fine di ottenere un risultato di segmentazione finale il più fedele possibile a quello atteso. Il modello probabilistico a grafo, sul quale si basa la formulazione dei MRF, permette di rappresentare efficacemente non solo le relazioni di vicinato che intercorrono tra i voxel ricavati, ma anche l'esito della classificazione effettuata singolarmente per ogni elemento.

Dato il ruolo di fondamentale importanza che i MRF ricoprono all'interno di questa tesi, inizialmente verrà presentata una trattazione di base della teoria alla base di questi

(con particolare attenzione al loro utilizzo nell’ambito dei processi di segmentazione) e saranno descritte le motivazioni per le quali questo strumento permette di ottenere in maniera elegante il risultato finale di etichettatura della scena.

Successivamente all’introduzione dei MRF si analizzeranno gli strumenti hardware utilizzati per l’acquisizione dei dati e verrà presentata al lettore una panoramica dei componenti sviluppati e librerie utilizzate per l’elaborazione delle informazioni acquisite mediante i sensori adoperati. Si descriverà l’architettura complessiva del sistema realizzato e verrà esplicitato il ruolo svolto dai vari componenti durante il normale flusso d’esecuzione dell’applicazione. Infine, saranno passate in rassegna tutte le varie fasi dell’algoritmo con particolare attenzione alla descrizione dei vari aspetti, teorici ed implementativi, che hanno portato alla sua realizzazione finale: saranno quindi coperti tutti gli step intermedi eseguiti, a partire dalla fase di raccolta dei dati sensoriali, fino all’ottenimento dell’output conclusivo cioè la scena segmentata.

3.1 Teoria ed impiego dei Markov Random Field in applicazioni di segmentazione

I Markov Random Field sono un caso speciale di grafo $\mathcal{G}(\mathcal{V}, \mathcal{E})$, non orientato, che fonda le proprie caratteristiche in ambito probabilistico, tale struttura ricopre un’importante ruolo nel campo dell’apprendimento automatico causa delle caratteristiche che verranno descritte a breve; per una trattazione completa e esauriente sui modelli a grafo (e come questi siano spesso legati al campo del *machine learning*) si rimanda il lettore a [54] e [55].

I MRF sono stati spesso utilizzati in diversi tipi di applicazioni (per esempio nell’analisi d’immagini, problemi di detection/recognition, ecc.) grazie alla loro capacità di catturare e rappresentare facilmente le relazioni di contesto o vicinato presenti tra gli elementi che lo formano e al buon comportamento che questi modelli hanno in eventuale presenza di rumore. Data l’estrema duttilità di questo particolare tipo di modello, è possibile definire grafi dalle più disparate topologie, questo al fine di generare un grado di connettività tra nodi che modelli al meglio le interazioni fra le entità che si vogliono

etichettare all'interno della scena. Il compito di etichettatura attraverso l'impiego dei MRF rappresenta un caso di *segmentazione basata su regioni* dove queste sono rappresentate dalle variabili aleatorie che compongono il modello, il quale a sua volta, viene risolto come problema d'inferenza tramite un criterio di *Stima del massimo a posteriori* (MAP) [56].

Un tipico Markov Random Field (in figura 3.1 un caso di MRF regolare a griglia) è costituito da un insieme di nodi, rappresentante un insieme di variabili casuali (o *variabili aleatorie VA*), $\mathbf{X} = \{X_1, X_2, \dots, X_n\}$ dove ad ogni variabile $X_i \in \mathbf{X}$ viene assegnato un valore dall'insieme delle possibili classi (o etichette) $\mathcal{L} = \{l_1, l_2, \dots, l_k\}$; come verrà descritto più avanti nel nostro caso queste categorie d'appartenenza saranno ricondotte a vari tipi di entità concrete quali *muro*, *porta*, *pavimento*, ecc. Una “etichettatura” \mathbf{x} consiste in una qualsiasi delle possibili configurazioni d'assegnamento dei nodi ai label appartenenti all'insieme $\mathbf{L} = \mathcal{L}^N$.

Il modello è definito da un grafo $\mathcal{V} = \{n_1, n_2, \dots, n_N\}$ dove ogni nodo n_i è associato ad una variabile aleatoria X_i . Nel seguito confonderemo il nodo con la VA facendo abuso di notazione.

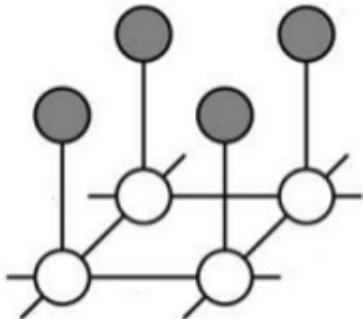


Figura 3.1: MRF bidimensionale a griglia: i vari nodi del random field (bianchi) e le relative etichette ad essi associati (grigio)

Sia \mathcal{N} l'insieme dei vicini del *random field* ed \mathcal{N}_i , l'insieme dei nodi adiacenti (e le VA ad esso associati) al nodo X_i , detto anche *vicinato di X_i* . Una *clique* c è definita come l'insieme di tutte le variabili \mathbf{X}_c condizionalmente dipendenti tra loro.

Si definisce inoltre la probabilità di un'etichettatura $\mathbf{X} = \mathbf{x}$ come $\text{Pr}(\mathbf{x})$ e quella di un'etichettatura $X_i = x_i$ come $\text{Pr}(x_i)$. Un random field è detto un Markov Random

Field rispetto ad un vicinato \mathcal{N} se e solo se soddisfa le seguenti proprietà:

$$\Pr(\mathbf{x}) > 0, \forall \mathbf{x} \in \mathbf{L} \quad (3.1)$$

$$\Pr(x_i | \{x_j : j \in \mathcal{V} \setminus \{i\}\}) = \Pr(x_i | \{x_j : j \in \mathcal{N}_i\}), \forall i \in \mathcal{V} \quad (3.2)$$

Dove la prima definisce la condizione di *positività* tipica della teoria probabilistica, mentre la seconda rappresenta un'estensione del concetto di *Markovianità*: la probabilità di assegnazione ad una certa classe di un nodo del grafo, condizionata all'etichettatura di tutti gli altri nodi, dipende esclusivamente dalla classificazione degli elementi appartenenti al vicinato del nodo in questione; in figura 3.2 una rappresentazione visuale della proprietà appena enunciata.

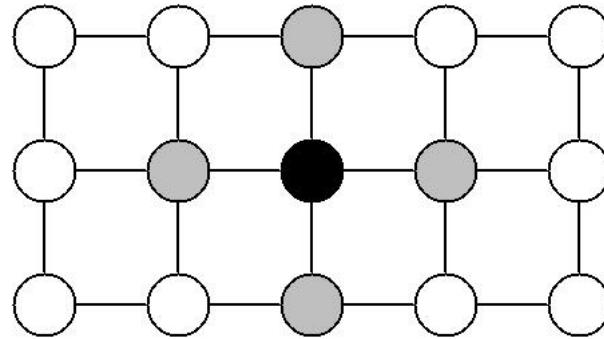


Figura 3.2: MRF bidimensionale a griglia e proprietà di Markovianità: dati i nodi (grigi) appartenenti al 4-vicinato del nodo centrale (nero), questo è condizionalmente indipendente da tutti gli altri elementi del grafo

Una volta trovata una possibile segmentazione $\mathbf{X} = \mathbf{x}$, che fa corrispondere ogni nodo del grafo ai corrispondenti label, la distribuzione di probabilità associata all'insieme delle VA può essere espressa dal prodotto di fattori $\Psi_c(\cdot)$ definiti sulle clique massimali del grafo \mathcal{G} .

Tale distribuzione è quindi espressa da:

$$\Pr(\mathbf{X} = \mathbf{x}) = \prod_{c \in \mathcal{C}} \Psi_c(\mathbf{x}_c) \quad (3.3)$$

Il termine $\phi_c(\mathbf{x}_c)$ prende il nome di *funzione potenziale* della clique c dove $\mathbf{x}_c = \{x_i, i \in c\}$. Inoltre, nel caso specifico in cui le funzioni potenziale siano strettamente positive, secondo il teorema di *Hammersley-Clifford* [57] il MRF può essere associato ad una *distribuzione di Gibbs* del tipo:

$$\Pr(\mathbf{X} = \mathbf{x}) = \frac{1}{Z} e^{-\sum_{c \in C} \Psi_c(\mathbf{x}_c)} \quad (3.4)$$

Dove Z è una costante di normalizzazione (o *partition function*) data da:

$$Z = \sum_{\mathbf{X} \in \mathbf{X}} e^{-\sum_{c \in C} \Psi_c(\mathbf{x}_c)} \quad (3.5)$$

Definita la 3.4 è possibile associare ad essa la relativa *energia di Gibbs* $E(\mathbf{x})$ definita come:

$$E(\mathbf{x}) = -\log \Pr(\mathbf{X} = \mathbf{x}) - \log Z = \sum_{c \in C} \Psi_c(\mathbf{x}_c) \quad (3.6)$$

Come già accennato all'inizio del capitolo è possibile ricavare l'etichettatura più probabile \mathbf{x}^* attraverso un criterio di massimo a posteriori (MAP) definito nel seguente modo:

$$\mathbf{x}^* = \arg \max_{x \in \mathbf{L}} E(x) \quad (3.7)$$

L'etichettatura ottimale della scena viene ricavata attraverso la minimizzazione, ottenibile mediante diverse possibili tecniche di inferenza, della funzione energia appena introdotta.

Arrivati a questo punto è quindi noto come un approccio che sfrutti i MRF si basi sulla definizione di una funzione energia (eq. 3.6), da “modellare” opportunamente, a seconda delle applicazioni per le quali si sta utilizzando questo particolare modello. Nei problemi di segmentazione automatica si ha la necessità di ricavare l'etichettatura migliore per la scena sulla base dei dati osservati, i quali però possono essere incompleti o affetti da rumore. In questi casi la funzione energia (da minimizzare) consiste nel-

la somma di *potenziali unari* ψ_i e *potenziali di coppia* ϕ_{ij} definiti tra i nodi del grafo; complessivamente la funzione energia assume una forma del tipo:

$$E(\mathbf{x}) = \sum_{i \in \mathcal{V}} \psi_i(x_i) + \sum_{(i,j) \in \mathcal{E}} \phi_{ij}(x_i, x_j) \quad (3.8)$$

Dove \mathcal{V} rappresenta l'insieme dei nodi e \mathcal{E} è indicata tutte le coppie di VA tra loro dipendenti le quali formano il set complessivo dei vicinati \mathcal{N} . Il vantaggio nell'utilizzo dei MRF sta quindi nella capacità di considerare, oltre al costo di associazione definito dai potenziali unari, anche il “contesto” nel quale si trova un determinato nodo in base alla composizione locale del suo vicinato e regolare appropriatamente il valore delle funzione energia in ogni zona del random field. Nello specifico scenario di applicazione dei MRF ad un processo di segmentazione, ogni label rappresenterà le diverse categorie di appartenenza (alle quali potranno essere associati i vertici del grafo) ed ogni possibile assegnazione dei nodi del MRF daranno origine ad una determinata segmentazione complessiva della scena.

3.1.1 Il potenziale unario

Il potenziale unario $\psi_i(x_i)$ rappresenta il costo derivante dell'assegnamento $X_i = x_i$, cioè il costo relativo all'assegnamento di un particolare label x_i alla variabile casuale X_i all'interno del grafo: a seconda degli strumenti a disposizione, i tipi di dati trattati, e del tipo di applicazione, possono essere definite svariate funzioni di costo capaci di quantificare l'onerosità dell'attribuzione di una classe ad un elemento del random field; siccome l'obiettivo finale è quello di minimizzare la quantità rappresentata da $E(\mathbf{x})$, la funzione scelta dovrà essere formulata in maniera tale da associare un costo basso (o eventualmente negativo in certi casi [43]) nel caso di assegnazione di un nodo ad un label “corretto”, mentre dovrà restituire un valore alto, quindi penalizzante, nel caso di corrispondenza con un'etichetta poco probabile. Nel corso di questa tesi sarà esposta la tecnica adoperata per la generazione di questo potenziale.

3.1.2 Il potenziale di coppia

Il potenziale di coppia $\phi_{ij}(x_i, x_j)$ esprime il costo derivante dagli assegnamenti congiunti $X_i = x_i$ e $X_j = x_j$; solitamente ci si riferisce a questo potenziale anche con il nome di *smoothness term*: questo perché tende a favorire elementi contigui appartenenti alla stessa classe mentre penalizza situazioni di discontinuità tra nodi adiacenti (cioè appartenenti allo stesso sottoinsieme di vicinato) ma assegnati a classi d'appartenenza diverse. Un esempio ricorrente di questo tipo di funzione è il modello di Potts [58]. Verranno ora esposti i modelli, relativi ai potenziali di coppia, più comunemente usati nell'ambito della segmentazione automatica e, di seguito, verrà riportata la definizione (ed il relativo grafico) dei tipi di modelli maggiormente utilizzati.

Il modello di Potts è quello concettualmente più semplice e matematicamente si traduce nel seguente set di equazioni:

$$\phi_{ij}(x_i, x_j) = \begin{cases} 0, & |x_i - x_j| \leq 1 \\ \lambda, & \text{altrimenti} \end{cases} \quad (3.9)$$

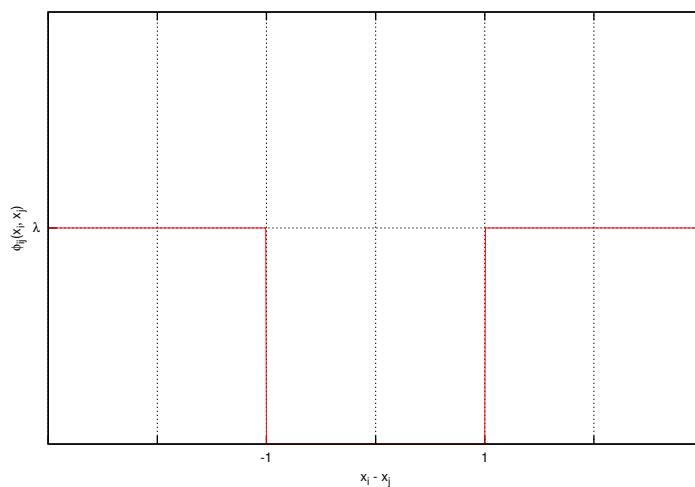


Figura 3.3: Grafico rappresentante un modello di Potts standard

Altri tipi di modello comunemente utilizzati sono i cosiddetti *modelli troncati* che solitamente sono “declinati” in *lineare* (figura 3.4):

$$\phi_{ij}(x_i, x_j) = \lambda \min(|x_i - x_j|, K) \quad (3.10)$$

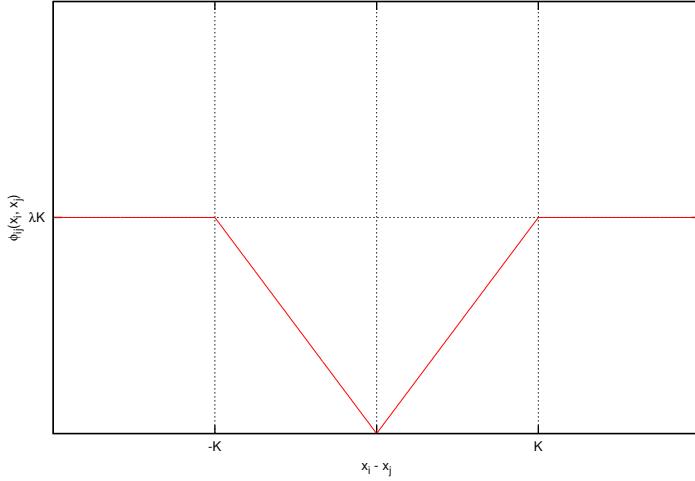


Figura 3.4: Grafico rappresentante un modello lineare troncato

e *quadratico* (figura 3.5):

$$\phi_{ij}(x_i, x_j) = \lambda \min((x_i - x_j)^2, K) \quad (3.11)$$

Ognuno di questi modelli presenta uno o più gradi di libertà che danno la possibilità di regolare il potenziale di coppia in base ai valori assegnati ai diversi parametri: il modello di Potts è una funzione binaria a parametro singolo λ . Questo valore controlla il grado di “smoothness” da applicare in caso di nodi adiacenti assegnati a label differenti. I modelli lineare e quadratico hanno invece un parametro extra K che permette di regolare anche il limite massimo di penalità della funzione potenziale. Ovviamente questi tipi di modello possono essere ulteriormente generalizzati definendo funzioni *ad-hoc* del tipo $\phi_{ij}(x_i, x_j; \Omega_{ij})$ che, oltre a considerare l’etichetta associata ai nodi “vicini” x_i e x_j , prendono in considerazione anche altri tipi di informazione derivanti, per esempio, da un insieme di features precalcolato $\Omega_{ij} = \{\omega_{ij}^1, \omega_{ij}^2, \dots, \omega_{ij}^m\}$ relativo ai nodi i e j ; soluzioni di questo tipo si possono trovare in [45] e [59]. Un’ulteriore considerazione può essere fatta su quale modello di potenziale possa risultare più “conveniente”: di fatto

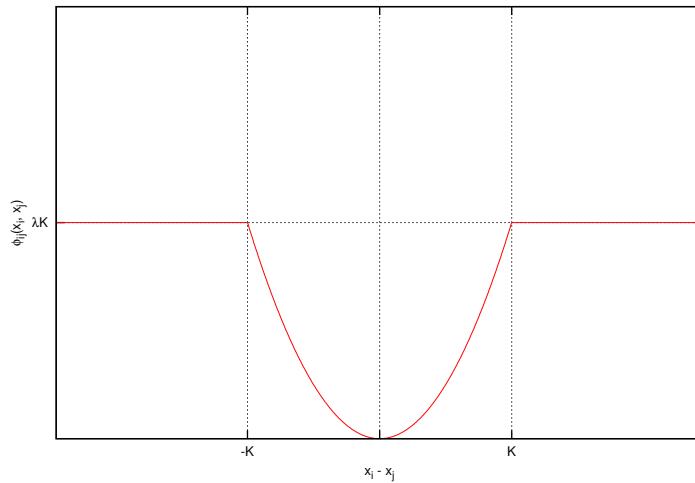


Figura 3.5: Grafico rappresentante un modello quadratico troncato

non ne esiste uno migliore in assoluto, a seconda del caso d'utilizzo del MRF possono essere più performanti alcuni tipi di modello piuttosto che altri.

In questa sezione si è voluto descrivere sommariamente la teoria alla base dei Markov Random Field ed una delle loro possibili modalità d'impiego. Esiste una vasta letteratura riguardante gli aspetti teorici dei MRF e le loro possibili applicazioni che, per ovvi motivi, non verrà esposta in questo lavoro; per ulteriori approfondimenti si rimanda il lettore a [3].

3.2 Il sistema d'acquisizione

Il sistema d'acquisizione adoperato per questo progetto è costituito dal sensore *Microsoft Kinect* [1]. Inizialmente finalizzato esclusivamente a scopo videoludico, l'uscita sul mercato di un sensore RGB-D a basso costo come questo ha avuto un impatto notevole anche in ambito scientifico dove ha trovato impiego in numerosi scenari applicativi.



Figura 3.6: Il sensore Microsoft Kinect

Il dispositivo è composto da svariati tipi di sensore come:

- un array di microfoni orientati verso il basso (tre sul lato destro e uno sul lato sinistro);
- tre apparati ottici utilizzati per ricavare le informazioni di *colore* e *range* di cui: un proiettore IR, una camera a colori ed un sensore ad infrarossi;
- un accelerometro *Kionix KXSD9* a tre assi utilizzato per controllare l'inclinazione e stabilizzare l'immagine;
- un chip *Prime Sense PS1080-A2* che rappresenta il “cuore” della tecnologia Kinect.

La disposizione dei principali componenti ottici e meccanici del sensore sono riportati in seguito in figura 3.7, mentre una raffigurazione ad alto livello dello schema architetturale dell'hardware del dispositivo è quello riportato in figura 3.8.

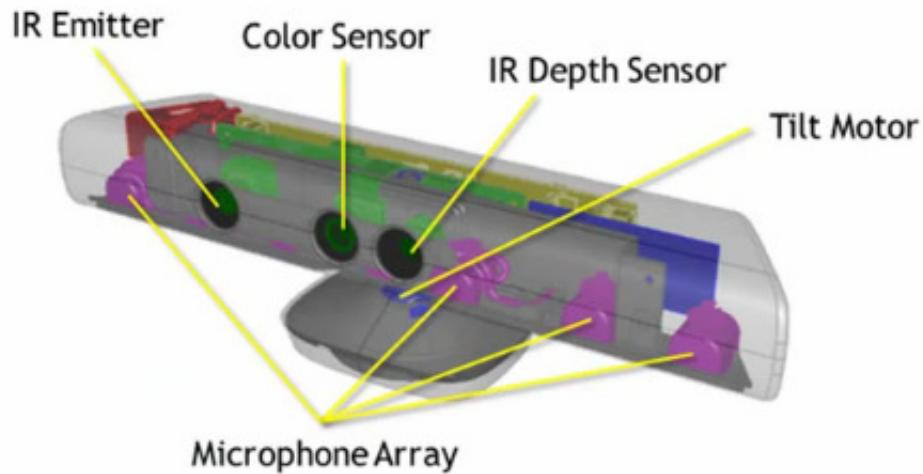


Figura 3.7: Schema a colori dei componenti principali del sensore Microsoft Kinect

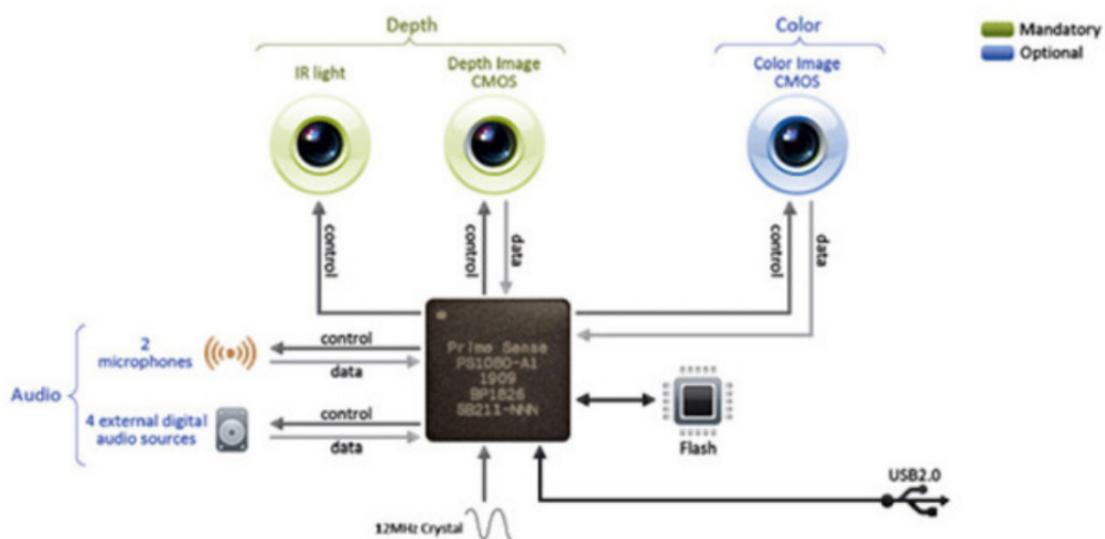


Figura 3.8: In figura è riportata l'architettura hardware del sensore Microsoft Kinect

Da questa rappresentazione viene evidenziato come il già citato *PSI080-A2* sovraintenda tutta la procedura di analisi della scena, controllando adeguatamente i sistemi ottici ed audio, al fine di raccogliere le informazioni di cui necessita.

Più in dettaglio il corredo di apparati ottici di Kinect si compone di una telecamera RGB ed un sensore di profondità a raggi infrarossi. Tale sensore è composto in realtà da due componenti distinti: un proiettore IR ed una camera sensibile alla stessa banda: la camera non fa altro che registrare il pattern di luce strutturata (non visibile) emesso dal proiettore ad infrarossi; data la natura del sensore range si possono rilevare informazioni di profondità anche in condizioni di luce non ottimali. Grazie all'utilizzo congiunto di diversi tipi di sensore (nello specifico la telecamera RGB ed il doppio sensore IR) questo dispositivo è in grado di restituire informazioni sincronizzate di colore e profondità relativamente alla scena osservata. Il Kinect non si limita quindi solamente alla generazione di un'immagine bidimensionale ma, grazie all'eterogeneità dei dati da esso ricavati è possibile ottenere una ricostruzione tridimensionale, densa e a colori dell'ambiente osservato, il tutto in tempo reale.

In un'ampia categoria di problemi la disponibilità di questo tipo d'informazione ha incrementato significativamente accuratezza e robustezza di sistemi sviluppati per differenti finalità; alcuni esempi di questi scenari di applicazione sono: stima di posture corporee [47], mapping 3D [60], riconoscimento d'oggetti [49], [59], ricostruzione 3D [48], e riconoscimento di *gesture* [61].

3.3 Software e librerie utilizzate

In questa sezione si vuole dare al lettore una panoramica sul software utilizzato durante lo sviluppo del progetto. Il sistema realizzato infatti integra sotto lo stesso framework tutte le varie librerie che verranno elencate in seguito: nella realizzazione di questo lavoro è stata quindi svolta un'importante attività tesa all'integrazione delle varie componenti software di cui ci si è serviti.

Ognuna di queste librerie assolve, all'interno del flusso di programma, uno o più ruoli ben precisi come, fungere da infrastruttura per la gestione della point cloud, elaborazione d'immagini, compiti di classificazione (supervisionata e non supervisionata), generazione di Markov Random Field e tecniche d'inferenza per la minimizzazione della funzione energia ad essi associata.

Nei prossimi paragrafi verrà data un breve descrizione delle funzioni principali assolute da ogni componente per poi poter passare alla descrizione dettagliata dell'algoritmo sviluppato.

3.3.1 Point Cloud Library

Point Cloud Library (PCL) [2] è un framework *open-source* standalone sviluppato da *Willow Garage*, che implementa al suo interno una moltitudine di algoritmi allo stato dell'arte per la gestione di point cloud n -dimensionali ed elaborazione geometrica 3D. La libreria si compone di algoritmi di filtraggio, calcolo di features, ricostruzione di superfici, registrazione di nuvole di punti, segmentazione, e altri strumenti per la lavorazione efficace ed efficiente di point cloud.

Le PCL rivestono ruoli di fondamentale importanza all'interno di questo progetto: in prima istanza, grazie alla classe *pcl::Grabber*, permette di ottenere immediatamente la nuvola di punti 3D, e la corrispondente immagine RGB, di ciò che si sta inquadrando con il Kinect; la praticità nell'utilizzo di questa classe consiste nel fatto che si astrae completamente lo sviluppatore dai problemi di interfacciamento a livello hardware col sensore, ritornando immediatamente i dati percepiti in strutture dati organizzate, immediatamente fruibili per lo sviluppo software.

Un altro aspetto, che rende essenziale l'utilizzo delle PCL, è quello di mettere a disposi-

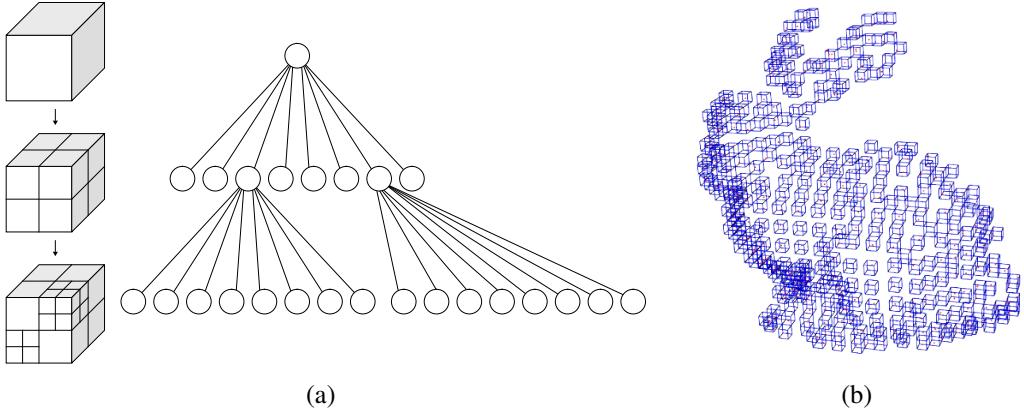


Figura 3.9: Esempio di octree: (a) il volume di spazio che si intende modellare e la struttura dati corrispondente, (b) Visualizzazione di una cloud d'esempio mediante la struttura dati octree di PCL

zione un particolare strumento dati per il partizionamento e l'organizzazione efficiente della nuvola di punti: tale strumento prende il nome di *octree* [62]. Gli octree, come si può intuire dal nome, sono strutture dati ad albero dove ogni nodo (non foglia) ha esattamente otto figli; a causa di questo tipo d'organizzazione gli octree sono spesso usati per partizionare lo spazio tridimensionale suddividendolo ricorsivamente in ottanti (fino al raggiungimento di una soglia minima ξ che rappresenta la lunghezza dello spigolo del cubo di dimensione “unitaria”).

Il vantaggi nell'utilizzo degli octree implementati in PCL (figura 3.8) sono diversi:

- permettono la scomposizione di una nuvola di punti in una griglia di *voxel* di volume ξ^3 ;
- danno la possibilità di accedere ad ogni voxel della cloud in maniera rapida (con complessità $N \log N$, dove N è il numero di voxel) consentendo controlli efficienti sull'esistenza di punti in precise zone dello spazio;
- possono ricavare la numerosità di punti limitatamente a volumi pari ad un voxel in una qualsiasi regione di ricerca.

La versione adottata è attualmente la 1.7.0.

3.3.2 OpenCV

Le *Open Computer Vision Library* [63] sono librerie sviluppate da *Intel* e ora mantenute da *Willow Garage* che costituiscono un framework completo per la visione artificiale. In questo progetto le OpenCV sono utilizzate principalmente per la generazione di istogrammi (che serviranno per rappresentare la distribuzione relativa a determinati sottospazi di features), conversione tra spazi di colore differenti e generazione/applicazione di filtri, anche custom, alle immagini acquisite. Gli istogrammi saranno necessari alla generazione dei vettori di *features* che verranno associati ad ogni voxel, la scelta di passare ad un altro spazio di colore è stata motivata dal fatto che lo spazio di destinazione scelto, cioè l'HSV, si rivela più robusto alle variazioni d'illuminazione: questo grazie alla separazione di ogni pixel in tre canali, uno per il colore (*H*), uno per la saturazione (*S*), cioè l'intensità del colore, e l'ultimo (*V*) che definisce la sua brillantezza. Il motivo per cui risulta conveniente il passaggio a questo spazio di colore è che quest'ultimo, grazie alla gestione del colore tramite un solo canale (e non tre come l'RGB), gode di proprietà di invarianza rispetto a cambiamenti d'illuminazione, ovviamente a patto che questi non siano eccessivi. Questa rappresentazione di colore è spesso utilizzata in situazioni dove si vuole aumentare la robustezza del sistema di percezione a fronte della presenza di ombre e/o effetti di mutevolezza d'illuminazione che potrebbero inficiare l'analisi dell'immagine [64].

Infine la libreria consente l'applicazione alle immagini dei filtri più comuni utilizzati nella Computer Vision (come il filtro gaussiano, mediano, Sobel, ecc...), ma rende possibile anche la definizione di *kernel custom* permettendo così l'implementazione di sistemi di filtraggio efficienti e completamente personalizzabili; più avanti sarà presentato il filtro implementato nella fase di estrazione delle features.

Infine le OpenCV costituiranno anche una parte importante di questo progetto nella realizzazione del processo di *classificazione non supervisionata* (tramite l'implementazione dell'algoritmo *K-Means* [65]).

Per questa libreria è stata adottata la versione 2 . 4 . 0.

3.3.3 FANN: Fast Artificial Neural Network

Come suggerisce il nome questa libreria [66] viene sfruttata per la realizzazione, addestramento e l’impiego di *Reti Neurali* (o Neural Network, NN) [67] anch’esse utilizzate all’interno della fase di *classificazione* ma in questo caso *supervisionata*. FANN permette la gestione di diversi tipi di reti multistrato: completamente o parzialmente connesse; implementa inoltre algoritmi di *back-propagation* allo stato dell’arte per il training della rete.

Per questo progetto è stata utilizzata FANN alla versione 2 . 2 . 0.

3.3.4 Tiny SVM

Tiny SVM [68] costituisce un’alternativa alle reti neurali per quanto riguarda la parte di classificazione supervisionata; essa infatti permette di realizzare “moduli” di classificazione mediante l’utilizzo di *Support Vector Machines* (SVM) [69].

La versione impiegata è la 0 . 0 9.

Tra le varie librerie esistenti in grado di realizzare NN e SVM, la scelta è ricaduta su queste due appena presentate per diversi motivi:

- semplicità di utilizzo ed “inserimento” all’interno di codice personale;
- il formato di file atteso per la procedura di addestramento, di facile generazione;
- possibilità di salvare su file di testo, in formato proprietario, i classificatori creati ed addestrati per poterli memorizzare e ricaricare in momenti distinti durante l’utilizzo del sistema.

3.3.5 GCO: Graph-Cut Optimization

La *Graph-Cut Optimization Library* [70] è una libreria sviluppata da Olga Veksler e Andrew Delong, per l’ottimizzazione di funzioni energia multi-label attraverso l’utilizzo di algoritmi d’inferenza come l’ α -*Expansion* ed l’ $\alpha\beta$ -*Swap* [71]. Il pacchetto consente la creazione di grafi con un numero arbitrario di nodi, da la possibilità di definire come

questi siano interconnessi tra loro e consente la gestione di potenziali unari e di coppia. Una volta definiti questi aspetti è possibile passare alla minimizzazione dell'energia associata al grafo (scegliendo una delle due tecniche sopracitate), per ottenere il risultato finale di labellizzazione. Per comprendere al meglio la teoria alla base del funzionamento di questa libreria, si rimanda il lettore a [72] e [73].

La versione di GCO utilizzata è la 3 . 0.

3.3.6 Riepilogo dell'architettura applicativa

Come può risultare evidente il grosso del lavoro è svolto dalle PCL, che fungono da infrastruttura principale (come una sorta di *backbone* applicativo), per la raccolta, visualizzazione e gestione/elaborazione dei dati acquisiti tramite Kinect; le OpenCV completano l'insieme dei moduli atti alla trasformazione delle informazioni avendo però anche un ulteriore compito nella parte di classificazione non supervisionata. Per quanto riguarda invece la parte supervisionata gli attori principali sono senza alcun dubbio FANN e TinySVM con le relative implementazioni di reti neurali e macchine a vettori di supporto. Infine GCO si occupa della fase relativa alla realizzazione del MRF associato alla cloud e mette a disposizione gli strumenti per poter fare inferenza su di esso.

Nel corso di questo capitolo sono state presentati i principali ruoli svolti da ognuna delle librerie utilizzate ma ovviamente, oltre all'impiego di queste, è stata necessaria un'attività di sviluppo di codice (aggiuntivo a quello scritto per la parte algoritmica) che fosse orientata a favorire l'integrazione sotto un unico progetto di tutti questi componenti. Le strutture della classi definite a supporto di quest'attività di collaborazione sarà esplicitata nel prossimo capitolo man mano che verranno esposti i vari step che compongono l'algoritmo implementato.

Per concludere, in figura 3.10 si presenta un diagramma che illustra, ad alto livello, le funzioni svolte dalle librerie finora descritte e presenta una mappa concettuale di come queste risultino funzionalmente interconnesse tra loro.

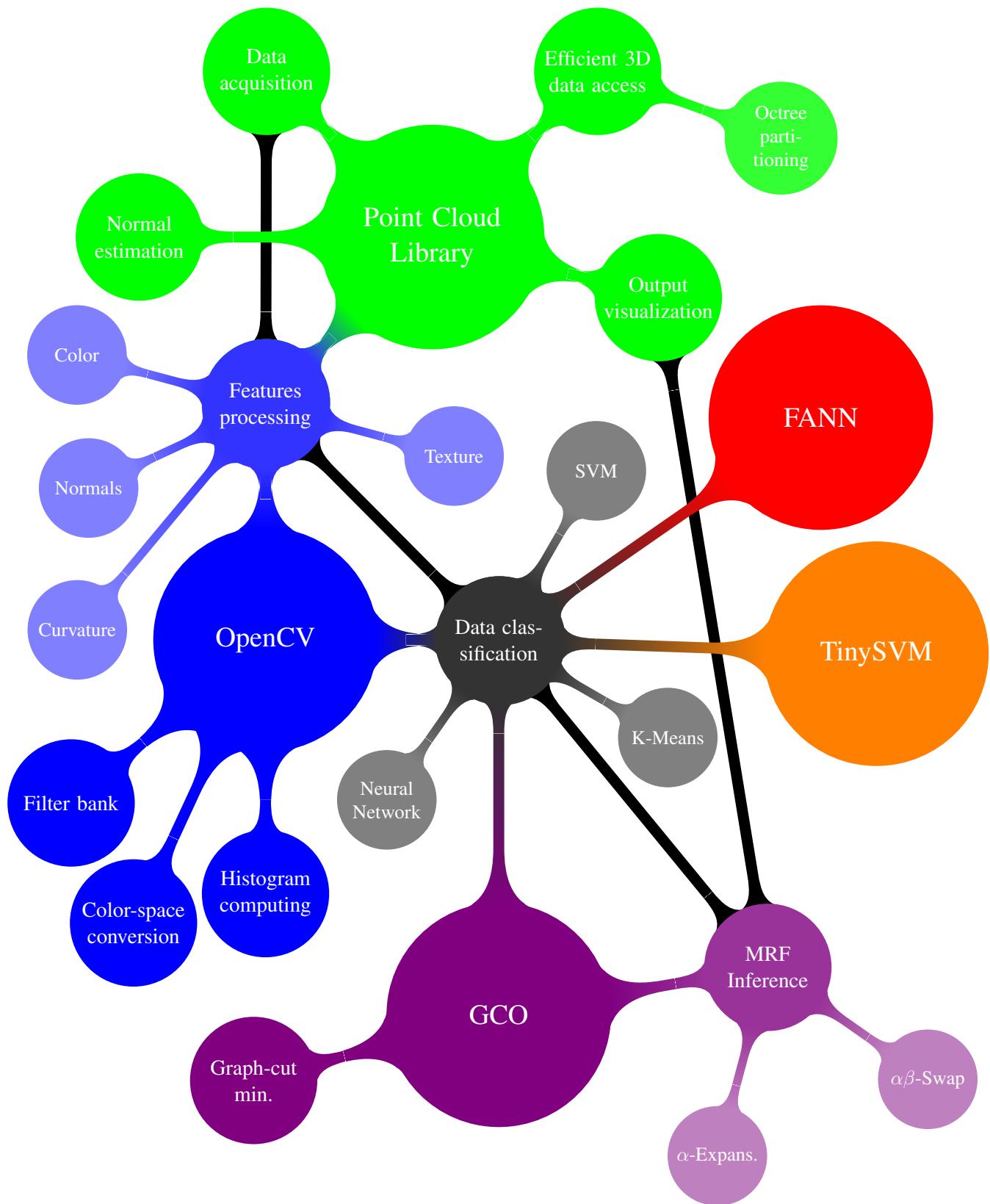


Figura 3.10: Il diagramma che esprime le relazioni funzionali tra le varie librerie utilizzate e descrive brevemente le operazioni compiute mediante il loro utilizzo (in nero un accenno al flusso principale dell'algoritmo)

Capitolo 4

L'algoritmo realizzato

In questo capitolo si darà al lettore una descrizione dettagliata dell'algoritmo che sta alla base del sistema realizzato, verranno spiegate tutte le operazioni seguite dal programma per ottenere il risultato finale e saranno riportate alcune parti di codice, relative alle principali classi implementate, al fine di dare una visione esaustiva ed il più completa possibile del lavoro svolto.

4.1 Overview dell'algoritmo

Prima di descrivere nel dettaglio l'algoritmo si riporta una descrizione ad alto livello i passi che lo compongono. Il flusso di programma realizzato può essere scomposto nelle cinque seguenti macro-fasi:

- *Data acquisition*: fase nella quale si prelevano le istantanee (immagine RGB e nuvola di punti) che verranno elaborate tramite l'utilizzo del Microsoft Kinect;
- *Preprocessing*: fase in cui vengono effettuate alcune operazioni preliminari necessarie alla “preparazione” dei dati (incapsulamento della cloud in un octree e la suddivisione in voxel) ed il calcolo di parametri utili all’elaborazione successiva (come l’operazione di stima delle normali della cloud), che presentano la caratteristica di dover essere calcolate solamente una volta;

- *Feature extraction*: si calcolano i vettori di features che costituiscono una sorta di “firma” distintiva per ogni voxel; questi descrittori multidimensionali esprimono le caratteristiche, di una parte minima (elementare) della scena, ed esprimono informazioni di colore, tessitura e forma;
- *Classification*: ottenuti i vettori di caratteristiche per tutti i voxel della scena si procede ad una fase di classificazione. Per questo compito si è scelto di adottare tecniche *supervisionate* e *non supervisionate* al fine di poter mettere a confronto i risultati e valutare quale sistema possa risultare migliore per lo scopo di segmentazione automatica;
- *MRF-based segmentation*: effettuata la fase di classificazione, si prendono i risultati finali di questa come punto di partenza per la generazione di un Markov Random Field associato ai voxel che compongono la nuvola di punti; vengono definiti i rapporti di vicinato, i potenziali unari, i potenziali di coppia e la conseguente funzione energia associata: una volta stabilita, quest’ultima viene minimizzata attraverso l’impiego di algoritmi di taglio a costo minimo.

Al termine di tutte queste fasi il processo restituisce come output una point cloud segmentata dove ogni colore corrisponde ad una determinata classe (ad es. *muro*, *pavimento*, *porta*, ecc).

4.2 Descrizione dell'algoritmo

Passiamo ora alla descrizione puntuale di ogni fase del processo; per completezza si riporta, in figura 4.1, un diagramma rappresentante l'intero flusso dell'algoritmo realizzato.

4.2.1 Acquisizione

Come già anticipato nel capitolo 3.2, i dati sui quali si svilupperà tutto il lavoro sono stati prelevati tramite l'impiego del sensore Microsoft Kinect. Il dispositivo permette l'acquisizione di immagini RGB "classiche" ad una risoluzione di 640 x 480 pixel (per un totale di 307200 punti) e restituisce una nuvola di punti, anch'essa organizzata per righe e colonne, avente la stessa risoluzione.

Siano \mathcal{S} e \mathcal{P} rispettivamente l'immagine e la point cloud ritornate dal sensore, è possibile definire questi due oggetti tramite le seguenti relazioni:

$$\begin{aligned} \mathcal{S} &= \left\{ \mathbf{s}_i \in \mathbb{N}_{[0,255]}^3 : \mathbf{s}_i = \begin{pmatrix} s_i^r \\ s_i^g \\ s_i^b \end{pmatrix} \right\} \\ \mathcal{P} &= \left\{ \mathbf{p}_i = \begin{pmatrix} \mathbf{r}_i \\ \mathbf{c}_i \end{pmatrix}, \mathbf{r}_i \in \mathbb{R}^3, \mathbf{c}_i \in \mathbb{N}_{[0,255]}^4 : \mathbf{r}_i = \begin{pmatrix} r_i^x \\ r_i^y \\ r_i^z \end{pmatrix}, \mathbf{c}_i = \begin{pmatrix} c_i^r \\ c_i^g \\ c_i^b \\ c_i^a \end{pmatrix} \right\} \end{aligned} \quad (4.1)$$

L'immagine \mathcal{S} è quindi composta dall'insieme di punti s_i che rappresentano i singoli pixel con le relative informazioni cromatiche nello spazio di colore RGB; la point cloud \mathcal{P} invece è composta da una struttura leggermente più complessa: il singolo punto \mathbf{p}_i , oltre alle informazioni di colore presenti in \mathbf{r}_i (dove però rispetto a 4.1 abbiamo un elemento in più per gestire la trasparenza), sono presenti anche informazioni legate alla collocazione spaziale del punto mediante la componente espressa da \mathbf{c}_i .

Tramite l'impiego della classe `pcl::Grabber` è possibile sfruttare appieno le capacità del Kinect per ricavare con semplicità l'immagine a colori e la point cloud relativa alla sce-

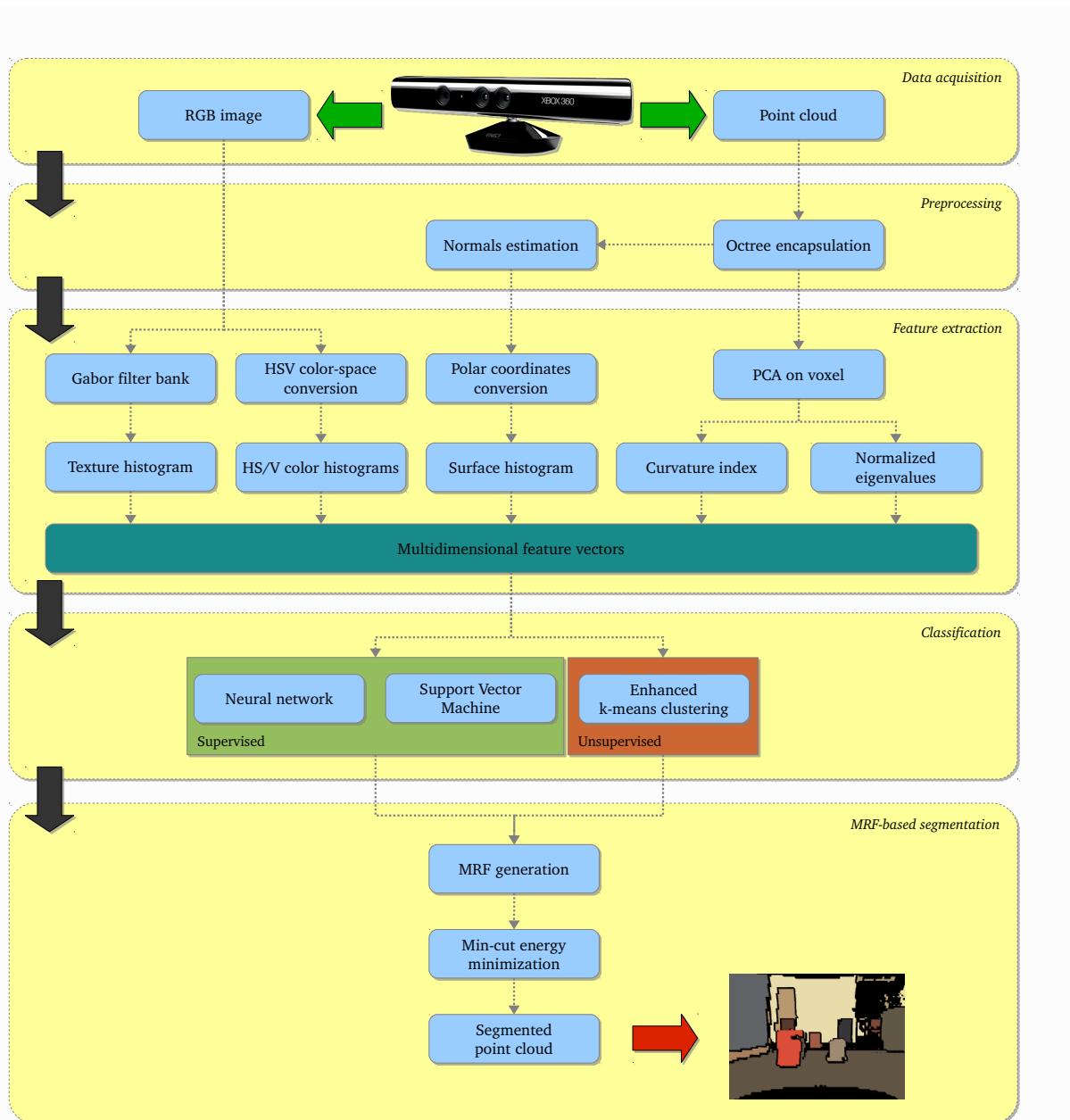


Figura 4.1: Schema di flusso dell'algoritmo realizzato

na osservata; per la realizzazione del sistema si è proceduto all’acquisizione di diversi fotogrammi riguardanti ambienti relativi a spazi domestici di interni.

Altri componenti utili alla realizzazione del sistema si acquisizione sono le classi *pcl::visualization::PCLVisualizer* e *pcl::visualization::ImageViewer*, che rispettivamente si occupano di presentare all’interno di finestre l’immagine RGB e la nuvola di punti durante l’utilizzo del sensore, il tutto in tempo reale.

Una volta ricavato un modo per visualizzare ciò che si sta inquadrando tramite il Kinect è stato necessario definire uno strumento per poter catturare istantanee dello spazio osservato: questo è stato possibile grazie alla libreria PCL che permette la definizione di funzioni delegate (o anche dette di *callback*), che possono essere associate alle due classi precedentemente citate, per gestire l’interazione con l’utente mediante l’utilizzo della tastiera e del mouse. In questo caso è stata definita una funzione di callback che (grazie all’utilizzo congiunto delle OpenCV e le PCL), a seguito delle pressione del tasto destro del mouse, permettesse la cattura sincronizzata dell’immagine a colori (salvata in formato non compresso *.ppm*) e della nuvola di punti corrispondente (memorizzata in formato proprietario *.pcd*); un esempio dei dati acquisiti è riportato in figura 4.2.

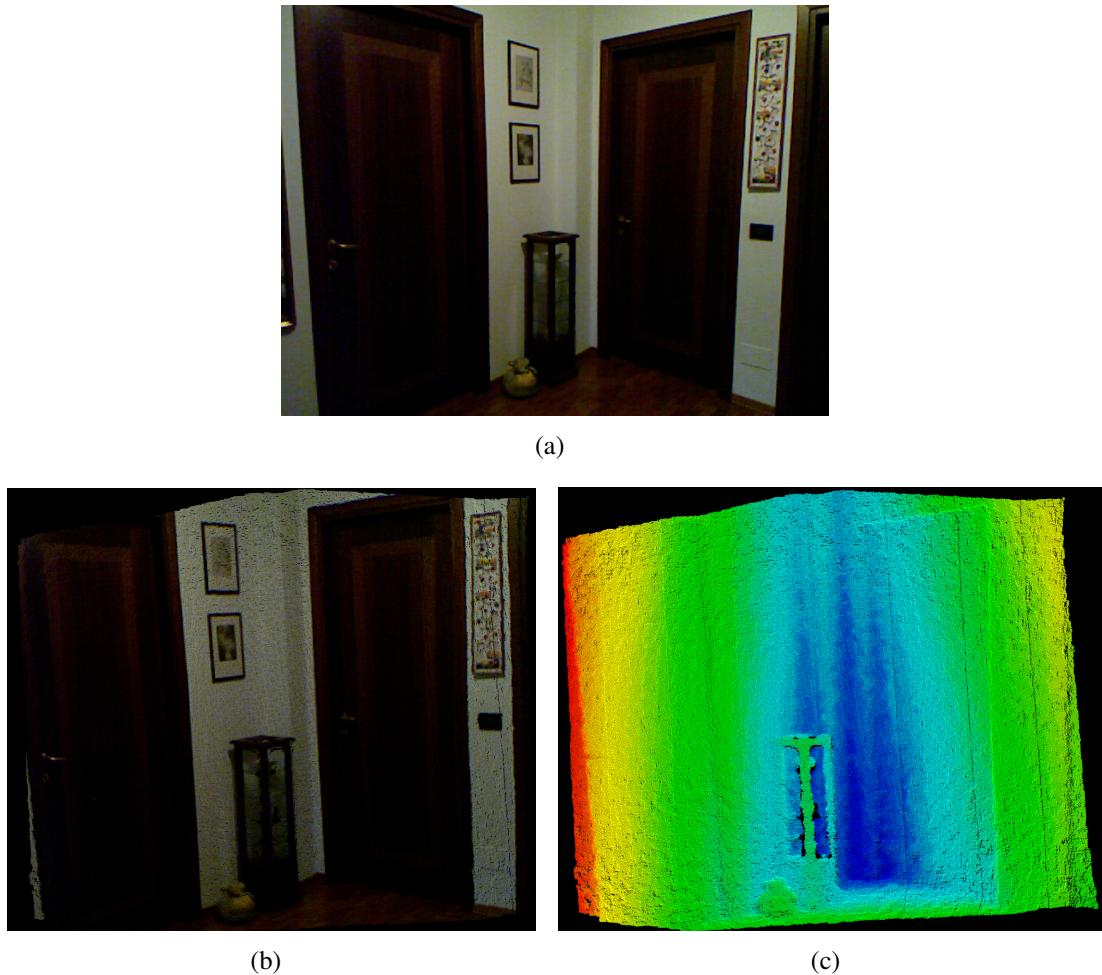


Figura 4.2: Esempio di dati rilevati durante la fase d'acquisizione: (a) Immagine RGB, (b) point cloud e (c) rappresentazione della nuvola mediante “falsi colori” dove zone calde indicano punti vicini mentre colori freddi appartengono a zone più lontane

4.2.2 Il preprocessing dei dati

Una volta acquisiti tutti i dati, sono necessarie alcune operazioni al fine di preparare tutte le informazioni a disposizione per le successive elaborazioni. La nuvola di punti viene incapsulata ed indicizzata tramite un'opportuna struttura dati, inoltre si effettua il calcolo delle normali della cloud che, a causa dell'elevata quantità di elementi, richiedono parecchio tempo per essere computate rendendo quindi conveniente il loro calcolo a monte di ogni altra fase dell'algoritmo.

4.2.2.1 Organizzazione della nuvola tramite octree

Come già accennato nel capitolo 3.3, per una gestione organizzata dei punti che compongono la cloud, si fa uso della struttura dati octree; questo consente una ripartizione della nuvola di punti in celle elementari (voxel), garantisce un accesso efficiente ad ogni suo elemento e permette di effettuare attività di ricerca efficienti su gruppi di punti in zone arbitrarie dello spazio. Grazie alla topologia ad albero con cui è implementato l'octree, i vantaggi relativi al tempo d'esecuzione sono evidenti, a maggior ragione quando si ha a che fare con centinaia di migliaia di punti e la ricerca deve essere eseguita ripetutamente durante l'esecuzione del programma.

Il parametro principale che caratterizza l'octree è la risoluzione ξ che rappresenta il grado massimo di dettaglio raggiungibile dalla struttura: lo spigolo dei voxel in cui la point cloud è partizionata è pari a ξ . Nello specifico durante la realizzazione del progetto si è fissato il valore di $\xi = 0.05m$: dato che d'ora in poi la nuvola non sarà più gestita punto per punto, bensì a livello di “aggregati di punti” (cioè tutti quelli che ricadono all'interno dello stesso voxel), questo valore si è rivelato un buon compromesso fra risoluzione (una cella di dimensioni eccessive discretizzerebbe in modo troppo grossolano la scena) e significatività di ciascuna cella (features ricavate da voxel composti da pochi punti portano un basso contenuto informativo).

Il metodo seguito per organizzare la nuvola e navigarla efficientemente tramite l'octree è il seguente (figura 4.3):

- incapsulare la point cloud all'interno di una struttura octree a risoluzione σ ;

- ricavare le coordinate dei centri di massa relativi ai voxel *non vuoti* ottenuti al punto precedente;
- interrogare, sfruttando l'octree, il voxel corrispondente al centroide indicato al fine di recuperare gli indici dei punti contenuti al suo interno;
- tramite la lista restituita accedere direttamente ai punti interessati all'interno della nuvola;
- ricavare gli indici dei voxel vicini e non vuoti.

Lo pseudo-codice equivalente alle operazioni appena elencate è riportato nel listato 4.1.

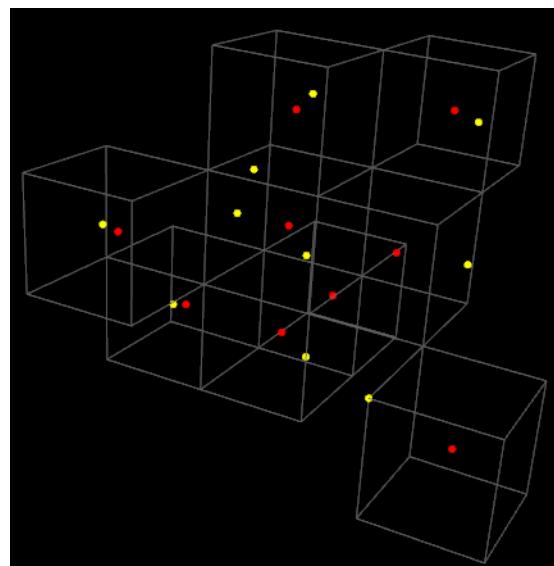


Figura 4.3: Nuvola di punti generata casualmente ed incapsulata in una struttura octree: i punti rossi rappresentano i centri di massa dei voxel

Listato 4.1 Accesso rapido ai punti compresi all'interno di un voxel

Input: Una nuvola di punti *PointCloud***Output:** Una lista *pointIdxList* degli indici dei punti compresi in un voxel arbitrario

```

1: OctreePointCloudSearch octree  $\leftarrow$  new OctreePointCloudSearch( $\sigma$ )
2: octree.setPointCloud(pointCloud)
3: octree.addPointsFromInputCloud()
4: octree.getOccupiedVoxelCenters(centroids)
5: octree.voxelSearch(centroids[i], pointIdxList)

6: for all index in pointIdxList do
7:   :
8:     DoStuff(pointCloud.points[index])
9:   :
10: end for

```

4.2.2.2 Calcolo delle normali

Le normali sono una proprietà importante delle superfici geometriche e sono spesso usate in campi come la *computer graphics*, dove vengono sfruttate per l'applicazione di ombre, fonti luminose e altri effetti. In questo progetto le normali serviranno per ricavare caratteristiche inerenti alla superficie locale di ogni voxel.

Per il calcolo delle normali solitamente si procede nel seguenti modi:

- determinare la superficie della cloud in modo esplicito, per esempio con tecniche di *meshing*, e conseguentemente calcolare la normale ad ogni faccia generata;
- usare metodi approssimati per stimare le normali alla superficie.

Il metodo principale per il calcolo delle normali, messo a disposizione dalle Point Cloud Library, è *approssimato* poiché calcola le normali dopo aver applicato l'*Analisi per Componenti Principali* (PCA) (un esempio in figura 4.4). Per ogni punto p_i della nuvola viene calcolata la PCA per un sottoinsieme di punti \mathcal{K} costituito da lui ed i suoi k -vicini: la ricerca di questi punti avviene efficientemente attraverso l'utilizzo di *kD-tree* [74].

Ottenuti sufficienti punti è possibile ricavare gli autovalori e autovettori associati alla matrice di covarianza \mathbf{C} , matematicamente:

$$PCA \rightarrow \begin{cases} \mathbf{C} = \frac{1}{k} \sum_{i=1}^k (\mathbf{p}_i - \bar{\mathbf{p}})(\mathbf{p}_i - \bar{\mathbf{p}})^T, & k = |\mathcal{K}| \\ \mathbf{C}\mathbf{v}_j = \lambda_j \mathbf{v}_j, & j \in \{0, 1, 2\} \end{cases} \quad (4.2)$$

A questo punto una buona stima della normale alla superficie al punto p_i è l'autovettore v_{min} associato all'autovalore minore λ_{min} tra quelli ricavati.

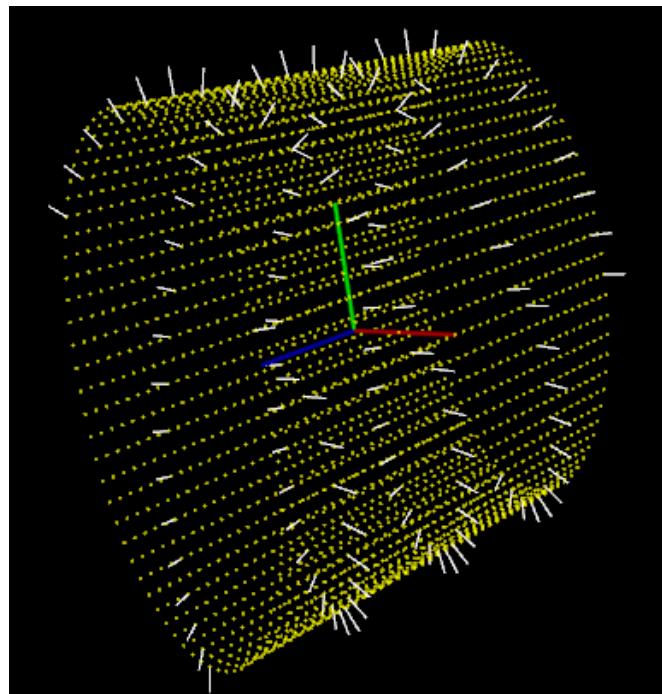


Figura 4.4: Risultato ottenuto dopo la stima delle normali alla point cloud mediante PCA, in figura è stata visualizzata una normale ogni 25 punti

4.2.3 Estrazione delle features

Ora che la nuvola di punti è stata suddivisa per voxel è necessario ricavare un descrittore, ossia un vettore, che rappresenti in forma compatta le principali caratteristiche del voxel. Inoltre è lecito supporre che a zone simili della scena corrispondano voxel con caratteristiche non molto diverse tra loro; conseguentemente a questa ipotesi si può quindi pensare di sfruttare i descrittori derivanti da questa fase di estrazione di *features*, per ottenere dei classificatori in grado di associare correttamente un intero voxel ad una determinata classe dato in input la firma che lo contraddistingue. In questo lavoro si è scelto di ricavare, per ogni voxel, un descrittore che potesse racchiudere al suo interno informazioni di:

- *colore*: informazioni di questo tipo rappresentano già un buon livello di caratterizzazione dei voxel, inoltre la struttura dati della nuvola che si andrà a manipolare rende facilmente fruibili le caratteristiche cromatiche di ogni suo punto;
- *texture*: ricavare solamente informazioni di colore, tipicamente attraverso un istogramma, potrebbe non risultare sufficientemente distintivo, in quanto a istogrammi uguali potrebbero corrispondere elementi con texture differenti (figura 4.5).
- *forma*: oltre al colore, le informazioni ricavabili per ogni punto comprendono anche la sua dislocazione spaziale, questi dati saranno utili per ricavare una descrizione della conformazione locale del voxel;

Nei prossimi paragrafi verranno esaminati tutte le varie componenti che andranno a costituire il vettore di features finale associato a ciascun voxel della cloud.

4.2.3.1 Features di colore

Per rappresentare la distribuzione di cromatica presente all'interno di un voxel si è scelto di adoperare istogrammi di colore normalizzati. L'informazione sul colore è restituita dal sistema in formato RGB, tale formato però presenta lo svantaggio principale di essere poco robusto a cambiamenti di illuminazione della scena: questo a causa del fatto che una qualsiasi tonalità di colore è dipendente dalla combinazione dei tre canali cromatici

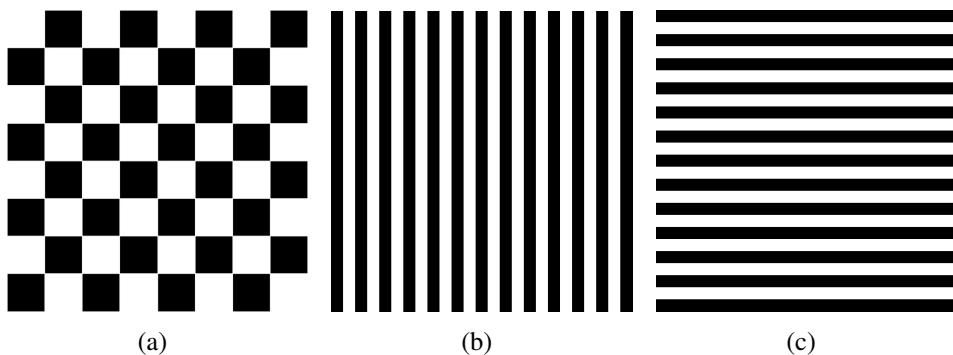


Figura 4.5: Esempio alcune semplici immagini aventi lo stesso istogramma di colore ma che presentano differenti texture

Color patch				
RGB values	(255, 0, 0)	(77, 0, 0)	(168, 0, 0)	(204, 102, 102)
HSV values	(0, 1, 1)	(0, 1, 0.3)	(0, 1, 0.6)	(0, 0.5, 0.8)

Figura 4.6: Varie tonalità di rosso contraddistinte da un diverso grado d'illuminazione: nello spazio di colore HSV il valore del canale H corrisponde sempre a “0” (rosso), mentre in RGB tutti e tre i canali subiscono variazioni al proprio valore

rosso, verde e blu. Conseguentemente a questo fatto i colori di oggetti identici ma illuminati in maniera differente saranno rappresentati da triplete di colori completamente differenti tra loro e, a causa di questo, contribuiranno alla generazione di istogrammi sensibilmente diversi l'uno dall'altro.

A fronte di questo tipo di problema è necessario passare dall'RGB ad uno spazio di colore differente che presenti una maggior robustezza alle problematiche sopracitate. Lo spazio di colore *Hue Saturation Value* (HSV) sfrutta sempre tre canali per la rappresentazione del colore ma, a differenza dell'RGB, la componente di tonalità è isolata in un unico canale (Hue) mentre gli altri due rappresentano il grado di saturazione del colore (Saturation) e la sua luminosità (Value).

Grazie a questa differenziazione, ad un determinato colore (indipendentemente dal suo grado di luminosità) corrisponderà sempre lo stesso valore di tonalità; in tabella 4.6 un esempio a dimostrazione di questa proprietà.

A differenza dell'RGB, le quali componenti sono normalmente tutte definite su un in-

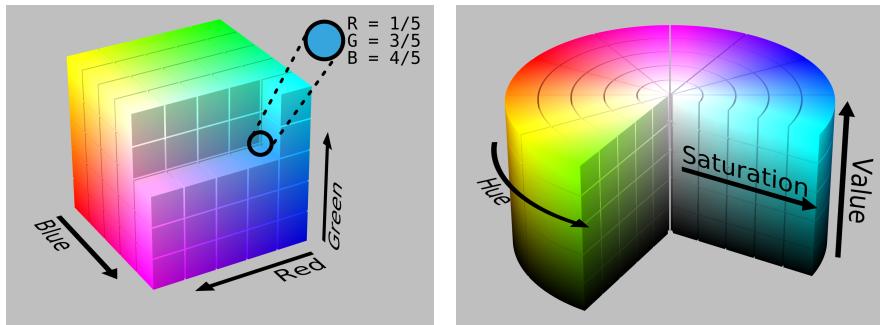


Figura 4.7: Spazi di colore a confronto: A sinistra il cubo che rappresenta lo spazio di colore Reg-Green-Blue (RGB) e destra il cilindro corrispondente per lo Hue-Saturation-Value (HSV)

intervallo di valori interi compresi tra $[0, 255]$ o valori reali tra $[0, 1]$, l'HSV mappa il canale di Hue su un intervallo di valori interi tra $[0, 360]$, mentre Saturation e Value solitamente hanno un dominio continuo su range compreso tra $[0, 1]$ (figura 4.7). Sfruttando le funzioni di conversione tra spazi di colore messe a disposizione dalle OpenCV è facile ottenere valori di colore in HSV con tutte e tre le componenti riscalate su un intervallo continuo $[0, 1]$. Nell'implementazione del progetto si è scelto, per ogni voxel, di convertire i valori di colore da RGB ad HSV e successivamente generare un istogramma bidimensionale 10×10 normalizzato per i canali H-S, ed un secondo istogramma (anch'esso composto da 10 intervalli) per il canale V, il tutto per un totale di $10 \times 10 + 10 = 110$ valori.

4.2.3.2 Features di tessitura

Come già accennato, lo stesso istogramma di colore può corrispondere a texture differenti; per questo è necessario introdurre un'altro strumento che sia in grado di esprimere anche questo tipo di caratteristica. Il processo per ricavare le informazioni riguardanti la tessitura dell'immagine è stato implementato attraverso l'utilizzo di un particolare filtro (già impiegato da Reynolds e Murphy [75]): il *filtro di Gabor*.

Il filtro di Gabor [76] è un filtro bidimensionale la cui risposta all'impulso è definita da una funzione armonica moltiplicata per una funzione Gaussiana (figura 4.8). In forza del teorema di convoluzione la trasformata di Fourier della risposta all'impulso di un filtro di Gabor risulta essere data dalla convoluzione fra la trasformata di Fourier della

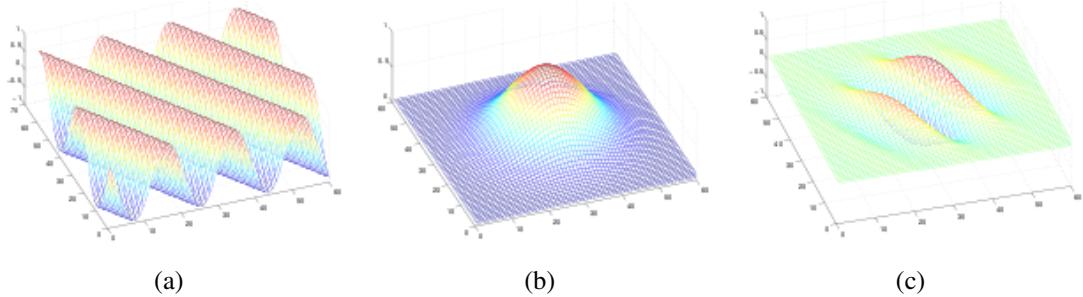


Figura 4.8: Il filtro di Gabor e le sue componenti: (a) componente armonica, (b) componente Gaussiana e (c) filtro di Gabor ottenuto dalla combinazione di (a) e (b)

funzione armonica e la trasformata di Fourier della funzione Gaussiana. Analiticamente è definito dalla seguente funzione:

$$g(x, y; \lambda, \theta, \psi, \sigma, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi \frac{x'}{\lambda} + \psi\right) \quad (4.3)$$

dove:

$$\begin{aligned} x' &= x \cos \theta + y \sin \theta \\ y' &= -x \sin \theta + y \cos \theta \end{aligned} \quad (4.4)$$

I parametri elencati all'interno dell'equazione 4.3 svolgono il seguente ruolo:

- λ rappresenta la lunghezza d'onda del fattore coseno;
- θ rappresenta l'orientazione globale del filtro;
- ψ è la fase del fattore coseno;
- σ è il parametro che regola la scala dell'inviluppo;
- γ specifica l'ellitticità del supporto della funzione di Gabor.

In virtù della definizione parametrica di questo tipo di filtro, solitamente per estrarre le informazioni relative alla tessitura di un'immagine, si ricorre un banco di filtri di Gabor: si generano più filtri, definiti per diverse scale e/o orientazioni, e si valuta la risposta complessiva al banco.

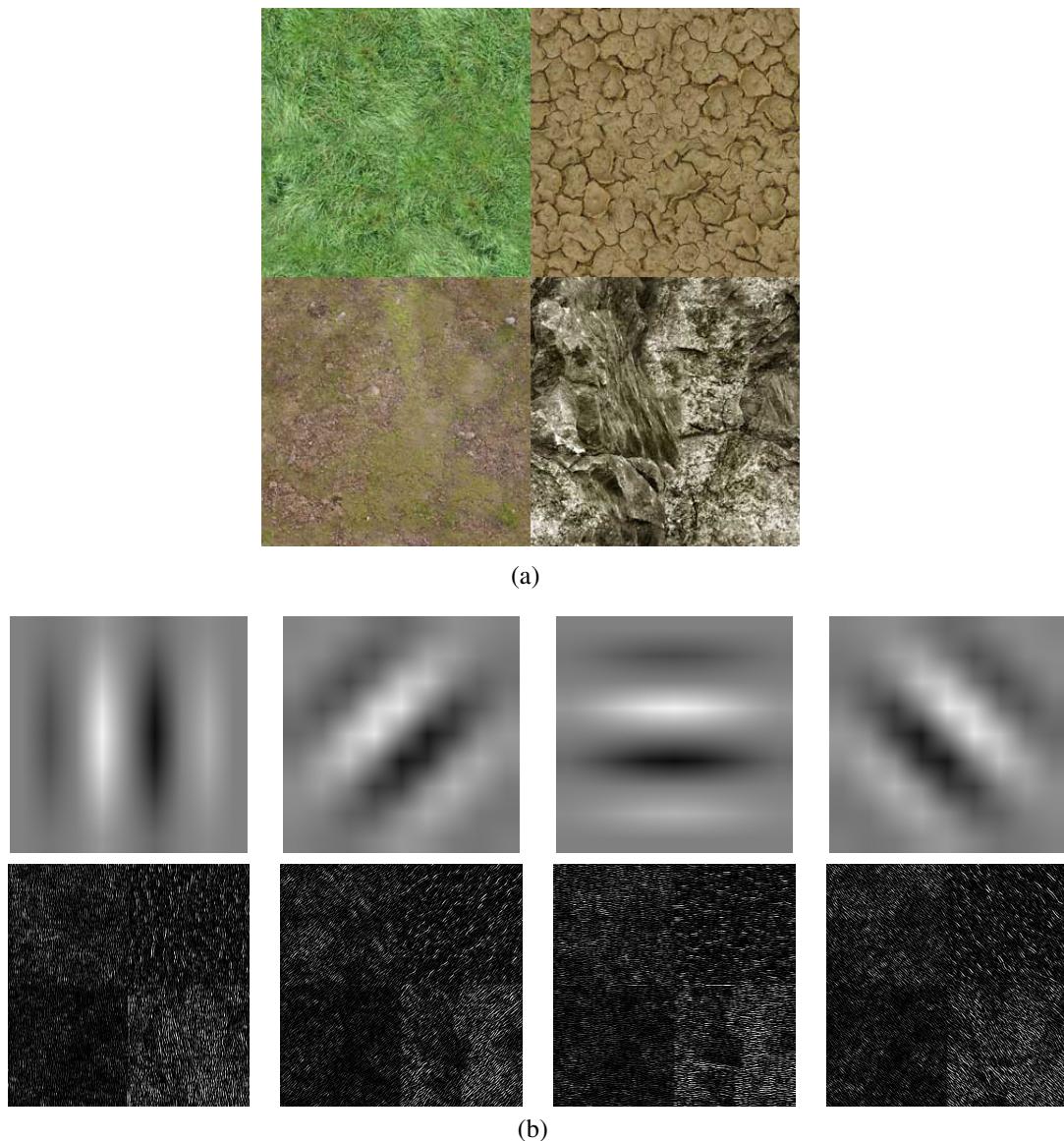


Figura 4.9: Applicazione di un banco di filtri di Gabor a quattro diverse orientazioni (0° , 45° , 90° e 135°): (a) immagine di input, (b) visualizzazione grafica del kernel applicato (in alto) e risposta al filtro ottenuta dopo l'applicazione del banco di filtri all'immagine originale (in basso)

Dall'esempio riportato in figura 4.9 risulta quindi evidente come i filtri di Gabor possano essere sfruttati per ottenere dei buoni risultati nell'analisi di texture di immagini bidimensionali. Per questo compito sono state impiegate ancora una volta le OpenCV grazie alla possibilità, che questa libreria offre, di definire kernel personalizzati e successivamente di applicarli ad immagini tramite l'impiego di poche istruzioni.

Nello specifico si è realizzato un *banco di Gabor* composto da un insieme di kernel di dimensione 9×9 pixel, a 6 diverse orientazioni Θ e 4 distinte scale Σ , nello specifico:

$$\begin{aligned}\Theta &= \{\theta_k \in \mathbb{R}, k \in \mathbb{N}^+ : \theta_k = \frac{k\pi}{6}, \forall k \in [1, 6]\} \\ \Sigma &= \{\sigma_l \in \mathbb{R}, l \in \mathbb{N}^+ : \sigma_l = l, \forall l \in [1, 4]\}\end{aligned}\tag{4.5}$$

Per i rimanenti parametri dei filtri del banco si è scelto di applicare i seguenti valori per ognuno: $\lambda = 1$, $\psi = \pi/2$ e $\gamma = 1$. Definito il banco si passa alla generazione delle informazioni che saranno processate da quest'ultimo: sono create patch d'immagine partendo dal frame 2D corrente e considerando solamente i pixel che hanno una corrispondenza con i punti (della cloud) interni al volume del voxel d'interesse. Ognuno di questi output viene poi rimappato su un istogramma normalizzato di 10 elementi dando così un descrittore di texture finale da $6 \times 4 \times 10 = 240$ valori reali.

4.2.3.3 Features di forma

Per quanto riguarda l'estrazione di features relative alla conformazione della cloud, per ogni voxel v_i si è scelto di considerare tre differenti indicatori in grado di esprimere le caratteristiche di forma di v_i . Prima di elencare questi indici, e riportare quale sia il procedimento seguito per ricavarli, è necessaria una fase preliminare di calcolo della PCA per quella porzione della cloud relativa unicamente ai punti che ricadono all'interno del volume dell' i -esimo voxel considerato.

Terminato questo passaggio, definiamo l'insieme $E = \{\ell_i^x, \ell_i^y, \ell_i^z\}$ l'insieme degli autovalori ricavati dall'analisi per componenti principali sugli elementi di v_i ; a questo punto è possibile definire:

- Gli autovalori normalizzati $\overline{\ell_i^x}, \overline{\ell_i^y}, \overline{\ell_i^z}$ associati a v_i :

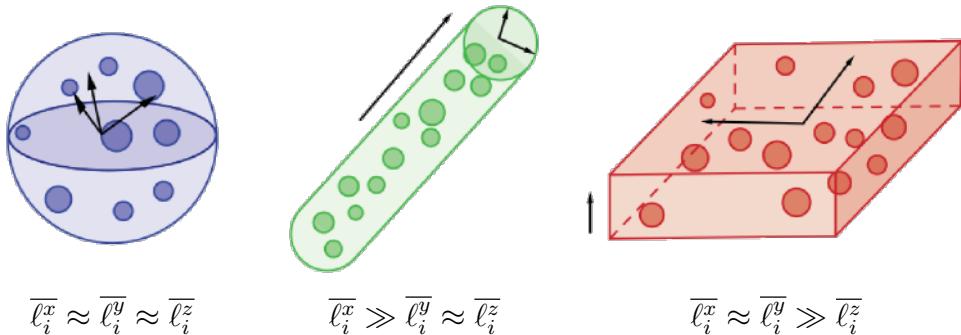


Figura 4.10: Rapporti di forma esprimibili attraverso gli autovalori normalizzati

$$\overline{\ell_i^x} = \frac{\ell_i^x}{\sum_{j=1}^{|E|} \ell_i^j} \quad \overline{\ell_i^y} = \frac{\ell_i^y}{\sum_{j=1}^{|E|} \ell_i^j} \quad \overline{\ell_i^z} = \frac{\ell_i^z}{\sum_{j=1}^{|E|} \ell_i^j} \quad (4.6)$$

Questi indicatori si rivelano particolarmente utili in quanto sono in grado di descrivere il rapporto di forma tridimensionale espresso dai punti racchiusi in v_i (figura 4.10).

- L'indice di curvatura ϱ_i definito nella seguente maniera:

$$\varrho_i = \frac{\min\{\ell_i^x \ell_i^y, \ell_i^z\}}{\sum_{j=1}^{|E|} \ell_i^j} \quad (4.7)$$

La curvatura della cloud viene quindi approssimata attraverso il rapporto tra l'autovalore minore tra quelli ricavati dall'analisi per componenti principali e la somma di tutti gli autovalori calcolati.

- L'istogramma bidimensionale, normalizzato, delle normali alla nuvola nei punti circoscritti dal voxel. È da osservare come queste normali (calcolate durante la fase di preprocessing) siano descritte nello spazio cartesiano da una tripletta di valori reali (x, y, z). Di fatto questa terna può essere tradotta equivalentemente ad una coppia di valori (ϕ, θ) attraverso il passaggio da coordinate cartesiane a coordinate sferiche. Questa riduzione di dimensionalità è possibile in virtù del fatto che le normali ricavate in precedenza hanno norma unitaria: sarebbe pos-

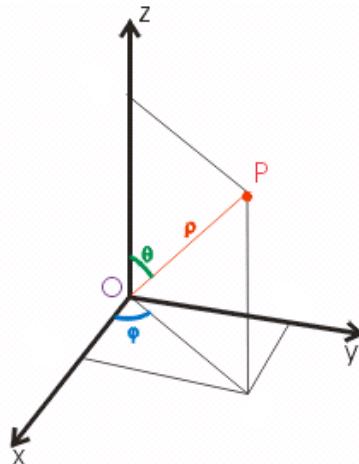


Figura 4.11: Rappresentazione grafica della correlazione tra coordinate cartesiane (x, y, z) e coordinate sferiche (ρ, ϕ, θ)

sibile sostituire i tre valori (x, y, z) nei corrispondenti (ρ, ϕ, θ) (figura 4.11) ma, siccome $\rho = 1$ per ogni normale calcolata, è sufficiente considerare solamente la coppia (ϕ, θ) . La conversione tra questi spazi di coordinate avviene attraverso la seguente set di equazioni (4.8):

$$\begin{aligned}\rho &= \sqrt{x^2 + y^2 + z^2} \\ \phi &= \arctan\left(\frac{y}{x}\right) \\ \theta &= \arccos\left(\frac{z}{\sqrt{x^2 + y^2 + z^2}}\right)\end{aligned}\tag{4.8}$$

Le coppie di valori (ϕ, θ) , così ottenute per ogni punto interno al voxel, vengono quindi utilizzate per generare un istogramma bidimensionale 10×10 rappresentante la “morfologia” locale della parte di cloud contenuta in v_i .

Alla luce di quanto detto finora, alla fine di questa attività inherente la generazione di caratteristiche di forma, si riesce ad ottenere un descrittore parziale di $10 \times 10 + 3 + 1 = 104$ elementi.

Tutto il modulo di estrazione di features prevede quindi la generazione di tre grandi classi di tratti distintivi per la cloud: il colore, la texture e la forma; sommando i con-

tributi dati dai vari descrittori parziali generati in ognuna di queste fasi, questo stadio di *feature extraction* genera complessivamente un descrittore $\mathbf{f}_i = (f_i^1, f_i^2, \dots, f_i^h)$ di

$$\overbrace{(10 \times 10 + 10)}^{colore} + \overbrace{(6 \times 4 \times 10)}^{texture} + \overbrace{(10 \times 10 + 3 + 1)}^{forma} = 454 \quad (4.9)$$

elementi a valore reale, $\mathbf{f}_i \in \mathbb{R}^{454}$. Durante questa procedura ognuno dei voxel, nei quali è stata preventivamente scomposta la nuvola, viene quindi tradotto in un vettore multivariato di 454 valori.

L'insieme di tutti questi vettori danno una descrizione completa della scena osservata e si prestano con facilità ad essere sfruttati come entità sulle quali è possibile impiegare un meccanismo di classificazione. In seguito, i risultati restituiti da quest'ultimo passaggio costituiranno la base di partenza necessaria per la generazione di un Markov Random Field ed il raggiungimento finale di una segmentazione globale della scena.

Nel prossimo capitolo verranno analizzate le tecniche di classificazione (supervisionata e non) sfruttate per riuscire ad associare ogni singolo voxel alla classe d'appartenenza più appropriata.

4.2.4 La classificazione

A seguito del calcolo dei vettori di features (uno per ogni voxel della cloud), è necessario l'impiego di un meccanismo di classificazione per riuscire ad assegnare ogni *parte* della nuvola ad una determinata classe d'appartenenza in maniera automatica. Le classi nella quali verranno ripartiti i vari voxel costuiranno, alla fine del processo, l'obiettivo ultimo dell'algoritmo, cioè la segmentazione globale della scena.

L'obiettivo di questo lavoro di tesi è l'attribuzione di ogni elemento, nei quali la scena è stata suddivisa (voxel), ad una classe classe specifica. A seconda dell'applicazione le etichette da attribuire a ciascun elemento possono essere definite a priori dall'utente oppure possono emergere da una partizione automatica della scena. Nel caso dell'identificazione di oggetti è possibile classificare oggetti noti (o categorie di oggetti noti) oppure cercare di segmentare la scena secondo criteri che facciano emergere gli oggetti.

In fase di realizzazione del progetto si è scelto di utilizzare due tipi differenti di classificatori supervisionati ed un metodo per il caso di classificazione non supervisionata; questo al fine di comparare le performance di classificazione tra i vari sistemi impiegati. Per quanto riguarda la classificazione supervisionata sono stati utilizzate *reti neurali artificiali* (Artificial Neural Networks, ANN) e *macchine a vettori di supporto* (Support Vector Machine, SVM), mentre per il clustering non supervisionato si è scelto di implementare una versione ottimizzata dell'algoritmo *k-means*.

4.2.4.1 Apprendimento supervisionato

L'Apprendimento supervisionato è una tecnica di machine learning che mira ad istruire un sistema informatico in modo da consentirgli di risolvere dei compiti in automatico; nel nostro caso la classificazione dei voxel.

Il procedimento è il seguente:

- Si definisce un insieme \mathcal{I} di dati d'ingresso, questi tipicamente sono rappresentati da vettori a più dimensioni;
- Si definisce l'insieme dei dati in uscita come insieme \mathcal{O} , gli output possono essere valori continui (in questo caso si parla di regressione) o una etichetta numerica;
- Si definisce una funzione ζ che associa ad ogni dato in ingresso di \mathcal{I} la sua risposta corretta in \mathcal{O} .

Tutti gli algoritmi di apprendimento supervisionato partono dal presupposto che se forniamo al sistema un numero adeguato di esempi, questo sarà in grado di creare una funzione ζ' che approssimerà la funzione ζ ; se l'approssimazione di ζ risulterà adeguata quando proporremo a ζ' dei dati in ingresso mai analizzati, la funzione dovrebbe essere in grado di fornire delle risposte in uscita simili a quelle fornite da ζ e quindi corrette (o quasi). Il fatto che la costruzione ed il buon funzionamento del classificatore richieda la definizione preliminare di un training set, indica come questo dipenda in modo significativo dai dati di addestramento. Un esempio generico del workflow seguito durante il processo di apprendimento supervisionato è riportato in figura 4.12.

Di solito un algoritmo di apprendimento viene allenato usando un certo numero di esempi (il training set appunto), cioè situazioni tipo di cui è già noto il risultato che interessa prevedere. Si assume che l'algoritmo di apprendimento raggiungerà, durante la fase di addestramento, uno stato in cui il classificatore sarà in grado di predire gli output per tutti gli altri esempi che ancora non ha visionato, cioè si assume che il modello di apprendimento sarà in grado di *generalizzare*. Tuttavia, soprattutto nei casi in cui l'addestramento sia stato effettuato troppo a lungo o qualora si fosse scelto uno scarso

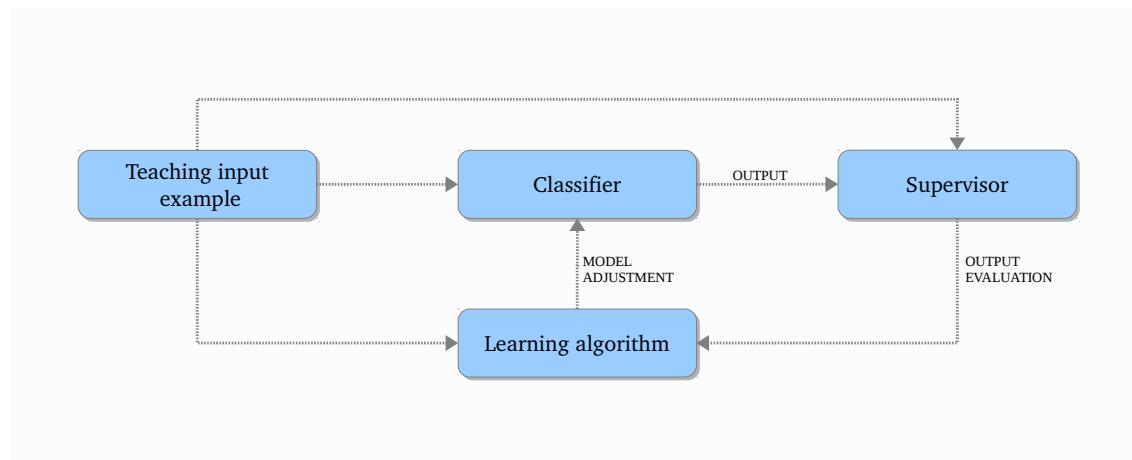


Figura 4.12: Rappresentazione del workflow per i processi d'apprendimento supervisionato: l'esempio viene dato in input al classificatore che da in output la propria risposta; il supervisore a sua volta da una risposta sulla base al risultato di classificazione sperato e quello ricevuto dal modello, in seguito a questo risponso l'algoritmo di apprendimento modifica i parametri del classificatore al fine di migliorare il suo comportamento finale

numero di esempi di allenamento, il modello potrebbe adattarsi a caratteristiche che sono specifiche solo del training set, ma che non hanno riscontro nel resto dei casi; perciò, in presenza di *overfitting* (figura 4.13), le prestazioni (cioè la capacità di adattarsi/prevedere) sui dati di allenamento aumenteranno, mentre le prestazioni sui dati non visionati saranno peggiori.

Generalmente i metodi di classificazione sono predisposti per la classificazione binaria. Per effettuare una classificazione multi-classe è stato necessario definire una batteria di K modelli (ognuno rappresentante una precisa classe) che hanno il compito di restituire in output un valore di confidenza, compreso tra $[-1, 1] \in \mathbb{R}$, che indica quanto il voxel analizzato sia verosimilmente appartenente alla k -esima classe: ogni vettore di features è fatto passare attraverso questo banco di classificazione e conseguentemente verrà associato alla classe rappresentata dal classificatore che ha restituito il valore di *confidence* più alto. Un esempio tramite pseudo-codice del processo appena citato è riportato nel listato 4.2.

La fase di addestramento prevederà quindi la generazione preliminare di K file di training dove sono presenti al loro interno N vettori di features per ogni classe, dando così

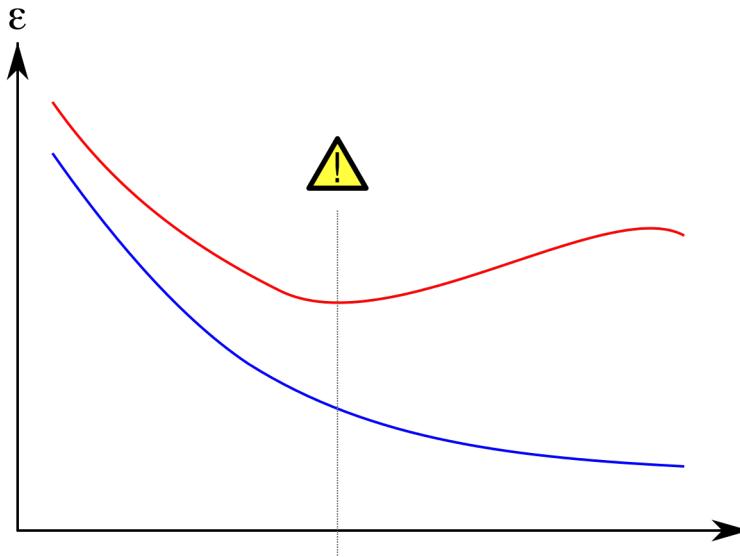


Figura 4.13: Overfitting: la curva blu mostra l'andamento dell'errore nel classificare i dati di training, mentre la curva rossa mostra l'errore nel classificare i dati di test o validazione. Se l'errore di validazione aumenta mentre l'errore sui dati di training diminuisce, ciò indica che siamo in presenza di un possibile caso di overfitting

luogo ad un file di $N \times K$ vettori d'esempio. Avendo scelto un pattern di classificazione dove ogni classificatore restituisce solo il grado di confidenza relativa alla classe che esso rappresenta, i K files di addestramento saranno identici a meno del valore d'appartenenza $\{\pm 1\}$ indicato a fianco di ogni vettore: per esempio il file di training processato dal classificatore per la classe *porta* vedrà come positivi tutti quegli esempi che si riferiscono a voxel effettivamente collocati in presenza di porte, al contrario il classificatore per la categoria *pavimento* considererà cattivi esempi i vettori “buoni” di porte e di tutte le altre classi meno che quelli riferiti a voxel che, nella scena inquadrata, compongono il pavimento (listato 4.3).

4.2.4.2 Il ruolo del ground truth

Solitamente con il termine *ground truth* (GT) ci si riferisce ad un insieme di informazioni che vengono assunte come modello di riferimento per validare o verificare osservazioni. Di fatto esso corrisponde al risultato desiderato e considerato corretto. Il ground truth

Listato 4.2 Classificazione dei vettori di features

Input: L'insieme di voxel $\mathcal{V} = \{v_1, v_2, \dots, v_n\}$ appartenenti alla nuvola *pointCloud* da classificare

Output: L'etichetta $l_k^* \in \mathcal{L} = \{l_1, l_2, \dots, l_m\}$ corrispondente la classe alla quale è stato associato ciascun voxel $v_i \in \mathcal{V}$

```

1: FeatureExtractor extractor  $\leftarrow$  new FeatureExtractor()
2: for all  $l_k$  in  $\mathcal{L}$  do
3:   Classifier  $class_k \leftarrow$  new SupervisedClassifier()
4:   ClassifierList  $\leftarrow$  ClassifierList  $\cup \{class_k\}$ 
5: end for

6: for all  $v_i$  in  $\mathcal{V}$  do
7:   FeatureVector  $featVector_i \leftarrow extractor.computeFeatures(v_i)$ 
8:   for all  $class_k$  in ClassifierList do
9:      $result_k \leftarrow class_k.classify(featVector_i)$ 
10:    ClassificationResults  $\leftarrow$  ClassificationResults  $\cup \{result_k\}$ 
11: end for

12:    $l_k^* \leftarrow \max_{k \in |\mathcal{L}|} ClassificationResults$ 
13: end for

```

Listato 4.3 Confronto tra due file di training per altrettante classi: α (a sinistra) e β (a destra)

$+1, f_\alpha^{11}, f_\alpha^{12}, \dots, f_\alpha^{1h}$	$-1, f_\alpha^{11}, f_\alpha^{12}, \dots, f_\alpha^{1h}$
$+1, f_\alpha^{21}, f_\alpha^{22}, \dots, f_\alpha^{2h}$	$-1, f_\alpha^{21}, f_\alpha^{22}, \dots, f_\alpha^{2h}$
$-1, f_\beta^{31}, f_\beta^{32}, \dots, f_\beta^{3h}$	$+1, f_\beta^{31}, f_\beta^{32}, \dots, f_\beta^{3h}$
$+1, f_\alpha^{41}, f_\alpha^{42}, \dots, f_\alpha^{4h}$	$-1, f_\alpha^{41}, f_\alpha^{42}, \dots, f_\alpha^{4h}$
$-1, f_\beta^{51}, f_\beta^{52}, \dots, f_\beta^{5h}$	$+1, f_\beta^{51}, f_\beta^{52}, \dots, f_\beta^{5h}$
\vdots	\vdots
$-1, f_\beta^{N1}, f_\beta^{N2}, \dots, f_\beta^{Nh}$	$+1, f_\beta^{N1}, f_\beta^{N2}, \dots, f_\beta^{Nh}$

è ottenuto dalla stima del supervisore stesso oppure tramite un modello di validazione indipendente dal sistema che si sta addestrando.

Nel caso della classificazione supervisionata un insieme di ground truth è quindi il mezzo attraverso il quale il *supervisore* determina se l'output ricevuto, prodotto dal classificatore durante la fase d'addestramento, sia un risultato corretto o meno considerando l'esempio dato in ingresso allo step corrente. Oltre al ruolo di fondamentale importanza che questo tipo d'informazione ricopre durante la fase di training, il ground truth viene spesso utilizzato per valutare la bontà di un sistema di classificazione: ottenuto il risultato finale dell'assegnazione di tutti gli elementi alle varie classi, è possibile saggiare la correttezza del sistema generato confrontando questo risultato con il GT. Questo tipo d'approccio consente la definizione di diversi benchmark di performance tra cui ricordiamo gli indicatori di *precisione* e *recall*.

Reti neurali Tradizionalmente il termine *rete neurale* viene utilizzato come riferimento ad una rete o ad un circuito di neuroni biologici, tuttavia ne è affermato l'uso anche in ambito informatico con riferimento alle *reti neurali artificiali* (o equivalentemente in inglese Artificial Neural Networks, ANN) [77], modelli matematici composti di “neuroni” artificiali. Sono modelli matematici che rappresentano l’interconnessione tra elementi definiti neuroni artificiali, ossia costrutti matematici che in qualche misura imitano le proprietà dei neuroni viventi. Questi modelli matematici possono essere utilizzati sia per ottenere una comprensione delle reti neurali biologiche, ma ancor di più per risolvere problemi ingegneristici di intelligenza artificiale come quelli che si pongono in diversi ambiti tecnologici (in elettronica, informatica, simulazione, e altre discipline).

Per il training di tali modelli si usa una approccio di apprendimento supervisionato: qualora si disponga di un insieme di dati per l'addestramento (o training set) comprendente esempi tipici d'ingressi con le relative uscite loro corrispondenti; in tal modo la rete può imparare ad inferire la relazione che li lega. Successivamente, la rete è addestrata mediante un opportuno algoritmo (tipicamente, la *backpropagation* che è appunto un algoritmo d'apprendimento supervisionato), il quale usa tali dati allo scopo di mo-

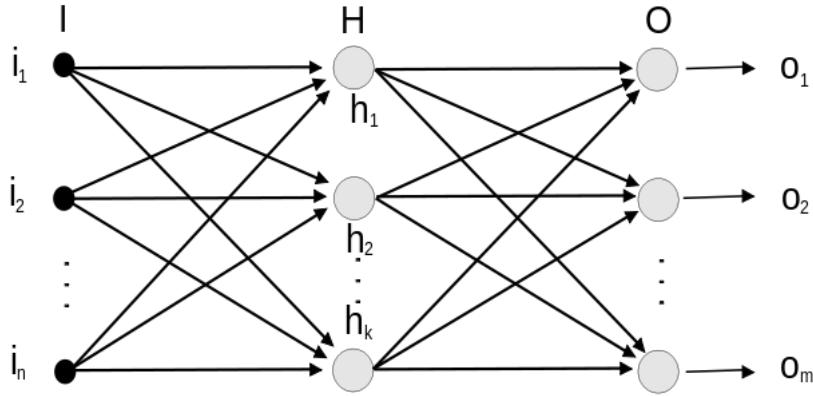


Figura 4.14: Struttura generica di una rete neurale artificiale

dificare i pesi ed altri parametri della rete stessa in modo tale da minimizzare l'errore di previsione relativo all'insieme d'addestramento. Se l'addestramento ha successo, la rete impara a riconoscere la relazione incognita che lega le variabili d'ingresso a quelle d'uscita, ed è quindi in grado di fare previsioni anche laddove l'uscita non è nota a priori. La struttura tipica di una rete neurale (figura 4.14) è composta dalla presenza di n elementi al livello d'ingresso I , questi sono tutti completamente connessi ad uno strato nascosto (o *hidden layer*) H che a sua volta può essere connesso ad altri strati della stessa natura o direttamente ad un layer d'uscita O composto da m neuroni; così come ogni nodo d'ingresso è collegato a tutti i neuroni nascosti del primo strato, in maniera duale ogni neurone dello strato d'uscita sarà collegato a tutti gli elementi dell'ultimo strato nascosto.

L'hidden layer gioca un ruolo di fondamentale importanza all'interno della NN in quanto rappresenta la capacità della rete di approssimare al meglio funzioni continue complesse: il teorema di Kolmogorov del 1975 [78] afferma infatti che qualsiasi funzione continua $y = f(x) : \mathbb{R}^n \mapsto \mathbb{R}^m$ può essere computata da una opportuna rete a 3 strati avente n unità nello strato di ingresso, $2n + 1$ nello strato nascosto ed m nello strato di uscita e totalmente connessa fra gli strati. In questo paragrafo si è voluta dare solamente una descrizione sommaria del funzionamento delle NN in quanto si presuppone che il lettore sia già a conoscenza dei fondamenti teorico-matematici che le descrivono; per ulteriori approfondimenti sull'argomento si rimanda a [77, 67].

Nello sviluppo di questo progetto sono state definite k reti neurali artificiali (una per ogni classe) mediante l'impiego della libreria *FANN* (Fast Artificial Neural Network); ogni rete ha una struttura a 3 strati ed è caratterizzata dalla presenza di 454 nodi di input (uno per ogni valore reale del vettore di features che identifica le caratteristiche del voxel), uno strato nascosto intermedio di 1000 neuroni che a loro volta risultano connessi allo strato d'uscita composto da solo elemento terminale. Il valore espresso da quest'ultimo rappresenta il grado di appartenenza dell' i -esimo voxel alla k -esima classe.

Support vector machines Le macchine a vettori di supporto possono essere pensate come una tecnica alternativa per l'apprendimento di classificatori polinomiali, contrapposta alle tecniche classiche di addestramento delle reti neurali. Le reti neurali ad un solo strato hanno un algoritmo di apprendimento efficiente, ma sono utili soltanto nel caso di dati linearmente separabili. Viceversa, le reti neurali multistrato possono rappresentare funzioni non lineari, ma sono difficili da addestrare a causa dell'alto numero di dimensioni dello spazio dei pesi e poiché le tecniche più diffuse, come la *back-propagation*, permettono di ottenere i pesi della rete risolvendo un problema di ottimizzazione non convesso e non vincolato che, di conseguenza, presenta un numero indeterminato di minimi locali. La tecnica di addestramento SVM tenta di risolvere entrambi i problemi: presenta un algoritmo efficiente ed è in grado di rappresentare funzioni non lineari complesse. I parametri caratteristici della rete sono ottenuti mediante la soluzione di un problema di programmazione quadratica convesso con vincoli di uguaglianza o di tipo box (in cui il valore del parametro deve essere mantenuto all'interno di un intervallo), che prevede un unico minimo globale [67].

La classificazione binaria può essere vista come un problema di separazione di dati (appartenenti a due classi differenti) nello spazio delle features; la classificazione tramite l'impiego di SVM si riduce al compito di ricavare l'iperpiano separatore ottimo, cioè che separa "al meglio" gli elementi dati in input al sistema. Data un training set di copie di valori esempio-etichetta (\mathbf{x}_i, y_i) , $i = 1, \dots, l$ dove $\mathbf{x}_i \in \mathbb{R}^n$ e $y \in \{\pm 1\}^l$ l'attività di addestramento per un classificatore di questo tipo si riduce al seguente problema di

ottimizzazione [69]:

$$\begin{aligned} \min_{\mathbf{w}, b, \xi} \quad & \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^l \xi_i \\ \text{subject to} \quad & y_i (\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \end{aligned} \tag{4.10}$$

La bontà relativa alla separazione è rappresentata dalla distanza che questo iperpiano presenta dagli elementi più vicini (i vettori di supporto o *support vectors*), appartenenti alle due classi, che si sta cercando di separare. Conseguentemente a quanto appena detto la separazione migliore sarà associata a quel piano che riesce a massimizzare la distanza tra iperpiano ed i vettori di supporto ricavati (figura 4.15); infine il termine $C > 0$ rappresenta l'indice di costo associato all'errore ξ . Può capitare che i dati raccolti non siano linearmente separabili e quindi il piano potrebbe non esistere: per ovviare a questo tipo di problema si ricorre a ciò che nel campo del machine learning viene chiamato *kernel trick*. Questa tecnica prevede l'impiego di funzioni particolari (chiamate per l'appunto *funzioni kernel*) per poter rimappare lo spazio originario, al quale appartengono i dati raccolti, in uno spazio di dimensione maggiore dove questi risultino separabili mediante un opportuno iperpiano; in figura 4.16, un esempio di quanto appena asserito. La definizione matematica di kernel, in relazione all'equazione 4.10, è la seguente:

$$\begin{aligned} \kappa : \mathbb{R}^n &\mapsto \mathbb{R}^m \\ \kappa(\mathbf{x}_i, \mathbf{x}_j) &= \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \end{aligned} \tag{4.11}$$

I tipi di kernel più comunemente utilizzati sono:

- lineare: $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$
- polinomiale: $\kappa(\mathbf{x}_i, \mathbf{x}_j) = (\gamma \mathbf{x}_i^T \mathbf{x}_j + r)^d$, $\gamma > 0$
- radial basis function (RBF): $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2)$, $\gamma > 0$
- sigmoide: $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\gamma \mathbf{x}_i^T \mathbf{x}_j + r)$

Dove γ, r e d sono i parametri che caratterizzano i vari tipi di kernel. Per ulteriori approfondimenti riguardanti la teoria alla base delle SVM, ed i compiti d'addestramento

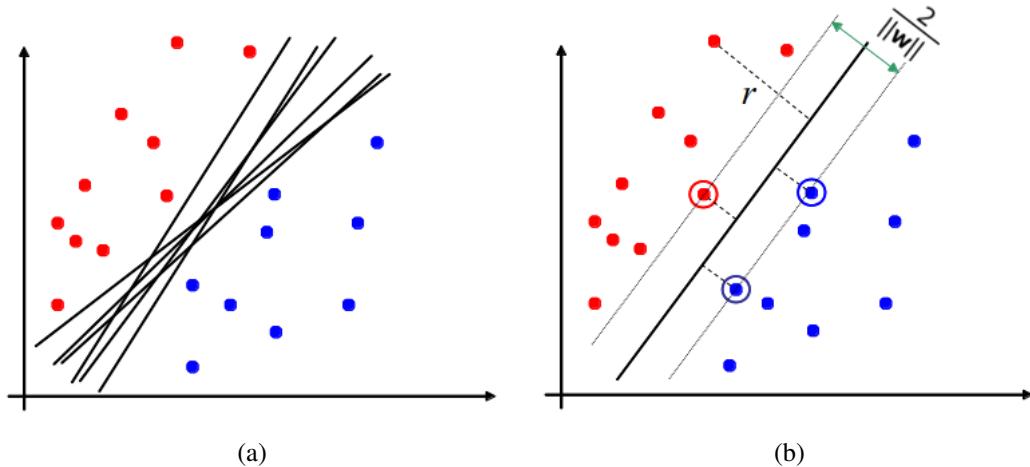


Figura 4.15: Separazione lineare di dati tramite SVM: (a) il piano in grado di separare correttamente le due classi non è univoco, (b) le SVM cercano di ricavare il piano di separazione ottimo che massimizza la distanza $\frac{2}{\|\mathbf{w}\|}$ tra le due classi. Cerchiati, i vettori di supporto che generano il piano

e classificazione ed esse collegati, si rimanda il lettore a [79].

Nello specifico, per la realizzazione di questo progetto si è scelto di utilizzare dei classificatori SVM con kernel di tipo RBF con $\gamma = 1$ e $C = 1$.

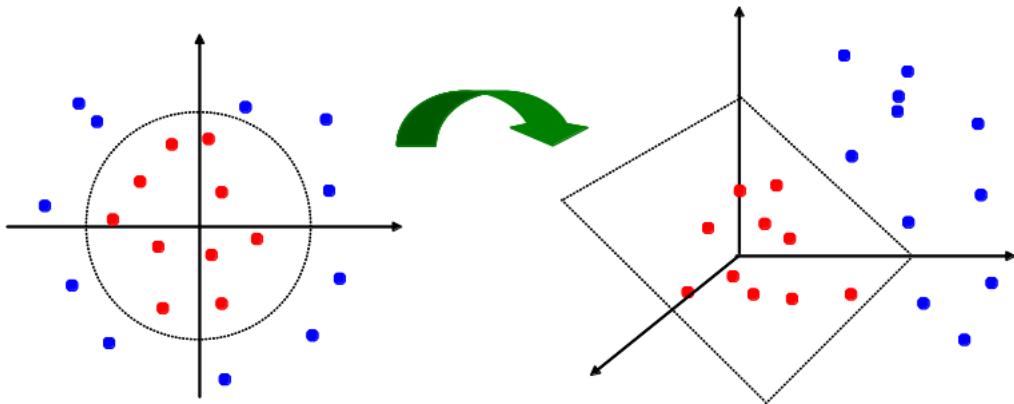


Figura 4.16: Remapping dei dati tra spazi differenti al fine di renderli linearmente separabili. Il kernel utilizzato nell'esempio è dato dalla funzione $\kappa : \mathbb{R}^2 \mapsto \mathbb{R}^3$, $\kappa(x_1, x_2) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$

4.2.4.3 Apprendimento non supervisionato

In apprendimento automatico, l'apprendimento non supervisionato consiste in una classe di problemi in cui si cerca di determinare automaticamente il modo in cui i dati sono organizzati. Al contrario del caso supervisionato, durante l'apprendimento vengono forniti al classificatore solo esempi non annotati, in quanto le classi non sono note a priori ma devono essere apprese automaticamente; questo aspetto permette quindi di poter effettuare un'azione di classificazione dei dati senza predisporre di un training set apposito per l'addestramento. Il workflow tipico seguito durante il processo di apprendimento non supervisionato è riportato in figura 4.17.

Le tecniche di apprendimento non supervisionato lavorano confrontando i dati e ricercando similarità o differenze. Sono molto efficienti con elementi di tipo numerico, dato che possono utilizzare tutte le tecniche derivate dalla statistica, ma risultano essere meno efficienti con dati non numerici. Se i dati sono dotati di un ordinamento intrinseco, gli algoritmi riescono comunque ad estrarre informazioni, al contrario possono fallire. Per identificare in maniera automatica le classi, nelle quali è possibile ripartire i dati a disposizione, gli algoritmi di apprendimento non supervisionato si basano spesso su tecniche di aggregazione o equivalentemente dette di *clustering*; tra questi, il più popolare (e semplice da implementare) è l'algoritmo *k-Means* [65].

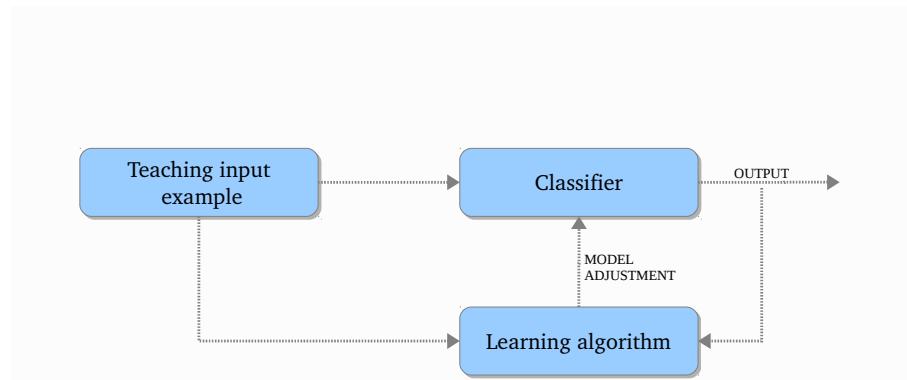


Figura 4.17: Rappresentazione del workflow per i processi d'apprendimento non supervisionato: a differenza del caso supervisionato rappresentato in figura 4.12 non è presente l'effetto del supervisore

Nel corso di questa sezione verrà brevemente illustrato l'algoritmo k-Means “base” e la definizione del modello matematico ad esso associato; verranno poi evidenziati pregi e difetti di tale approccio ed infine verrà descritto una variante dell'algoritmo k-Means, ottimizzato, implementata esclusivamente per cercare di superare ai limiti che la forma “normale” dell'algoritmo presenta.

Algoritmo k-Means base L'algoritmo k-Means è un algoritmo di clustering partizionale che permette di suddividere gruppi di oggetti in k partizioni sulla base dei loro attributi. Si assume che gli attributi degli oggetti possano essere rappresentati come vettori, e che quindi formino uno spazio vettoriale d -dimensionale.

L'obiettivo che l'algoritmo si propone è di minimizzare la varianza totale intra-cluster, matematicamente: dato un insieme di osservazioni $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, dove $\mathbf{x}_i \in \mathbb{R}^d$ l'algoritmo k-Means cerca di partizionare i dati osservati in $k \leq n$ gruppi $\mathbf{S} = \{S_1, S_2, \dots, S_k\}$ minimizzando la quantità:

$$\arg \min_{\mathbf{S}} \sum_{i=1}^k \sum_{\mathbf{x}_j \in S_i} \|\mathbf{x}_j - \mu_i\|^2 \quad (4.12)$$

dove $\mu_i \in \mathbb{R}^d$ è il centro di massa dei punti appartenenti ad $S_i \in \mathbf{S}$; ognuno di questi

insiemi viene quindi identificato mediante un centroide o punto medio.

L'algoritmo si compone di pochissimi passi ed è formulato in maniera iterativa, di conseguenza si presta facilmente ad essere implementato in sistemi di calcolo; nel listato 4.4 vengono esposti i passi di tale processo.

Listato 4.4 Algoritmo k-Means *base*

Input: Il numero di classi K nelle quali si vuole partizionare lo spazio dei dati \mathbf{X}

Output: La lista degli insiemi $\mathbf{S} = \{S_1, S_2, \dots, S_K\}$ associati ai centroidi $\bar{\mu} = \{\mu_1, \mu_2, \dots, \mu_K\}$

- 1: Inizializzazione dei centroidi $\bar{\mu}^{(0)} = \{\mu_1^{(0)}, \mu_2^{(0)}, \dots, \mu_k^{(0)}\}$
 - 2: **repeat**
 - 3: Utilizzare i centri di massa calcolati per associare i dati ai cluster secondo
 - 4: $S_i^{(t)} = \{\mathbf{x}_p : \|\mathbf{x}_p - \mu_i^{(t)}\| \leq \|\mathbf{x}_p - \mu_j^{(t)}\|, \forall j \in [1, k]\}$
 - 5:
 - 6: **for** i from 1 to k **do**
 - 7: Ricalcola ogni centroide $\mu_i^{(t+1)} = \frac{1}{|S_i^{(t)}|} \sum_{\mathbf{x}_j \in S_i^{(t)}} \mathbf{x}_j$
 - 8: **end for**
 - 9: **until** $\bar{\mu}^{(t+1)} - \bar{\mu}^{(t)} \neq \mathbf{0}$
 - 10: **return** $\mathbf{S}^{(t)} = \{S_1^{(t)}, S_2^{(t)}, \dots, S_k^{(t)}\}$
-

Tra i principali vantaggi che questo algoritmo presenta troviamo:

- Processo di semplice comprensione;
- Facilità di implementazione a causa della natura iterativa dell'algoritmo stesso;
- Rapidità di convergenza elevata, solitamente in un numero di iterazioni minore del numero di punti.

D'altro canto il k-Means presenta alcuni limiti (figura 4.18): in prima istanza presuppone la conoscenza *a priori* del numero k di classi nelle quale si desidera partizionare i dati disponibili; mentre, in termini di qualità delle soluzioni, il k-Means non garantisce il raggiungimento dell'ottimo globale: questo a causa di una possibile "cattiva" inizializzazione dei valori associati ai centroidi di partenza. Nel prossimo paragrafo verrà presentata una variante del k-Means realizzata allo scopo di superare ai limiti appena esposti.

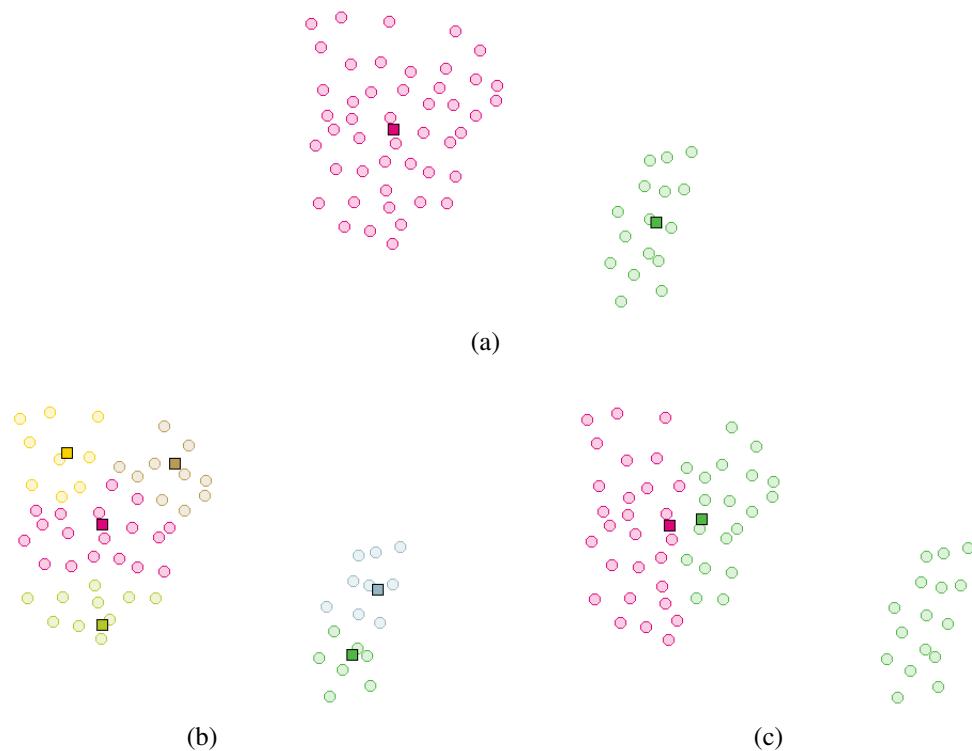


Figura 4.18: Limiti dell'algoritmo k-Means base: (a) un esempio di soluzione corretta tramite k-Means, (b) algoritmo eseguito con un numero ($K = 6$) errato di classi e (c) partizionamento scorretto ottenuto a causa di una cattiva scelta dei centroidi iniziali

Algoritmo k-Means ottimizzato Questa variante ottimizzata dell'algoritmo k-Means è stata resa necessaria a fronte dei limiti che la versione “base” presenta: nel paragrafo precedente sono stati riportati esempi di come una scelta sbagliata del numero di cluster e/o la cattiva inizializzazione dei centroidi possa portare ad una soluzione che non rappresenti l'ottimo globale; in seguito verranno esposte le tecniche adottate per superare ad entrambi i limiti.

L'ultimo dei problemi elencati, cioè la scelta poco efficiente dei centri di massa iniziali, può essere risolto adottando comunque una selezione randomica tra i punti a disposizione ma prestando attenzione a come questi vengono scelti; infatti seguendo un metodo di campionamento, comunque casuale, ma più “accorto” (che tenga in considerazione i vertici già scelti come centroidi) può migliorare notevolmente le performance dell'algoritmo. L'idea è quella di prendere casualmente il primo centro di massa tra i dati presenti, mentre la scelta dei successivi avverrà tra i dati disponibili con una probabilità proporzionale al quadrato della distanza tra l'ultimo centroide considerato ed i dati stessi [80]. Questo sarà anche l'approccio che verrà seguito durante la fase iniziale della versione migliorata del k-Means che è stato realizzato; eseguita la clusterizzazione per K classi, un indicatore (seppur particolarmente semplice) della bontà del risultato ottenuto è dato dall'indice di performance $\pi(K)$ così definito:

$$\pi(K) = \sum_{i=1}^K ||\mathbf{x}_i - \mu_i||^2 \quad (4.13)$$

Il tipo d'inizializzazione, per quanto migliorata, presenta comunque una componente non deterministica: fissato il valore di K ed eseguito il k-Means diverse volte, i risultati finali differiranno comunque tra loro. Per questo, a parità di K , si è deciso di ripetere l'algoritmo per n iterazioni e si terrà come soluzione l'istanza che ha restituito l'indice migliore (cioè il minimo tra quelli prodotti); questo al fine di incrementare ulteriormente la robustezza all'aleatorietà nella selezione dei centri di massa iniziali. In particolare nell'implementazione prodotta si è scelto $n = 25$.

Per valutare la miglior scelta del numero di classi da utilizzare si è invece proceduto attraverso l'impiego di un indice specifico, spesso utilizzato per misurare la qualità della

soluzione restituita da un algoritmo di clustering, il *Davies-Bouldin Index* (DBI) [81]. La definizione di tale indice presuppone i concetti di *varianza intra-cluster* σ_i^2 e la *distanza inter-cluster* $M_{i,j}$.

Sia C_i un cluster di elementi e sia $\mathbf{x}_j \in \mathbb{R}^d$ un elemento di C_i , definiamo σ_i^2 la *varianza intra-cluster* di C_i come:

$$\sigma_i^2 = \sqrt[q]{\frac{1}{|C_i|} \sum_{j=1}^{|C_i|} |\mathbf{x}_j - \mu_i|^q} \quad (4.14)$$

Dove $\mu_i \in \mathbb{R}^d$ è il centroide di C_i e $|C_i|$ rappresenta il numero di elementi assegnati all' i -esimo cluster. La quantità σ_i^2 rappresenta lo *scatter* del cluster, ovvero una stima del grado di dispersione dei punti appartenenti a C_i .

Definiamo $M_{i,j}$ la *distanza inter-cluster* tra i cluster C_i e C_j come:

$$M_{i,j} = \|\mu_i - \mu_j\|_q = \sqrt[q]{\sum_{k=1}^d |\mu_i^{(k)} - \mu_j^{(k)}|^q} \quad (4.15)$$

Di fatto $M_{i,j}$ rappresenta il grado di separazione tra i cluster C_i e C_j , mentre $\mu_i^{(k)}$ rappresenta la k -esima componente del vettore μ_i .

È importante sottolineare che, al fine di ottenere risultati consistenti, è necessario che la metrica utilizzata durante l'esecuzione dell'algoritmo k-Means sia la stessa utilizzata per il calcolo di σ_i^2 e $M_{i,j}$; nello specifico è stata utilizzata la *distanza euclidea*, quindi in questo caso $q = 2$.

Definiamo inoltre la quantità $R_{i,j}$ come la stima di quanto “buono” è il risultato restituito dall'algoritmo di clustering utilizzato. Questa misura, per definizione deve prendere in considerazione i parametri σ_i^2 e $M_{i,j}$ introdotti in precedenza, il primo esprime la compattezza dei dati all'interno del cluster (più questo risulta compatto più il suo valore è basso), mentre il secondo indica la distanza reciproca tra le classi ricavate la quale, idealmente, deve essere la più alta possibile. Di conseguenza l'indice di Davies Bouldin è legato al rapporto $R_{i,j}$, che a sua volta è definito da σ_i^2 e $M_{i,j}$, nella seguente maniera:

$$R_{i,j} = \frac{\sigma_i^2 + \sigma_j^2}{M_{i,j}} \quad (4.16)$$

Osserviamo che per $R_{i,j}$ valgono le seguenti proprietà:

1. $R_{i,j} \geq 0$: condizione di non negatività;
2. $R_{i,j} = R_{j,i}$: condizione di simmetria;
3. $\sigma_j^2 \geq \sigma_k^2 \wedge M_{i,j} = M_{i,k} \Rightarrow R_{i,j} \geq R_{i,k}$: a parità di distanza inter-cluster la varianza intra-cluster determina il rapporto $R_{i,j}$;
4. $\sigma_j^2 = \sigma_k^2 \wedge M_{i,j} \leq M_{i,k} \Rightarrow R_{i,j} \geq R_{i,k}$: a parità di varianza intra-cluster la distanza inter-cluster determina il rapporto $R_{i,j}$.

Dalla formulazione e le proprietà esposte per $R_{i,j}$ si evince che, più questo rapporto è basso, migliore è la separazione tra i cluster e maggiore è la loro compattezza interna.

Fissato il numero di cluster K , l'indice di Davies-Bouldin può essere quindi definito dalle seguenti relazioni:

$$\begin{aligned} D &= \max_{j:i \neq j} R_{i,j} \\ DBI(K) &\equiv \frac{1}{K} \sum_{i=1}^K D_i \end{aligned} \tag{4.17}$$

La quantità D_i sceglie il rapporto $R_{i,j}$ più grande, cioè che rappresenta lo scenario più sfavorevole per ogni cluster C_i ; il DBI è quindi ottenuto come la media aritmetica tra i vari D_i calcolati. Pertanto, trovare il valore di K per cui l'indice di Davies-Bouldin è minimo coincide con il trovare il numero ottimo di cluster nei quali sono raggruppabili i dati da classificare. Questo indice risulta particolarmente utile in quanto permette di applicare un processo di classificazione non supervisionata senza dover conoscere a priori il numero di cluster da costituire, anzi il numero di classi ottimale è individuato dal procedimento stesso.

Il listato 4.5 presenta infine lo pseudo-codice relativo all'algoritmo k-Means ottimizzato per la classificazione non supervisionata dei vettori di features calcolati sui voxel costituenti la point cloud. Il procedimento sviluppato sfrutta l'implementazione k-Means già presente, e quindi immediatamente fruibile, inclusa nelle librerie OpenCV; le API

messe a disposizione permettono, tramite l'invocazione di un'unica funzione (corredata di opportuni parametri), l'esecuzione della clusterizzazione con un'inizializzazione "intelligente" dei centroidi (secondo il metodo esposto in precedenza), la reiterazione dell'algoritmo per n volte (a parità di cluster K) ed è capace di restituire automaticamente il raggruppamento associato all'indice di performance $\pi(K)$ migliore.

Combinando questo effetto al calcolo del DBI e ad un incremento progressivo di K , siamo in grado di definire una strategia di classificazione non supervisionata che richiede esclusivamente la definizione dei dati di input ed è capace di raggrupparli automaticamente nel miglior modo possibile.

Listato 4.5 Algoritmo k-Means ottimizzato

Input: Lo spazio dei dati $\mathbf{X} = \{\mathbf{x}_i : \mathbf{x}_i \in \mathbb{R}^d\}$ sul quale si vuole effettuare la clusterizzazione, il limite massimo $K_{max} \leq |\mathbf{X}|$ di raggruppamenti desiderati ed il numero n di iterazioni k-Means

Output: Il numero ottimo di cluster K^* e la lista degli insiemi $\mathbf{S} = \{S_1, S_2, \dots, S_{K^*}\}$ associati ai centroidi $\bar{\mu} = \{\mu_1, \mu_2, \dots, \mu_{K^*}\}$

```

1:  $K \leftarrow 1$ 
2:  $K^* \leftarrow 1$ 
3:  $DBI^* \leftarrow +\infty$ 
4:
5: while  $K \leq K_{max}$  do
6:    $(\mathbf{S}, \bar{\mu}) \leftarrow cvKMeans(\mathbf{X}, K, n)$ 
7:    $DBI(K) \leftarrow ComputeDBI(K, \mathbf{S}, \bar{\mu})$ 
8:   if  $DBI(K) < DBI^*$  then
9:      $DBI^* \leftarrow DBI(K)$ 
10:     $K^* \leftarrow K$ 
11:   end if
12: end while
13:
14:  $(\mathbf{S}, \bar{\mu}) = cvKMeans(\mathbf{X}, K^*, n)$ 
15: return  $(K^*, \mathbf{S}, \bar{\mu})$ 

```

Il limite principale del classificatore non supervisionato presentato in questo paragrafo, ma anche degli altri classificatori, è quella di prendere decisioni sulla base del singolo voxel senza considerare il contesto. Il Markov Random Field permette di raffinare il risultato prendendo in considerazione non solo la classe di appartenenza del voxel cor-

rente, ma anche la configurazione degli elementi, ad esso adiacenti, che compongono il suo vicinato.

Nel prossimo capitolo verrà quindi introdotto il MRF utilizzato al fine di migliorare la segmentazione ottenuta dopo la fase di classificazione. I risultati derivanti da questo processo di apprendimento rappresenteranno quindi la base di partenza per la definizione di un MRF associato alla scena; l'inferenza su tale modello ci consentirà infine di ricavare l'output finale relativo all'etichettatura dello spazio osservato.

4.2.5 Segmentazione mediante Markov Random Field (MRF)

Come già descritto nel Capitolo 3.1, il compito dei problemi di segmentazione cercano di categorizzare i punti interni della scena assegnando ad ognuno una precisa etichetta appartenente ad un insieme finito di classi. Il risultato finale di questo processo solitamente presenta un'elevata frammentazione e discontinuità nella etichettatura. Una soluzione è quella di introdurre dei vincoli di contesto che riducano gli errori, introdotti durante l'assegnamento delle etichette, e gli effetti di rumore accidentali.

All'interno della scena è possibile ipotizzare che le caratteristiche assegnate ad ogni punto varino dolcemente se questi appartengono alla stessa entità; al contrario, cambiamenti di queste features si verificheranno invece in corrispondenza *bordi di transizione* tra pezzi di scena reciprocamente appartenenti a categorie differenti. L'ipotesi che la classe alla quale può venir associato ogni punto della scena, dipenda esclusivamente dallo stato dei punti a lui vicini, rende possibile la definizione di un problema di minimizzazione attraverso i Markov Random Field. Boykov et al. [71] hanno proposto un algoritmo per trovare la miglior etichettatura ricavabile per un insieme di dati sia equivalente a risolvere un problema MAP. Nella formulazione proposta occorre definire la struttura del vicinato intorno ad ogni singolo punto; inoltre è necessario ricavare la distribuzione di probabilità a priori che governa l'assegnazione dei dati all'insieme delle possibili etichette. Il processo di classificazione svolto al passo precedente è ciò che ci permetterà di definire il potenziale unario $\psi_i(v_i)$ (il potenziale di coppia $\phi_{ij}(v_i, v_j)$ è semplicemente una costante) introdotto in precedenza alla Sezione 3.1.1; il modo in cui verranno definiti questi potenziali caratterizzerà la funzione energia $E(\mathcal{V})$ che si andrà

a minimizzare.

Infine sarà possibile ottenere i risultati finali di segmentazione risolvendo il problema di ottimizzazione attraverso l'impiego di algoritmi di taglio come l' $\alpha\beta$ -Swap e l' α -Expansion.

4.2.5.1 Platt scaling

Va ricordato che allo step di classificazione supervisionata, per ogni voxel v_i , il valore di appartenenza ad una determinata categoria è un numero reale che varia all'interno dell'intervallo $I = [-1, 1]$. Per poter operare con valori con valori compatibili con l'espressione di un valore di probabilità è necessario rimappare questo intervallo I in un secondo range $I' = [0, 1]$: questo è ottenibile attraverso una tecnica che prende il nome di *Platt scaling* [82]. Questo mapping può essere ottenuto definendo una funzione $\Gamma : \mathbb{R}_{[-1,1]} \mapsto \mathbb{R}_{[0,1]}$ tale per cui:

$$\Gamma(c|v_i) = \frac{1}{1 + e^{-\nu s_c(v_i) + \zeta}} \quad (4.18)$$

Dove $s_c(v_i) \in I$ rappresenta il risultato ottenuto tramite processo di classificazione supervisionata per la classe c e l' i -esimo voxel v_i , mentre ν e ζ rappresentano i parametri di regolazione della funzione; in fase di implementazione sono stati scelti $\nu = 5$ e $\zeta = 0$ (figura 4.19).

Le categorie scelte per la classificazione sono essenzialmente tre: *Floor*, *Wall* e *Door*; per queste la probabilità di appartenenza dell' i -esimo voxel è data dall'equazione 4.18; in più è stata prevista un'ulteriore classe, ribattezzata *Other*, utilizzata per rappresentare tutte quelle parti della scena non associabili alle tre macro-classi "principali". Per ottenere la probabilità di appartenenza a quest'ultima classe osserviamo che, avendo definito tre classi principali, la seguente relazione è senz'altro vera per *Other*:

$$\Gamma(c_{other}|v_i) \leq 1 - \sum_{c=1}^C \Gamma(c|v_i) \quad \forall c \in \{1, \dots, C\} \quad (4.19)$$

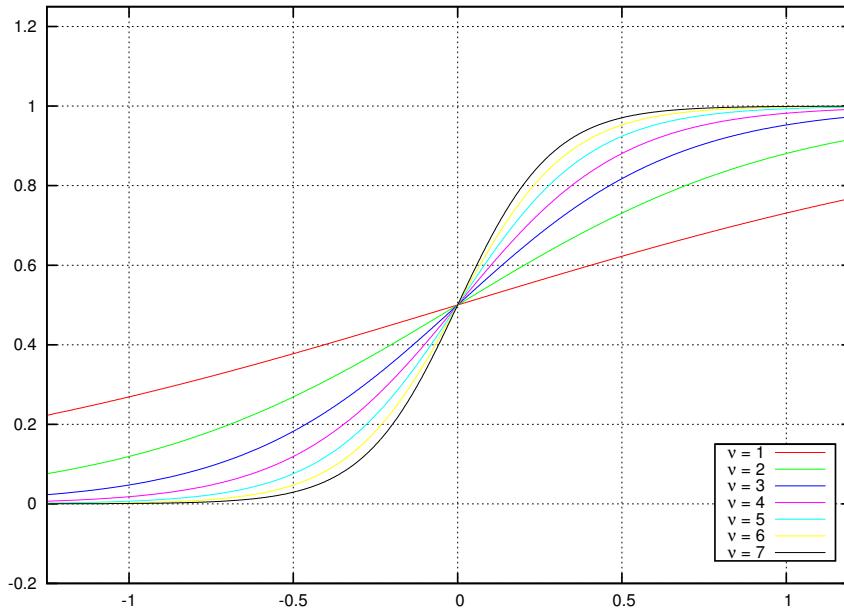


Figura 4.19: La funzione di Platt scaling Γ per diversi valori di ν e $\zeta = 0$

Quindi è possibile utilizzare il più piccolo dei C valori ottenibili come limite superiore per la probabilità di appartenenza alla classe *Other*, formalmente:

$$\Gamma(c_{other}|v_i) = \alpha \min_{1 \leq c \leq C} \{1 - \Gamma(c|v_i)\} \quad \alpha \in \mathbb{R} \quad (4.20)$$

Dove il parametro $\alpha \in (0, 1]$ modula il valore dell'*upper bound* appena introdotto.

4.2.5.2 Distance scaling

Per quanto riguarda il processo di classificazione non supervisionata, terminato il calcolo dei centroidi, è necessario recuperare il valore di probabilità d'appartenenza di ogni voxel rispetto alle classi ricavate. Questo tipo d'informazione è facilmente ottenibile calcolando la distanza tra il vettore di features $f_i \in \mathbb{R}^d$ ed ogni elemento $\bar{\mu}^{(c)} \in \mathbb{R}^d$ della lista di centroidi $\bar{\mu}$ ricavata; una volta ottenute queste distanze bisognerà effettuare un'operazione di riscalatura per poter ottenere una lista di valori (reali, non negativi e compresi nell'intervallo $[0, 1]$) che possano esprimere le probabilità d'appartenenza di un voxel ad ognuna delle classi ricavate.

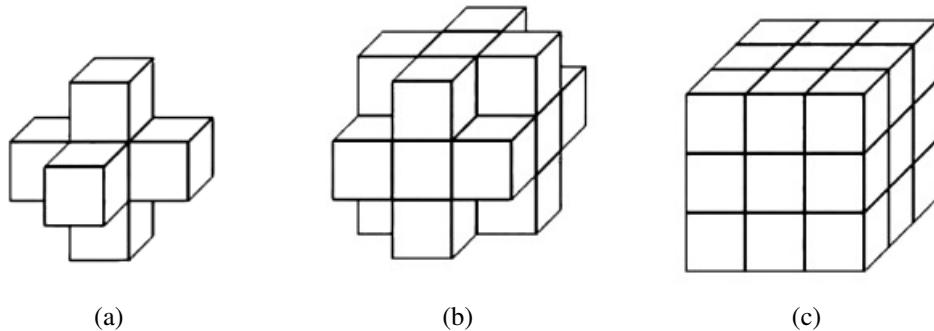


Figura 4.20: Sistemi di vicinato tridimensionale: (a) 6-vicinato, (b) 18-vicinato e (c) 26-vicinato

Considerato il voxel v_i ed il numero di cluster K , l'insieme delle distanze associate \mathcal{D}_i è definibile come:

$$\mathcal{D}_i = \{d_i^{(c)} \in \mathbb{R} : d_i^{(c)} = \|\mathbf{f}_i - \bar{\mu}^{(c)}\|^2, \forall c \in [1, K]\} \quad (4.21)$$

Successivamente per ogni voxel v_i viene quindi ricavato il valore d'appartenenza alla classe c attraverso la seguente relazione:

$$\Gamma(c|v_i) \propto 1 - \frac{d_i^{(c)}}{\sum_{c=1}^K d_i^{(c)}}, \quad \forall c \in [1, K] \quad (4.22)$$

In altre parole, $\Gamma(c|v_i)$ esprime il valore di probabilità d'appartenenza del voxel v_i alla classe c a meno di un fattore moltiplicativo.

4.2.5.3 Definizione del grafo e funzione energia associata

Una volta applicata la trasformazione Γ , i potenziale unario $\psi_i(v_i)$ ed il potenziale di coppia $\phi_{ij}(v_i, v_j)$ sono definiti nel seguente modo:

$$\begin{aligned} \psi_i(v_i) &= -\omega_U \log \Gamma(c|v_i) \\ \phi_{ij}(v_i, v_j) &= \omega_P \delta(v_i, v_j) \end{aligned} \quad (4.23)$$

Dove ω_U e ω_P sono gradi di libertà introdotti per regolare meglio il fattore d'incidenza dei due potenziali (unario e di coppia rispettivamente), mentre per $\phi_{ij}(v_i, v_j)$ si è

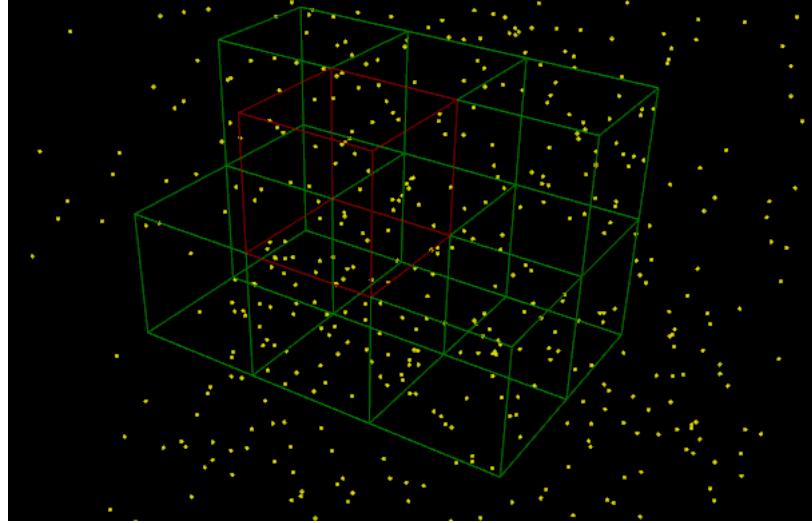


Figura 4.21: 26-vicinato variabile: in rosso l' i -esimo voxel considerato ed in verde i voxel vicini, disponibili solo 9 dei 26 elementi appartenenti al vicinato

scelto un modello di Potts nel quale $\delta(v_i, v_j)$ è una funzione gradino che vale 1 qualora la classe assegnata a v_i e v_j (appartenenti allo stesso vicinato \mathcal{N}) fosse la stessa, 0 altrimenti. Avendo a che fare con dati tridimensionali, il sistema di vicinato \mathcal{N} (figura 4.20) definito per ogni voxel $v_i \in \mathcal{V}$ è costituito dal *26-vicinato variabile* di ogni elemento; *variabile* siccome in questo caso abbiamo a che fare con una nuvola di punti che per sua natura non presenta una topologia definibile a priori e, di conseguenza, ogni voxel potrà avere un vicinato \mathcal{N} composto da un numero mutevole di elementi compreso tra 0 e 26. In figura 4.21, un esempio di point cloud nella quale è stato evidenziato un suo voxel, scelto a caso, ed i relativi elementi vicini.

Conseguentemente a tutto ciò la funzione energia $E(\mathcal{V})$ da minimizzare diventa quindi:

$$\begin{aligned} E(\mathcal{V}) &= \sum_{i \in \mathcal{V}} \psi_i(v_i) + \sum_{(i,j) \in \mathcal{N}} \phi_{ij}(v_i, v_j) \\ &= -\omega_U \sum_{i \in \mathcal{V}} \log \Gamma(c|v_i) + \omega_P \sum_{(i,j) \in \mathcal{N}} \delta(v_i, v_j) \end{aligned} \quad (4.24)$$

A questa funzione energia viene quindi associato un grafo \mathcal{G} che conterrà due tipi di vertici:

- Vertici di tipo p (o *p-vertici*), ognuno dei quali sarà associato ad un voxel $v_i \in \mathcal{V}$;

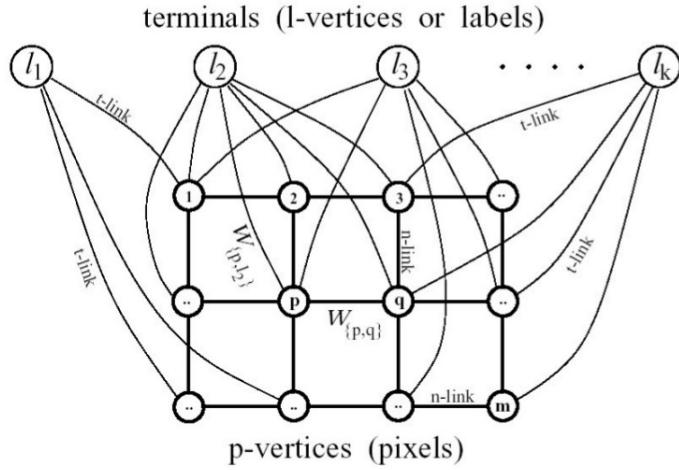


Figura 4.22: Esempio di un semplice grafo bidimensionale 3×4 dove sono presenti p -vertici, l -vertici e collegamenti sia di tipo n che di tipo t

- Vertici di tipo l (o l -vertici), che corrisponderanno all’etichetta alla quale sarà associabile ogni voxel e coincideranno con i nodi terminali del grafo durante l’applicazione degli algoritmi di taglio.

Tutti gli archi che appartengono al sistema di vicinato \mathcal{N} sono anche archi di \mathcal{G} ; questi collegamenti sono chiamati collegamenti di tipo n (o n -links), mentre gli archi tra vertici di tipo p e di tipo n prendono il nome di collegamenti di tipo t (o t -links); gli archi di quest’ultimo tipo vengono aggiunti a \mathcal{G} per poter associare al grafo un problema di taglio a costo minimo: ai t -links sono assegnati determinati pesi in base al primo termine di $E(\mathcal{V})$ (il potenziale unario $\psi_i(v_i)$), invece il peso associato ai n -links è caratterizzato dal potenziale di coppia $\phi_{i,j}(v_i, v_j)$. Mentre i collegamenti di tipo n sono bidirezionali i collegamenti di tipo l hanno un’orientazione ben definita: partiranno dal nodo sorgente e andranno a collimare sul nodo destinazione. La figura 4.22 rappresenta un semplice esempio di grafo bidimensionale 3×4 associato alla funzione energia appena definita.

4.2.5.4 Processo d’inferenza e minimizzazione dell’energia

La difficoltà principale che risiede nei problemi di minimizzazione è il costo computazionale che la loro risoluzione può richiedere. In genere la funzione energia da minimizzare è una funzione *non convessa* che presenta diversi punti di minimo locale e, definito

\mathcal{V} l'insieme degli elementi da etichettare, il suo spazio delle soluzioni ha dimensionalità $|\mathcal{V}|$ (che solitamente è dell'ordine delle migliaia). Sfortunatamente minimizzare una funzione energia arbitraria è un problema, in generale *NP-hard*, che richiede un tempo esponenziale per essere risolto. In alcuni casi particolari $E(\mathcal{V})$ può essere minimizzata in maniera esatta, ma sotto la condizione che l'insieme \mathcal{L} dei possibili label abbia cardinalità 2; in questo caso può essere trovata la soluzione in tempo polinomiale calcolando il taglio a costo minimo su un grafo appositamente creato [83]. Se \mathcal{L} fosse un insieme finito di dimensione unitaria e $\phi_{ij}(v_i, v_j) = |l_i - l_j|$ anche in questo caso il punto di minimo può essere ricavato in maniera esatta attraverso tagli a costo minimo [84], [85]; un'altra condizione per la quale è possibile ottenere una soluzione esatta in tempo polinomiale è il caso in cui il grafo non presenti cicli al suo interno, ma questa condizione non rappresenta il caso sul quale stiamo lavorando: di fatto la topologia del grafo costruito è strettamente dipendente dalla scena osservata e quindi del tutto arbitraria. In generale la soluzione esatta del problema, per un qualsiasi insieme di etichette \mathcal{L} , risulta avere complessità *NP-hard* [86].

In questa sezione verranno presentati due algoritmi che minimizzano in maniera approssimata $E(\mathcal{V})$, per un qualsiasi insieme di etichette \mathcal{L} , sotto la condizione che il potenziale d'interazione $\phi_{ij}(v_i, v_j)$ sia una *metrica* o una *semi-metrica*. Una generica funzione potenziale Φ può essere considerata una *semi-metrica* se per ogni coppia di etichette α e β valgono le prime due delle tre seguenti due proprietà:

1. $\Phi(\alpha, \beta) = \Phi(\beta, \alpha) \geq 0$;
2. $\Phi(\alpha, \beta) = 0 \Leftrightarrow \alpha = \beta$;
3. $\Phi(\alpha, \beta) \leq \Phi(\alpha, \gamma) + \Phi(\gamma, \beta)$.

Nel caso in cui Φ goda anche della terza proprietà per ogni $\alpha, \beta, \gamma \in \mathcal{L}$ (disegualanza triangolare) allora si dice che Φ è una *metrica*. Osserviamo, per esempio, che il *modello quadratico troncato* ed il *modello di Potts standard* introdotti nella sezione 3.1.2 sono entrambi delle metriche [6].

Partendo da quanto detto alla sezione precedente, il taglio a costo minimo di \mathcal{G} (figura 4.23) garantirà la minimizzazione di $E(\mathcal{V})$ lasciando connesso ogni *p*-vertice esclusivamente a vertici simili, cioè che faranno parte della stessa regione etichettata, e tutti

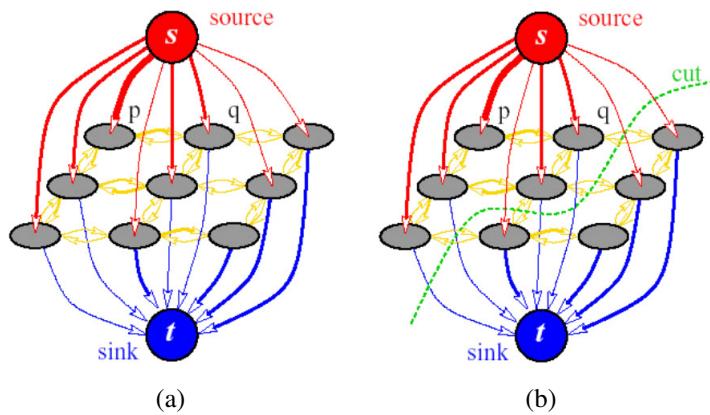


Figura 4.23: Esempio di taglio a costo minimo: partendo dal grafo creato (a), il taglio ottenuto minimizza la funzione energia $E(V)$ recidendo t -links ed n -links lasciando i collegamenti che rappresenteranno la segmentazione finale (b)

questi saranno collegati ad un unico *l-vertice* che rappresenterà il label ad essi associato. Gli algoritmi che permetteranno una minimizzazione approssimata delle funzione energia, attraverso la risoluzione di un problema di taglio a costo minimo, sono l' $\alpha\beta$ -Swap e l' α -Expansion. In seguito verranno esplicite le caratteristiche alla base di entrambi i metodi, per ognuno di essi verrà descritto sommariamente lo pseudo-codice seguito e le operazioni elementari (chiamate anche *mosse*) effettuate per l'ottenimento del risultato finale. Di fatto entrambi gli algoritmi seguono un processo di tipo iterativo e differiscono sostanzialmente per il tipo di mossa compiuta ad ogni passaggio.

Partizioni e spazio delle mosse In questo paragrafo si da una breve introduzione del concetto di *mossa* che rappresenterà l'operazione elementare e compiuta ad ogni ciclo durante l'esecuzione di entrambi gli algoritmi. Di fatto la mossa compiuta è l'unico tratto distintivo tra le due tecniche di inferenza che verranno analizzate nel corso di questa sezione.

Innanzi di tutto è necessario introdurre il concetto di *partizione*: ogni etichettatura della scena ε può essere rappresentata da una partizione dei voxel $\mathbf{P} = \{\mathcal{P}_l : l \in \mathcal{L}\}$ dove $\mathcal{P}_l = \{p \in \mathcal{P} : \varepsilon_p = l\}$ è un sottoinsieme di voxel assegnati all'etichetta l .

Data una coppia di label α e β una mossa da una partizione P (etichettatura ε) ad una partizione P' (etichettatura ε') è chiamata un $\alpha\beta$ -Swap se $P_l = P'_l$ per ogni $l \neq \alpha, \beta$ e

$\mathcal{P}_\alpha \cap \mathcal{P}'_\beta \neq \emptyset \vee \mathcal{P}_\beta \cap \mathcal{P}'_\alpha \neq \emptyset$. Questo significa che l'unica differenza tra \mathbf{P} e \mathbf{P}' sono alcuni voxel che prima dello scambio erano associati all'etichetta α mentre ora sono associati al label β e viceversa.

Dato un label α , una mossa da una partizione \mathbf{P} (etichettatura ε) ad una partizione \mathbf{P}' (etichettatura ε') è chiamata un' α -*Expansion* se $\mathcal{P}_\alpha \subset \mathcal{P}'_\alpha$ e $\mathcal{P}_l \subseteq \mathcal{P}'_l$ per ogni etichetta $l \neq \alpha$; in altre parole un' α -*Expansion* permette ad un sottoinsieme di voxel di cambiare la propria etichetta associata in α .

L'algoritmo di $\alpha\beta$ -Swap Iniziamo presentando il listato dello pseudo-codice alla base del processo di $\alpha\beta$ -Swap:

Listato 4.6 Algoritmo di $\alpha\beta$ -Swap

Input: Una labellizzazione arbitraria ε

Output: La labellizzazione ε corrispondente alla soluzione del problema di minimizzazione dell'energia

```

1: success  $\leftarrow$  false
2:
3: for all coppia di label  $\{\alpha, \beta\}$  in  $\mathcal{L}$  do
4:    $\hat{\varepsilon} \leftarrow \arg \min E(\varepsilon')$  con  $\varepsilon'$  scelto tra tutte le etichettature  $\varepsilon$  ottenibili mediante
     una mossa di  $\alpha\beta$ -Swap
5:   if  $E(\hat{\varepsilon}) < E(\varepsilon')$  then
6:      $\varepsilon \leftarrow \hat{\varepsilon}$ 
7:     success  $\leftarrow$  true
8:   end if
9: end for
10:
11: if success = true then
12:   goto 1
13: end if
14:
15: return  $\varepsilon$ 

```

Per rendere più agevole la descrizione dell'algoritmo introduciamo i concetti di *ciclo* e *iterazione*: con *ciclo* si intende l'insieme di istruzioni compiute dalla riga 1 alla riga 13, mentre useremo il termine *iterazione* per le operazioni compiute nell'intervallo di righe dalla 4 alla 8. Ad ogni ciclo l'algoritmo compie un'iterazione per ogni coppia di

etichette (α e β appunto), l'ordine nel quale queste vengono considerate può essere prefissato o casuale. Un ciclo può considerarsi concluso con successo se, tra le iterazioni considerate, si è riuscito a trovarne una che definisce una nuova etichettatura $\hat{\varepsilon}$ in grado di restituire un valore d'energia $E(\hat{\varepsilon})$ strettamente minore di quello attuale; l'algoritmo si arresta alla fine del primo ciclo che si conclude senza riuscire a minimizzare ulteriormente la funzione energia. Osserviamo infine che un ciclo all'interno dell'algoritmo di $\alpha\beta$ -Swap richiede esattamente $|\mathcal{L}|^2$ iterazioni.

Descritto il flusso d'esecuzione, e tenuto a mente quanto detto all'inizio di questa sezione, diamo ora ad la descrizione formale del significato di $\alpha\beta$ -Swap; successivamente vedremo secondo quali criteri verranno generati i pesi (o capacità) associati ai vari t -link ed n -links; questi valori sono di fondamentale importanza per la corretta esecuzione dell'algoritmo: il taglio a costo minimo verrà infatti generato sulla base delle capacità associate agli archi del grafo.

Data un'etichettatura ε (partizione \mathbf{P}) e una coppia di etichette (α, β) , si vuole ricavare una nuova etichettatura $\hat{\varepsilon}$, che minimizzi al meglio l'energia E , con un $\alpha\beta$ -Swap di ε . Per far questo si calcola l'etichettatura ottenuta dal taglio a costo minimo per il grafo $\mathcal{G}_{\alpha\beta} = \langle \mathcal{V}_{\alpha\beta}, \mathcal{E}_{\alpha\beta} \rangle$, la struttura di questo grafo è determinata dinamicamente dalla partizione corrente \mathbf{P} e dalla coppia di etichette (α, β) scelta.

L'insieme dei vertici $\mathcal{V}_{\alpha\beta}$ è quindi formato dalla coppia di label considerati per quell'iterazione, che fungeranno da nodo sorgente e nodo terminale, ed tutti i nodi corrispondenti ai voxel etichettati secondo tali label (cioè i voxel appartenenti agli insiemi \mathcal{P}_α e \mathcal{P}_β); formalmente $\mathcal{V}_{\alpha\beta} = \{\alpha, \beta\} \cup \mathcal{P}_{\alpha\beta}$ con $\mathcal{P}_{\alpha\beta} = \mathcal{P}_\alpha \cup \mathcal{P}_\beta$. Riassumendo, ogni voxel $v \in \mathcal{P}_{\alpha\beta}$ è quindi connesso ai nodi terminali attraverso t -link mentre ogni coppia di voxel $\{p, q\} \subset \mathcal{P}_{\alpha\beta}$ vicini (cioè $\{p, q\} \in \mathcal{N}$) è connessa da archi $e_{\{p,q\}}$ chiamati n -link (figura 4.24). L'insieme degli archi $\mathcal{E}_{\alpha\beta}$ è costituito da $\bigcup_{p \in \mathcal{P}_{\alpha\beta}} \{t_p^\alpha, t_p^\beta\}$ (t -link) e $\bigcup_{\{p,q\} \in \mathcal{N}, p, q \in \mathcal{P}_{\alpha\beta}} e_{\{p,q\}}$ (n -link); le capacità assegnate a tali archi sono riportati in tabella 4.1.

Ogni taglio \mathcal{T} su $\mathcal{G}_{\alpha\beta}$ genera il grafo indotto $\mathcal{G}(\mathcal{T}) = \langle \mathcal{V}, \mathcal{E} - \mathcal{T} \rangle$ e recide esattamente un t -link per ogni voxel $v_p \in \mathcal{P}_{\alpha\beta}$: se così non fosse allora esisterebbe un percorso ammissibile dal nodo sorgente α al nodo terminale β e verrebbe a meno la definizione stessa

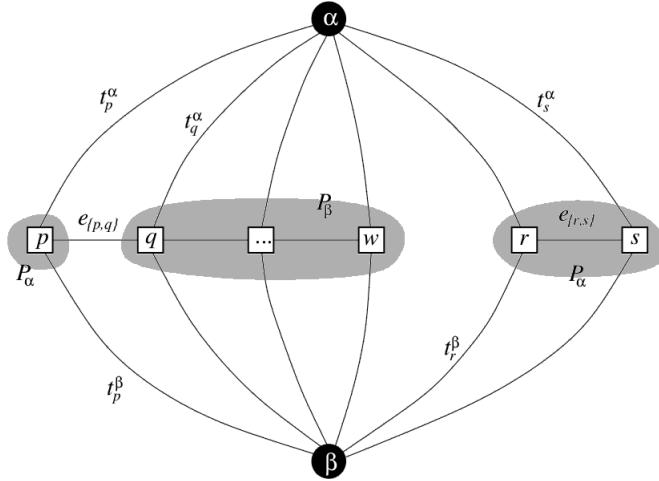


Figura 4.24: Esempio di grafo associato all'algoritmo di $\alpha\beta$ -Swap per un'immagine 1D. L'insieme degli elementi (pixel) è $\mathcal{P}_{\alpha\beta} = \mathcal{P}_\alpha \cup \mathcal{P}_\beta$ dove $\mathcal{P}_\alpha = \{p, r, s\}$ e $\mathcal{P}_\beta = \{q, \dots, w\}$

Edge	Weight	For
t_p^α	$\psi_p(\alpha) + \sum_{q \in \mathcal{N}_p, q \notin \mathcal{P}_{\alpha\beta}} \phi_{p,q}(\alpha, \varepsilon_q)$	$p \in \mathcal{P}_{\alpha\beta}$
t_p^β	$\psi_p(\beta) + \sum_{q \in \mathcal{N}_p, q \notin \mathcal{P}_{\alpha\beta}} \phi_{p,q}(\beta, \varepsilon_q)$	$p \in \mathcal{P}_{\alpha\beta}$
$e_{\{p,q\}}$	$\phi_{p,q}(\alpha, \beta)$	$\{p, q\} \in \mathcal{N} \wedge p, q \in \mathcal{P}_{\alpha\beta}$

Tabella 4.1: Le capacità associate agli archi $\mathcal{E}_{\alpha\beta}$ per l'algoritmo $\alpha\beta$ -Swap

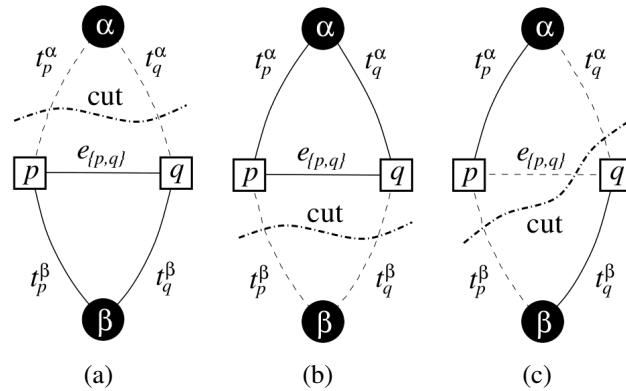


Figura 4.25: Esempio di assegnazione dei nodi p e q dopo il taglio \mathcal{T} : (a) entrambi i nodi vengono associati all’etichetta α , (b) i nodi vengono associati al label β e (c) q viene associato ad α e p a β

di taglio. Il taglio \mathcal{T} definisce quindi immediatamente una nuova possibile etichettatura $\varepsilon^{\mathcal{T}}$ della scena, formalmente:

$$\varepsilon_p^{\mathcal{T}} = \begin{cases} \alpha & \text{se } t_p^\alpha \in \mathcal{T} \text{ per } p \in \mathcal{P}_{\alpha\beta} \\ \beta & \text{se } t_p^\beta \in \mathcal{T} \text{ per } p \in \mathcal{P}_{\alpha\beta} \\ \varepsilon_p & \text{per } p \in \mathcal{P}, p \notin \mathcal{P}_{\alpha\beta} \end{cases} \quad (4.25)$$

In altre parole, se il voxel p dovesse appartenere a $\mathcal{P}_{\alpha\beta}$ allora sarà etichettato con il label α nell'eventualità in cui venisse separato dal nodo terminale α tramite \mathcal{T} ; allo stesso modo un voxel verrà associato all'etichetta β nel caso in cui appartenesse a $\mathcal{P}_{\alpha\beta}$ e \mathcal{T} recidesse l'arco di connessione tra quel nodo ed il terminale β . Per quei nodi che non appartengono alla partizione $\mathcal{P}_{\alpha\beta}$ (che quindi non sono etichettati con i due label considerati per l'iterazione corrente) verrà mantenuto label ε_p . In figura 4.25 alcune delle situazioni verificabili dopo l'applicazione del taglio \mathcal{T} .

L'algoritmo di α -Expansion Questo algoritmo è applicabile esclusivamente a quelle funzioni energia in cui il generico potenziale d'interazione Φ è una *metrica* [6]; anche in questo caso presenteremo lo pseudo-codice che descrive il processo d'inferenza e successivamente analizzeremo più in dettaglio le caratteristiche di tale procedimento.

Listato 4.7 Algoritmo di α -Expansion

Input: Una labellizzazione arbitraria ε

Output: La labellizzazione ε corrispondente alla soluzione del problema di minimizzazione dell'energia

```
1: success  $\leftarrow$  false
2:
3: for all label  $\alpha$  in  $\mathcal{L}$  do
4:    $\hat{\varepsilon} \leftarrow \arg \min E(\varepsilon')$  con  $\varepsilon'$  scelto tra tutte le etichettature  $\varepsilon$  ottenibili mediante
    una mossa di  $\alpha$ -Expansion
5:   if  $E(\hat{\varepsilon}) < E(\varepsilon')$  then
6:      $\varepsilon \leftarrow \hat{\varepsilon}$ 
7:     success  $\leftarrow$  true
8:   end if
9: end for
10:
11: if success = true then
12:   goto 1
13: end if
14:
15: return  $\varepsilon$ 
```

Come è facile notare, l'unica differenza dal listato 4.6 è rappresentata dal passo di iterazione compiuto: al posto di una mossa di $\alpha\beta$ -Swap ne troviamo invece una di α -Expansion.

La differenza principale di questo algoritmo rispetto a quello precedentemente esposto, è che ogni iterazione contempla esclusivamente un unico label α invece che una coppia di etichette (α, β) : a causa di tutto ciò ogni ciclo di α -Expansion richiederà esattamente $|\mathcal{L}|$ iterazioni (invece che $|\mathcal{L}|^2$ richieste nel caso dell' $\alpha\beta$ -Swap). In seguito, anche per questo caso, la descrizione formale della procedura di α -Expansion e le regole definite per l'assegnazione della capacità ai vari archi del grafo.

Data un'etichettatura ε (partizione \mathbf{P}) ed un label α , vogliamo trovare una nuova etichettatura $\hat{\varepsilon}$ che minimizzi E tra tutte le etichettature ottenibili attraverso una α -Expansion di ε . Questo procedimento si può ricondurre al calcolo del taglio a costo minimo \mathcal{T} sul grafo $\mathcal{G}_\alpha = \langle \mathcal{V}_\alpha, \mathcal{E}_\alpha \rangle$; la struttura di tale grafo è determinata dalla partizione corrente \mathbf{P} e dall'etichetta corrente α . Come nel caso precedente, il grafo verrà generato dinamicamente ad ogni iterazione.

Un esempio della struttura del grafo creato è riportata in figura 4.26 (in particolare un semplice esempio d'immagine 1D): in questo caso l'insieme dei vertici \mathcal{V}_α sarà costituito dagli l -vertici sorgente e destinazione α ed $\bar{\alpha}$, e dai nodi (p -vertici $\in \mathcal{P}$) corrispondenti a voxel ricavati. Inoltre, per ogni coppia di vertici vicini $(p, q) \in \mathcal{N}$ separati all'interno della partizione corrente (cioè $\varepsilon_p \neq \varepsilon_q$) introduciamo un vertice *ausiliario* $a_{\{p,q\}}$; questi nodi ausiliari sono quindi generati nelle regioni di bordo tra le partizioni \mathcal{P}_l , $l \in \mathcal{L}$, l'insieme totale dei vertici è quindi definito da:

$$\mathcal{V}_\alpha = \{\alpha, \bar{\alpha}, \mathcal{P}, \bigcup_{\{p,q\} \in \mathcal{N}, \varepsilon_p \neq \varepsilon_q} a_{\{p,q\}}\} \quad (4.26)$$

Ogni nodo in \mathcal{P} è connesso ai terminali α e $\bar{\alpha}$ tramite i t -link t_p^α e $t_p^{\bar{\alpha}}$ rispettivamente; Ogni coppia di nodi vicini $\{p, q\} \in \mathcal{N}$ che non sono separati dalla partizione \mathbf{P} (cioè $\varepsilon_p = \varepsilon_q$) è connessa da un n -link $e_{\{p,q\}}$. Diversamente, per ogni coppia di vertici $\{p, q\} \in \mathcal{N}$ tali per cui $\varepsilon_p = \varepsilon_q$, creiamo una tripletta di archi $\mathcal{E}_{(p,q)} = \{e_{(p,a)}, e_{(a,q)}, t_a^{\bar{\alpha}}\}$ dove $a = a_{(p,q)}$ è il nodo ausiliario corrispondente. Gli archi $e_{(p,a)}$ ed $e_{(a,q)}$ connettono i

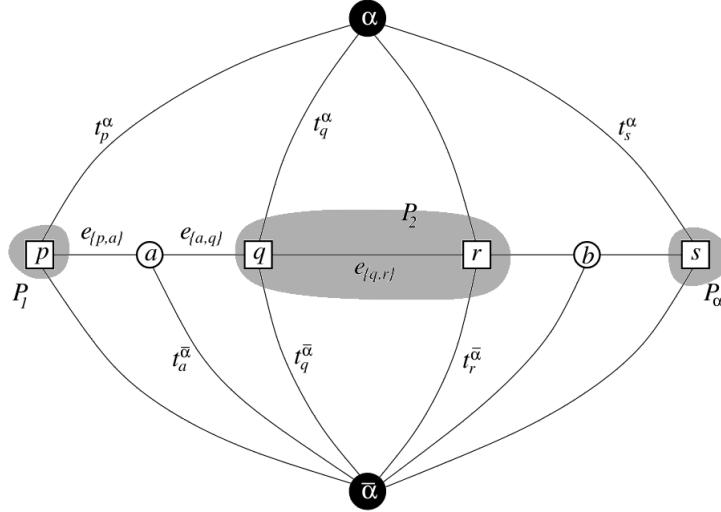


Figura 4.26: Esempio di grafo associato all’algoritmo di α -Expansion per un’immagine 1D. L’insieme dei pixel è costituito da $\mathcal{P} = \{p, q, r, s\}$, mentre la partizione corrente \mathbf{P} è data da $\mathbf{P} = \{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_\alpha\}$ dove $\mathcal{P}_1 = \{p\}$, $\mathcal{P}_2 = \{q, r\}$, $\mathcal{P}_\alpha = \{s\}$. Due nodi ausiliari $a = a_{(p,q)}$ e $b = a_{(r,s)}$ sono stati introdotti tra i pixel vicini separati dalla partizione corrente; questi sono nodi di bordo per gli insiemi \mathcal{P}_l per $l \in \mathcal{P}$

p -vertici p e q ad $a_{(p,q)}$ ed il t -link $t_a^{\bar{\alpha}}$ connette il nodo ausiliario $a_{(p,q)}$ al nodo terminale $\bar{\alpha}$. Detto questo possiamo riscrivere l’insieme di tutti gli archi secondo la relazione:

$$\mathcal{E}_\alpha = \left\{ \bigcup_{p \in \mathcal{P}} (t_p^\alpha, t_p^{\bar{\alpha}}), \bigcup_{(p,q) \in \mathcal{N}} \mathcal{E}_{(p,q)}, \bigcup_{(p,q) \in \mathcal{N}} e_{(p,q)} \right\} \quad (4.27)$$

Mentre i pesi (o equivalentemente, le capacità) assegnati ai vari archi sono riportati in tabella 4.2.

Come nel paragrafo precedente, ogni taglio \mathcal{T} recide un t -link per ogni p -vertice appartenente a \mathcal{P} ; quest’operazione definisce l’etichettatura $\varepsilon^{\mathcal{T}}$ conseguente al taglio \mathcal{T} su \mathcal{G}_α , formalmente:

$$\varepsilon_p^{\mathcal{T}} = \begin{cases} \alpha & \text{se } t_p^\alpha \in \mathcal{T}, \forall p \in \mathcal{P} \\ \varepsilon_p & \text{se } t_p^{\bar{\alpha}} \in \mathcal{T}, \forall p \in \mathcal{P} \end{cases} \quad (4.28)$$

In pratica, ogni taglio \mathcal{T} su \mathcal{G}_α genera il grafo indotto $\mathcal{G}(\mathcal{T}) = \langle \mathcal{V}, \mathcal{E} - \mathcal{T} \rangle$ dove ogni voxel p sarà assegnato all’etichetta α se il taglio \mathcal{T} separa p dal terminale α ; viceversa p

Edge	Weight	For
$t_p^{\bar{\alpha}}$	$+\infty$	$p \in \mathcal{P}_{\alpha\beta}$
$t_p^{\bar{\alpha}}$	$\psi_p(\varepsilon_p)$	$p \notin \mathcal{P}_{\alpha\beta}$
t_p^{α}	$\psi_p(\alpha)$	$p \in \mathcal{P}_{\alpha\beta}$
$e_{\{p,a\}}$	$\phi_{p,q}(\varepsilon_p, \alpha)$	$\{p, q\} \in \mathcal{N}, \varepsilon_p \neq \varepsilon_q$
$e_{\{a,q\}}$	$\phi_{p,q}(\alpha, \varepsilon_p)$	
$t_a^{\bar{\alpha}}$	$\phi_{p,q}(\varepsilon_p, \varepsilon_q)$	$\{p, q\} \in \mathcal{N}, \varepsilon_p = \varepsilon_q$
$e_{\{p,q\}}$	$\phi_{p,q}(\varepsilon_p, \alpha)$	

Tabella 4.2: Le capacità associate agli archi \mathcal{E}_α per l'algoritmo α -Expansion

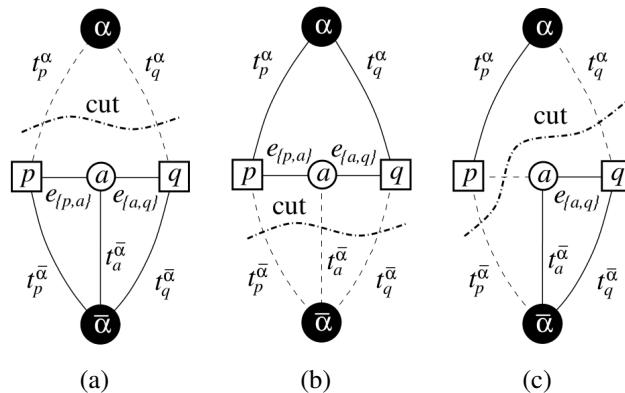


Figura 4.27: Esempi di taglio per i nodi $p, q \in \mathcal{N}$ tali per cui $\varepsilon_p \neq \varepsilon_q$. Le linee tratteggiate mostrano gli archi recisi dal taglio \mathcal{T} mentre quelle continue rappresentano gli archi del grafo indotto $\mathcal{G}(\mathcal{T})$

rimarrà assegnato alla sua “vecchia” etichetta ε_p nel caso in cui \mathcal{T} separi p da $\bar{\alpha}$; alcuni casi di taglio possibile sono riportati in figura 4.27. Osserviamo inoltre che per $p \notin \mathcal{P}_\alpha$ il terminale $\bar{\alpha}$ rappresenta le etichette assegnati ai voxel nell’etichettatura iniziale ε .

In questo capitolo si è voluto fornire al lettore una descrizione degli algoritmi di taglio utilizzati sui MRF per minimizzare la funzione energia E e ricavare la segmentazione finale della scena. Tali algoritmi sono gli stessi implementati ed utilizzati all’interno della libreria GCO [70]. Per un approfondimento e la trattazione formale con dimostrazione di tutti i teoremi e le proprietà alla base di questi processi, si rimanda il lettore a [6, 73, 86].

Nel prossimo capitolo verranno presentati i risultati, ottenuti dall’applicazione del sistema sviluppato, ad uno scenario di test rappresentante un comune ambiente interno domestico: per la stessa scena verranno messi a confronto alcuni indici di performance per tutti i vari metodi di classificazione utilizzati.

Capitolo 5

Risultati sperimentali

Il sistema realizzato in questa tesi permette di acquisire immagini e point cloud relative alla scena osservata tramite l'utilizzo del sensore Microsoft Kinect. Il sistema è in grado di segmentare in maniera automatica la nuvola di punti così ottenuta restituendo all'utente un'immagine tridimensionale opportunamente etichettata. In questo capitolo vengono presentati i risultati ottenuti dagli esperimenti effettuati e la valutazione delle prestazioni degli algoritmi realizzati.

5.1 Setup sperimentale

I test sono stati effettuati acquisendo alcuni frame da scene di interno, nello specifico, relative a scenari domestici. Il sensore utilizzato per l'acquisizione è stato mosso liberamente a mano per consentire di ottenere facilmente tutti i campioni prelevati dalla scena. Il codice prodotto è stato eseguito su un *PC laptop* con le specifiche hardware elencate in tabella 5.1 ed è stato compilato su un sistema operativo *Xubuntu 12.04 Precise Pangolin* a 64-bit. Infine, il progetto è stato scritto interamente in linguaggio C/C++ e realizzato sotto forma di *console application* che integra ed utilizza la librerie elencate nella sezione 3.3.

Hardware specifications	
Processor	2x Intel Core2 Duo T9300 @ 2.50GHz
RAM Memory	4Gb DDR2 Dual Channel
Graphics	NVIDIA GeForce 8600M GS 512Mb
Hyperthreading	NO - 1 process/core

Tabella 5.1: Specifiche tecniche dell'hardware su cui è stato compilato ed eseguito il progetto realizzato

5.2 Sequenza acquisita

Il *dataset* acquisito si compone di 13 immagini RGB bidimensionali ed altrettante point cloud 3D (figura 5.1): la sequenza riporta la stessa scena d'interno inquadrata da punti diversi e/o utilizzando orientazioni diverse per il sensore.



Figura 5.1: Esempio di alcuni fotogrammi acquisiti relativi alla scena di test

Le immagini RGB sono state acquisite impostando la camera del Kinect ad una risoluzione VGA di 640×480 pixel, le nuvole di punti associate sono cloud *organizzate*

costituite da altrettanti punti; questo ha permesso la gestione di una corrispondenza immediata *pixel-punto* (tra cloud ed immagine) per tutto il tempo d'esecuzione del processo. Non avendo utilizzato alcun tipo di tecnica di *refinement* per la nuvola, questa manterrà la propria organizzazione per righe e colonne durante tutto il ciclo di vita dell'algoritmo.

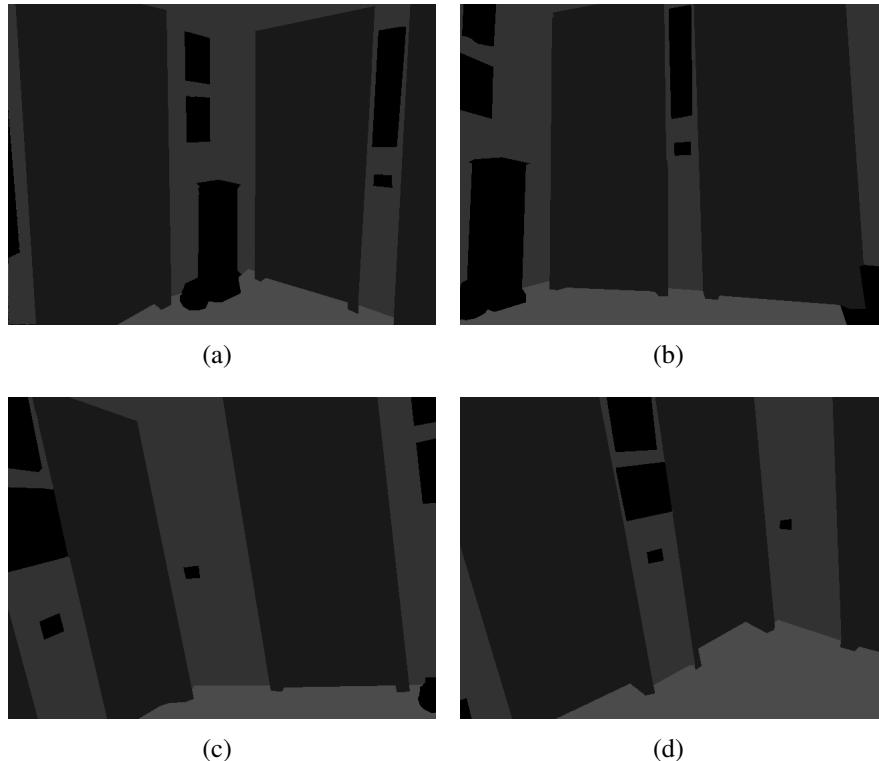


Figura 5.2: Alcuni esempi d'immagini di ground truth relative ai dati acquisiti, diversi colori rappresentano le varie categorie previste per il processo di classificazione: *Floor*, *Wall*, *Door*; le patch nere rappresentano zone associate alla classe *Other*

Nonostante il numero di punti delle cloud rilevate sia costante va sottolineato che la disposizione dei punti varia a seconda della scena e, quindi, varia anche la struttura dati octree che la indicizza ed il corrispondente numero di voxel in cui essa è suddivisa. Inoltre, può accadere che alcuni pixel dell'immagine di profondità corrispondano a misure non valide. La tabella 5.2 mostra il numero medio di voxel e la corrispondente deviazione standard per il training set ed il test set. Inoltre, le point cloud acquisite contengono al loro interno in media circa 45.16 ± 6.218 voxel.

	Training set	Test set
Avg no. of voxels	5703	6106.3
Std deviation	774.59	910.24

Tabella 5.2: Numero medio e deviazione standard del numero di voxel presenti nel training set ed il test set

5.2.1 Training set e test set

Prerogativa principale di ogni sistema di classificazione supervisionata è la costituzione preliminare di un *training set* mediante il quale addestrare i classificatori che verranno utilizzati (*Reti Neurali* e *Support Vector Machines*). Per evitare l’overfitting nell’addestramento dei classificatori, si è scelto di utilizzare solamente 4 delle 13 immagini a disposizione dopo la fase d’acquisizione; le rimanenti sono state impiegate come *test set*. Per l’addestramento degli algoritmi supervisionati e per valutare le performance, è stato necessario generare anche il *groud-truth* relativo a tutti i fotogrammi acquisiti (figura 5.2): la classi previste per la classificazione sono 3 (*Floor*, *Wall* e *Door*) più una classe che contempla tutto ciò che non ricade nelle prime tre (*Other*).

I file di training sono stati costituiti campionando casualmente le immagini utilizzate come train set: in ognuna di queste sono stati prelevati N voxel (nello specifico $N = 20$) contenenti almeno τ punti ($\tau = 150$); definito C il numero di classi, per la fase d’addestramento sono quindi stati prodotti C files contenenti al loro interno $C \times N$ campioni d’esempio. Il processo di classificazione prevede un approccio del tipo *One vs. All* in quanto ogni classificatore darà il proprio voto di *confidence* sull’appartenenza dei voxel campionati ad una certa categoria. In fase di addestramento ogni classificatore riceverà come esempio “positivo” i vettori di features estratti da voxel della sua stessa categoria, mentre gli verranno indicati come “negativi” tutti gli esempi relativi a parti di scena che non appartengono alla classe da esso rappresentata. Per ogni voxel, una volta ottenuti i responsi da tutti i classificatori, questo sarà associato alla classe per la quale ha avuto il valore di confidence più alto. Per quanto riguarda la classe *Other* occorre effettuare una precisazione: per questa categoria non è stato addestrato un classificatore ad-hoc ma si è scelto di associare il voxel corrente a questa classe nel momento in cui nessuno dei tre classificatori precedenti dovesse prevalere, in termini di confidence restituita, in

maniera significativa sugli altri.

Nelle tabelle 5.3 e 5.4 riportiamo alcuni valori relativi al numero di voxel ricavati per ogni frame utilizzato nella composizione di train e test set.

Frame name	Voxel number	Frame type
<i>DataSetFrame0</i>	6269	Train
<i>DataSetFrame1</i>	5404	Test
<i>DataSetFrame2</i>	5221	Train
<i>DataSetFrame3</i>	4917	Train
<i>DataSetFrame4</i>	4775	Test
<i>DataSetFrame5</i>	5878	Test
<i>DataSetFrame6</i>	6837	Test
<i>DataSetFrame7</i>	6915	Test
<i>DataSetFrame8</i>	5686	Test
<i>DataSetFrame9</i>	5832	Test
<i>DataSetFrame10</i>	5849	Test
<i>DataSetFrame11</i>	7781	Test
<i>DataSetFrame12</i>	6405	Train

Tabella 5.3: Numero di voxel ricavati da ogni frame di train e test set durante la fase di preprocessing

	Average voxel no. μ	Standard deviation σ
Train set	5703	744,60
Test set	6106	910,24

Tabella 5.4: Numero medio e deviazione standard dei voxel del train e test set durante la fase di preprocessing

In seguito presenteremo i risultati di segmentazione ottenuti tramite l'utilizzo di tecniche di classificazione supervisionata e non supervisionata. Per ognuna di queste casistiche verranno presentati i risultati ottenuti con e senza l'impiego dei Markov Random Field: questo al fine di dare ulteriore evidenza ai benefici che l'applicazione di questo modello può portare nei task di segmentazione automatica.

5.2.2 Risultati di classificazione supervisionata

In questo ambito sono stati utilizzati due diversi sistemi di classificazione (presentati alla Sezione 4.2.4): le Reti Neurali e le Support Vector Machines; in seguito analizzeremo i risultati di classificazione ottenuti con entrambe le tecniche calcolando alcuni indici di performance che indichino quale tipo di classificatore sia in grado di dare i risultati migliori per questo particolare tipo di applicazione. Alla base dei risultati che verranno presentati ci sono le *matrici di confusione* ricavate in seguito all'attività di classificazione.

La matrice di confusione sintetizza l'esito di un processo di classificazione. Ogni riga della matrice rappresenta la corretta classe di appartenenza degli item, mentre ciascuna colonna rappresenta il possibile esito della classificazione. Pertanto nella cella alla riga i -esima e alla colonna j -esima viene riportato il numero di volte in cui un item di classe i è stato classificato come appartenente alla classe j . Il contatore di ciascuna cella è talvolta normalizzato rispetto al numero totale di elementi della riga.

Questa matrice misura il grado di “confusione” nella classificazione di diverse classi e da essa si possono ricavare alcuni indici di performance quali l'indice di *precision* e *recall*. Questi due parametri indicano rispettivamente l'accuratezza dell'algoritmo, ovvero la capacità di non commettere errori di riconoscimento, e la sensibilità, ovvero la capacità di riconoscere tutti gli oggetti. Sono entrambi parametri con valori compresi tra 0 e 1 e forniscono un'indicazione sulle prestazioni globali del classificatore utilizzato. Il caso ideale è rappresentato da entrambi gli indicatori tendenti al valore massimo che corrisponde ad aver riconosciuto tutti gli oggetti ($\text{recall}=1$) correttamente ($\text{precision}=1$). Formalmente, data una matrice di confusione come la seguente:

		<i>classe stimata</i>		
		A	B	C
<i>classe conosciuta</i>	A	A_{tp}	AB_e	AC_e
	B	BA_e	B_{tp}	\dots
	C	CA_e	\dots	C_{tp}

La *precision* relativa della classe A si calcola utilizzando i valori della corrispondente colonna

$$P_A = \frac{TP_A}{TP_A + FP_A} = \frac{A_{tp}}{A_{tp} + BA_e + CA_e + \dots} \quad (5.1)$$

Invece il valore di *recall* per la classe si calcola operando lungo le righe

$$R_A = \frac{TP_A}{TP_A + FN_A} = \frac{A_{tp}}{A_{tp} + AB_e + AC_e + \dots} \quad (5.2)$$

Oltre a questi due indicatori è possibile ricavare gli indici di *precision media* e *recall media* attraverso le seguenti relazioni:

$$\begin{aligned} P_{avg} &= \frac{1}{C} \sum_{i=1}^C P_i \\ R_{avg} &= \frac{1}{C} \sum_{i=1}^C R_i \end{aligned} \quad (5.3)$$

Un altro parametro utile a stimare la bontà della classificazione è il valore di *accuracyzza*: definita la matrice di confusione $M \in \mathbb{N}^{C \times C}$, l'indice è calcolabile come la somma degli elementi sulla diagonale principale rapportata al totale di tutti i valori della matrice. Formalmente:

$$ACC = \frac{\sum_{i=1}^C m_{ii}}{\sum_{i=1}^C \sum_{j=1}^C m_{ij}} \quad (5.4)$$

Infine verranno riportati anche i tempi impiegati per l’addestramento della rete e la procedura di classificazione.

5.2.2.1 Reti Neurali

Come già anticipato alla sezione 4.2.4 una volta ricavati i vettori di features per ogni voxel, questi vengono elaborati da un banco di C classificatori. In questo caso i classificatori sono reti neurali multistrato con 454 nodi d’ingresso, uno strato nascosto composto da un migliaio di percettroni, i quali vanno tutti a collimare su un unico nodo d’uscita. Quest’ultimo restituirà il grado di confidence del voxel per la categoria considerata. In fase di training i parametri sui quali è possibile intervenire sono:

- **learning rate** α : è una valore compreso in nell’intervallo reale $[0, 1]$ che determina in che proporzio-ne debbano essere aggiornati i pesi dei singoli neuroni durante l’addestramento;
- **learning momentum** μ : anche questo vari in un intervallo tra $[0, 1]$ e serve per aggiungere una frazione del valore precedente del peso durante lo step di aggiornamento del valore corrente; questo valore è usato per prevenire la convergenza della rete ad un punto di minimo locale o di sella;
- **numero di epoch** n : è la prima condizione di terminazione, rappresenta il numero massimo di iterazioni stabilito per la procedura di training;
- **errore minimo** ϵ : è la seconda condizione di terminazione, indica la minima di errore desiderata per la conclusione dell’addestramento.

In tabella 5.5 sono riportati tutti i valori dei parametri utilizzati per l’addestramento delle reti.

In seguito riportiamo le matrici di confusione relative alla segmentazione ottenuta utilizzando esclusivamente il passo di classificazione supervisionata senza e con l’ausilio di MRF (tabella 5.6 e 5.7). Gli indici di precision e recall ricavati da questa sono invece riportati in tabella 5.8 e nei grafici seguenti (figura 5.3 e 5.4).

ANN training parameters	
No. of input layer perceptrons	454
No. of hidden layer perceptrons	1000
No. of output layer perceptrons	1
Learning rate α	0.3
Learning momentum μ	0.5
Max no. of epochs n	500
Minimum error threshold ϵ	10^{-5}

Tabella 5.5: Elenco di tutti i parametri impostati per la fase di training dei classificatori basati su reti neurali

	Door	Wall	Floor	Other
Door	17855	1018	2419	7833
Wall	1401	8676	226	2703
Floor	291	233	6824	1399
Other	684	1419	411	1565

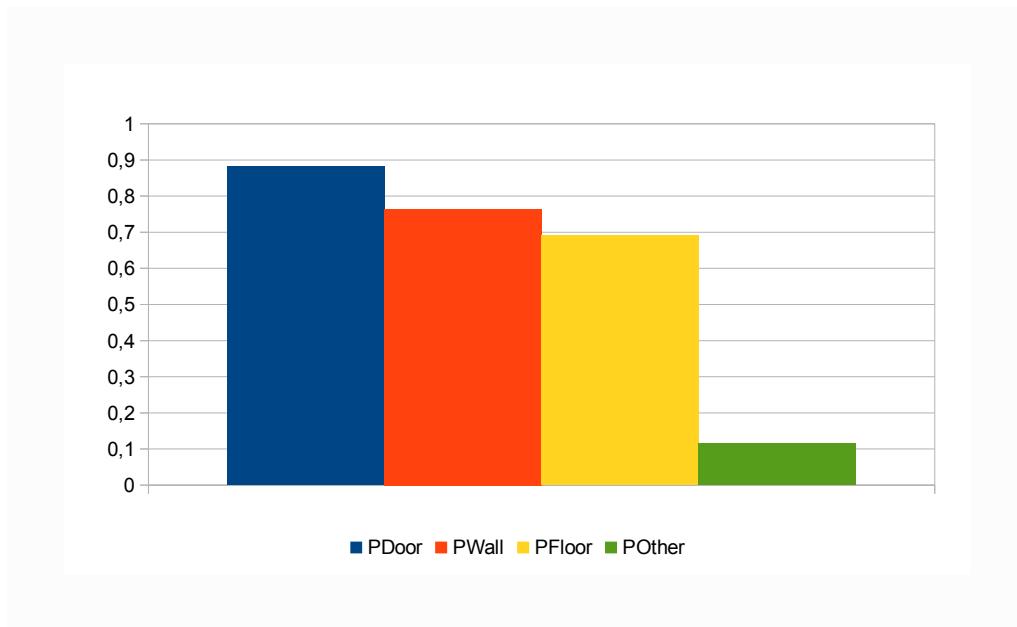
Tabella 5.6: Matrice di confusione relativa al processo di classificazione supervisionata attraverso reti neurali senza l'ausilio di MRF

	Door	Wall	Floor	Other
Door	23126	700	3480	1819
Wall	390	11769	239	608
Floor	160	15	8548	24
Other	705	2088	737	549

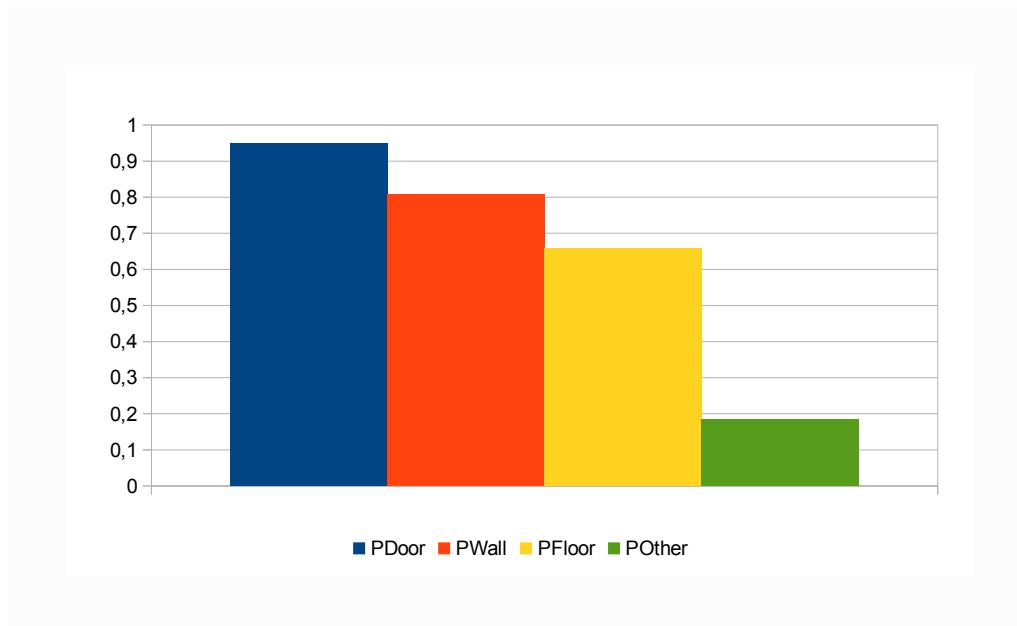
Tabella 5.7: Matrice di confusione relativa al processo di classificazione supervisionata attraverso l'impiego di reti neurali e MRF

	<i>ANN only</i>	<i>ANN with MRF</i>
P_{Door}	0.883	0.949
P_{Wall}	0.765	0.808
P_{Floor}	0.691	0.657
P_{Other}	0.116	0.183
P_{avg}	0.613	0.649
R_{Door}	0.613	0.794
R_{Wall}	0.667	0.905
R_{Floor}	0.780	0.977
R_{Other}	0,384	0.135
R_{avg}	0.611	0.703
ACC	0.635	0.801

Tabella 5.8: Indici di *Precision*, *Recall* ed *Accuracy* relativi alla classificazione eseguita attraverso l'utilizzo di reti neurali con e senza l'ausilio di MRF

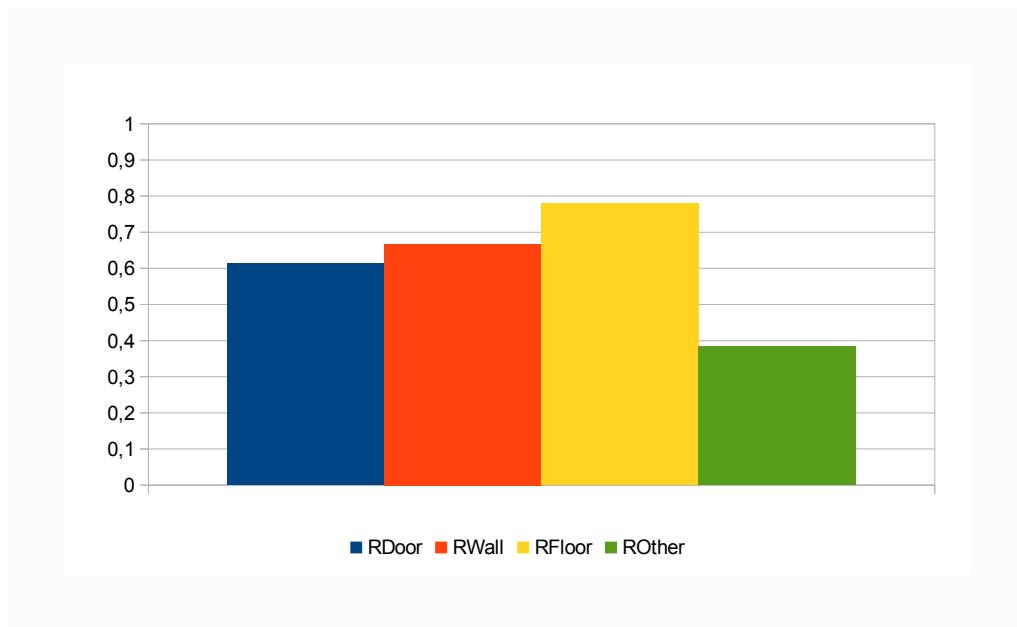


(a)

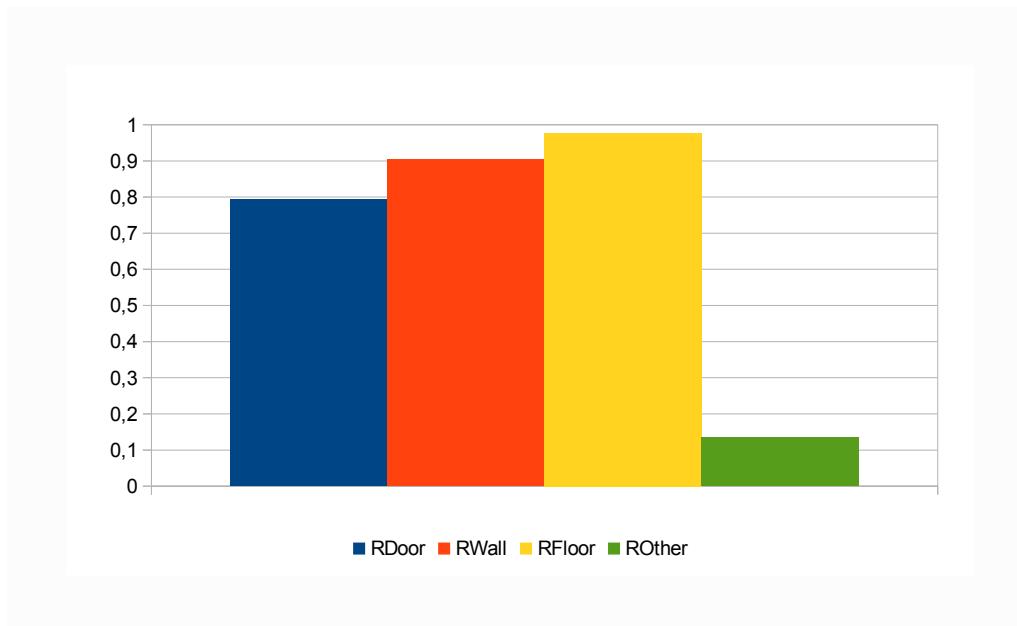


(b)

Figura 5.3: Precision: (a) indici di performance relativi all'utilizzo esclusivo di reti neurali e (b) con l'aggiunta di MRF



(a)



(b)

Figura 5.4: Recall: (a) indici di performance relativi all'utilizzo esclusivo di reti neurali e (b) con l'aggiunta di MRF

Come possiamo notare dai dati ed i grafici appena riportati, l'utilizzo dei Markov Random Field ha un effetto positivo sul processo globale di segmentazione in quanto aumenta in tutti i casi (tutti ad eccezione di uno) il grado di precision e recall. Le relazioni di vicinato tra i voxel consentono un significativo miglioramento del processo di etichettatura che, se considerasse esclusivamente i risultati ottenuti mediante i classificatori addestrati, si rivelerebbe maggiormente impreciso e rumoroso. Notiamo anche come, impiegando il, i valori medi di precision e recall subiscano anche loro un miglioramento e l'accuracy globale da circa il 60% raggiunge valori superiori all'80%. In seguito vengono riportate alcune immagini raffiguranti le istanze di segmentazione ottenute con e senza l'utilizzo di MRF (figura 5.5).

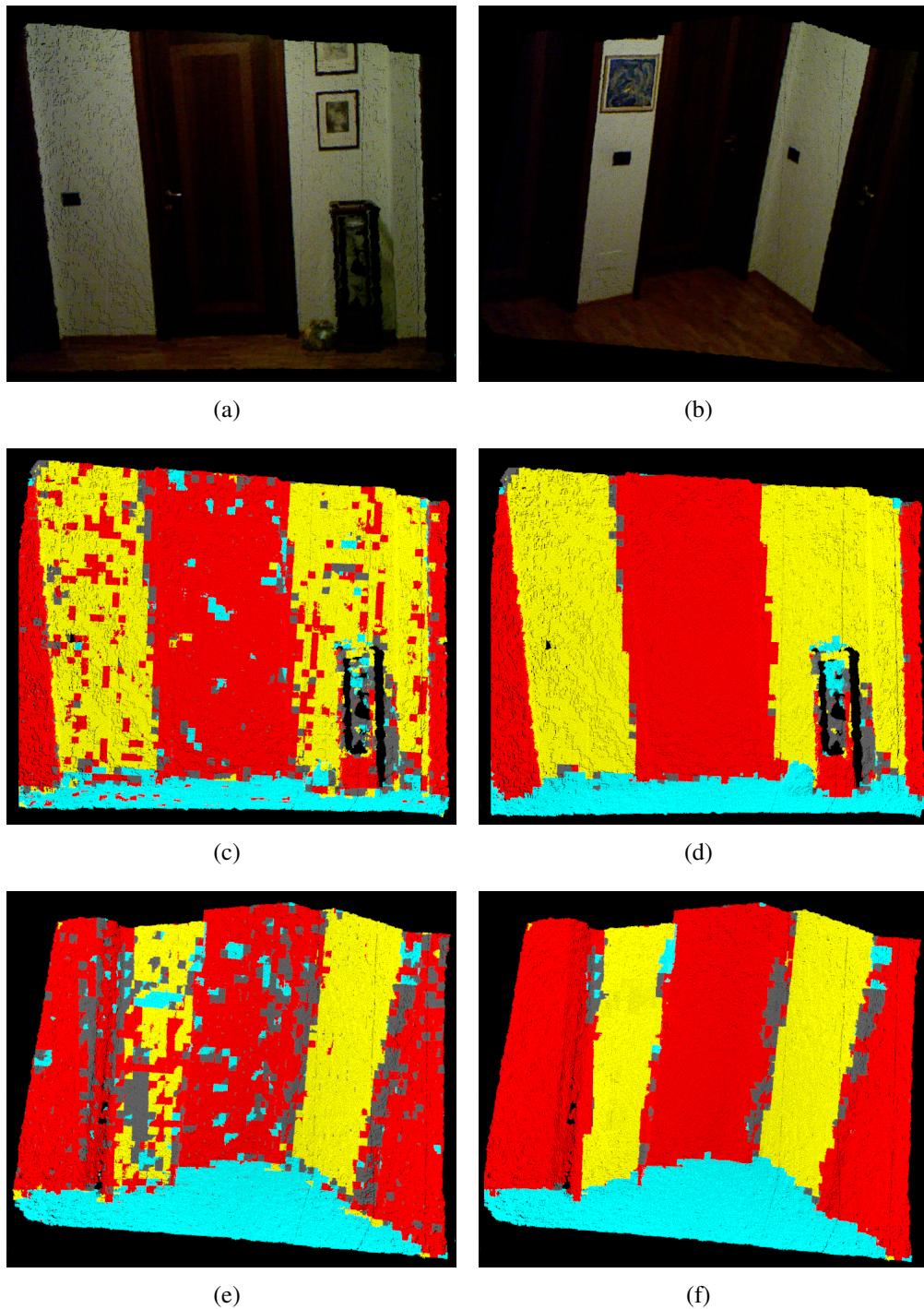


Figura 5.5: Risultati di segmentazione: (a, b) point cloud originale, (c, e) scena etichettata dopo l'utilizzo esclusivo di reti neurali e (d,f) i risultati di etichettatura ottenuti con l'impiego aggiuntivo di MRF

5.2.2.2 Support Vector Machines

Le Support Vector Machines cercano di stimare l'iperpiano separatore ottimo dato un insieme multivariato di elementi. In questa sezione verranno ripetute le stesse considerazioni appena fatte per le reti neurali: proponiamo i risultati ottenuti attraverso l'utilizzo delle SVM come mezzo di classificazione dei dati ottenuti dalla fase di *feature extraction* valutando gli stessi indici di performance di precision, recall (per classe e medi) ed accuracy.

Scelto un diverso classificatore, il passo successivo è la definizione dei relativi parametri; il problema di ottimizzazione alla base delle SVM (equazione 4.10) è definito principalmente da:

- il fattore di costo C che modula l'incidenza dell'errore ξ ammesso;
- le grandezze che caratterizzano la *funzione kernel* $\kappa(\mathbf{x}_i, \mathbf{x}_j)$ scelta.

Nello specifico il kernel utilizzato è di tipo *Radial Basis Function* (abbreviato RBF) è una funzione del tipo:

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2) \quad (5.5)$$

dove la variabile γ gioca un ruolo di fondamentale importanza nel definire l'efficacia del kernel scelto.

I valori scelti nell'addestramento delle SVM sono i seguenti:

SVM training parameters	
Kernel type	RBF
Kernel parameter γ	1
Soft-error margin cost C	1

Tabella 5.9: Elenco dei parametri impostati per la fase di training dei classificatori basati su SVM

Effettuato il training del classificatore si riportano i risultati ottenuti nelle seguenti matrici di confusione:

	Door	Wall	Floor	Other
Door	12836	233	624	15432
Wall	0	10398	0	2608
Floor	134	3	4845	3765
Other	544	1061	52	2422

Tabella 5.10: Matrice di confusione relativa al processo di classificazione supervisionata attraverso SVM senza l’ausilio di MRF

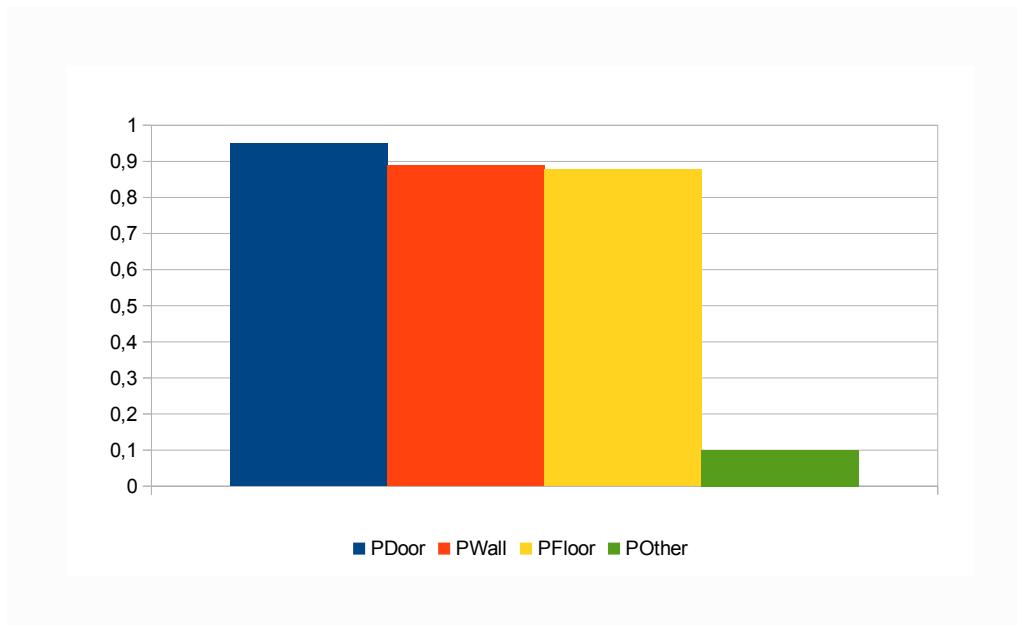
	Door	Wall	Floor	Other
Door	24736	1553	524	2312
Wall	54	12890	35	27
Floor	307	47	7908	485
Other	884	2693	107	395

Tabella 5.11: Matrice di confusione relativa al processo di classificazione supervisionata attraverso l’impiego di SVM e MRF

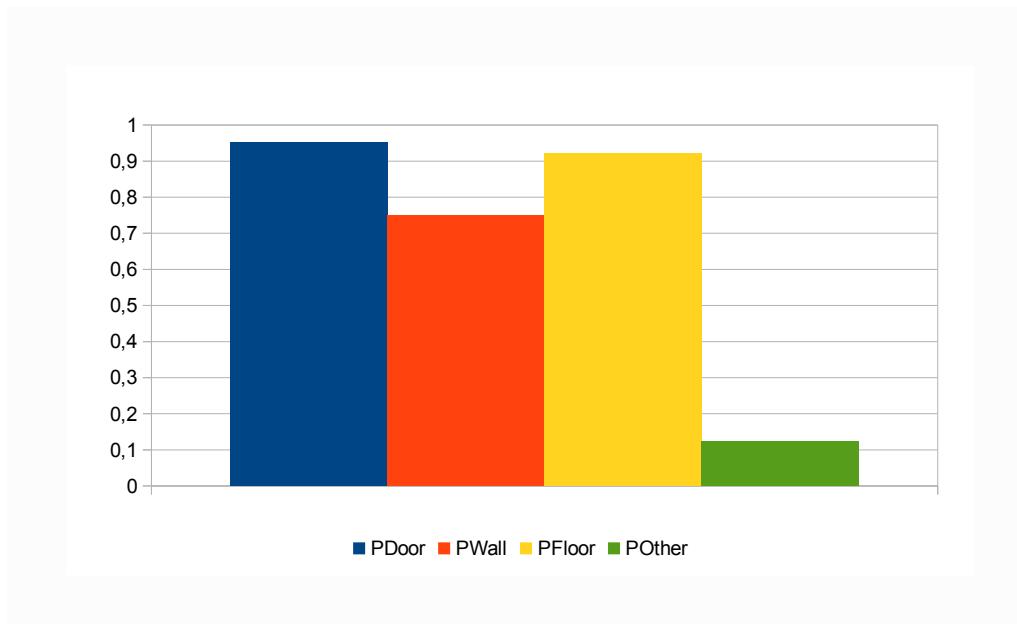
Analogamente alla sezione precedente, in tabella 5.12 e nelle figure 5.6, 5.7, riportiamo i dati di performance relativi all’esito del processo di classificazione eseguito in assenza di MRF e successivamente ad una segmentazione ottenuta dall’utilizzo congiunto di SVM e Markov Random Field.

	<i>SVM only</i>	<i>SVM with MRF</i>
P_{Door}	0.949	0.952
P_{Wall}	0.889	0.750
P_{Floor}	0.878	0.922
P_{Other}	0.099	0.123
P_{avg}	0.704	0.687
R_{Door}	0.441	0.849
R_{Wall}	0.799	0.991
R_{Floor}	0.554	0.904
R_{Other}	0.594	0.097
R_{avg}	0.597	0.711
ACC	0.555	0.836

Tabella 5.12: Indici di *Precision*, *Recall* ed *Accuracy* relativi alla classificazione eseguita attraverso l'utilizzo di SVM con e senza l'ausilio di MRF

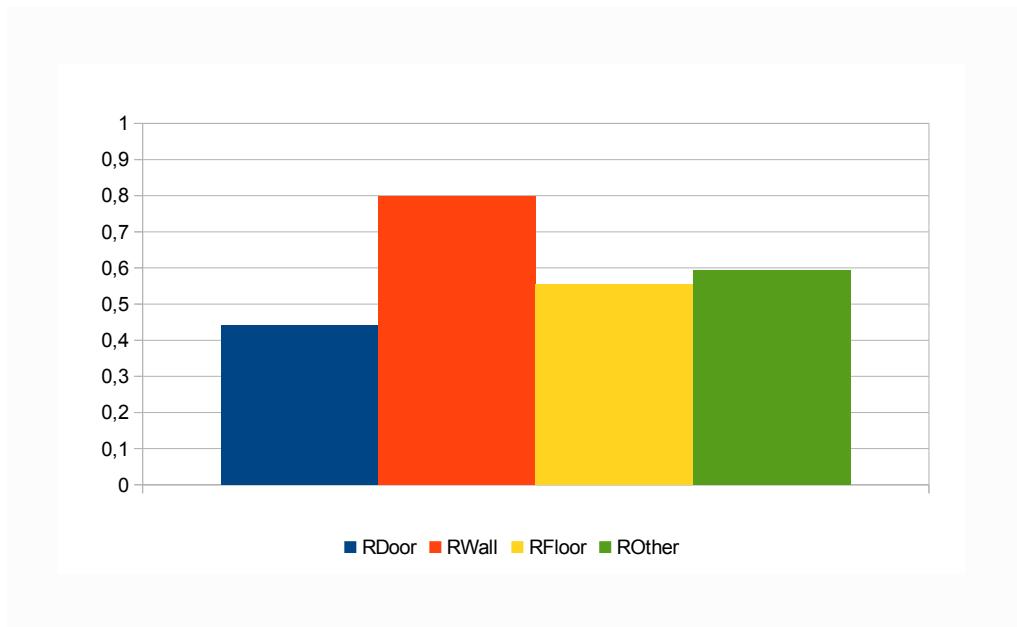


(a)

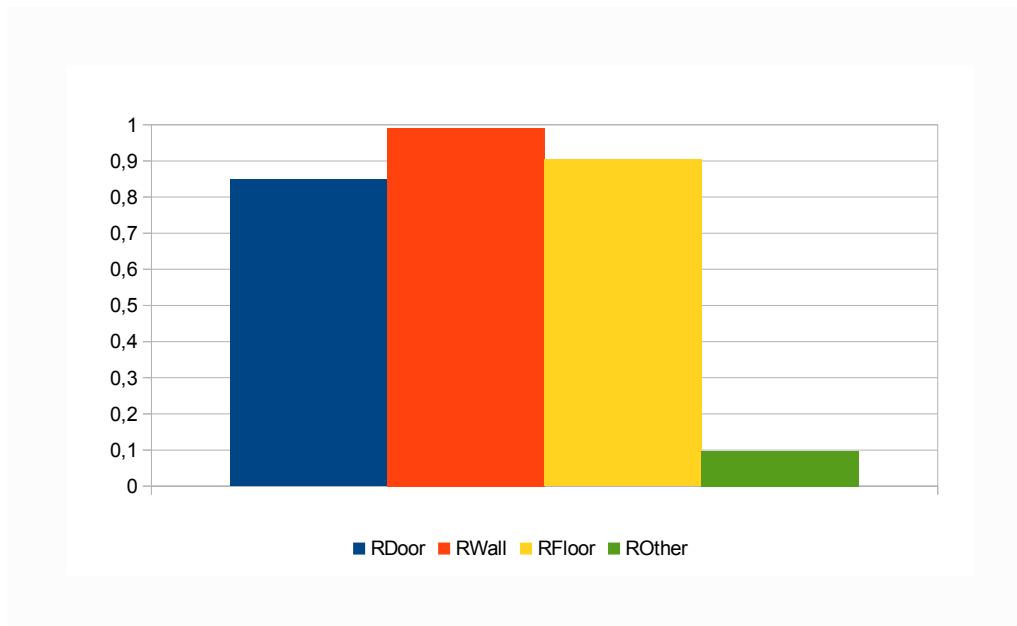


(b)

Figura 5.6: Precision: (a) indici di performance relativi all'utilizzo esclusivo di SVM e (b) con l'aggiunta di MRF



(a)



(b)

Figura 5.7: Recall: (a) indici di performance relativi all'utilizzo esclusivo di SVM e (b) con l'aggiunta di MRF

Come si può riscontrare dai dati appena riportati, anche in questo caso la segmentazione ottenuta esclusivamente attraverso l'utilizzo di classificatori, in questo caso SVM, risulta piuttosto imprecisa e discontinua (figure 5.9(c) e 5.9(e)). A seguito dell'applicazione di MRF si ottengono risultati decisamente migliori dove la maggior parte delle "imperfezioni" vengono corrette grazie alla minimizzazione della funzione energia associata; alcuni esempi del risultato finale di etichettatura è vengono riportati in figura 5.9(d) e 5.9(f).

Anche in questo caso l'applicazione dei MRF porta ad un aumento degli indici di precision e recall per la maggior parte delle classi raggiungendo addirittura valori superiori al 90%; il valor medio di specificità subisce un aumento di circa una decina di punti percentuale mentre quello di precisione subisce una leggera flessione, rimanendo comunque su valori vicini al 70%, a causa della leggera perdita di performance osservabile per la classe *Wall*. Sottolineiamo comunque un valore di accuratezza media molto buono che, dopo l'introduzione dell'inferenza sul grafo, passa da circa il 50% ad un valore superiore all'80%; un effetto analogo era già stato riscontrato al paragrafo precedente dove sono state usate le reti neurali come classificatori iniziali.

In figura 5.8 presentiamo i grafici riportanti gli indici di performance ricavati finora e messi a confronto relativamente alle varie configurazioni proposte: utilizzo di reti neurali e SVM, con l'appoggio o meno di MRF.

	<i>SVM only</i>	<i>SVM with MRF</i>
P_{Door}	0.949	0.952
P_{Wall}	0.889	0.750
P_{Floor}	0.878	0.922
P_{Other}	0.099	0.123
P_{avg}	0.704	0.687
R_{Door}	0.441	0.849
R_{Wall}	0.799	0.991
R_{Floor}	0.554	0.904
R_{Other}	0.594	0.097
R_{avg}	0.597	0.711
ACC	0.555	0.836

Tabella 5.13: Indici di *Precision*, *Recall* ed *Accuracy* relativi alla classificazione eseguita attraverso l'utilizzo di SVM con e senza l'ausilio di MRF

Nel complesso possiamo concludere che il MRF migliora generalmente gli indici di precision e recall: in qualche caso sacrifica leggermente la recall della classe *Other*, questo è dovuto al fatto che il potenziale associato a tale classe è quello più debole. Infine, la scelta del classificatore (rete neurale o SVM) non incide particolarmente sui risultati di classificazione. Sebbene entrambi i tipi di classificatori presentino performance comparabili, notiamo come le SVM restituiscano indici leggermente migliori.

5.2.2.3 Tempi di addestramento e classificazione supervisionata

Le prestazioni dei due metodi di classificazione utilizzati sono essenzialmente comparabili, sebbene la SVM abbiano un leggero vantaggio. Le SVM hanno però un tempo di elaborazione più basso rispetto le reti neurali nell’etichettatura dell’intera nuvola. Nonostante i due tipi di classificatori presentino tempi d’esecuzione analoghi, questi risultano ancora troppo lunghi per applicazioni di tipo real time: come si evince dai dati riportati in tabella 5.14, il tempo necessario per l’ottenimento dei risultati finali si aggira intorno ai 30 secondi per le reti neurali e 24 secondi per le support vector machines.

L’introduzione dell’ulteriore step di “rifinitura” attraverso i MRF introduce ulteriore *overhead* che si ripercuote direttamente sui tempi richiesti dell’applicazione per presentare i risultati finali di segmentazione. Questo incremento risulta comunque piuttosto limitato e non inficia in maniera significativa i tempi d’esecuzione: di fatto l’aumento dei tempi d’attesa è dello 0.4/0.5%.

	Average value μ	Standard deviation σ
ANN class. time (without MRF)	30.06	0.662
SVM class. time (without MRF)	24.68	1.086
ANN class. time (with MRF)	30.22	0.763
SVM class. time (with MRF)	24.78	1.555

Tabella 5.14: Tempi di classificazione mediante l’utilizzo di reti neurali e support vector machines, con e senza l’impiego di MRF

Sebbene il divario tra i tempi di classificazione non sia così importante, lo è invece quello relativo all’addestramento, come è facilmente osservabile dai dati riportati in tabella 5.15, il divario tra i tempi impiegati per il training delle reti neurali è di ben tre ordini di grandezza rispetto a quello richiesto per l’apprendimento delle SVM. Questo

risultato mette in evidenza la maggior efficienza dei classificatori basati su SVM durante la fase di apprendimento.

	Training time [sec]
Artificial Neural Networks (ANN)	96.33
Support Vector Machines (SVM)	0.24

Tabella 5.15: Tempi di addestramento a confronto tra reti neurali e support vector machines

L'approccio attraverso classificatori di tipo supervisionato ha dato risultati positivi ma, per via della sua natura, presuppone la definizione *a priori* dell'insieme di categorie previste per la segmentazione. In più notiamo che, per come è definita il processo di assegnazione dei voxel alle diverse classi, l'unica classe che presenta un valore molto basso di precisione e specificità è la classe di *background* ribattezzata come *Other*. In ultima istanza, un altro problema che può insorgere con approcci di questo tipo è quello della capacità del classificatore di generalizzare (conceitto già esposto precedentemente al punto 4.2.4.1).

Per sopperire a tutte queste limitazioni, nella prossima sezione verranno illustrati i risultati ottenuti attraverso la sostituzione del tipo di classificatore utilizzato: da un modelli ottenuti da un apprendimento di tipo supervisionato passiamo all'impiego di classificatori ricavati attraverso tecniche di apprendimento non supervisionato.

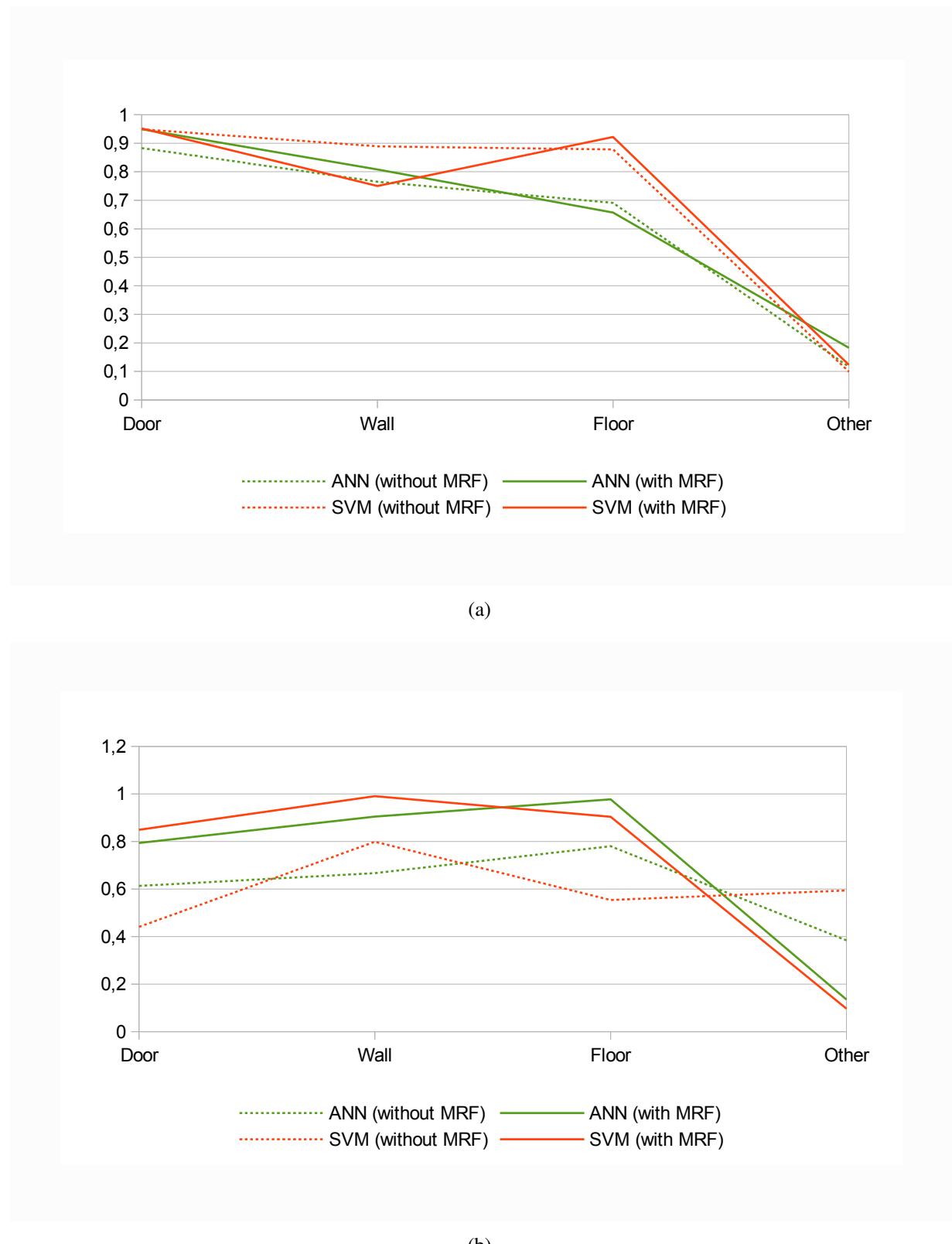


Figura 5.8: Confronto degli indici di performance ottenuti per le varie configurazioni proposte: valori di (a) precisione e (b) recall

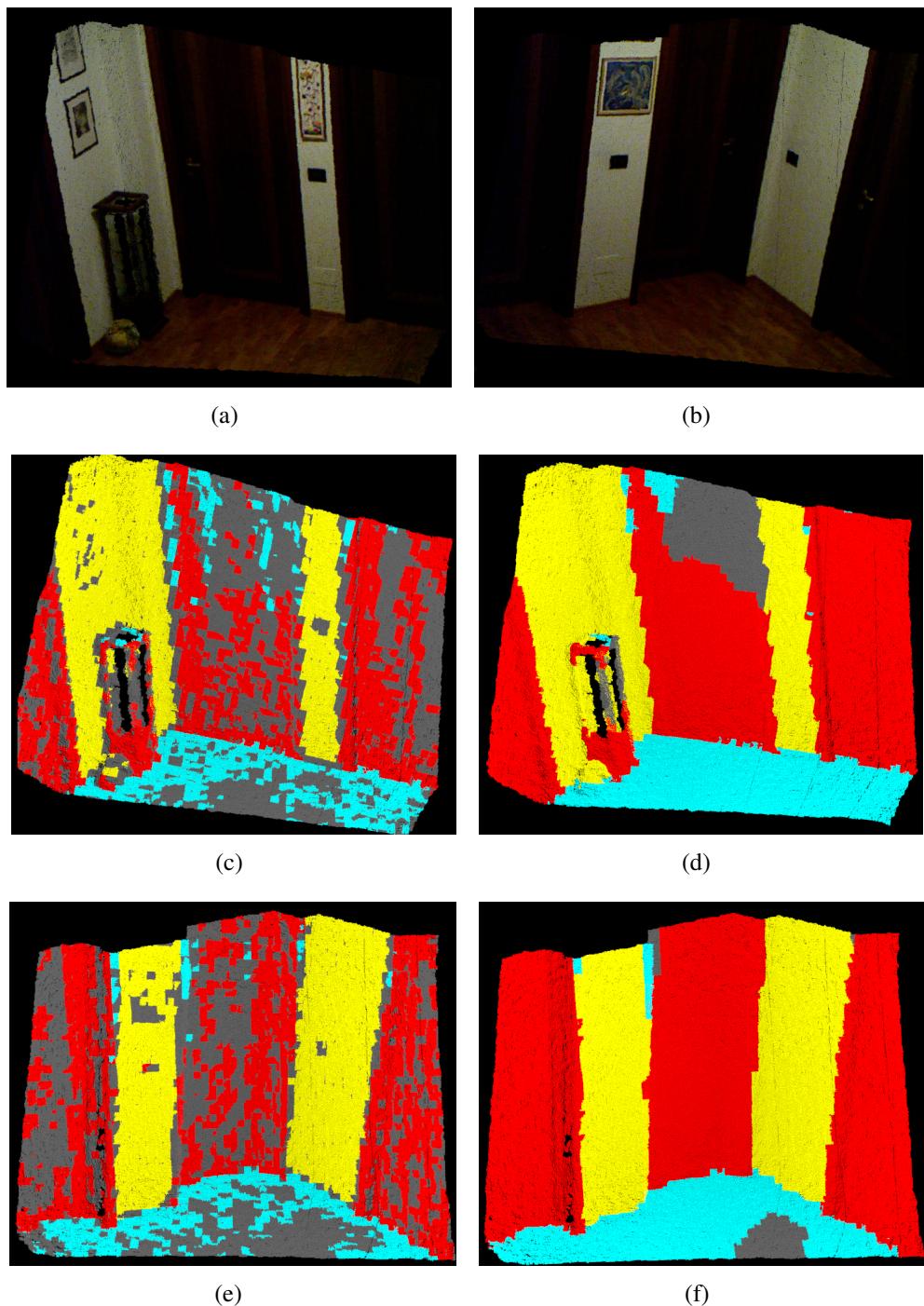


Figura 5.9: Risultati di segmentazione: (a, b) point cloud originale, (c, e) scena etichettata dopo l'utilizzo esclusivo di SVM e (d,f) i risultati di etichettatura ottenuti con l'impiego aggiuntivo di MRF

5.2.3 Risultati di classificazione non supervisionata

In questa sezione sono illustrati i risultati ottenuti applicando l'algoritmo di segmentazione non supervisionato. A differenza del precedente caso supervisionato, questo tipo d'approccio non definisce a priori le categorie rispetto alle quali ogni elemento della scena viene classificato.

Il vantaggio principale risiede nell'indipendenza dall'intervento di un operatore e nella possibilità di applicare il metodo ad una scena con elementi non noti a priori. Al contempo la segmentazione non supervisionata pone problemi di interpretazione semantica dei risultati (per esempio nel far corrispondere un'etichetta ad un oggetto o ad una categoria) e, di conseguenza, di valutazione quantitativa dei risultati. Nello specifico si pongono due problemi principali:

- valutare un metodo per determinare il numero di partizioni in cui suddividere la scena. Dato che l'algoritmo di aggregazione è il k-Means, questo corrisponde alla scelta del numero di cluster nello spazio delle feature;
- applicare un criterio di consistenza tra il ground truth fornito dall'utente ed il partizionamento automatico generato dall'algoritmo.

Nel seguito di questa sezione verranno esposti i risultati finali di segmentazione ricavati tramite l'adozione di un procedimento di classificazione non supervisionata. Come già introdotto precedentemente, tale metodo è stato appositamente sviluppato per sopprimere alle limitazioni dell'algoritmo k-Means “classico” il quale soffre principalmente sotto due aspetti:

- è fortemente influenzato dalla scelta iniziale dei centroidi;
- necessita della definizione a priori del numero di classi nelle quali suddividere i dati.

L'algoritmo *k-Means ottimizzato* è stato sviluppato proprio per sopprimere ai problemi sopraccitati: attraverso un'inizializzazione “intelligente” dei centroidi (che segue la stessa tecnica presentata in [80]) ed il calcolo del Davies-Bouldin Index, è possibile ottenere

un algoritmo di classificazione non supervisionata in grado di separare al meglio, e nel giusto numero di classi, i dati esistenti dopo la fase di feature extraction.

Per l’addestramento del modello vengono prelevati casualmente N campioni d’esempio (voxel) da M immagini scelte come training set, ottenendo così uno spazio dei dati costituito da $M \times N$ vettori di features. I frame scelti per la definizione del training set sono gli stessi utilizzati alla sezione precedente per l’addestramento nel caso supervisionato. È da sottolineare come, di fatto, il concetto di addestramento venga a meno nel caso non supervisionato. In questo caso è stato scelto un campionamento casuale delle immagini del precedente training set sia per ridurre la complessità di calcolo, che per dare continuità con quanto eseguito per il caso supervisionato. Si sarebbe potuto ripetere la generazione dello spazio delle features anche campionando completamente ogni singola immagine o decidere di ricavare i descrittori da ogni voxel presente nella scena.

Una volta ricavati questi vettori è sufficiente applicare il procedimento esposto nel listato 4.5 per ottenere rapidamente la separazione dello spazio dei dati nel numero ottimo di classi. In seguito, in figura 5.10, si riporta l’andamento del DBI in corrispondenza dell’intervallo di valori $k \in \mathbb{N}$ ($k \in [2, 50]$) per cui è stato ricavato il numero ideale k^* di cluster.

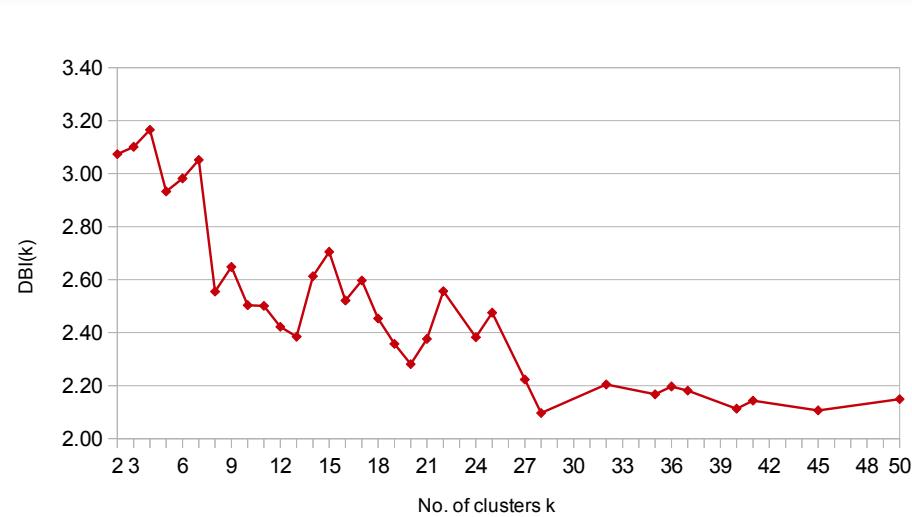


Figura 5.10: Numero ottimo k^* di clusters ottenuto in corrispondenza del minimo valore del DBI per $k \in [2, 50]$

Come si può facilmente verificare sul grafico, il numero ottimo di centroidi si trova in corrispondenza del valore minimo registrato per il DBI che, in questo caso è $k^* = 28$.

Una volta ricavato il giusto numero di classi nelle quali suddividere lo spazio delle features in nostro possesso, è possibile passare alla fase di classificazione della scena; nelle prossime figure saranno riportati alcuni esempi, relativi ai risultati di segmentazione, mediante l'utilizzo esclusivo del passo di classificazione (non supervisionata) e senza l'ausilio dei MRF.

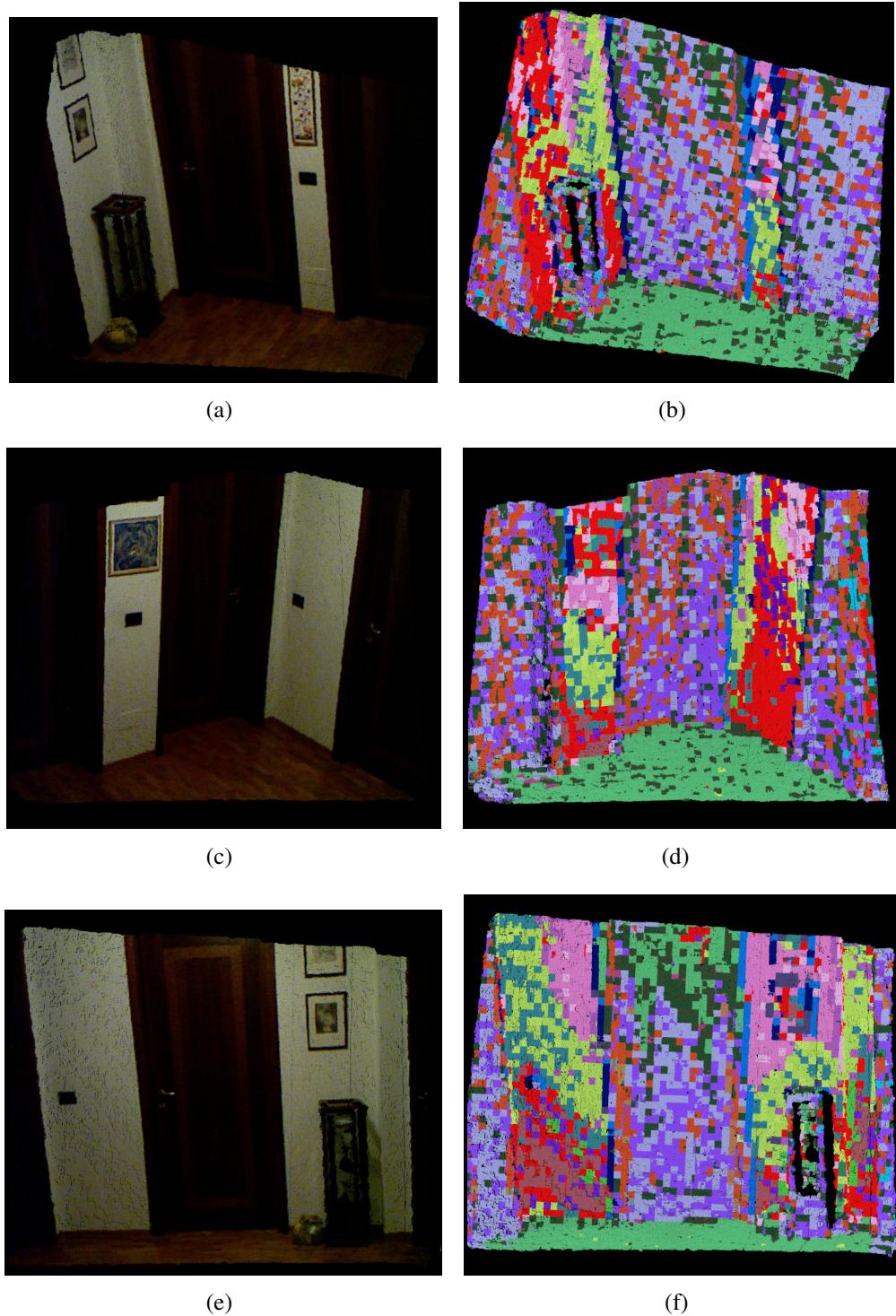


Figura 5.11: Risultati di segmentazione ottenuti mediante il processo di classificazione non supervisionata senza l'utilizzo di MRF: (a, c, e) point cloud originale e (b, d, f) scena etichettata dopo l'utilizzo esclusivo del modello di classificazione non supervisionata con k^* classi

Il risultato più evidente della segmentazione con classificazione non supervisionata è l'estrema frammentazione e la presenza di forti discontinuità tra voxel vicini, l'immagine finale restituita appare quindi molto "rumorosa". Una volta etichettati tutti i voxel della scena ci si presenta uno scenario in cui si riesce a fatica a distinguere le parti principali che si vorrebbe mettere in evidenza, cioè muri, pavimento, porte e così via.

Al fine di mitigare l'estrema irregolarità riscontrata in questi scenari si è fatto ricorso al modello MRF che, grazie alla capacità di utilizzare le relazioni di vicinato rendono l'etichettatura più omogenea come si può vedere per esempio in figura 5.12. La segmentazione ottenuta dopo il passo di minimizzazione dell'energia, presenta agglomerati di voxel omologhi che si identificano facilmente in parti ben definite della scena.

A differenza dei risultati ottenuti nella versione supervisionata dell'algoritmo, in questo caso le etichette assegnate non sono state associate a classi definite a priori. È possibile, però, valutare la compatibilità tra l'etichettatura definita dall'operatore (il ground truth) e quella ottenuta dall'algoritmo. Tipicamente il numero di classi k^* trovato con il metodo non supervisionato è superiore al numero delle macro-categorie definite nel ground truth. È quindi possibile considerare le categorie definite dall'utente come macro-classi e misurare la consistenza verificando che ogni etichetta sia contenuta prevalentemente in un'unica macro-classe.

Questo tipo di classificazione svincola l'utilizzatore del sistema dalla dichiarazione preventiva delle classi desiderate, non è in grado di restituire un partizionamento finale della scena nelle stesse categorie definite per il caso supervisionato, ma è in grado generare una prima, consistente, ripartizione dello spazio osservato in modo che questa risulti immediatamente fruibile come dato di input per sistemi a valle di questo processo, i quali potranno sfruttare questo tipo di informazioni per ulteriori elaborazioni.

Il risultato ottenuto mette in risalto la possibilità di sfruttare i dati derivanti dal processo di segmentazione (svolto tramite l'utilizzo di un classificatore non supervisionato) per poi convergere, tramite altre opportune tecniche di aggregazione, ad una labellizzazione finale della scena che sia in grado di superare ai limiti tipici indotti da un apprendimento di tipo supervisionato: eventuale scarsa capacità di generalizzazione, necessità di un ampio dataset per l'addestramento e dipendenza dal ground-truth in fase di training.

Per cercare di esprimere in che misura le classi trovate siano ripartite tra le macro-

categorie attese dopo l'applicazione del MRF, sia C_{ij} il numero di voxel classificati come appartenenti all' i -esimo cluster e associati alla j -esima macro-categoria; definiamo quindi l'*indice di ripartizione relativo* η_i nella seguente maniera:

$$\eta_i = \frac{T_i}{\sum_j C_{ij}} \quad (5.6)$$

dove $T_i = \max_j C_{ij} \quad \forall j \in \{Door, Wall, Floor, Other\}$

In tabella 5.16 si presentano i risultati finali di segmentazione ottenuti dalla comparazione tra le etichette assegnate ai vari voxel e le macro-classi presenti nelle immagini di ground-truth. Questo al fine di ricavare una misura di quanto i voxel associati ad una certa classe, tra le k^* ricavate, siano ripartiti sulle macro-classi *Door*, *Wall*, *Floor* e *Other* che si intende evidenziare nel risultato finale.

Oltre al calcolo degli indici di partizione per ogni singola classe definiamo anche l'*indice di ripartizione medio* $\bar{\eta}$ nella seguente maniera:

$$\begin{aligned} \bar{\eta} &= w_{tot}^{-1} \sum_{i=1}^{k^*} w_i \cdot \eta_i \\ &= \frac{\sum_{i=1}^{k^*} \sum_j C_{ij} \eta_i}{\sum_{i=1}^{k^*} \sum_j C_{ij}} \end{aligned} \quad (5.7)$$

Questo indice altro non è che la media aritmetica dei singoli indici di ripartizione, pesata sul numero di voxel associati ad ogni classe.

È da osservare come questi aggregati di voxel rispecchino comunque un partizionamento della scena che, seppur maggiormente abbondante in termini di numero di suddivisioni ottenute, ricalca prevalentemente gli stessi risultati ottenuti nel caso dell'impiego di reti neurali artificiali e SVM: le parti che nel caso supervisionato erano comprese all'interno di un'unica classe, risultano comunque ottenibili mediante l'unione di più partizioni ottenute attraverso il processo non supervisionato (figura 5.13). I voxel appartenenti ai cluster ricavati non vengono totalmente ribaltati su un'unica macro-classe ma le percentuali di ripartizione calcolate presentano comunque dei valori molto buoni indicando che, per ognuna delle classi ricavate, la maggioranza assoluta dei voxel viene

riversato su un'unica macro-categoria. L'indice di ripartizione medio, calcolato sulla totalità dei dati riportati in tabella 5.16, evidenzia un altro dato positivo, e cioè come mediamente oltre l'80% dei voxel appartenenti ad una delle k^* classi venga associato ad un'unica macro-classe, confermando così il buon comportamento dell'algoritmo di segmentazione.

Notiamo infine come alcune delle k^* classi ricavate non “sopravvivano” al processo di normalizzazione indotto dai MRF e scompaiano completamente a seguito dell'applicazione dell'algoritmo di taglio; il motivo di questo fenomeno può essere giustificato dal fatto che i cluster per i quali si verifica il fatto raccolgono un basso numero di voxel: questi risultano sparsi all'interno della scena e possono trovarsi circondati da voxel con un'etichetta differente. Successivamente l'effetto di smoothing, introdotto dai MRF, induce una soppressione di questi elementi penalizzando le discontinuità di label associati a voxel vicini.

	Before MRF				After MRF				η_i
	Door	Wall	Floor	Other	Door	Wall	Floor	Other	
Class 1	7	13	4	3	0	0	0	0	0
Class 2	510	1545	7442	290	222	1067	8432	141	0.855
Class 3	26	2	20	6	0	0	0	0	0
Class 4	1245	154	487	329	15	2	1	1	0.789
Class 5	27	128	165	4	0	0	0	0	0
Class 6	513	1374	1052	5854	1699	1746	1454	18794	0.793
Class 7	1716	313	606	573	11	2	1	4	0.6111
Class 8	5	0	1	2	0	0	0	0	0
Class 9	102	536	219	141	165	769	136	73	0.628
Class 10	23	157	65	24	2	12	2	1	0.706
Class 11	264	875	808	3168	78	250	77	2721	0.871
Class 12	63	484	246	52	0	0	0	0	0
Class 13	103	812	294	401	52	1369	42	280	0.785
Class 14	37	172	83	32	0	0	0	0	0
Class 15	61	751	530	62	0	0	0	0	0
Class 16	49	16	24	27	0	0	0	0	0
Class 17	466	770	111	599	0	0	0	0	0
Class 18	70	474	340	42	0	0	0	0	0
Class 19	98	161	80	536	35	41	31	172	0.616
Class 20	17	45	17	20	0	0	0	0	0
Class 21	31	155	284	43	0	0	0	0	0
Class 22	37	433	148	173	0	7	2	0	0
Class 23	63	378	166	101	74	546	45	194	0.636
Class 24	61	235	114	30	0	0	0	0	0
Class 25	71	8	56	0	0	0	0	0	0
Class 26	21	18	13	38	0	0	0	0	0
Class 27	278	613	3922	406	201	441	4669	663	0.782
Class 28	57	644	275	43	57	902	81	76	0.808
Average allocation index $\bar{\eta}$								0.803	

Tabella 5.16: Ripartizione delle classi generate sulle macro-classi del ground-truth

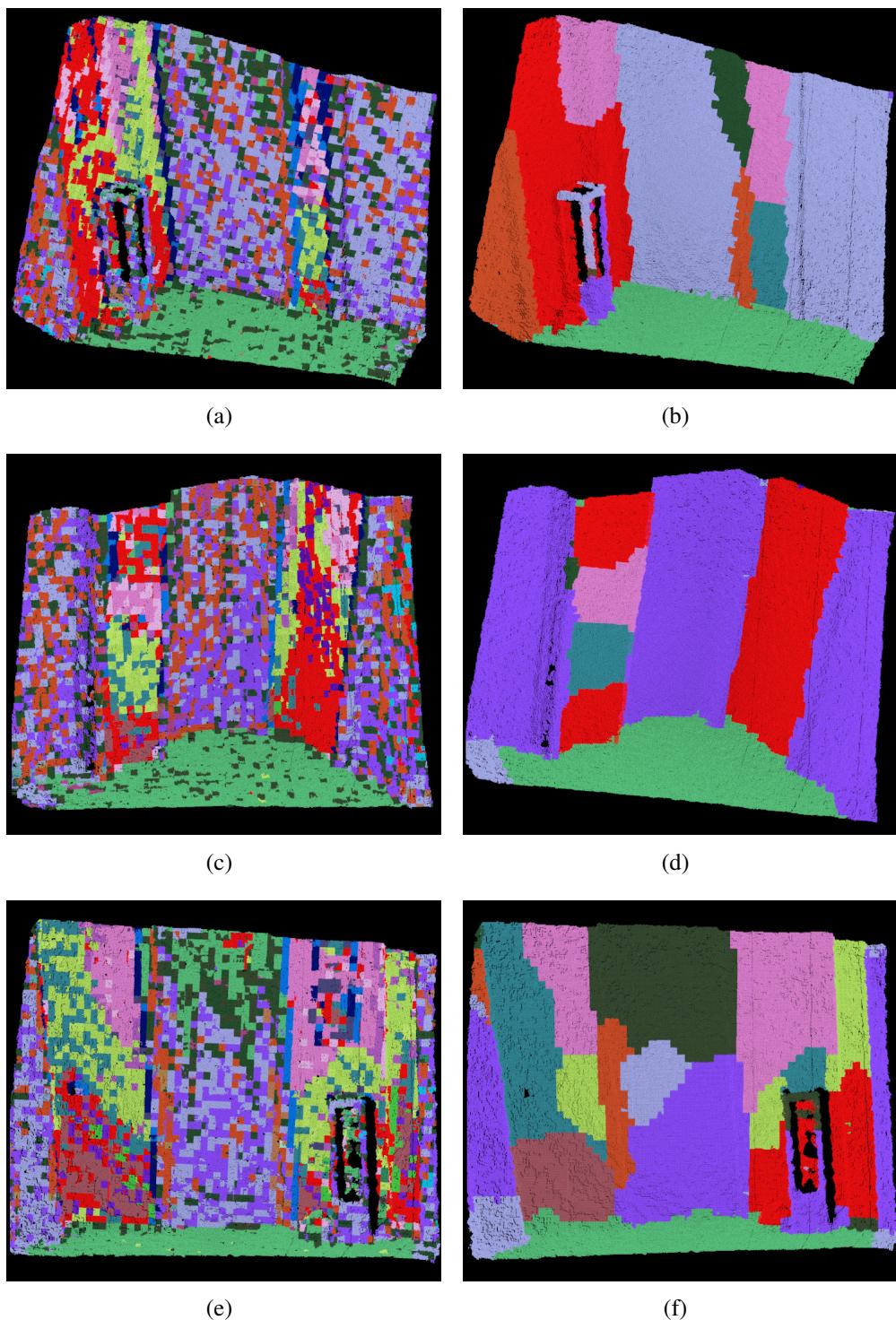


Figura 5.12: Risultati di segmentazione ottenuti mediante il processo di classificazione non supervisionata con l'utilizzo di MRF: (a, c, e) segmentazione di partenza e (b, d, f) scena etichettata dopo la minimizzazione dell'energia associata al MRF

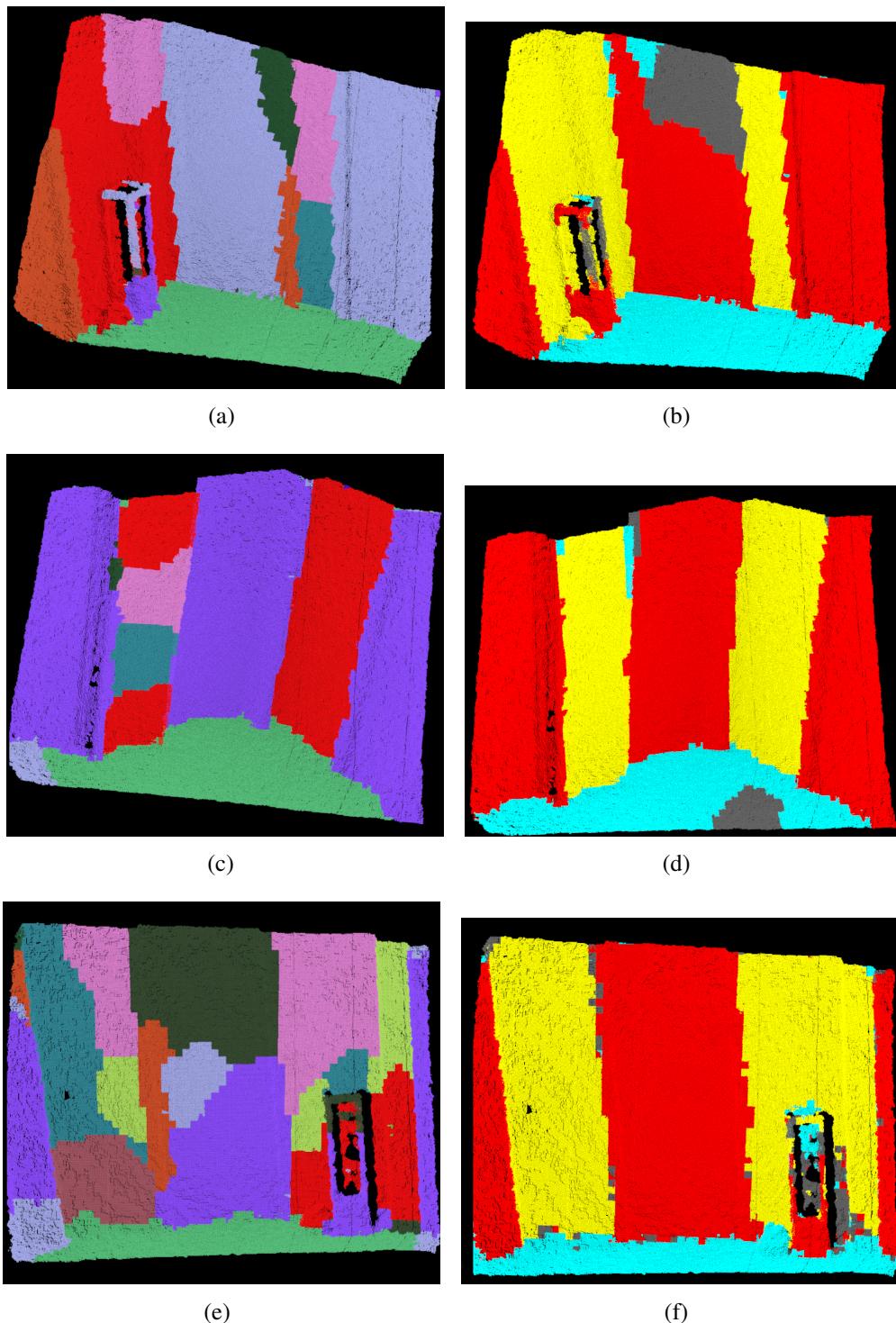


Figura 5.13: Comparazione dei risultati finali di segmentazione ottenuti mediante il processo di classificazione non supervisionata (a, c, e) e supervisionata (b, d, f); in entrambi i casi vi è stato l'impiego di MRF

5.2.3.1 Tempi di addestramento e classificazione non supervisionata

In questa sezione si presentano alcune brevi statistiche sui tempi di addestramento e classificazione per il caso non supervisionato; nello specifico, i dati che verranno riportati a breve riguarderanno l'utilizzo dell'algoritmo k-Means "ottimizzato" posto in essere per questo lavoro di tesi.

	Average value μ	Standard deviation σ
Enhanced k-Means (without MRF)	18.963	1.660
Enhanced k-Means (with MRF)	21.1	1.374
Enhanced k-Means training		88.87

Tabella 5.17: Tempi di classificazione mediante l'utilizzo dell'algoritmo k-Means ottimizzato con e senza l'impiego di MRF

Dai dati riportati in tabella 5.17 vediamo come in questo caso il processo di segmentazione risulti più rapido rispetto allo stesso eseguito tramite classificatori supervisionati, sia in caso di utilizzo di MRF che in assenza del loro impiego. La giustificazione di questo risultato è giustificato dal fatto che il processo di classificazione risulta più veloce in quanto consiste nel semplice calcolo della distanza euclidea tra il voxel preso in considerazione ed i centroidi ricavati in fase di addestramento.

Notiamo infine come la fase di addestramento risulti meno onerosa in termini temporali per il caso non supervisionato che per quello supervisionato se si confronta con i risultati temporali riguardanti il training di reti neurali. Nel complesso le SVM rimangono invece le più veloci con un tempo di addestramento inferiore al secondo. Nel grafico riportato in figura 5.14 sono stati paragonati tutti i tempi di classificazione (con e senza MRF) ottenuti finora, sia per il caso supervisionato che per quello non supervisionato.

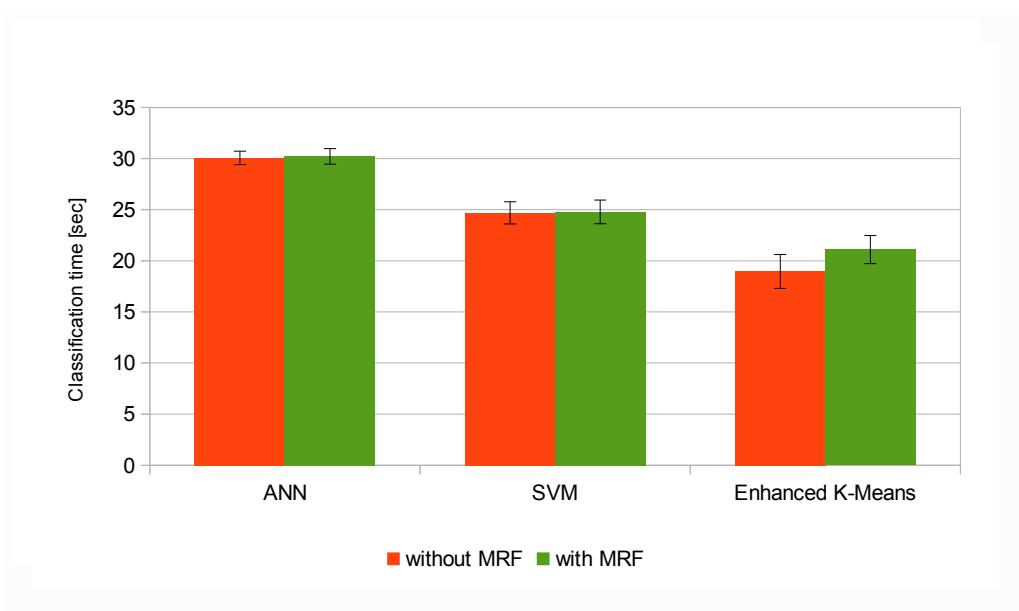


Figura 5.14: Comparazione dei tempi di classificazione a confronto ottenuti finora con i vari tipi di classificatori utilizzati con e senza l'ausilio di MRF

5.2.4 Tempi d'esecuzione dell'algoritmo

L'algoritmo presentato finora è composto da diverse fasi e sotto-fasi (come incapsulamento della cloud in voxel, estrazione dei vettori di caratteristiche, classificazione, ecc.) che possono risultare anche molto differenti in termini di tempo d'esecuzione. In seguito si propone un'analisi quantitativa della ripartizione su base temporale delle varie fasi che costituiscono l'algoritmo sviluppato.

Si è potuto osservare come, a causa delle differenti tecniche di classificazione adottate (supervisionata con rete neurale/SVM e non supervisionata con k-Means ottimizzato), i tempi d'esecuzione dell'algoritmo possano subire variazioni anche significative; a parte questo nulla è stato ancora detto su come le singole parti dell'algoritmo pesino sul tempo complessivo richiesto per l'ottenimento del risultato finale. Questo tipo d'analisi è utile per avere una visione della ripartizione sotto il profilo temporale delle singole fasi, portare in evidenza eventuali colli di bottiglia all'interno del sistema e ottenere indicazioni di dove poter andare ad intervenire per cercare di migliorare il tempo di risposta complessivo che intercorre tra l'acquisizione dei dati e l'output conclusivo. Conseguentemente a questo fatto riportiamo ora alcuni diagrammi atti a raffigurare in che misura le singole fasi d'elaborazione pesino all'interno dell'intero ciclo di vita del processo di segmentazione.

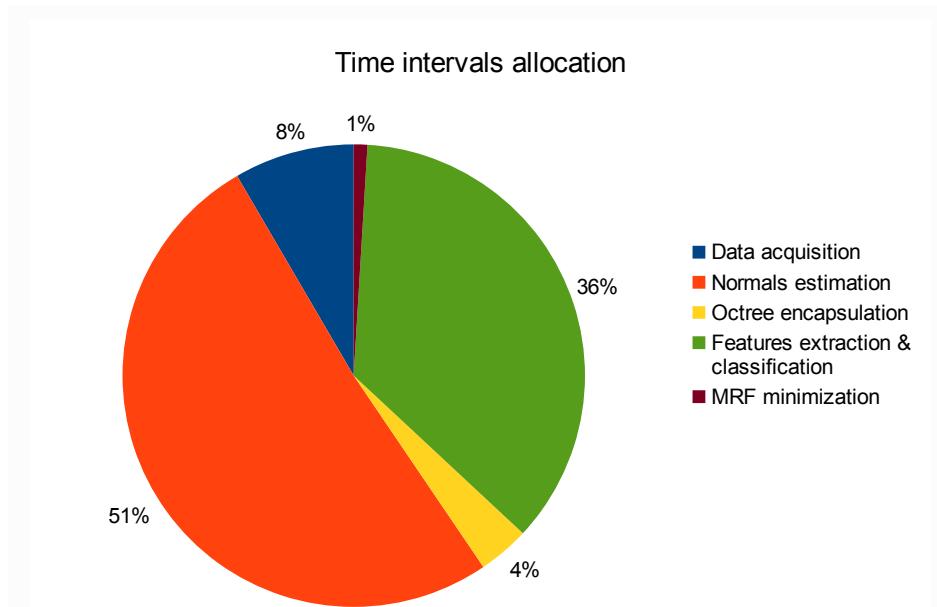


Figura 5.15: Tempi d'esecuzione dell'algoritmo: caso supervisionato, classificazione ottenuta mediante rete neurale artificiale

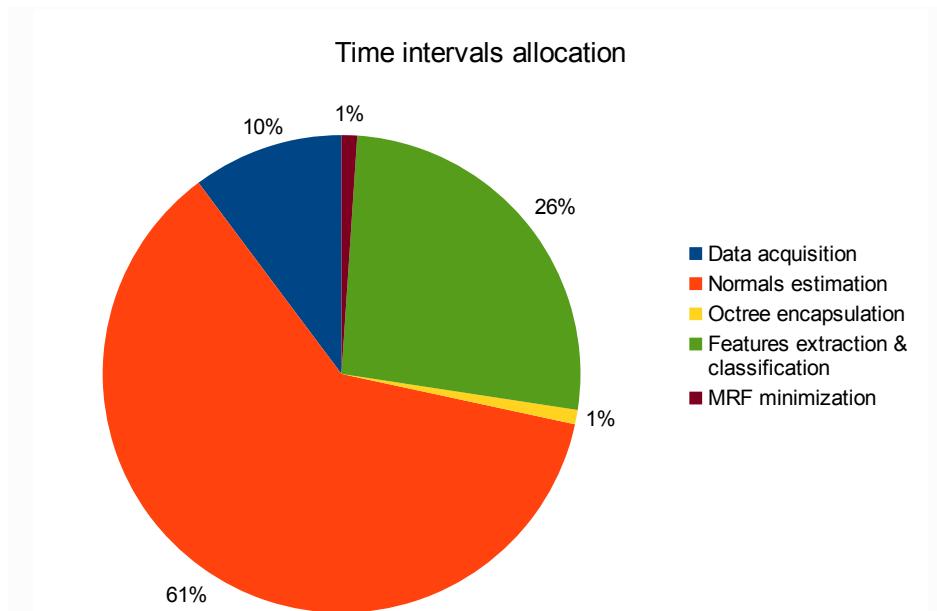


Figura 5.16: Tempi d'esecuzione dell'algoritmo: caso supervisionato, classificazione ottenuta mediante support vector machines

Come possiamo notare dai diagrammi riportati nelle figure 5.15 e 5.16 le fasi di caricamento dei dati acquisiti, l'incapsulamento in octree e la fase di inferenza tramite MRF occupano complessivamente circa il 12-13% del tempo d'esecuzione, mentre circa un terzo del tempo è impegnato dalle operazioni di estrazioni di caratteristiche e conseguente classificazione. La parte più onerosa in termini temporali risulta essere la fase di preprocessing che, in entrambi i casi, da sola occupa più del 50% del tempo d'esecuzione: in questa fase vengono calcolate le normali relative a tutti i punti che compongono la nuvola; questo tipo di operazione risulta computazionalmente molto dispendiosa introducendo così un ritardo non indifferente in fase d'esecuzione: dato che l'intero ciclo di segmentazione prende in questi casi circa 25-30 secondi, la stima delle normali introduce un ritardo di circa 12-15 secondi.

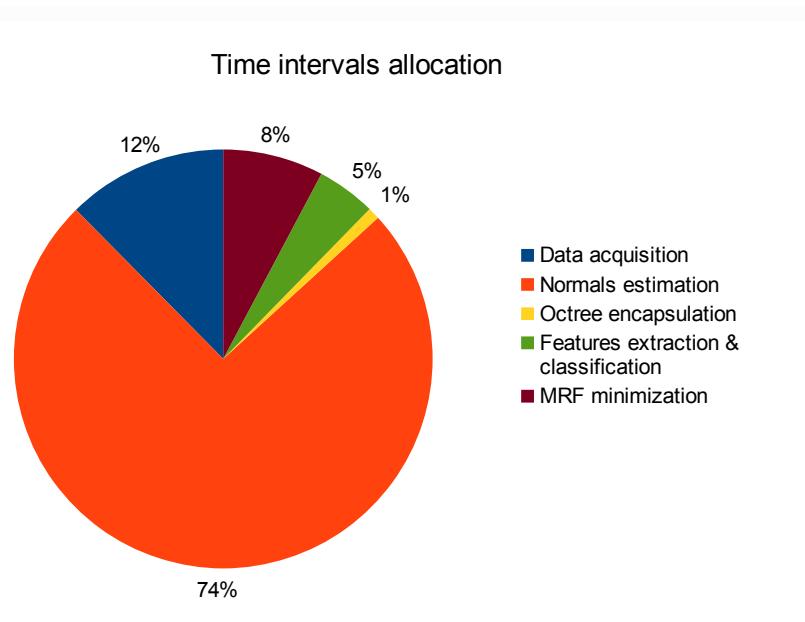


Figura 5.17: Tempi d'esecuzione dell'algoritmo: caso non supervisionato, classificazione ottenuta mediante k-Means ottimizzato

Per quanto riguarda il caso non supervisionato anche qui è possibile notare come il 25% circa di tutto il tempo d'esecuzione racchiuda di fatto quasi la totalità delle operazioni eseguite. Come già descritto nelle sezioni precedenti, l'attività di classificazione dopo l'applicazione dell'algoritmo k-Means ottimizzato risulta molto più veloce rispetto alla controparte supervisionata: il tutto si riconduce al calcolo della distanza tra i vettori di caratteristiche ed ogni centroide ricavato; questo è anche il motivo principale del perché sia possibile ottenere una segmentazione della scena in tempi comunque inferiori al caso supervisionato più veloce (ottenuto con le SVM).

Comunemente a quanto già osservato poco fa la parte più pesante di tutto il processo è costituita dal calcolo dei vettori normali ai punti della cloud: in questa circostanza questa fase arriva ad occupare addirittura oltre il 70% di tutto il ciclo di vita dell'algoritmo; considerando un tempo globale medio d'esecuzione di circa 20 secondi, la stima delle normali introduce una latenza di almeno 15 secondi.

Concludendo, possiamo identificare nella fase di preprocessing il collo di bottiglia principale di tutto il sistema: riuscire a diminuire il tempo di calcolo sotto questo aspetto significherebbe abbassare di altrettanti secondi l'attesa necessaria all'ottenimento dell'output finale di segmentazione. Pertanto, in un'ottica mirata al miglioramento del processo implementato, questa è la fase sulla quale è preferibile intervenire per poter ottenere i benefici maggiori in termini di risposta del sistema. Verranno presentati nel prossimo capitolo conclusioni e sviluppi futuri.

Capitolo 6

Conclusioni e sviluppi futuri

In questo lavoro di tesi è stato presentato un algoritmo per la classificazione e segmentazione di immagini di profondità attraverso l'estrazione di features e l'applicazione di Markov Random Field. L'immagine di profondità che è stata acquisita con un sensore Kinect viene efficientemente decomposta in componenti elementari rappresentati da voxel. Ciascun componente viene caratterizzato da proprietà di colore, tessitura e forma che sono impiegate per assegnare un'etichetta con metodi supervisionati o non supervisionati. Infine il risultato della classificazione viene ulteriormente raffinato con il MRF tenendo conto delle relazioni di vicinato dei singoli voxel. Dati sperimentali hanno dimostrato che il metodo presentato è in grado di segmentare in modo consistente la scena inquadrata.

Il sistema proposto si è rivelato capace di funzionare in maniera corretta indistintamente dal tipo di classificatore utilizzato, indipendentemente dal fatto che questo sia stato generato attraverso un procedimento di addestramento supervisionato o no. Nel primo caso siamo in grado di ricavare una ripartizione della scena direttamente in macro-categorie arbitrarie (definite a priori), mentre nel caso non supervisionato viene restituita una riorganizzazione della scena in gruppi di voxel omologhi. In entrambi i casi l'etichettatura permette di partizionare la scena e semplificare ulteriori compiti, come ad esempio l'individuazione ed il riconoscimento di uno specifico oggetto. L'approccio supervisionato già consente di interpretare semanticamente la scena, quello non supervisionato è invece più generale, identificando solo gruppi di zone omologhe. I risultati

derivanti da questo processo di segmentazione tramite classificazione non supervisionata, possono quindi essere sfruttati come un solido punto di partenza all'interno di un sistema più complesso operante, ad esempio, nell'ambito del riconoscimento di oggetti. I risultati ottenuti sperimentalmente dimostrano che un framework basato su Markov Random Field può essere sfruttato per raffinare risultati di segmentazione, anche piuttosto rumorosi ed incerti, grazie alla capacità di riuscire a cogliere le relazioni di vicinato che intercorrono tra gli elementi interessati durante la fase di classificazione. L'applicazione dei MRF permette di aumentare in maniera significativa il grado di accuratezza di un sistema che si serve esclusivamente di classificatori per l'ottenimento della segmentazione finale. Inoltre, è stato dimostrato dai dati ricavati che questo risultato è raggiungibile senza inficiare eccessivamente i tempi di risposta complessivi.

Dal punto di vista progettuale è possibile utilizzare il sistema realizzato astraendo completamente l'utilizzatore dalla sua struttura interna: il sistema può essere infatti visto come una sorta di *black-box* che non fa altro che ricevere una nuvola di punti, e relativa immagine RGB, e restituire una seconda nuvola di punti già etichettata secondo le classi designate. Altro punto di forza è che, per natura stessa dei dati di input richiesti, il software realizzato può essere impiegato non solo con il Microsoft Kinect: di fatto risulta utilizzabile in combinazione con qualsiasi tipo di sensore (o apparato sensoriale) di profondità, ovviamente a patto che questo sia in grado di fornire dati di input del tipo atteso (cloud e immagine).

Dall'analisi temporale eseguita sull'algoritmo è emerso che allo stato di realizzazione attuale non è possibile utilizzarlo in applicazioni real-time; lo studio di occupazione temporale dei vari passi eseguiti ha però evidenziato come la fase critica, sotto il profilo temporale, sia quella di pre-elaborazione in cui viene eseguito il calcolo dei vettori normali alla superficie della nuvola di punti; questo è un primo indizio utile per capire dove intervenire per diminuire i tempi d'attesa richiesti allo stato attuale.

6.1 Sviluppi futuri

I possibili sviluppi per il sistema realizzato sono diversi. Il processo, per come è stato strutturato, permette l’elaborazione di un’unica scena per poi terminare: una prima modifica evolutiva potrebbe essere quella di cambiare la logica d’esecuzione in modo che il sistema sia in grado di elaborare un flusso continuo di dati senza dover terminare una volta ottenuta la segmentazione finale. Contestualmente a questo tipo d’intervento, risulterebbe conveniente l’integrazione di tutto il processo realizzato all’interno di un’infrastruttura software progettata in maniera da risultare particolarmente flessibile, facilmente riutilizzabile, mantenibile ed espandibile come il framework *Robot Operating System* (ROS [87]). Questo framework, appositamente sviluppato per la realizzazione di sistemi su architetture robotiche, mette a disposizione un sistema di comunicazione basato su nodi (comunicanti sulla base di un paradigma di tipo *publish-subscribe*) all’interno del quale potrebbe essere calato il sistema. In un contesto del genere si potrebbe andare a racchiudere tutta la logica realizzata all’interno di un nodo in grado di cooperare con altri in maniera distribuita.

Sebbene il sistema abbia restituito buoni risultati, esso presenta comunque margini di miglioramento per quanto riguarda i tempi d’esecuzione: nonostante l’hardware sul quale è stato testato non sia molto recente, le latenze richieste per l’ottenimento dell’output finale indicano che si è ancora parecchio lontani da un utilizzo in un contesto di tipo real time.

Avendo individuato il principale collo di bottiglia nella fase di preprocessing (nella fat-tispecie nel calcolo delle normali alla cloud), è sicuramente possibile diminuire i tempi di risposta intervenendo sulla tecnica adottata per il calcolo delle normali; questo in favore di un procedimento alternativo che risulti meno oneroso su base temporale o di una tecnica ugualmente efficace nel descrivere le caratteristiche di forma ma comunque eseguibile in meno tempo.

Bibliografia

- [1] Microsoft kinect. <http://www.xbox.com/en-us/kinect/>.
- [2] Point cloud library. <http://pointclouds.org/>.
- [3] Ross Kindermann and J. Laurie Snell. *Markov Random Fields and Their Applications*. Contemporary Mathematics. American Mathematical Society, Providence, Rhode Island, 1980.
- [4] Richard Szeliski, Ramin Zabih, Daniel Scharstein, Olga Veksler, Vladimir Kolmogorov, Aseem Agarwala, Marshall Tappen, and Carsten Rother. A Comparative Study of Energy Minimization Methods for Markov Random Fields. In *95h European Conference on Computer Vision*, volume 30, pages 1068–1080, Los Alamitos, CA, USA, jun 2006. IEEE.
- [5] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. Technical Report TR-2001-22, Mitsubishi Electric Research Laboratories, San Francisco, CA, USA, January 2002.
- [6] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.
- [7] Robert B. Fisher and Kurt Konolige. Range sensors. In Bruno Siciliano and Ousama Khatib, editors, *Springer Handbook of Robotics*, pages 521–542. Springer, 2008.

- [8] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.
- [9] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *In ECCV*, pages 404–417, 2006.
- [10] Gregory T. Flitton, Toby P. Breckon, and Najla Megherbi Bouallagu. Object recognition using 3D SIFT in complex CT volumes. In Frédéric Labrosse, Reyer Zwiggelaar, Yonghuai Liu, and Bernie Tiddeman, editors, *BMVC*, pages 1–12. British Machine Vision Association, 2010.
- [11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An Efficient Alternative to SIFT or SURF. In *International Conference on Computer Vision*, Barcelona, 2011.
- [12] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *In European Conference on Computer Vision*, pages 430–443, 2006.
- [13] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: binary robust independent elementary features. In *Proceedings of the 11th European conference on Computer vision: Part IV*, ECCV’10, pages 778–792, Berlin, Heidelberg, 2010. Springer-Verlag.
- [14] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In *CVPR*, pages 510–517. IEEE, 2012.
- [15] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (FPFH) for 3D registration. In *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ICRA’09, pages 1848–1853. IEEE Press, 2009.
- [16] Andrew Edie Johnson and Martial Hebert. Using spin images for efficient object recognition in cluttered 3D scenes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(5):433–449, 1999.

- [17] Li Fei-Fei, Rob Fergus, and Antonio Torralba. Recognizing and learning object categories. ICCV short course, 2009.
- [18] R. Adams and L. Bischof. Seeded region growing. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(6):641–647, jun 1994.
- [19] Stuart Geman and Donald Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6:721–741, 1984.
- [20] Tai-Pang Wu, Kam-Lun Tang, Chi-Keung Tang, and Tien-Tsin Wong. Dense photometric stereo: A markov random field approach. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(11):1830 –1846, nov. 2006.
- [21] Anand Rangarajan, Rama Chellappa, and Anand Rangarajan. Markov random field models in image processing, 1995.
- [22] C. D’Elia, G. Poggi, and G. Scarpa. A tree-structured Markov random field model for bayesian image segmentation. *Image Processing, IEEE Transactions on*, 12(10):1259 – 1273, oct. 2003.
- [23] D.E. Melas and S.P. Wilson. Double Markov random fields and Bayesian image segmentation. *Signal Processing, IEEE Transactions on*, 50(2):357 –365, feb 2002.
- [24] Wojciech Pieczynski and Abdel-Nasser Tebbache. Pairwise Markov random fields and segmentation of textured images. In *Machine Graphics and Vision*, pages 705–718, 2000.
- [25] S. Gould, R. Fulton, and D. Koller. Decomposing a scene into geometric and semantically consistent regions. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1 –8, 29 2009-oct. 2 2009.
- [26] M. Pawan Kumar and Daphne Koller. Efficiently selecting regions for scene understanding, 2010.

- [27] Chris Russell, Philip H. S. Torr, and Pushmeet Kohli. Associative hierarchical crfs for object class image segmentation. In *In Proc. ICCV*, 2009.
- [28] Joseph Tighe and Svetlana Lazebnik. Superparsing: Scalable nonparametric image parsing with superpixels. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *ECCV (5)*, volume 6315 of *Lecture Notes in Computer Science*, pages 352–365. Springer, 2010.
- [29] Charles Sutton and Andrew Mccallum. An introduction to conditional random fields for relational learning. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*. 2007.
- [30] Stephen Gould, Jim Rodgers, David Cohen, Gal Elidan, and Daphne Koller. Multi-class segmentation with relative location prior. *Int. J. Comput. Vision*, 80(3):300–316, dec 2008.
- [31] Xuming He Richard, Richard S. Zemel, and Miguel A. Carreira. Multiscale conditional random fields for image labeling. In *In CVPR*, pages 695–702, 2004.
- [32] Charles Sutton, Khashayar Rohanimanesh, and Andrew Mccallum. Dynamic Conditional Random Fields: Factorized Probabilistic Models for Labeling and Segmenting Sequence Data, 2004.
- [33] Context M. For. TextonBoost: Joint Appearance, Shape and Context Modeling for Multi-Class Object Recognition and Segmentation.
- [34] Jakob Verbeek and William Triggs. Scene Segmentation with CRFs Learned from Partially Labeled Images. In John C. Platt, Daphne Koller, Yoram Singer, and Sam Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20, pages 1553–1560, Vancouver, Canada, January 2008. NIPS Fondation.
- [35] Jamie Shotton. Contour-based learning for object detection. In *In Proc. ICCV*, pages 503–510, 2005.
- [36] Sanjiv Kumar and M. Hebert. Discriminative random fields: a discriminative framework for contextual interaction in classification. In *Computer Vision, 2003*.

- Proceedings. Ninth IEEE International Conference on*, pages 1150 –1157 vol.2, oct. 2003.
- [37] Charles Bouman and Bede Liu. Multiple resolution segmentation of textured images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:99–113, 1991.
 - [38] Z. Kato, M. Berthod, and J. Zerubia. Parallel image classification using multi-scale markov random fields. In *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*, volume 5, pages 137 –140 vol.5, april 1993.
 - [39] Derek Hoiem, Alexei A. Efros, and Martial Hebert. Geometric context from a single image. In *In ICCV*, pages 654–661, 2005.
 - [40] Lin Yang, Peter Meer, and David J. Foran. Multiple class segmentation using a unified framework over mean-shift patches. In *In CVPR*, 2007.
 - [41] B.C. Russell, W.T. Freeman, A.A. Efros, J. Sivic, and A. Zisserman. Using multiple segmentations to discover objects and their extent in image collections. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 1605 – 1614, 2006.
 - [42] Joseph J. Lim, Pablo Arbeláez, Chunhui Gu, and Jitendra Malik. Context by region ancestry.
 - [43] Marcel Haselich, Marc Arends, Dagmar Lang, and Dietrich Paulus. Terrain classification with markov random fields on fused camera and 3d laser range data. In *Proceedings of the 5th European Conference on Mobile Robotics (ECMR)*, pages 153–158, 2011.
 - [44] Sunando Segupta, Paul Sturgess, Lubor Ladicky, and Philip H. S. Torr. Automatic dense visual semantic mapping from street-level imagery, 2011.

- [45] Paul Sturgess, Kartek Alahari, Lubor Ladicky, and Philip H. S. Torr. Combining appearance and structure from motion features for road scene understanding. In *BMVC*. British Machine Vision Association, 2009.
- [46] Primesense. <http://www.primesense.com/>.
- [47] Jamie Shotton, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake. Real-Time Human Pose Recognition in Parts from Single Depth Images. June 2011.
- [48] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *In Proc. UIST*, pages 559–568, 2011.
- [49] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. A large-scale hierarchical multi-view RGB-D object dataset. In *ICRA*, pages 1817–1824. IEEE, 2011.
- [50] Abhishek Anand, Hema Koppula, Thorsten Joachims, and Ashutosh Saxena. Contextually guided semantic labeling and search for 3d point clouds. 2012.
- [51] N. Silberman and R. Fergus. Indoor scene segmentation using a structured light sensor. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 601–608, nov. 2011.
- [52] Nyu depth dataset v2.
- [53] Pushmeet Kohli, L'Ubor Ladický, and Philip H. Torr. Robust higher order potentials for enforcing label consistency. *Int. J. Comput. Vision*, 82(3):302–324, may 2009.
- [54] Michael I. Jordan. Graphical models. *Statistical Science*, 2004.
- [55] Michael I. Jordan. Learning in graphical models. 19(1):140–155, 2004.
- [56] Rui Huang. *Graphical models for object segmentation*. PhD thesis, New Brunswick, NJ, USA, 2008. AAI3349688.

- [57] J. M. Hammersley and P. Clifford. Markov field on finite graphs and lattices. 1971.
- [58] F. Y. Wu. The Potts model. *Reviews of Modern Physics*, 54(1):235–268, January 1982.
- [59] Kevin Lai, Liefeng Bo, Xiaofeng Ren, and Dieter Fox. Detection-based object labeling in 3d scenes. In *ICRA*, pages 1330–1337. IEEE, 2012.
- [60] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgbd mapping: Using depth cameras for dense 3d modeling of indoor environments. In *In RGB-D: Advanced Reasoning with Depth Cameras Workshop in conjunction with RSS*, 2010.
- [61] Zhou Ren, Jingjing Meng, Junsong Yuan, and Zhengyou Zhang. Robust hand gesture recognition with kinect sensor. In *Proceedings of the 19th ACM international conference on Multimedia*, MM ’11, pages 759–760, New York, NY, USA, 2011. ACM.
- [62] D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, jun 1982.
- [63] Opencv. <http://opencv.org/>.
- [64] R. Cucchiara, C. Grana, M. Piccardi, A. Prati, and S. Sirotti. Improving shadow suppression in moving object detection with hsv color information. In *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pages 334 –339, 2001.
- [65] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In L. M. Le Cam and J. Neyman, editors, *Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. University of California Press, 1967.
- [66] Fast artificial neural network (fann). <http://leenissen.dk/fann/wp/>.
- [67] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition, 2003.

- [68] Tiny svm. <http://chisen.org/~taku/software/tinysvm/>.
- [69] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.
- [70] Graph-cut optimization library (gco). <http://vision.csd.uwo.ca/code/>.
- [71] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23:2001, 2001.
- [72] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(9):1124 –1137, sept. 2004.
- [73] V. Kolmogorov and R. Zabin. What energy functions can be minimized via graph cuts? *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(2):147 –159, feb. 2004.
- [74] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, sep 1975.
- [75] Jordan Reynolds and Kevin Murphy. Figure-ground segmentation using a hierarchical conditional random field. In *Proceedings of the Fourth Canadian Conference on Computer and Robot Vision, CRV '07*, pages 175–182, Washington, DC, USA, 2007. IEEE Computer Society.
- [76] J. R. Movellan. Tutorial on Gabor Filters. *Tutorial paper* <http://mplab.ucsd.edu/tutorials/pdfs/gabor.pdf>, 2008.
- [77] Simon Haykin. *Neural Networks: A Comprehensive Foundation (2nd Edition)*. Prentice Hall, 2 edition, jul 1998.
- [78] A. K. Kolmogorov. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 114:369–373, 1957.

- [79] Nello Cristianini and John Shawe-Taylor. *An introduction to support Vector Machines: and other kernel-based learning methods.* Cambridge University Press, New York, NY, USA, 2000.
- [80] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeing. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '07, pages 1027–1035, Philadelphia, PA, USA, 2007. Society for Industrial and Applied Mathematics.
- [81] David L. Davies and Donald W. Bouldin. A cluster separation measure. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-1(2):224 –227, april 1979.
- [82] John C. Platt. Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, pages 61–74. MIT Press, 1999.
- [83] D. M. Greig, B. T. Porteous, and A. H. Seheult. Exact maximum a posteriori estimation for binary images. 1989.
- [84] H. Ishikawa and D. Geiger. Segmentation by grouping junctions. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 125 –131, jun 1998.
- [85] Y. Boykov, O. Veksler, and R. Zabih. Markov random fields with efficient approximations. In *Computer Vision and Pattern Recognition, 1998. Proceedings. 1998 IEEE Computer Society Conference on*, pages 648 –655, jun 1998.
- [86] Olga Veksler. *Efficient graph-based energy minimization methods in computer vision.* PhD thesis, Ithaca, NY, USA, 1999. AAI9939932.
- [87] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

- [88] Xiaofeng Ren, Liefeng Bo, and D. Fox. RGB-(D) scene labeling: Features and algorithms. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2759 –2766, june 2012.