Alma Mater Studiorum – Università di Bologna

DIPARTIMENTO DI INFORMATICA - SCIENZA E INGEGNERIA (DISI)
Scuola di Ingegneria e Architettura

Tesi di laurea
in
Calcolatori Elettronici-T

Sperimentazione con algoritmi per l'analisi di nuvole di punti per applicazioni di guida autonoma

Candidato:
Andrea Garbugli

Relatore:
Prof. Stefano Mattoccia

Correlatore:
Dott. Matteo Poggi

Anno Accademico 2014-2015
Sessione III

Sommario

Al giorno d'oggi quasi tutte le persone possiedono un mezzo motorizzato che utilizzano per spostarsi. Tale operazione, che risulta semplice per una persona, può essere compiuta da un robot o un autoveicolo in modo autonomo? La risposta a questa domanda è si, ma se ad una persona serve solo un po' di pratica per guidare, questa azione non risulta altrettanto immediata per dei veicoli motorizzati. In soccorso ad essi vi è la *Computer Vision*, un ramo dell'informatica che, in un certo senso, rende un elaboratore elettronico in grado di percepire l'ambiente circostante, nel modo in cui una persona fa con i propri occhi. Oggi ci concentreremo su due campi della computer vision, lo SLAM o Simultaneous Localization and Mapping, che rende un robot in grado di mappare, attraverso una camera, il mondo in cui si trova ed allo stesso tempo di localizzare, istante per istante, la propria posizione all'interno di esso, e la *Plane Detection*, che permette di estrapolare i piani presenti all'interno di una data immagine.

Indice

1 Introduzione **1**

1.1 Obbiettivo della tesi 1

1.2 Computer Vision 2

1.2.1 Cenni sulla visione stereo 2

2 SLAM **5**

2.1 Sistemi SLAM 5

2.2 Sensore per l'acquisizione delle immagini 6

2.3 Stereo SLAM 7

3 Risultati sperimentali **11**

3.1 Test con camera stereo 11

3.1.1 Schermate di controllo 14

4 Algoritmi per la Plane Detection **17**

4.1 Plane Detection 17

4.2 Cenni sugli algoritmi 18

4.2.1 Algoritmi basati su RANSAC 18

4.2.2 Algoritmo basato sulle Normali 19

4.2.3 Algoritmi basato sulla trasformata di Hough 20

4.2.4 Algoritmo basato sul Region Growing 20

5 Programmi per i test **21**

5.1 Programma di test con GUI 21

6 Risultati Sperimentali **23**

6.1 Definizioni usate nei test 23

6.2 KITTI dataset 24

6.3 Stereo Camera dataset 40

7 Conclusioni **43**

A Qt Framework **45**

A.1 Qmake 46

A.2 Programma d'Esempio 47

B CMake **51**

B.1 Esempio di file CMakeLists.txt 51

Elenco delle figure

1.1 Esempio di camera stereo ideale. 2

2.1 Sensore basato su FPGA realizzato presso il DISI. 6
2.2 Pipeline di elaborazione. 7

3.1 Esempio di immagine left e di disparità. 12
3.2 Nuvola di punti ottenuta con bundle adjustment e loop closures. 12
3.3 Nuvola di punti ottenuta senza bundle adjustment e loop closures. 13
3.4 Visualizzatore di due nuvole in sequenza. 14
3.5 Finestra con informazioni sui keypoints. 14

4.1 Esempio utilizzo di RANdom SAmple Consensus (RANSAC). 19

5.1 Schermata principale del programma *PlaneDetection*. 22

6.1 Immagine 000000_10 25
6.2 Piani trovati per l'immagine 000000_10. 27
6.3 Immagine 000005_10 29
6.4 Piani trovati per l'immagine 000005_10. 31
6.5 Immagine 000016_10 32
6.6 Piani trovati per l'immagine 000016_10. 35
6.7 Immagine 000036_10 37
6.8 Piani trovati per l'immagine 000036_10. 38
6.9 Piani trovati per l'immagine con la panchina e il tronco. . . . 41

A.1 Programma d'esempio. 47
A.2 Signal and Slot. 48

Elenco delle tabelle

3.1 Tempi di esecuzione dei singoli step. 13

6.1 Parametri per l'immagine 000000_10. 26
6.2 Risultati per l'immagine 000000_10. 28
6.3 Parametri per l'immagine 000005_10. 30
6.4 Risultati per l'immagine 000005_10. 33
6.5 Parametri per l'immagine 000016_10. 34
6.6 Risultati per l'immagine 000016_10. 36
6.7 Parametri per l'immagine 000016_10. 37
6.8 Risultati per l'immagine 000036_10. 39
6.9 Parametri per l'immagine 000005_10. 42

Capitolo 1

Introduzione

In questo capitolo introduttivo occorre precisare che il lavoro di tesi presentato nel seguente testo è stato diviso in due parti distinte, ma facenti parte della stessa macrocategoria, ovvero la *Computer Vision*.

La prima parte del lavoro si è incentrata sul problema dello SLAM, un problema ampiamente studiato in robotica, e nello specifico sullo studio del programma *StereoSLAM* [2], ovvero un'implementazione di questo metodo che facente uso di immagini catturate da una *stereo camera*.

Nella seconda parte, invece, verranno esposti i risultate dei test effettuati sulla libreria *lib_plane_detection*, contenente un insieme di metodi che implementano degli algoritmi per la *Plane Detection*, ovvero l'individuazione di piani in immagini contenenti dati che permettono tale operazione.

1.1 Obbiettivo della tesi

Come prima accennato, la tesi ha due obbiettivi separati, che verranno affrontati secondo la seguente struttura:

Parte I Studio e test del programma StreoSLAM al fine di migliorare la qualità delle mappe ricostruite a partire da immagini stereo.

Parte II Obbiettivo della seconda parte è quello di unificare, all'interno di una libreria di funzioni, diversi algoritmi per la plane detection, testandoli e cercando di rendere il codice il più portabile possibile.

1.2 Computer Vision

La *Computer Vision* è un campo della scienza che include metodi di acquisizione, processazione e analisi di immagini bidimensionali. Lo scopo principale di questa disciplina è quello di ottenere i dati del mondo reale attraverso immagini e video, per poi tradurli in informazioni numeriche o simboliche in grado di essere processate da un elaboratore elettronico.

Esempi di applicazione della computer vision sono:

- In un processo industriale per controllare la bontà del prodotto;
- Nella navigazione di veicoli autonomi o robot mobili;
- Video sorveglianza o tracciamento di persone;
- Per organizzare database di immagini;
- Analisi di immagini mediche.

1.2.1 Cenni sulla visione stereo

La *visione stereo* è una tecnica della computer vision, mirata alla ricostruzione di una scena tridimensionale tramite l'ausilio di una camera composta da uno o più sensori.

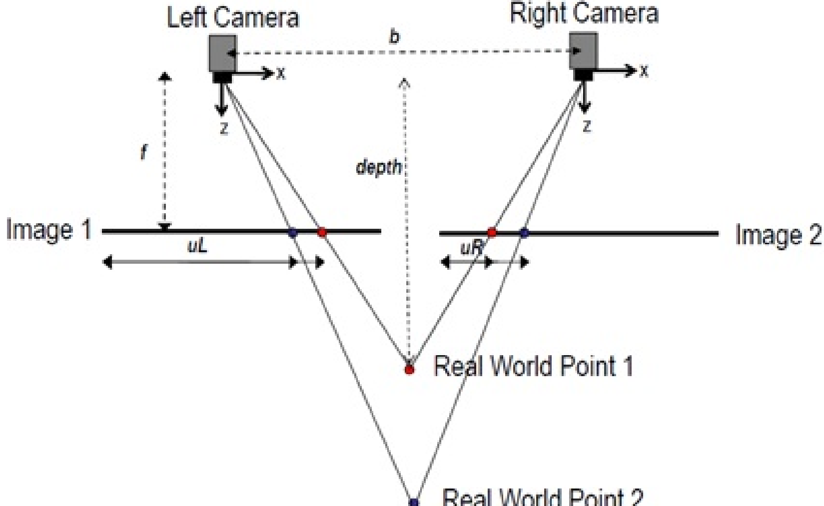


Figura 1.1: Esempio di camera stereo ideale.

Per esempio preso un sistema stereo composto da due camere, sotto determinate ipotesi, quali: piani immagine della camere perfettamente allineati, assi ottici paralleli, sensori con identica lunghezza focale; due frame con assi paralleli differiscono solo per una traslazione lungo l'asse orizzontale, detta *baseline* (b). La figura 1.1 riporta lo schema semplificato della configurazione di una stereo camera ideale.

Come prima accennato sotto le ipotesi di idealità del sistema stereo è la sola coordinata orizzontale ad essere diversa. Questa differenza permette di determinare la distanza del punto nel sistema di riferimento tridimensionale osservato. In particolare, si ha:

$$u_L = x_L \frac{f}{z}$$

$$u_R = x_R \frac{f}{z}$$

Dalla differenza dei due valori calcolati sopra è possibile ricavare il parametro detto *disparità*:

$$d = u_L - u_R$$

La disparità può essere utilizzata nel metodo della triangolazione, questo permette di ricavare i dati tridimensionali relativi ai punti delle immagini acquisite dalla camera stereo.

Capitolo 2

SLAM – Simultaneous localization and mapping

Lo SLAM o *Simultaneous Localization and Mapping* è un problema di robotica, che si chiede se sia possibile per un robot, o qualsiasi altro veicolo con possibilità di movimento, ricostruire una mappa dell'ambiente circostante, ed allo stesso momento determinare la propria posizione all'interno dello scenario di azione.

2.1 Sistemi SLAM

La maggior parte dei sistemi pensati per risolvere il problema dello SLAM sono basati su un approccio strutturale comune; in particolare sono eseguite una serie di operazioni in sequenza che, a partire da dati in ingresso come immagini e dati di calibrazione dalle camera usata, danno in uscita una mappa 3D; per questo si parla di pipeline di elaborazione. Il primo passo di elaborazione è sempre l'acquisizione di immagini, e appunto dei dati necessari per determinare la posizione dei punti in uno spazio tridimensionale. Successivamente, sono sfruttate *features*[1] visuali per individuare punti di interesse nelle immagini ed eseguire un allineamento tridimensionale relativo fra coppie di frame. Tale allineamento è possibile anche sfruttando direttamente i dati 3D, ad esempio usando l'algoritmo *Iterative Closest Point (ICP)*. A volte i passaggi prima elencati non sono sufficienti per ottenere un buon risultato,

[1] Per feature si intende una specifica struttura, come possono essere un vertice, punto o oggetto appartenente ad un'immagine, che contiene informazioni rilevanti per la risoluzione di un calcolo computazionale al fine di risolvere un determinato problema.

quindi spesso si ricorre a tecniche per incrementare la robustezza delle mappe ottenute, come l'individuazione di *loop closures* oppure il *bundle adjustment*, come spiegato in Cavina [2]. Infine, grazie alle stime sulle trasformazioni calcolate si attua un processo di allineamento dei dati 3D rispetto ad un sistema globale, che da come risultato la mappa finale.

2.2 Sensore per l'acquisizione delle immagini

Prima di addentrarsi nella spiegazione del programma occorre soffermarsi sul componente senza il quale lo SLAM non sarebbe possibile; stiamo parlando dell'elemento elettronico in grado acquisire i dati per la pipeline di elaborazione, che nel nostro caso trattasi di una *stereo camera*, più precisamente del sensore 3D realizzato presso il DISI, presentato in Mattoccia, Marchio e Casadio [5] e mostrato in figura 2.1.

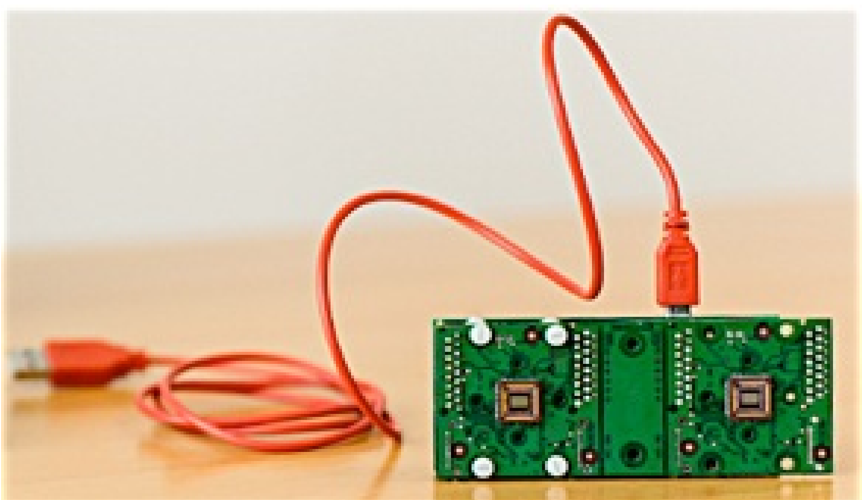


Figura 2.1: Sensore basato su FPGA realizzato presso il DISI.

La camera stereo è un sensore passivo, ciò significa che non fa uso di pattern luminosi proiettati sulla scena per determinare i dati tridimensionali. Un sensore stereo, per essere tale, deve comporsi almeno di due sensori ottici (camere), che come è facilmente intuibile devono essere sincronizzate nell'operazione di acquisizione delle immagini, inoltre i due sensori devono essere sempre calibrati prima di procedere con una qualsiasi sessione di registrazione dei dati. Una volta che tutti i dati sono disponibili, data la posizione reciproca dei sensori ottici sulla scheda, è possibile calcolare il

valore della distanza di un punto nella scena catturata; questo grazie alla triangolazione delle proiezioni del punto stesso nelle immagini catturate rispettivamente dal primo e dal secondo sensore.

2.3 Stereo SLAM

Iniziamo ora a parlare del sistema SLAM pensato da Cavina, e dei livelli che ne compongono la *pipeline* di elaborazione (vedi figura 2.2).

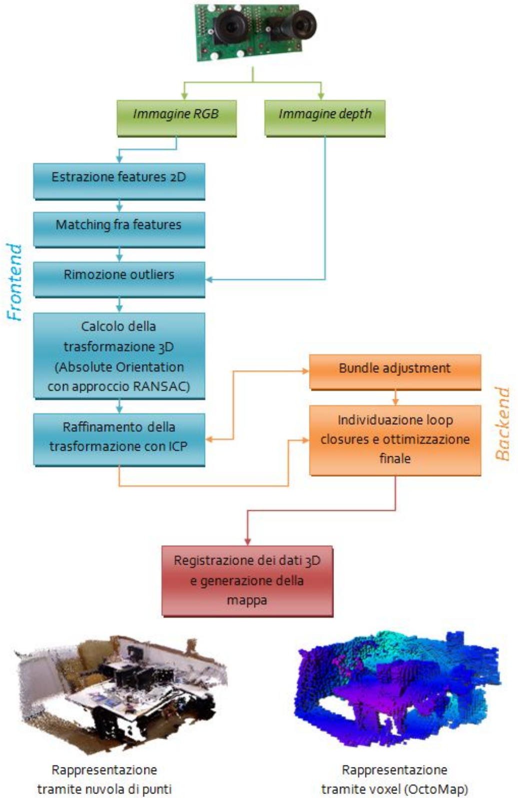


Figura 2.2: Pipeline di elaborazione.

Il sistema si compone principalmente di due parti, il *frontend* e il *backend*. La prima fornisce una stima della trasformazione fra coppie di frame adiacenti, mentre la seconda ha come obiettivo l'ottimizzazione del mapping sulla base delle informazioni ottenute dal frontend e ridurre al minimo il drift accumulato nel corso dell'esecuzione.

Senza addentrarsi una spiegazione approfondita di tutti i passaggi, per la quale si rimanda a [2], le operazione svolte dal programma sono riassumibili secondo il seguente schema:

Generazione nuvola di punti A partire dalle immagini RGB-D viene estratta grazie ad una libreria apposita una nuvola di punti tridimensionale.

Estrazione dei keypoints Per ogni frame vengono estratte delle features, questa fase, detta anche feature detection, consiste nell'individuazione di aree di interesse all'interno dell'immagine.

Descrizione dei keypoints In questa fase ogni regione intorno ai punti di interesse rivelati viene trasformata in un descrittore compatto il più possibile stabile e invariante rispetto alle condizioni in cui è stata acquisita l'immagine.

Matching fra descrittori Per fare si che due nuvole di punti, prese da due immagini successive, vengano allineate fra loro è necessario trovare un collegamento fra i rispettivi keypoints, ed è proprio questo lo scopo di questa fase.

Filtraggio degli outliers Si tratta di una fase molto importante, infatti non è sempre possibile ottenere degli ottimi risultati dal passaggio precedente; a volte possono essere presenti dei match errati, detti outliers, che vanno appunto filtrati.

Stima della trasformazione 3D Una volta che le features sono state calcolate e i match sono stati individuati in modo corretto, è possibile procedere con il calcolo della trasformazione rigida che consente di proiettare i punti 3D relativi ai keypoints della prima immagine nei corrispondenti punti 3D della seconda immagine.

Ottimizzazione della registrazione Anche se i passi fin qui analizzati sono sufficienti per ottenimento delle buone mappe, a volte è possibile incappare in errori sulla stima della posa relativa fra due nuvole. Per risolvere questo problema, il programma sfrutta due elementi: un agente che esegue il cosiddetto bundle adjustment, sfruttando i punti visti da più pose della camera, e un agente che rileva le chiusure del loop

e aggiunge vincoli nel caso in cui il robot passi più volte nello stesso punto della scena.

Capitolo 3

Risultati sperimentali su Stereo SLAM

Il programma in questione è stato sviluppato con l'obiettivo di essere implementato in un sistema che utilizzi il sensore di visione stereo realizzato presso il DISI, introdotto nella sezione 2.2.

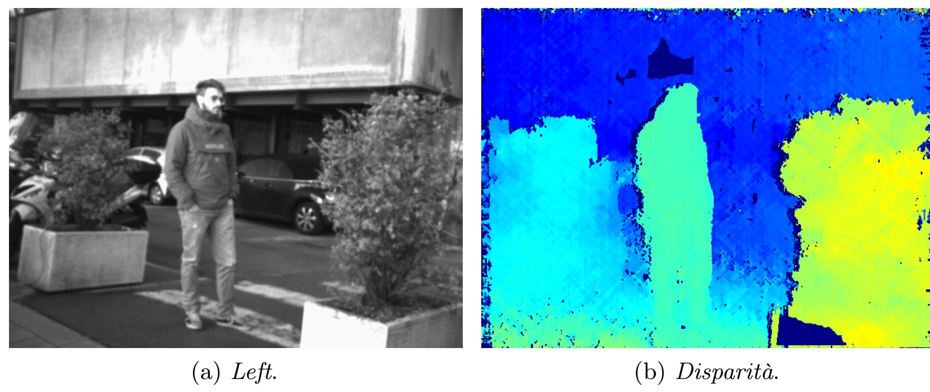
In questo capitolo saranno presentati alcuni dei risultati sperimentali, questi ultimi sono stati ottenuti utilizzando i set di immagini acquisite dalla camera stereo.

Tutti i test sono stati eseguiti su un calcolatore con processore *Intel*® *Core*™ i7-2670QM con 4 core da 2,2 GHz, memoria RAM da 8 GB. Il sistema operativo scelto per l'esecuzione del programma è Ubuntu 14.04.

3.1 Test con camera stereo

I test sul sistema sono stati effettuati sfruttando il sensore di visione stereo realizzato presso il DISI. In particolare, l'implementazione realizzata consente l'elaborazione di dati RGB-D acquisiti tramite tale sensore: si utilizzano immagini RGB con risoluzione 640 x 480 e le relative mappe di disparità generate dal sensore tramite calcolo su FPGA. Condizione necessaria per ottenere dati utilizzabili per l'esecuzione della pipeline SLAM è aver effettuato una calibrazione stereo accurata, quindi avere a disposizione i parametri necessari per ottenere immagini rettificate e per poter generare dati 3D (nuvole di punti) a partire dalle mappe di disparità fornite dal sensore.

Nella figura 3.1 viene mostrato il risultato in uscita dalla scheda FPGA della camera stereo.



(a) *Left.* (b) *Disparità.*

Figura 3.1: Esempio di immagine left e di disparità.

I risultati ottenuti dai test con la camera stereo possono essere valutati solo a carattere qualitativo, in quanto non era possibile avere a priori delle informazioni rispetto alle pose dalla camera, in ogni caso verranno riportati i tempi impiegati dai singoli step della pipeline di elaborazione.

Per testare la bontà del programma, la pipeline è stata sottoposta all'elaborazione di una sequenza di circa 40 immagini. L'ottimo risultato di figura 3.2 è stato ottenuto grazie all'esecuzione congiunta del frontend e del backend, quindi nel programma erano state attivate anche le parti relative al bundle adjustmentl e al loop closures.

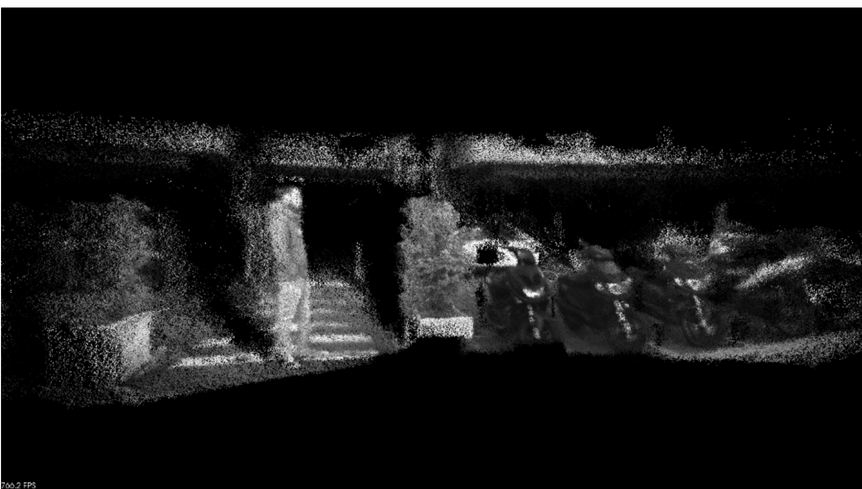


Figura 3.2: Nuvola di punti ottenuta con bundle adjustment e loop closures.

Nella la seconda immagine (figura 3.3), ottenuta utilizzato solo le tecniche del frontend, è possibile notare del rumore (dovuto probabilmente al drift delle pose) nella parte destra.

Figura 3.3: Nuvola di punti ottenuta senza bundle adjustment e loop closures.

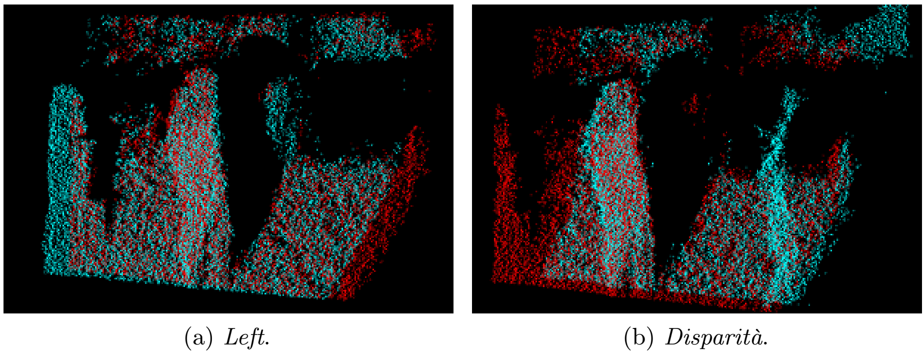
In generale il sistema ottiene degli ottimi risultati anche in merito ai tempi di esecuzione. Analizzando i dati in tabella 3.1 si riscontra che l'attivazione del backend incide sulle prestazioni del programma per poco più di 3 s; e ricordando che sono state utilizzate 40 immagini per l'elaborazione è immediato calcolare che il tempo per processare un singolo frame è di soli 700 ms.

Tabella 3.1: Tempi di esecuzione dei singoli step.

Step	Tempo Totale (ms)	Tempo Medio (ms)
Extract Features	13 966,279	349,157
Match Features	1500,553	38,476
Remove Outliers	385,433	9,883
Absolute Orientatin RANSAC	10 066,070	14,652
Remove Outliers	744,660	1,084
Loop Closures	292,543	10,448
Bundle Adjustment	18,432	0,658
Final Loop Closures	2095,391	
Final Bundle Adjustment	1207,936	
Programma Completo	32 412,7	

3.1.1 Schermate di controllo

Per controllare meglio i passaggi intermedi sono stati aggiunti alcuni strumenti di debug visuale. Il primo, di cui sono riportati due esempi in figura 3.4, mostra a schermo la sovrapposizione di due nuvole ottenute da due frame successivi; quella del primo frame colorata in rosso, mentre la nuvola del secondo frame viene proposta in azzurro. Nella figura 3.4b la situazione si capovolge, ma questa volta vengono prese in considerazione le nuvole della seconda e terza immagine della sequenza immessa nel programma.



(a) *Left.* (b) *Disparità.*

Figura 3.4: Visualizzatore di due nuvole in sequenza.

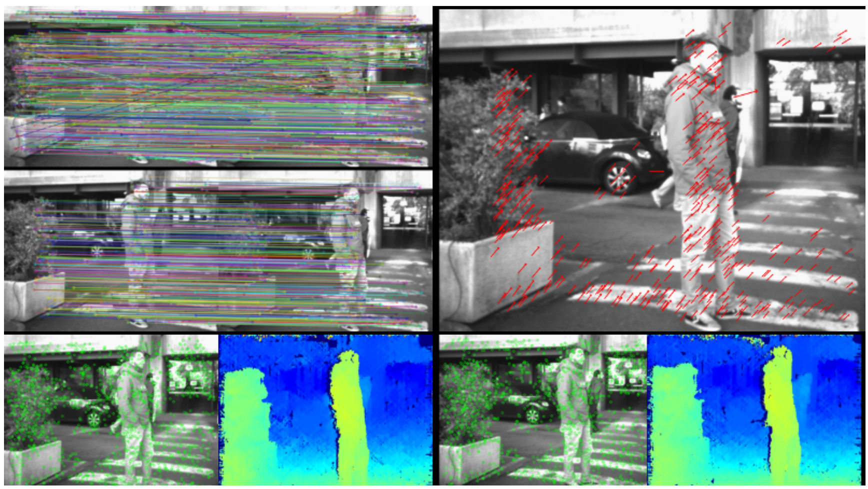


Figura 3.5: Finestra con informazioni sui keypoints.

Il secondo strumento si compone di una finestra a griglia, visibile in figura 3.5, la quale contiene informazioni relative ai keypoints, come:

- matching dei descrittori con e senza outliers;

- posizione delle features (puntini verdi);
- mappe di disparità;
- immagine con lo spostamento (nel verso delle frecce rosse) dei keypoints in immagini adiacenti.

Capitolo 4

Algoritmi per la Plane Detection

4.1 Plane Detection

Nel campo della guida autonoma, ma non solo, la percezione dell'ambiente circostante, è indispensabile per la corretta riuscita di questa operazione. Essendo le geometrie dello spazio in continua mutazione quando si compie un'azione di movimento, è essenziale che un'intelligenza artificiale sia in grado di determinare, in tempo reale, le informazioni del mondo circostante. Per fare in modo che un robot o un veicolo sia consapevole degli oggetti intorno ad esso, si ricorre in prima battuta alla determinazione della superficie sulla quale il soggetto si muove; questo problema è conosciuto col nome di *plane detection*, e per essere risolto a bisogno di informazioni 3D sull'ambiente di lavoro. Esistono diverse tecnologie che permettono di ottenere dati 3D, si pensi ad esempio a camere RGB-D, come il *Microsoft Kinect*, a sistemi come quello *LIDAR* oppure a camere stereo come quella presentata nella sezione 2.2. Una volta ottenute le informazioni 3D è possibile creare una nuvola di punti tridimensionale organizzata, dalla quale è possibile ricavare le informazioni dei piani presenti nella scena. Tutto questo avviene tramite degli algoritmi che saranno discussi nella sezione 4.2.

4.2 Cenni sugli algoritmi

La libreria *lib_plane_detection*, testata in questa seconda parte della tesi, è composta da metodi che implementano diversi algoritmi per l'individuazione dei piani, perciò in questo capitolo verrà fatta una breve premessa su ogni algoritmo in essa presente, rimandando alla bibliografia, in cui sono presenti le tesi degli autori di tali metodi, per degli eventuali approfondimenti.

4.2.1 Algoritmi basati su RANSAC

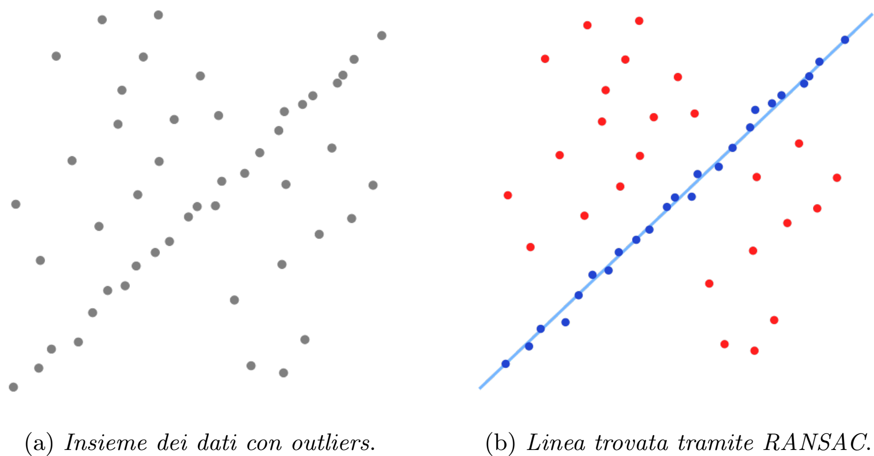
L'algoritmo RANSAC, è un metodo di regressione iterativo nato per la determinazione dei parametri di un modello matematico. La peculiarità di RANSAC è quella di ottenere un buon risultato anche a partire da un insieme di dati contenenti *outlier*[1], in ogni caso occorre precisare che si tratta di un algoritmo non deterministico, ovvero che la bontà del risultato è di tipo probabilistico (aumentando il numero di iterazioni aumenta la probabilità di ottenere un buon risultato).

L'esempio di figura 4.1 mostra come il metodo RANSAC riesca, a partire da un insieme di dati affetti da rumore (figura 4.1a), ad individuare una retta. Nell'immagine 4.1b è possibile vedere la retta risultante di colore blu, mentre i punti di colore rosso rappresentano gli *outliers*, o punti non appartenenti alla retta. I punti, o in generale i dati che fanno parte del modello cercato, prendono il nome di inliers.

All'interno della libreria in esame sono presenti varie implementazioni di RANSAC, quindi è più giusto parlare di una famiglia di algoritmi. Nei prossimi paragrafi daremo un piccolo spazio ad ognuna di queste implementazioni.

Standard RANSAC Il metodo standard per RANSAC consiste nel prendere 3 punti casuali dalla nuvola, calcolare il piano che passa fra loro e determinare quanti dei punti di tutto l'insieme appartengono al suddetto piano. Questa operazione viene ripetuta per il numero massimo di iterazioni impostate, ed ogni volta che viene trovato un piano migliore, questo viene salvato come valore di ritorno della funzione.

[1] Termine utilizzato in statistica per definire dei valori anomali, a volte indicato anche come rumore



(a) *Insieme dei dati con outliers.* (b) *Linea trovata tramite RANSAC.*

Figura 4.1: Esempio utilizzo di RANSAC.

Fast RANSAC Questa variante si differenzia da quella standard solo per il fatto che viene preso come risultato il primo piano valido trovato, quindi la funzione non esegue quasi mai il numero massimo di iterazioni, rendendo questa implementazione la più veloce delle 4.

LS É una versione modificata di RANSAC che oltre a prendere 3 punti in modo casuale ed individuare il piano passante, calcola anche la media della distanza di tutti i punti da tale piano. Ad ogni iterazione viene ripetuto il calcolo e viene tenuto come risultato finale il piano con il valore della distanza media più basso.

Optimized RANSAC Anche questa implementazione è molto simile a quella standard, ma il controllo degli inliers viene effettuato solo su i punti non appartenenti all'insieme dei punti utilizzati per calcolare il piano, cioè quelli del subset.

4.2.2 Algoritmo basato sulle Normali

Il concetto alla base di questo algoritmo è quello di raggruppare i punti aventi la normale molto simile fra loro. Per prima cosa occorre trovare un contorno, possibilmente esente da rumore, per ogni singolo punto come indicato in Barchi [1]; successivamente grazie ai punti del contorno è attuabile il calcolo dei vettori tangenti, ed è proprio da questi ultimi che si ottiene la normale del punto.

4.2.3 Algoritmi basato sulla trasformata di Hough

La trasformata di Hough è una tecnica di estrazione utilizzata nel campo dell'elaborazione digitale delle immagini. Permette di riconoscere diverse forme arbitrariamente definite, logicamente nel caso in esame la forma da trovare è quella di un piano. Partendo dal presupposto che per un punto passano infiniti piani, il metodo della trasformata controlla quale piano passa per il maggior numero di punti. Il concetto è logicamente più complesso di quanto espresso sopra, e nella tesi di Golfieri [4] se ne può trovare una spiegazione più approfondita.

Come RANSAC, anche il metodo basato su Hough è proposto in più versioni all'interno della libreria.

Hough Standard La versione standard della trasformata prevede di testare un numero prestabilito di piani su ogni punto, se il piano è una soluzione per quel punto il contatore ad esso relativo viene incrementato; alla fine il piano con più voti viene preso come quello cercato.

Hough Randomized La variante randomized prevede l'estrazione di terne di punti casuali, dalle quali si ricava il piano che le interseca. In questo modo è possibile trovare piani di grandi dimensioni in tempi più brevi, ma soprattutto di prevedere a priori il momento nel quale la probabilità di aver trovato il piano, permette di interrompere la ricerca.

4.2.4 Algoritmo basato sul Region Growing

Trattasi di un metodo per la segmentazione delle immagini, il cui processo inizia con la localizzazione di punto di interesse, detto *seed point*, all'interno dell'immagine da processare. Dopo aver individuato il punto si procede con il determinare se i pixel nelle sue vicinanze fanno parte dalla regione cercata.

Capitolo 5

Programmi per i test

PER testare gli algoritmi della libreria *lib_plane_detection* sono stati creati due programmi scritti in *C++*, uno dotato di una Graphic User Interface (GUI) creata tramite il framework *Qt*, si veda l'appendice A, mentre l'altro è un classico programma utilizzabile da *console*. In entrambi i casi si è cercato di mantenere il codice più portabile possibile, infatti entrambi i programmi possono essere compilati su diverse piattaforme.

5.1 Programma di test con GUI

Per agevolare sia l'inserimento dei parametri utilizzati in fase di esecuzione dagli algoritmi della libreria, sia la raccolta dei dati ottenuti nei vari test, si è pensato di creare un'apposita interfaccia grafica.

Come mostrato in figura 5.1, la finestra principale del programma è divisa in 3 parti principali. Nella prima parte si trova l'area in cui verrà visualizzata l'immagine di partenza (in bianco e nero) assieme ai punti che fanno parte del piano individuato; tali punti cambiano colore in base all'algoritmo utilizzato per la ricerca. Inoltre sopra l'area prima descritta è possibile notare due pulsati; quello con l'icona 💾, serve per salvare l'immagine del piano trovato; mentre quello con l'icona quando premuto stampa l'immagine con il piano reale (in rosso) nello spazio grigio sotto la selezione del dataset, e avvia una funzione per il confronto fra piano reale e quello individuato dagli algoritmi.

La seconda parte è divisa a sua volta in due sottoparti. La prima contenente un'area di testo in sola lettura per stampare a video i dati del piano individuato, mentre l'altra permette: tramite il tasto arancione di avviare

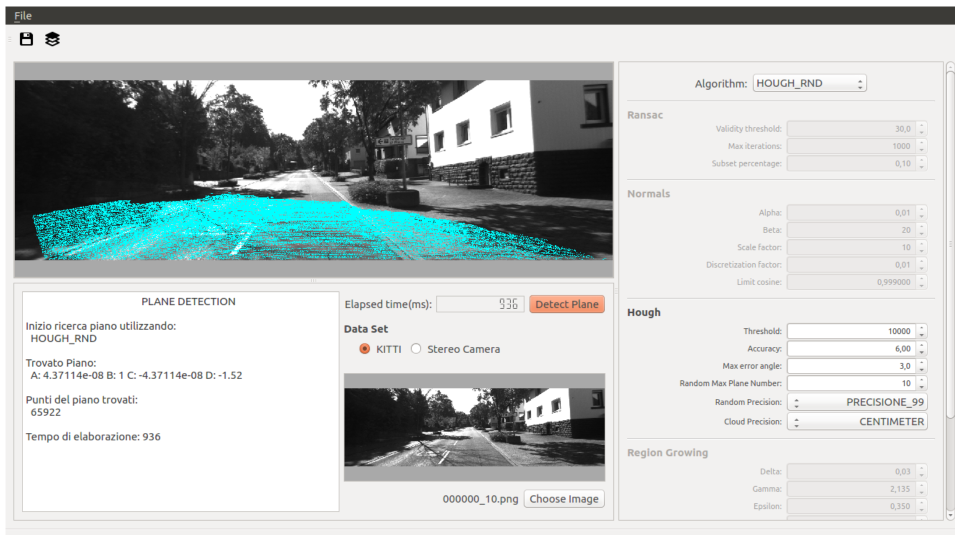


Figura 5.1: Schermata principale del programma *PlaneDetection*.

l'algoritmo selezionato, e con il tasto in grigio chiaro sia di aprire l'immagine da elaborare sia di vederne l'anteprima.

L'ultima area (quella che occupa la destra della finestra) viene utilizzata dall'utente per selezionare, tramite combobox, l'algoritmo. Quest'ultima azione si traduce nell'attivazione della sezione adibita all'inserimento dei parametri dell'algoritmo scelto.

Capitolo 6

Risultati Sperimentali su Plane Detection

Dopo aver presentato sia gli algoritmi che il programma per eseguire i test è tempo di mostrare i risultati derivanti dall'utilizzo della libreria. Dato che le prove sono state fatte su due dataset diversi, il capitolo sarà diviso in due sezioni, la prima riporta i test effettuati su *KITTI*, mentre nella seconda su alcune immagini prese grazie al sensore descritto nella sezione 2.2.

6.1 Definizioni usate nei test

Prima di procedere occorre specificare il significato di alcuni termini che verranno utilizzati nell'esposizione dei risultati. Dati i coefficienti esatti dei piani presenti nell'immagine e il numero dei punti che ne fanno parte, è possibile valutare l'algoritmo mediante:

Veri Positivi Insieme dei punti presenti sia nel piano reale che in quello rilevato dall'algoritmo.

Veri Negativi Insieme dei punti non facenti parte di nessuno dei due piani.

Falsi Positivi Insieme dei punti che fanno parte del piano rilevato, ma non di quello reale.

Falsi Negativi Insieme dei punti presenti nel piano reale, ma che invece non sono stati considerati dall'algoritmo.

Indice di Precisione In un processo di classificazione statistica, la precisione per una classe C è definito come:

$$\mathrm{I_P} = \frac{\text{Veri Positivi}}{\text{Veri Positivi} + \text{Falsi Positivi}}$$

Un indice di precisione di 1.0 sta a significare che ogni oggetto che è stato etichettato come appartenente alla classe C ne fa realmente parte.

Indice di Richiamo In un processo di classificazione statistica, la precisione per una classe C è definito come:

$$\mathrm{I_R} = \frac{\text{Veri Positivi}}{\text{Veri Positivi} + \text{Falsi Negativi}}$$

Un indice di richiamo di 1.0 implica che gli elementi della classe C siano stati etichettati tutti in modo corretto.

6.2 KITTI dataset

Il set di immagini KITTI è una collezione di immagini e mappe di disparità, quest'ultime sono state elaborate sfruttando un sensore LIDAR, un sensore di tipo attivo. Inoltre il set di prova fornisce anche le coordinate esatte del piano, cosicché sia possibile confrontarle con quelle ottenute dagli algoritmi. Le immagini hanno una risoluzione di circa 1240×370 px, di conseguenza gli algoritmi elaborano informazioni per 458 800 punti.

Per ogni immagine varranno mostrati: i parametri utilizzati da ogni singolo algoritmo per la rilevazione, le immagini contenenti il piano individuato (colorato in base all'algoritmo utilizzato), e i valori dei risultati ottenuti.

Dato che non è possibile riportare tutte le immagini del set, ci si è concentrati su quelle che mostrano delle situazioni di particolare interesse, come strade occupate da macchie o stradine di campagna; questo permette di confrontare gli algoritmi nei possibili scenari che si possono presentare.

Prima di iniziare l'esposizione dei risultati occorre aprire una piccola parentesi sul nome dei parametri:

Soglia rappresenta la percentuale minima rispetto al totale dei punti, affinchè un piano venga considerato valido.

Iter. numero di iterazioni eseguite da RANSAC.

Subset numero dei punti, in percentuale rispetto al totale, sul quale viene effettuato il controllo.

Prof. spessore del piano espresso in centimetri, ovvero la distanza massima per la quale un punto viene considerato parte del piano.

α, β **e Scala** valori legati all'intorno nei quali calcolare la normale, per dettagli si rimanda a Barchi [1].

Coseno vincolo sulla differenza fra le normali per essere considerate appartenenti allo stesso piano.

Disc. fattore di discretizzazione.

Soglia di Hough numero di punti minimo per cui il piano è considerabile valido.

Acc. accuratezza intesa come spessore del piano.

Ang. fattore di discretizzazione dell'inclinazione dei piani espresso in gradi.

N Piani numero dei piani su cui effettuare il test.

Prec. (R) percentuale di precisione richiesta nel calcolo dei piani.

Prec. (C) scala di misura per l'accuratezza.

δ, γ **e** ϵ parametri di vincolo sull'appartenenza alla regione.

Dist. Est. Ris. $\bigcup$ **e Punti** sono rispettivamente distanza dal piano, estensione, risoluzione, valore dell'unione e numero dei punti; per la spiegazione si veda Rucci [7].

Figura 6.1: Immagine 000000_10

Immagine 000000_10 La figura 6.1, mostra come il piano di questa immagine si sviluppi molto sia in larghezza che profondità. Inoltre si noti come la differenza di spessore fra strada e marciapiede sia molto piccola; questa peculiarità farà si che gli algoritmi non distinguano queste due zone.

I parametri utilizzati sono visibili nella tabella 6.1, si faccia caso al fatto che i parametri sono stati raggruppati per famiglie, ovvero per i 4 algoritmi basati su RANSAC sono stati utilizzati gli stessi parametri. Lo stesso discorso vale anche per le varianti di Hough.

Tabella 6.1: Parametri per l'immagine 000000_10.

(a) *Parametri RANSAC.*

Soglia	Iter.	Subset	Prof.
30	1000	0,10	0,15

(b) *Parametri Normals.*

ε	β	Scala	Discr.	Coseno	Prof.
0,01	20	10	0,01	0,999	0,15

(c) *Parametri Hough.*

Soglia	Acc.	Ang.	N Piani	Prec. (R)	Prec. (C)
10 000	6,00	3,0	25	PRE_99	CENT

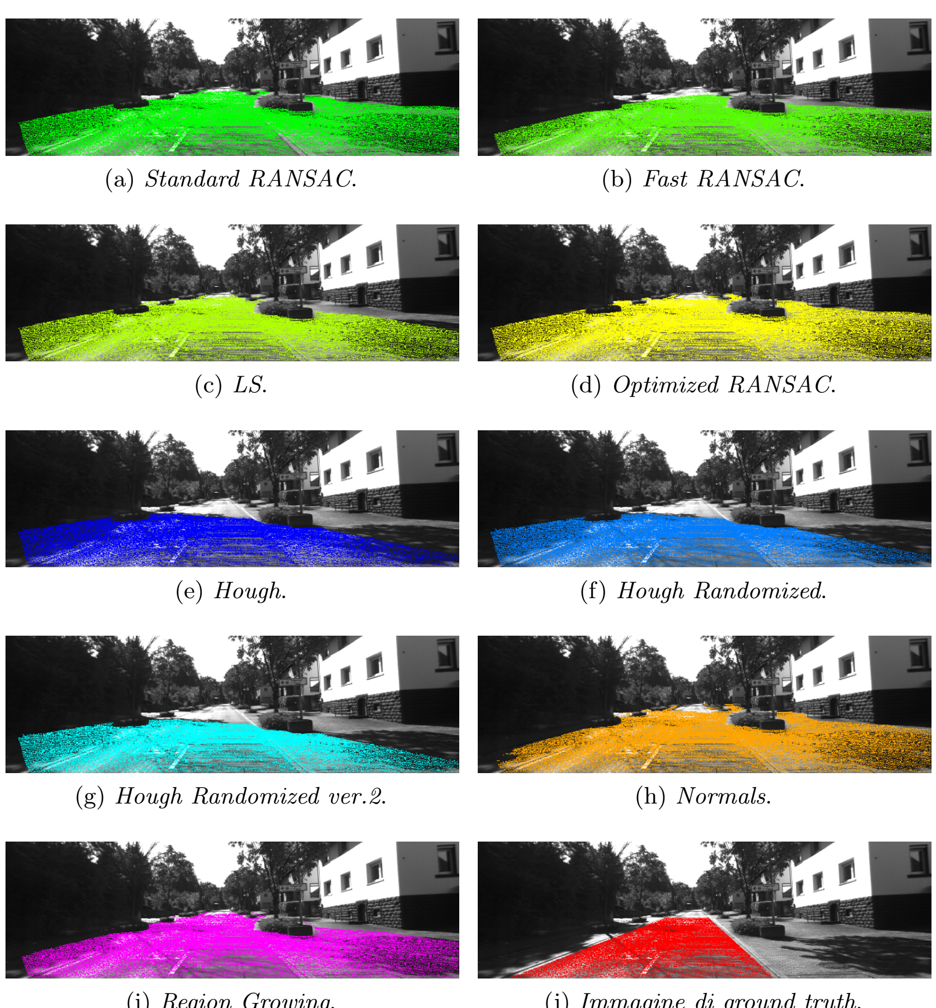
(d) *Parametri Region Growing.*

δ	γ	ϵ	Dist.	Est.	Ris.	$\bigcup$	Punti
0,05	0,735	0,350	40	3	5	0,20	4000

Nella figura 6.2 è possibile vedere i piani ottenuti per ogni singolo algoritmo, mentre nella figura 6.2j, in rosso, il piano reale. In generale gli algoritmi si sono comportati piuttosto bene nell'individuazione del piano relativo a questa immagine, infatti quasi tutti i piani trovati si discostano da quello reale di pochi gradi, ma probabilmente questa differenza deriva da una bassa approssimazione per lo spessore del piano.

Rispetto agli altri algoritmi, le varianti di Hough non sono riuscite ad individuare il piano oltre ad una certa distanza.

Tutti i dati in output sono visibili nella tabella 6.2, ma è possibile riassumere i risultati nei seguenti punti:



(a) *Standard RANSAC.* (b) *Fast RANSAC.*

(c) *LS.* (d) *Optimized RANSAC.*

(e) *Hough.* (f) *Hough Randomized.*

(g) *Hough Randomized ver.2.* (h) *Normals.*

(i) *Region Growing.* (j) *Immagine di ground truth.*

Figura 6.2: Piani trovati per l'immagine 000000_10.

Tabella 6.2: Risultati per l'immagine 000000_10.

(a) *Coefficienti dei piani.*

Algoritmi	Parametri Piano				Tempo	$\cos\theta$
	A	B	C	D	(ms)	
Standard RANSAC	$-8{,}4214$	$-373{,}746$	$-4{,}0343$	564,082	170	0,9794
Fast RANSAC	$-0{,}4941$	$-20{,}0783$	$-0{,}1357$	31,0107	15	0,9798
LS	$-0{,}4631$	$-35{,}1491$	$-0{,}4486$	56,5544	1338	0,9774
Optimized RANSAC	$-1{,}3793$	$-64{,}5862$	$-0{,}9738$	101,057	64	0,9791
Hough	$4{,}3711 \times 10^{-8}$	1	$-4{,}3711 \times 10^{-8}$	1,53	35 882	0,9747
Randomized Hough	$4{,}3711 \times 10^{-8}$	1	$-4{,}3711 \times 10^{-8}$	$-1{,}53$	1633	0,9747
Randomized Hough V.2	$4{,}3711 \times 10^{-8}$	1	$-4{,}3711 \times 10^{-8}$	$-1{,}53$	692	0,9747
Normals	$-0{,}6619$	$-25{,}5306$	$-0{,}4952$	43,5567	132	0,9799
Region Growing	0,0451	2,3072	0,0288	$-3{,}7346$	1060	0,9788
Piano Reale	0,0193	6,0183	0,0840	$-9{,}9901$	ND	1

(b) *Punti dei piani.*

Algoritmi	Punti Piano	Veri Positivi	Veri Negativi	Falsi Positivi	Falsi Negativi
Standard RANSAC	100 454	43 103	354 377	57 351	386
Fast RANSAC	89 654	43 035	365 109	46 619	454
LS	93 160	43 489	362 057	49 671	0
Optimized RANSAC	102 135	43 489	353 082	58 646	0
Hough	74 903	39 723	376 548	35 180	3766
Randomized Hough	74 903	39 723	376 548	35 180	3766
Randomized Hough V.2	74 903	39 723	376 548	35 180	3766
Normals	93 915	43 449	361 262	50 466	40
Region Growing	91 034	43 489	364 183	47 545	0

(c) *Indici.*

Algoritmi	Indice di Precisione	Indice di Richiamo
Standard RANSAC	0,429 082	0,991 124
Fast RANSAC	0,480 012	0,989 561
LS	0,466 821	1
Optimized RANSAC	0,425 799	1
Hough	0,530 326	0,913 403
Randomized Hough	0,530 326	0,913 403
Randomized Hough V.2	0,530 326	0,913 403
Normals	0,462 642	0,999 08
Region Growing	0,477 723	1

- L'algoritmo più veloce come auspicabile è Fast RANSAC, ma dato che ritorna il primo piano valido come risultato, ci sono voluti più tentativi per ottenere quello corretto.

- Da notare la differenza di velocità, nonostante il risultato sia lo stesso, fra le varianti di Hough, infatti utilizzando gli stessi parametri abbiamo una differenza di 34 s nel tempo di esecuzione fra quella standard e la versione uno della random; mentre uno scarto di circa 1 s fra le due randomized.

- Le normali dei piani trovati sono quasi parallele a quella del piano reale, infatti $\cos\theta$ vale in media 0,97. Il risultato meno preciso è dato da Hough in tutte le sue forme, mentre il metodo delle Normali ottiene il risultato migliore con un coseno di 0,9799.

- Il valore dei falsi negativi è sempre molto basso, questo significa che quasi tutti i punti del piano reale sono stati trovati dagli algoritmi.

Anche se l'indice di precisione è mediamente di 0.5, questo valore non deve trarre in inganno, infatti basti vedere che i piani trovati si estendono anche al di fuori del manto stradale, per lo stesso motivo spiegato prima.

Immagine 000005_10 Come seconda immagine per i test, si è optato per una che presenti un piano molto ravvicinato, visibile in figura 6.3. Anche qui la piccola differenza di spessore fra strada e marciapiede, porta gli algoritmi a vedere i punti delle due aree come appartenenti ad un'unico piano.

Figura 6.3: Immagine 000005_10

I parametri sono posti, come per l'immagine di prima, nell'apposita tabella (6.3). Questa volta si è provato ad abbassare il valore dell'accuratezza di Hough, parametro per lo spessore del piano cercato, oltre che aumentare la discretizzazione degli angoli, nel tentativo di ottenere solo la strada (cosa che

non si è rivelata vera). Per gli altri algoritmi la situazione cambia abbastanza poco; RANSAC utilizza più o meno gli stessi parametri, e la variante Fast ha sempre bisogno di più tentativi per trovare un ottimo piano.

Tabella 6.3: Parametri per l'immagine 000005_10.

(a) *Parametri RANSAC.*

Soglia	Iter.	Subset	Prof.
30	1000	0,35	0,10

(b) *Parametri Normals.*

ε	β	Scala	Discr.	Coseno	Prof.
0,03	25	16	0,03	0,9999	0,10

(c) *Parametri Hough.*

Soglia	Acc.	Ang.	N Piani	Prec. (R)	Prec. (C)
10 000	4,00	2,0	25	PRE_99	CENT

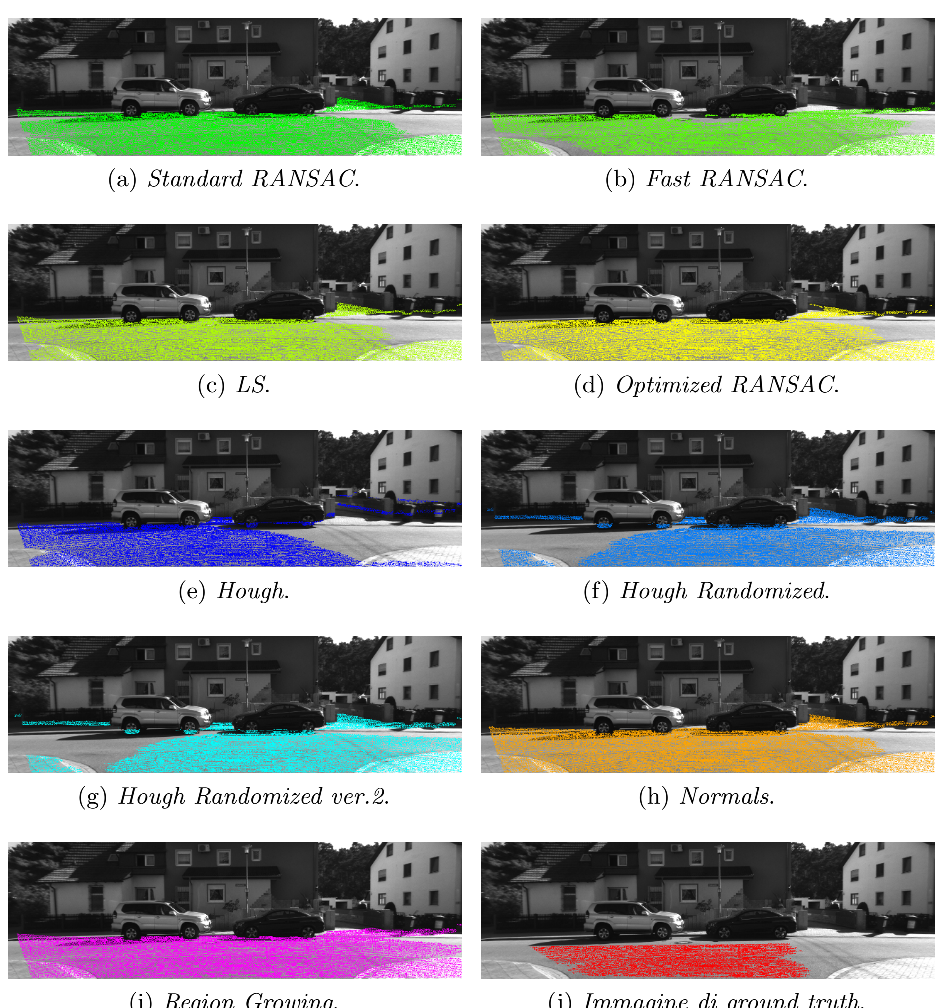
(d) *Parametri Region Growing.*

δ	γ	ϵ	Dist.	Est.	Ris.	$\bigcup$	Punti
0,06	1,335	1,750	10	5	3	0,20	4000

Come per la prima immagine gli algoritmi hanno trovato con buona approssimazione il piano richiesto, ma anche qui i piani presenti nella figura 6.4 mostrano che sono stati presi in considerazione più punti del dovuto, molti dei quali fanno parte o dei due marciapiedi, rispettivamente nell'angolo in basso a destra ed in basso a sinistra, oppure del marciapiede che si trova dietro le macchine. In alcuni casi anche una piccola porzione delle ruote delle macchine viene vista come appartenente al piano, questo comportamento è probabilmente dato da una bassa approssimazione dello spessore del piano; cosa che però si richiede necessaria al fine di ottenere un insieme di punti abbastanza popolato.

Il seguente elenco riporta brevemente le conclusioni relative ai test di questa immagine in particolare:

- Come nel primo caso Fast RANSAC si propone come l'algoritmo più veloce, ma questa volta lo scarto di tempo rispetto alle altre implementazioni di RANSAC è abbastanza basso da essere trascurabile.



(a) *Standard RANSAC.*
(b) *Fast RANSAC.*
(c) *LS.*
(d) *Optimized RANSAC.*
(e) *Hough.*
(f) *Hough Randomized.*
(g) *Hough Randomized ver.2.*
(h) *Normals.*
(i) *Region Growing.*
(j) *Immagine di ground truth.*

Figura 6.4: Piani trovati per l'immagine 000005_10.

- Anche qui la funzione standard di Hough ottiene un tempo troppo alto per essere utilizzata in una situazione reale; mentre le versioni randomized ottengono dei tempi, sicuramente più bassi, ma ancora troppo distanti dagli altri algoritmi.
- Nonostante standard Hough sia l'algoritmo più lento, ha il miglior valore per il $\cos\theta$.
- Tutti gli algoritmi ottengono una bassa precisione del piano a livello di numero di punti, questo per la motivazione discussa prima.
- Il Region Growing ha individuato il piano con il maggior numero di punti.
- Anche se all'apparenza i piani trovati sembrano paralleli a quello reale, il calcolo del $\cos\theta$ indica un differenza di inclinazione non troppo bassa; per questo parametro di valutazione il risultato peggiore è stato ottenuto dalle varianti randomized di Hough.

Nella tabella 6.4 è riportato l'elenco completo dei risultati per l'immagine 000005_10.

Immagine 000016_10 Il piano da individuare nella terza immagine è una strada, non troppo larga, con ai lati una serie di macchine parcheggiate, come è possibile vedere in figura 6.5. Trattasi di uno scenario in cui il piano va individuato con molta accuratezza; infatti per permettere ad un veicolo autonomo di muoversi all'interno di un ambiente come questo gli ostacoli vanno tracciati con un'alta precisione.

Figura 6.5: Immagine 000016_10

Come è possibile vedere dalla tabella 6.5 non sono stati prestati particolari accorgimenti per i parametri di RANSAC e Hough, mentre per le Normals e Region Growing occorre sempre fare attenzione alla situazione che si presenta.

Tabella 6.4: Risultati per l'immagine 000005_10.

(a) *Coefficienti dei piani.*

Algoritmi	Parametri Piano				Tempo	$\cos\theta$
	A	B	C	D	(ms)	
Standard RANSAC	0,5130	33,5916	0,7402	−55,6017	45	0,7111
Fast RANSAC	0,9370	78,7337	0,4869	−120,346	25	0,7089
LS	−0,5457	−30,2021	−0,6443	49,8589	382	0,7131
Optimized RANSAC	1,0006	56,1705	0,9711	−91,8215	33	0,7129
Hough	0,0348	0,9987	0,0348	−1,71	56 781	0,7245
Randomized Hough	$4{,}3684 \times 10^{-8}$	0,9993	0,0348	−1,72	3244	0,7000
Randomized Hough V.2	$4{,}3684 \times 10^{-8}$	0,9993	0,0348	−1,72	1257	0,7000
Normals	−0,2488	−13,9094	−0,2399	22,2135	94	0,7130
Region Growing	1,0234	71,3497	0,9270	−114,253	160	0,7106
Piano Reale	1,2979	61,637	1,2740	−100,53	ND	1

(b) *Punti dei piani.*

Algoritmi	Punti Piano	Veri Positivi	Veri Negativi	Falsi Positivi	Falsi Negativi
Standard RANSAC	67 077	32 628	388 140	34 449	0
Fast RANSAC	63 376	32 062	391 275	31 314	566
LS	65 674	32 628	389 543	33 046	0
Optimized RANSAC	66 496	32 628	388 721	33 868	0
Hough	53 923	31 523	400 189	22 400	1105
Randomized Hough	57 915	27 540	392 214	30 375	5088
Randomized Hough V.2	57 915	27 540	392 214	30 375	5088
Normals	70 636	32 628	384 581	38 008	0
Region Growing	72 848	32 628	382 369	40 220	0

(c) *Indici.*

Algoritmi	Indice di Precisione	Indice di Richiamo
Standard RANSAC	0,486 426	1
Fast RANSAC	0,505 901	0,982 653
LS	0,496 818	1
Optimized RANSAC	0,490 676	1
Hough	0,584 593	0,966 133
Randomized Hough	0,475 524	0,844 06
Randomized Hough V.2	0,475 524	0,844 06
Normals	0,461 917	0,996 939
Region Growing	0,447 892	1

Tabella 6.5: Parametri per l'immagine 000016_10.

(a) *Parametri RANSAC.*

Soglia	Iter.	Subset	Prof.
30	1000	0,10	0,15

(b) *Parametri Normals.*

ε	β	Scala	Discr.	Coseno	Prof.
0,01	20	10	0,01	0,999	0,15

(c) *Parametri Hough.*

Soglia	Acc.	Ang.	N Piani	Prec. (R)	Prec. (C)
10 000	6,00	3,0	25	PRE_99	CENT

(d) *Parametri Region Growing.*

δ	γ	ϵ	Dist.	Est.	Ris.	$\bigcup$	Punti
0,05	0,735	1,550	22	3	5	0,20	4000

Nelle immagini di figura 6.6 sono stati riportati tutti i piani identificati dagli algoritmi.

Alcune osservazioni sono riportate nel seguente elenco:

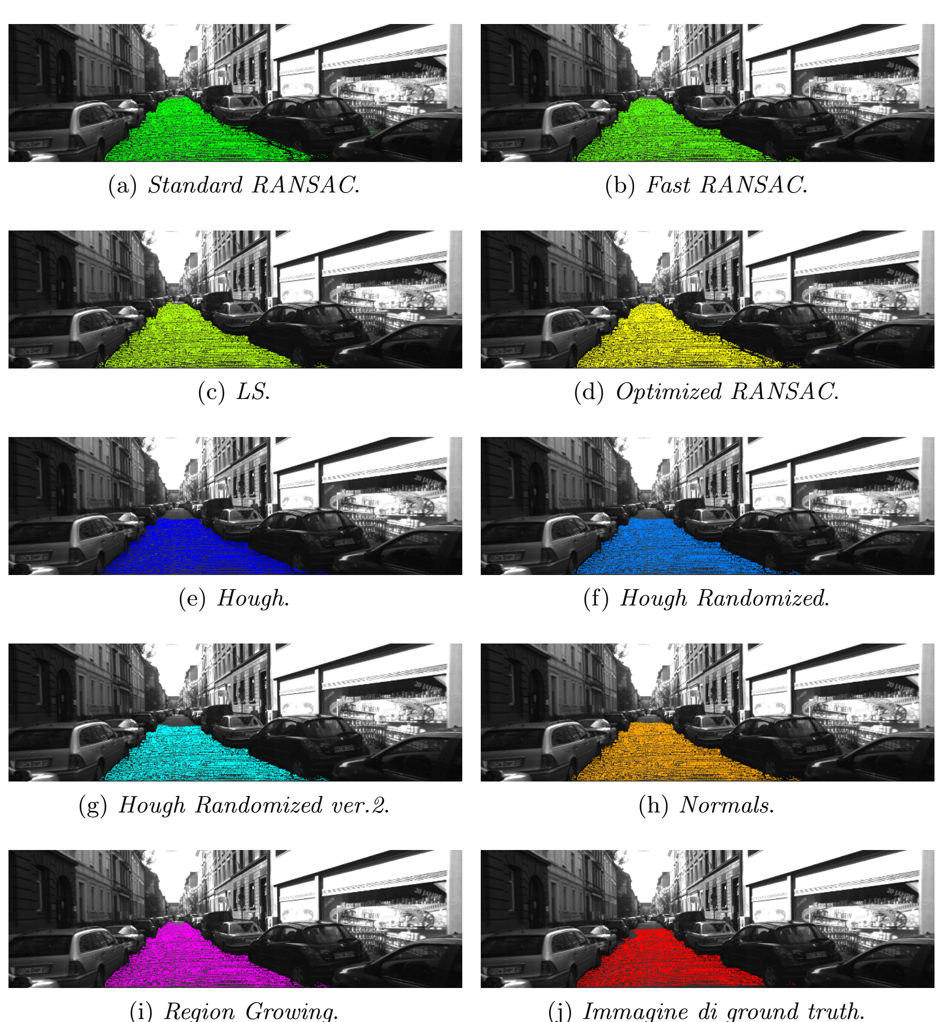
- Tutti gli algoritmi si sono comportati in maniera egregia, infatti i piani trovati godono di un'altissima precisione sia in termini di punti che di gradi di inclinazione.

- Come sempre la funzione standard di Hough ottiene un tempo troppo alto per un utilizzo real time.

- Per quanto riguarda i tempi di esecuzione, tutti gli algoritmi di RANSAC si sono rivelati estremamente veloci, addirittura la variante LS risulta più veloce di quella standard.

- Il miglior $\cos\theta$ è dato dall'algoritmo basato su standard RANSAC, mentre quello peggiore dalla versione LS.

Immagine 000036_10 L'ultima figura testata in questa lavoro rappresenta uno scenario in cui è molto difficile ottenere un buon piano risultate.



(a) *Standard RANSAC.* (b) *Fast RANSAC.*

(c) *LS.* (d) *Optimized RANSAC.*

(e) *Hough.* (f) *Hough Randomized.*

(g) *Hough Randomized ver.2.* (h) *Normals.*

(i) *Region Growing.* (j) *Immagine di ground truth.*

Figura 6.6: Piani trovati per l'immagine 000016_10.

Tabella 6.6: Risultati per l'immagine 000016_10.

(a) *Coefficienti dei piani.*

Algoritmi	Parametri Piano				Tempo	$\cos\theta$
	A	B	C	D	(ms)	
Standard RANSAC	3,1385	77,938	1,2685	−124,645	22	0,9817
Fast RANSAC	0,0186	−1,9389	−0,0104	3,1320	9	0,9711
LS	−0,0657	2,9412	0,0140	−4,743	17	0,9681
Optimized RANSAC	−0,4085	−12,0649	−0,1882	19,4709	12	0,9805
Hough	$4{,}3711 \times 10^{-8}$	1	$-4{,}3711 \times 10^{-8}$	−1,56	21 853	0,9734
Randomized Hough	$4{,}3711 \times 10^{-8}$	1	$-4{,}3711 \times 10^{-8}$	−1,56	1046	0,9734
Randomized Hough V.2	$4{,}3711 \times 10^{-8}$	1	$-4{,}3711 \times 10^{-8}$	−1,57	1507	0,9734
Normals	−0,0972	19,7502	0,0277	−30,8394	114	0,9723
Region Growing	−0,0613	−15,9069	−0,1884	26,663	168	0,9742
Piano Reale	0,0192	7,8061	0,0820	−12,8844	ND	1

(b) *Punti dei piani.*

Algoritmi	Punti Piano	Veri Positivi	Veri Negativi	Falsi Positivi	Falsi Negativi
Standard RANSAC	32 681	28 908	435 382	3773	171
Fast RANSAC	31 693	29 079	436 541	2614	0
LS	31 986	29 079	436 248	2907	0
Optimized RANSAC	32 708	29 079	435 526	3629	0
Hough	30 298	28 424	437 281	1874	655
Randomized Hough	30 298	28 424	437 281	1874	655
Randomized Hough V.2	30 045	28 253	437 363	1792	826
Normals	31 054	28 990	437 091	2064	89
Region Growing	31 895	29 079	436 339	2816	0

(c) *Indici.*

Algoritmi	Indice di Precisione	Indice di Richiamo
Standard RANSAC	0,884 551	0,994 119
Fast RANSAC	0,917 521	1
LS	0,909 116	1
Optimized RANSAC	0,889 049	1
Hough	0,938 148	0,977 475
Randomized Hough	0,938 148	0,977 475
Randomized Hough V.2	0,940 356	0,971 595
Normals	0,933 535	0,996 939
Region Growing	0,911 71	1

Come si può vedere nell'immagine 6.7 la superficie da trovare è una strada di campagna molto stretta e anche poco definita.

Figura 6.7: Immagine 000036_10

Tabella 6.7: Parametri per l'immagine 000016_10.

(a) *Parametri RANSAC.*

Soglia	Iter.	Subset	Prof.
30	1000	0,20	0,2

(b) *Parametri Normals.*

ε	β	Scala	Discr.	Coseno	Prof.
0,06	28	12	0,03	0,9999	0,2

(c) *Parametri Hough.*

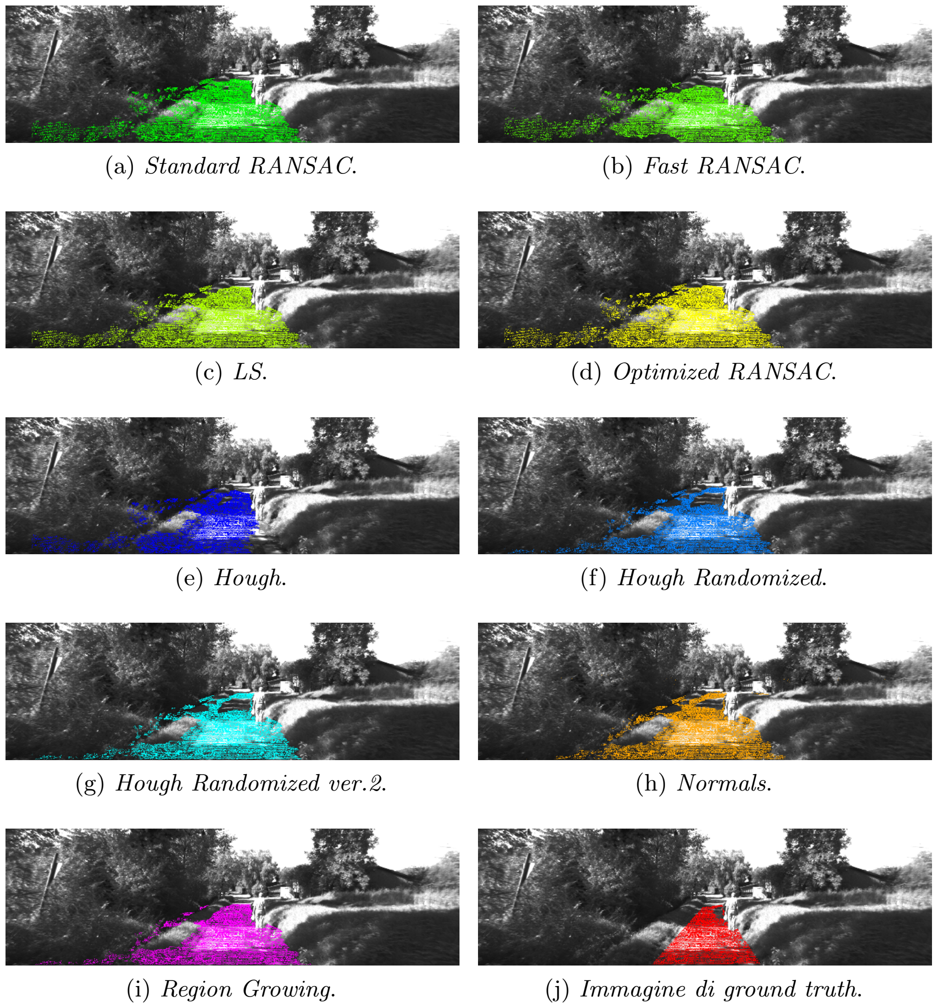
Soglia	Acc.	Ang.	N Piani	Prec. (R)	Prec. (C)
10 000	10,00	1,0	15	PRE_99	CENT

(d) *Parametri Region Growing.*

δ	γ	ϵ	Dist.	Est.	Ris.	$\bigcup$	Punti
0,02	0,735	1,350	12	6	3	0,20	4000

Le possibili osservazioni per questa immagine sono:

- Per ottenere un numero sufficiente di punti si è utilizzato un valore abbastanza alto per lo spessore; cosa che si traduce in piani non perfettamente centrati con la strada.
- Anche questa volta gli indici di precisione risultano bassi, questo dovuto alla particolare posizione del piano.



(a) *Standard RANSAC.* (b) *Fast RANSAC.*

(c) *LS.* (d) *Optimized RANSAC.*

(e) *Hough.* (f) *Hough Randomized.*

(g) *Hough Randomized ver.2.* (h) *Normals.*

(i) *Region Growing.* (j) *Immagine di ground truth.*

Figura 6.8: Piani trovati per l'immagine 000036_10.

- Come sempre i risultati migliori sui i tempi di esecuzione, sono dati dagli algoritmi RANSAC, di cui Fast è il più veloce.
- Il miglior $\cos\theta$ è dato dall'algoritmo basato su standard Hough, mentre quello peggiore dal metodo delle Normali.
- Per un buon risultato con standard Hough si è mostrato necessario utilizzare una discretizzazione dell'angolo di un solo grado, questo fatto si traduce in un altissimo tempo di elaborazione.

Tabella 6.8: Risultati per l'immagine 000036_10.

(a) *Coefficienti dei piani.*

Algoritmi	Parametri Piano				Tempo	$\cos\theta$
	A	B	C	D	(ms)	
Standard RANSAC	0,4169	−5,0329	0,0303	7,8575	30	0,8405
Fast RANSAC	0,1764	−1,9001	0,0439	2,6353	14	0,8456
LS	4,7052	−70,6453	0,6929	109,179	82	0,8316
Optimized RANSAC	0,2032	−2,5476	0,0187	3,9641	28	0,8388
Hough	−0,1218	0,9925	$-4{,}3711 \times 10^{-8}$	−1,72	603 459	0,8612
Randomized Hough	−0,0349	0,9994	$-4{,}3711 \times 10^{-8}$	−1,66	7309	0,8137
Randomized Hough V.2	−0,0349	0,9994	$-4{,}3711 \times 10^{-8}$	−1,66	1801	0,8137
Normals	10 345,5	$-3{,}2612 \times 10^{7}$	8822,29	$5{,}4513 \times 10^{7}$	135	0,7931
Region Growing	−0,1108	4,113	−0,0407	−6,5139	186	0,8090
Piano Reale	−0,0124	−1,4561	0,0162	2,315	ND	1

(b) *Punti dei piani.*

Algoritmi	Punti Piano	Veri Positivi	Veri Negativi	Falsi Positivi	Falsi Negativi
Standard RANSAC	32 787	15 691	434 555	17 096	26
Fast RANSAC	29 563	14 737	436 825	14 826	980
LS	31 675	15 717	435 693	15 958	0
Optimized RANSAC	32 657	15 717	434 711	16 940	0
Hough	24 553	12 040	439 138	12 513	3677
Randomized Hough	26 531	14 694	439 814	11 837	1023
Randomized Hough V.2	26 531	14 694	439 814	11 837	1023
Normals	27 809	15 127	438 969	12 682	590
Region Growing	26 399	15 717	440 969	10 682	0

(c) *Indici.*

Algoritmi	Indice di Precisione	Indice di Richiamo
Standard RANSAC	0,478 574	0,998 346
Fast RANSAC	0,498 495	0,937 647
LS	0,496 196	1
Optimized RANSAC	0,481 275	1
Hough	0,490 368	0,766 05
Randomized Hough	0,553 843	0,934 911
Randomized Hough V.2	0,553 843	0,934 911
Normals	0,543 961	0,962 461
Region Growing	0,595 363	1

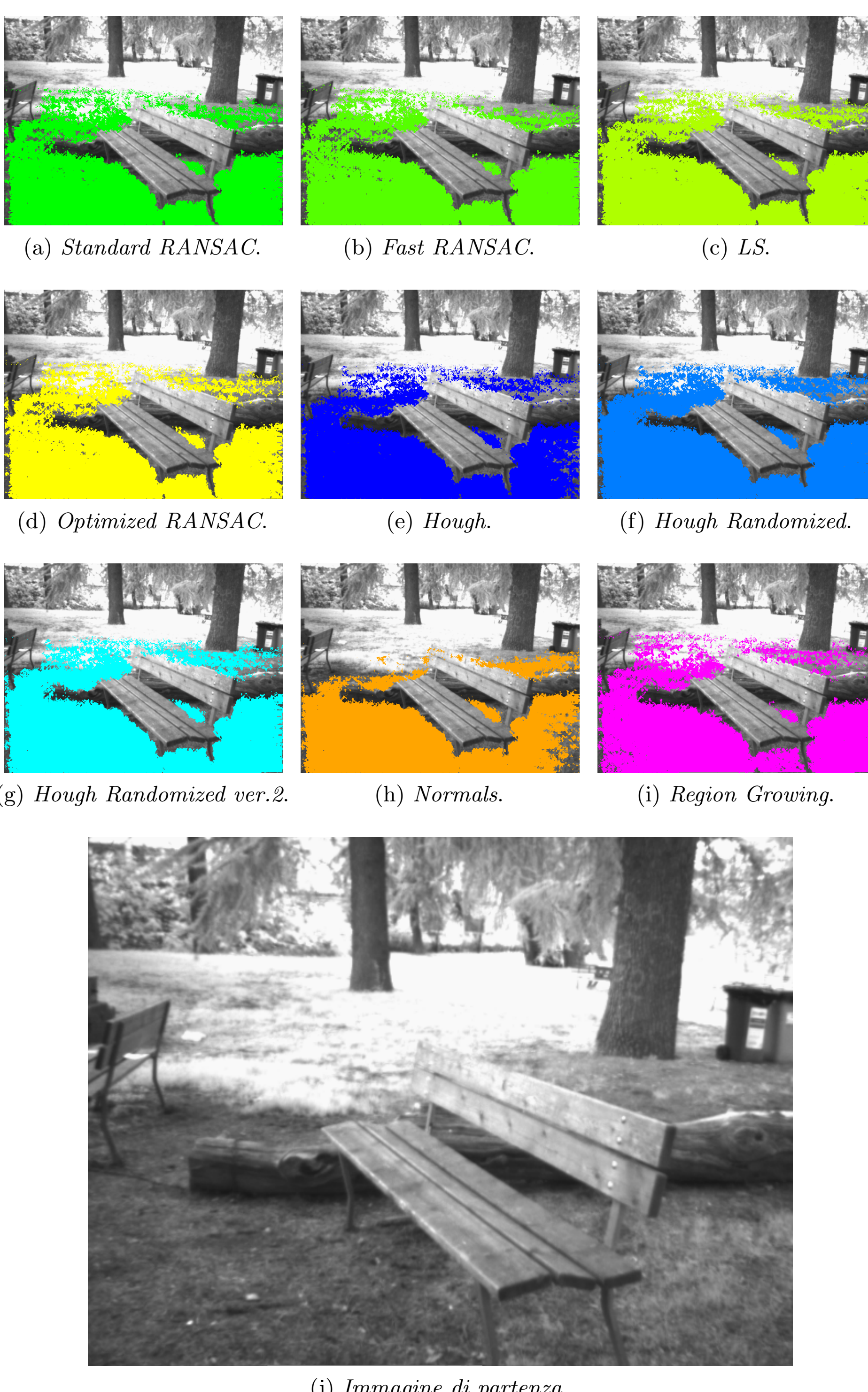
6.3 Stereo Camera dataset

Come nella prima parte della tesi i risultati ottenuti dai test con la camera stereo possono essere valutati solo a carattere qualitativo, non essendo questi set di immagini dotati di informazioni sui piani presenti, inoltre la prova si limiterà ad una sola immagine in quanto i dati già raccolti risultano sufficienti per un confronto fra gli algoritmi.

Panchina e tronco Per questa immagine verranno riportati solo i parametri in ingresso per i singoli algoritmi (tabella 6.9), ed i tempi di esecuzione dei singoli algoritmi:

- Standard RANSAC: 131 ms;
- Fast RANSAC: 15 ms;
- LS: 1461 ms;
- Optimized RANSAC: 72 ms;
- Standard Hough: 75 598 ms;
- Hough Randomized ver.1: 2264 ms;
- Hough Randomized ver.2: 808 ms;
- Normals: 114 ms;
- Region Growing: 179 ms.

Nella figura 6.9 sono visibili i piani individuati tramite la libreria di funzioni. Tutti gli algoritmi, tranne la versione standard di Hough, sono stati in grado di trovare il piano al primo tentativo, eliminando dall'equazione i due ostacoli principali dell'immagine. Occorre precisare che



(a) *Standard RANSAC.* (b) *Fast RANSAC.* (c) *LS.*

(d) *Optimized RANSAC.* (e) *Hough.* (f) *Hough Randomized.*

(g) *Hough Randomized ver.2.* (h) *Normals.* (i) *Region Growing.*

(j) *Immagine di partenza.*

Figura 6.9: Piani trovati per l'immagine con la panchina e il tronco.

Tabella 6.9: Parametri per l'immagine 000005_10.

(a) *Parametri RANSAC.*

Soglia	Iter.	Subset	Prof.
30	1000	0,10	0,10

(b) *Parametri Normals.*

ε	β	Scala	Discr.	Coseno	Prof.
0,05	36	18	0,01	0,9999	0,10

(c) *Parametri Hough.*

Soglia	Acc.	Ang.	N Piani	Prec. (R)	Prec. (C)
30 000	4,00	2,0	15	PRE_99	CENT

(d) *Parametri Region Growing.*

δ	γ	ϵ	Dist.	Est.	Ris.	$\bigcup$	Punti
0,03	1,735	0,750	10	6	6	0,20	4000

Capitolo 7

Conclusioni sugli algoritmi e su Stereo SLAM

Nel capitolo conclusivo di questo lavoro di tesi occorre fare un breve resoconto sui due problemi, relativi alla *Computer Vision*, affrontati nelle pagine precedenti.

Stereo SLAM Si tratta sicuramente di un buon sistema di SLAM, ma non completamente privo di difetti. Nonostante i buoni tempi di elaborazione sui singoli frame, nel momento in cui viene disattivata la parte di backend della pipeline, il sistema soffre di problemi dovuti al drift delle pose, problemi che possono essere risolti con un miglioramento della stima per la trasformazione 3D. Comunque, come mostrano i risultati sperimentali del capitolo 3, l'attivazione del bundle adjustment e della loop closure incrementa solo lievemente il tempo per la creazione delle mappe 3D, ma ne fornisce una versione molto più precisa.

Plane Detection Delle quattro implementazioni di RANSAC quella che si distingue di più per qualità dei risultati e velocità di esecuzione è sicuramente l'Optimized RANSAC, in ogni caso si tratta della famiglia di algoritmi che completano l'individuazione del piano nel minor tempo in quasi tutte le immagini, forse solo la versione LS ne risente nei tempi quando il piano cercato è di grandi dimensioni.

L'algoritmo basato su trasformata di Hough nella sua versione standard è molto distante dalla possibilità di essere utilizzato in situazioni reali; questo perchè il tempo che impiega per trovare un piano con la dovuta precisione

si aggira in media nella trentina di secondi. Al contrario le due versioni randomized, ed in particolare la seconda versione, si sono rivelate molto utili in alcuni casi, dato che non occorre portare particolare attenzione ai parametri utilizzati.

Anche se basati su concetti molto diversi, gli algoritmi basati sulle Normai e quello sul Region Growing godono delle stesse osservazioni; ovvero si tratta di metodi molto precisi e allo stesso tempo veloci, ma sono fortemente dipendenti dai parametri con i quali vengono settati. A volte Prima di ottenere un risultato ottimale è necessario un controllo sulle caratteristiche della scena in cui è presente il piano da identificare.

Sviluppi futuri Grazie al tool di testing visto nel capitolo 5 è possibile eseguire altri test in modo semplice e rapido anche su un numero elevato di immagini. Si potrebbe pensare di estendere il programma per la gestione di altri dataset, magari acquisiti da tipo di sensori diversi da quello LIDAR, come ad esempio il *Microsoft Kinect* o un sistema stereo come quello della sezione 2.2, e testare in modo più approfondito la bontà degli algoritmi della libreria.

Appendice A

Qt Framework

Qt è un framework multi-piattaforma adibito allo sviluppo di applicazioni desktop, embedded o mobile. Le piattaforme supportate sono molteplici, e vanno da sistemi desktop come Windows, Linux o OS X, fino a stitemi mobile come possono essere Android, iOS e BlackBerry.

Il framework è completamente scritto in C++, quindi non è costruito sopra un proprio linguaggio, ma estende il C++ tramite un preprocessore chiamato Meta-Object Compiler (MOC). Questo sistema aggiunge un sistema chiamato *signal and slots*, approfondito nella sezione A.2

Prima della compilazione vera e propria, il MOC analizza il codice sorgente scritto in C++ esteso da Qt e genera dei file in C++ standard. In questo le applicazioni create con Qt possono essere compilate da qualsiasi compilatore standard, come *gcc* o *msvc*. Anche se è possibile utilizzare qualsiasi sistema di compilazione con Qt, quest'ultimo viene distribuito con il proprio sistema, chiamato *qmake* (sezione A.1)

Oltre al sistema di compilazione Qt possiede anche il proprio Integrated Development Environment (IDE), chiamato *Qt Creator*. Grazie a questo programma e Qt è possibile creare GUI direttamente in C++, utilizzando il modulo dei Widgets. Sempre all'interno del programma esiste un tool chiamato *Qt Designer*, con lo scopo di aiutare l'utente nella creazione delle interfacce; infatti si tratta di uno strumento visuale che permette di aggiungere, con un semplice *drag and drop*, i vari elementi alla la finestra, e di modificarne tutte le proprietà principali.

Qt non è solo un insieme di strumenti per creare GUI, ma prevede anche dei moduli per la gestione della rete, di database, tecnologie web, sensori, protocolli di comunicazione, e molto altro.

A.1 Qmake

Qmake è uno strumento che semplifica il processo di compilazione di progetti multi-piattaforma, in pratica svolge la stessa funzione di altri sistemi come ad esempio CMake (appendice B). Qmake automatizza la creazione dei file utilizzati da make in ambiente Unix, ma può anche essere utilizzato in progetti fatti con Visual Studio.

Codice A.1: Esempio di progetto Qt.

```
#-------------------------------------------------
#
# Project created by QtCreator 2016-02-11T13:44:49
#
#-------------------------------------------------

QT       += core gui

CONFIG += c++11

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = PlaneDetection
TEMPLATE = app

INCLUDEPATH += /usr/local/include/opencv

LIBS += -L/usr/local/lib -lopencv_highgui -
    lopencv_core -lopencv_imgproc -lopencv_contrib

SOURCES += main.cpp\
        mainwindow.cpp \
    lib_plane_detection.cpp \
    lib_pointcloud.cpp \
    lib_plane_regions.cpp \
    kitti.cpp

HEADERS  += mainwindow.h \
    lib_plane_detection.h \
    lib_pointcloud.h \
    kitti.h

FORMS    += mainwindow.ui

RESOURCES += \
    images.qrc

DISTFILES += \
    CMakeLists.txt
```

Nel codice A.1 viene riportato il codice del file processato per la creazione del programma visto nella sezione 5.1.

A.2 Programma d'Esempio

Per spiegare in poche parole le basi di Qt e del suo sistema di signal and slot, verrà fatto uso di un piccolo programma di esempio.

Il concetto di signal and slot è allo stesso tempo semplice quanto potente. Questi due elementi vengono utilizzati per la comunicazione fra gli oggetti che compongono le finestre del programma. Spesso nei programmi dotati di interfaccia grafica, si vuole che il cambiamento di stato di un determinato widget si rifletta in una notifica ad un'altro widget. L'azione di avvisare un secondo elemento del proprio cambiamento può portare all'esecuzione di diverse funzioni.

Per prima cosa occorre costruire una finestra; in questo programma d'esempio si è scelto di inserire due bottoni, uno slider orizzontale, e uno spinbox in sola lettura.

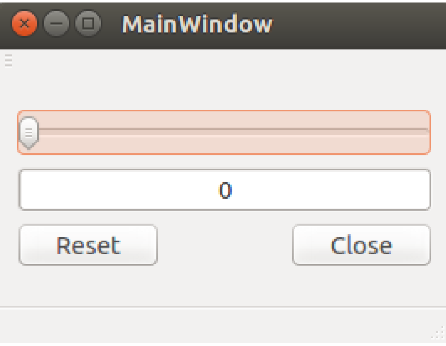


Figura A.1: Programma d'esempio.

Il main nel codice A.2 mostra come in poche righe di codice sia possibile mostrare a schermo una finestra. Quest'ultima è un'istanza delle classe `MainWindow` visibile nel codice A.3.

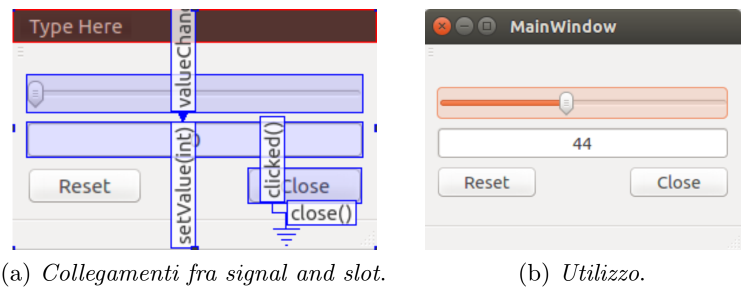
Codice A.2: Esempio di progetto Qt.

```
1 #include "mainwindow.h"
2 #include <QApplication>
3
4 int main(int argc, char *argv[])
5 {
6     QApplication a(argc, argv);
```

```
7       MainWindow w;
8       w.show();
9
10      return a.exec();
11  }
```

Tornando al concetto di signal and slot, verranno spiegati due metodi per utilizzare questo potente meccanismo. Il primo avviene per via grafica; grazie all'editor *Qt Creator* è possibile collegare due oggetti della finestra fra loro e decidere sia l'azione che scatena il segnale, sia la reazione del widget che lo riceve. La figura A.2a mostra due di questi collegamenti visuali.

Il secondo metodo è possibile grazie alla funzione `connect()`, di cui un possibile utilizzo è riportato nella figura A.3, la quale prende come primo argomento il widget che invia il segnale, che a sua volta viene passato come secondo argomento; il terzo e il quarto parametro sono rispettivamente l'oggetto ricevente e la funzione da eseguire.



(a) *Collegamenti fra signal and slot.* (b) *Utilizzo.*

Figura A.2: Signal and Slot.

Codice A.3: Esempio di progetto Qt.

```
1   #include "mainwindow.h"
2   #include "ui_mainwindow.h"
3
4   MainWindow::MainWindow(QWidget *parent) :
5       QMainWindow(parent),
6       ui(new Ui::MainWindow)
7   {
8       ui->setupUi(this);
9
10      connect(ui->pushButton_2, SIGNAL(clicked(bool)),
            this, SLOT(on_resetButton_clicked()));
11  }
12
13  MainWindow::~MainWindow()
14  {
15      delete ui;
```

```
16 }
17
18 void MainWindow::on_resetButton_clicked()
19 {
20     ui->horizontalSlider->setValue(0);
21 }
```

Il framework Qt non si ferma alle sole funzioni mostrate nelle righe precedenti, ma si rimanda alla documentazione ufficiale per gli approfondimenti del caso.

Appendice B

CMake

CMake è un sistema open-source con la possibilità di gestire il processo di compilazione indipendentemente dal sistema operativi nel quale lo si sta eseguendo. CMake è stato pensato per essere eseguito insieme al compilatore nativo del sistema. Un progetto CMake va creato tramite un file chiamato *CMakeLists.txt*, il discorso su tale file verrà approfondito nella sezione B.1.

CMake può generare sistemi di compilazione atti alla creazione di librerie o eseguibili in modo arbitrario; supporta anche multiple build a partire da un singolo albero delle risorse. Tornando alla compilazione di librerie, CMake supporta sia quelle statiche che quelle dinamiche.

Un'altra importante caratteristica di questo sistema è quella di creare, alla prima compilazione, un file cache che può essere utilizzato via editor grafico. Questo file contiene le configurazioni utili alla compilazione, che quindi possono essere cambiate per ottenere diversi tipi di risultati.

B.1 Esempio di file CMakeLists.txt

Per comprendere meglio il concetto di un progetto CMake, verrà fatto uso di un esempio di file CMakeLists.txt (codice B.1). Come accennato in precedenza questo file è il cuore pulsante di un progetto CMake, infatti contiene tutte le informazioni necessarie per creare una soluzione, specifica per ogni sistema, che successivamente potrà essere compilata tramite il compilatore nativo del sistema.

Il codice B.1 comprende poche ma indispensabili funzioni per un progetto CMake. La direttiva più importante è sicuramente `project(project_name)` che definisce il nome del progetto. La funzione `cmake_minimun_required(VERSION`

`2.8)` impone un requisito sulla versione minima di CMake che è possibile utilizzare; altre funzioni importanti sono: la `add_executable(executable_name surce_code.cxx surce_code2.cxx)`, che aggiunge un eseguibile da compilare dai file sorgente indicati nell'invocazione; la funzione `include_directories(directory)` che aggiunge la cartella indicata a quelle in cui il sistema cerca i file sorgente; `target_link_libraries(executable_name lib1 lib2)` che collega all'eseguibile passato come argomento le librerie *lib1* e *lib2*.

Una caratteristica molto utile di CMake è quella di creare delle *macro* per utilizzare librerie esterne tramite la funzione `find_package(package_name)`. Per utilizzare questo funzione, all'interno del sistema, deve essere presente un file che indichi a CMake la posizione della libreria nel file system. La clausola *REQUIRED* indica che l'utilizzo della libreria in questione è indispensabile per la corretta compilazione del progetto. É possibile vedere l'utilizzo di una macro risultate dall'utilizzo di questo funzione all'interno della funzione alla riga 11 del codice B.1.

Codice B.1: Esempio di CMakeLists.txt

```
1  cmake_minimum_required(VERSION 2.8)
2
3  project(planedetection)
4
5  include_directories("${PROJECT_BINARY_DIR}")
6
7  find_package(OpenCV REQUIRED)
8
9  add_executable(planedetection main.cpp
       lib_plane_detection.cpp lib_plane_regions.cpp
       lib_pointcloud.cpp lib_plane_detection.h
       lib_pointcloud.h)
10
11 target_link_libraries( planedetection ${OpenCV_LIBS})
```

Prendendo in esempio il sistema operativo *Ubunutu*, utilizzato in questo lavoro di tesi, il quale incorpora l'utility *make*[1]; il risultato finale dato dell'esecuzione di un progetto CMake è un file detto *makefile* che può essere processato appunto da make.

Nel caso di Windows la scelta più ovvia ricadrebbe nella creazione di una soluzione per Visual Studio.

[1] L'utility make è usata soprattutto per la compilazione di codice sorgente in codice oggetto, unendo e poi linkando il codice oggetto in programmi eseguibili o in librerie. Esso usa file chiamati makefile per determinare il grafo delle dipendenze per un particolare output, e gli script necessari per la compilazione da passare alla shell.

Bibliografia

[1] Davide Barchi. «Algoritmo basato su normali e dati stereo». Alma Mater Studiorum · Università di Bologna, 2015.

[2] Michael Cavina. «3D SLAM per applicazioni real-time». Alma Mater Studiorum · Università di Bologna, 2014.

[3] H. Durrant-Whyte e T. Bailey. «Simultaneous localization and mapping: part I». In: *IEEE Robotics Automation Magazine* 13.2 (giu. 2006), pp. 99–110. ISSN: 1070-9932. DOI: 10.1109/MRA.2006.1638022.

[4] Enrico Golfieri. «Studio e valutazione di metodologie per la rilevazione di piani da nuvole di punti mediante trasformata di Hough». Alma Mater Studiorum · Università di Bologna, 2015.

[5] S. Mattoccia, I. Marchio e M. Casadio. «A Compact 3D Camera Suited for Mobile and Embedded Vision Applications». In: *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*. Giu. 2014, pp. 195–196. DOI: 10.1109/CVPRW.2014.36.

[6] Valerio Poli. «Individuazione di sistemi planari e sistemi di riferimento in nuvole di punti generate da un sistema 3D». Alma Mater Studiorum · Università di Bologna, 2014.

[7] Manuel Rucci. «Plane detection from pointclouds by means of region growing approach». Alma Mater Studiorum · Università di Bologna, 2015.