Università degli Studi di Padova

Facoltà di Ingegneria

Corso di Laurea in Ingegneria Informatica

Tesi di Laurea Magistrale

Segmentazione congiunta di immagini ed informazione geometrica e suo utilizzo all'interno di algoritmi di visione stereoscopica

Relatore: Prof. Pietro Zanuttigh
Correlatore: Ing. Carlo Dal Mutto

Laureando: Mauro Tubiana

17 Ottobre 2011

Questo documento é stato scritto in LaTeX su Debian GNU/Linux.

Indice

Elenco delle figure

Sommario

Questo lavoro ha lo scopo di indagare due aspetti: da una parte vuole cercare di capire come la segmentazione di un insieme formato da elementi composti sfruttando informazioni di colore provenienti da una coppia di fotocamere, accoppiate alla geometria desumibile dalla natura stessa del sistema stereo, possa aiutare questo task in certi contesti; dall'altra, ha lo scopo di cimprendere se sia possibile risolvere in modo più preciso il problema del calcolo di mappe di disparità, iterando il processo di segmentazione di colore e geometria come ingresso di un algortimo stereo basato su segmentazione.

Capitolo 1

Descrizione del problema

Il primo passo fondamentale per indagare proficuamente rispetto ad un problema tecnicamente rilevante, è rappresentato dalla sua descrizione nei termini consentiti dalla sua espressione in linguaggio naturale. L'esistenza di questo capitolo è da imputare a questa esigenza progettuale.

1.1 Introduzione al problema

Cosa succederebbe se unissero le proprie forze la segmentazione e la stereopsi computazionale, con lo scopo di rafforzarsi l'un l'altro?
Il lavoro di questa tesi è volto proprio ad indagare la bontà di questa intuizione introducendo un paio di elementi di novità rispetto a quanto già proposto dalla letteratura.
La segmentazione costituisce uno dei problemi maggiormente studiati in visione computazionale ed elaborazione delle immagini; essa prevede che un'immagine venga suddivisa in regione di interesse e i risvolti applicativi sono molteplici: i programmi di editing grafico e video, alcune tecniche di *feature detection* e alcune applicazioni mediche, rappresentano un campionario già di per se piuttosto vario per esprimere la portata dell'argomento.
La potenza dello strumento richiede però, come controparte, di capire da un punto di vista quantitativo cosa significhi ottenere una separazione in porzioni interessanti; molto spesso si tratta fondamentalmente di individuare una partizione dell'immagine in sottoinsiemi, compatibile con quella che produrrebbe un essere umano, il quale tenderà a distinguere gli oggetti presenti in una scena e a raggruppare i pixel appartenenti ad un certo oggetto. Di fatto il cervello umano, almeno per quanto si possa sapere, si appoggia ad informazioni correlate all'esperienza, non limitandosi dunque all'analisi dell'immagine, aspetto che non

viene codificato dall' elaborazione delle immagini.
Per ottenere risultati che rispecchino l'obbiettivo sopra, è necessario spesso codificare metriche di raggruppamento molto complesse e magari molto specifiche, oltre che computazionalmente onerose.
La stereopsi, ovvero la stima della struttura geometrica di una certa scena a partire da una collezione di riprese della stessa, costituisce un'altro problema ampiamente studiato sia dal lato scentifico che tecnico.
Nello specifico ci si orienterà al problema della costruzione di mappe di disparità o *disparity map* data una coppia di immagini, chiamate rispettivamente *reference image* e *target image* e la cui natura verrà svelata al momento opportuno, ottenute da due telecamere opportunamente posizionate.
Oltre a permette il calcolo delle componenti spaziali di ciascuno o di una porzione dei pixel ricavati dalle camere, le mappe di disparità inducono anche una corrispondenza tra i pixel appartenente all'immagine di riferimento e quelli dell'immagine target; questa considerazione aiuta a capire come sia dunque possibile unire una molteplicità di informazioni, le quali risultano utili ai fini della segmentazione, senza peraltro richiedere modelli troppo complessi per definire delle metriche appropriate di discriminazione per i cluster.
Oltre a ciò, e sulla base di alcune considerazioni desumibili dalle particolarità che caratterizzano la segmentazione, si può pensare non solo di usare quest'ultima per ottenere migliori risultati in ambito stereo, ma si può anche cercare di capire, unendo sinergicamente i due moduli, se iterare il processo di calcolo di mappe di disparità e segmentazione, facendo nutrire dell'output dell'uno l'input dell'altro, porti a dei vantaggi reciproci.

1.2 Composizione della tesi

Oltre al presente Capitolo introduttivo, questo relazione sull'attività di tesi si presenta suddivisa come segue:

Capitolo 2 Questo capitolo cerca di dare un'infarinatura rispetto alla questioni legate della segmentazione, introducendo prima i formalismi che definiscono il generico modello matematico che descrive il problema, per poi offrire una carrellata sullo stato dell'arte, fino a trattare il particolare algoritmo usato, ovvero *Spectral Clustering* ponendo l'attenzione sulla variante che utilizza il metodo di Nystrom per ricavare una soluzione approssimata al problema generalizzato degli autovalori.

Capitolo 3 Tratta il problema, figlio della stereopsi computazionale, noto come calcolo delle corrispondenze e si pone l'obbiettivo di far capire al lettore

come sia possibile calcolare, attraverso triangolarizzazione, le coordinate spaziali associabili ad un insieme di pixel. Anche in questo caso, ritenendolo utile se non altro per potersi rendere conto di quali direzioni scientifiche sia state intraprese, si è deciso di condurre un rapido *tour* che descriva, senza pretesa di completezza, quali siano le più diffuse tecniche nel settore. Il *focus* verrà infine ancora una volta posto su una specifica soluzione, fornita da un gruppo di ricercatori italiani: *Segment Support*, interessante perché fà uso della segmentazione per attuare i suoi piani di conquista delle mappe di disparità

Capitolo 4 Dati i costituenti teorici, si è cercato di produrre una piccola libreria per indagare la bontà delle idee maturate e per realizzare il porting di una porzione già esistente e scritta in MATLAB. In questo capitolo si descrive perciò la struttura ad altro livello della libreria e dell'eventuale applicazione da realizzare, analizzando qualche problematica progettuale

Capitolo 5 Il contenuto di questo capitolo vuole essere una sorta di documentazione, di *reference*, al codice realizzato durante il periodo di tesi

Capitolo 6

Capitolo 2

Segmentazione

Lo scopo della segmentazione, inteso nel senso dell'elaborazione di immagini, può essere visto come un modo per separare i dati stessi in regioni significative, in accordo con una certa metrica; lo scopo di questo capitolo, è trasmettere al lettore un'idea più o meno precisa di come questo problema sia affrontato da un punto di vista scientifico, concentrando gli sforzi su particolari utili a comprendere quanto riportato in seguito.

2.1 Formalizzazione del modello

La segmentazione di immagini è l'azione di dividere l'immagine in regioni che la compongono.
Più formalemente si definisce regione una collezione di pixel che soddisfi i seguenti vincoli:

- $I = \cup_{i=1}^{N} R_i$
- R_i regione connessa $\forall i$
- $R_i \cap R_j = \emptyset \; \forall i \neq j$
- Sia Q un predicato, applicabile ad un sottoinsieme dell'immagine, ovvero ad una regione o ad un'unione di regioni, allora dev'essere vero che:

$$\begin{aligned} Q(R_i) &= \mathbf{true} \quad \forall\, i \in 1, 2, \ldots, N \\ Q(R_i \cup R_j) &= \mathbf{false} \qquad \forall i \neq j \end{aligned}$$

dove I rappresenta l'insieme, di cardinalità N sul quale definire la partizione e R_i indica il sottoinsieme di punti che costituisce una regione.

La seconda condizione molto spesso, o comunque non nel caso di questa tesi, non viene considerata o ne viene rilassato il relativo vincolo. L'elenco appena espresso, dice fondamentalmente che la divisione ottenuta deve essere partizione, nel senso matematico del termine, dell'insieme rappresentato dai pixel che compongono l'immagine; il quarto elemento, infine, specifica il fatto che i pixel appartenenti ad una regione, debbano avere in comune la veridicità di un certo predicato codificando la proprietà, che idealmente dovrebbero esprimere gli algoritmi o le procedure di segmentazione, di raggruppare elementi opportunamente. Un esempio di predicato potrebbe essere quello che misura se i pixel appartenenti ad una certa regione siano tutti di uno stesso colore.

2.2 Panoramica sulle tecniche più diffuse

Raggiungere tutti gli obbiettivi descritti sopra, non è compito facile specialmente considerata la generalità con la quale i vincoli sono stati specificati; proprio questa caratteristica permette di raggruppare i dati nel modo più consono e induce un livello di sfida che rende questo problema uno dei più studiati e longevi nell'ambito della *Computer Vision*.
Risulta allora utile capire come sia stata affrontata la questione relativa alla segmentazione attraverso una veloce panoramica sulle tecniche di maggior successo, senza peraltro pretesa di esaustività considerando anche l'immensa mole di letteratura disponibile sull'argomento, per la quale è possibile trovare alcuni spunti in [Sze10].

2.2.1 Clustering

Il *clustering* è una tecnica di *machine learning* per la classificazione non supervisionata di oggetti caratterizzati da un vettore di *feature*, attraverso la definizione di una misura di similarità tra gli elementi appartenenti all'insieme da suddividere (ad esempio distanza euclidea tra i colori che caratterizzano i pixel in questione, ma anche metriche più sofisticate che tirino in ballo caratteristiche meno locali o comunque adatte ad uno scopo specifico).

Si può definire una tassonomia delle strategie di *clustering* a seconda della tipologia caratterizzante le regioni prodotte:

clustering esclusivo ogni pixel dell'immagine appertiene ad almeno una regione ed esattamente ad una regione

overlapping clustering un determinato pixel può appartenere a più di una regione, viola una dei vincoli definiti sopra.

clustering gerarchico i pixel vengono divisi in sottoregioni e queste ultime vengono subiscono raggruppamenti in insiemi sempre più ampi

clustering probabilistico assegnazione delle regioni con metodi probabililstici

Una delle tecniche più conosciute quando si parla di clustering, è l'algoritmo *k-means*, inseribile nella classe delle procedure di clustering esclusivo. Fissata la funzione che determini la somiglianza tra punti e la cardinalità k dell'insieme delle regioni, si procede in questo modo:

1. si selezionano casualmente k centroidi in modo da inizializzare la partizione
2. ogni punto di interesse viene assegnato al centroide più vicino secondo la metrica scelta
3. si ricalcolano i centroidi, secondo la nuova disposizione e si ripete la procedura partendo dal punto 2, fino a che l'assegnazione non viene più modificata (la posizione del centroide dipende infatti dalla cardinalità delle regioni)

L'algoritmo non conduce necessariamente ad una soluzione ottima globale, poiché l'assegnazione iniziale dei cluster condiziona l'evolvere della procedura e la suddivisione è buona soprattutto nel caso in cui la disposizione dei dati sia di tipo gaussiano.
A questi contro si affiancano alcuni aspetti positivi che rendono *k-means* appetibile, ovvero, la velocità di convergenza nel caso medio e la semplicità di implementazione.

2.2.2 Histogram based

Questa tipologia prevede che l'immagine venga suddivisa attraverso l'analisi dell'istogramma associato.
Anche in questo caso si può verificare la presenza di varie tecniche, più o meno efficaci a seconda del contesto, in particolare:

sogliatura globale si cerca all'interno dell'istogramma di un'immagine un punto di minimo locale e si suddivide a partire da questo. La sogliatura globale è tuttavia molto sensibile al rumore e ai cambi di luminosità, condizioni che tendono ad appiattire l'istogramma rendendolo difficile la separazione tra zone

metodo di Otsu il metodo di Otsu cerca di suddividere l'istogramma in modo da massimizzare la varianza tra regioni diverse e allo stesso tempo cercando di minimizzare la varianza interclasse. Esistono varie estensioni a questo metodo che permettono di ottenere risultati più accurati

2.2.3 Segmentazione basata sulle regioni

La segmentazione basata sulle regioni ha come obbiettivo la ricerca di regioni uniformi rispetto ad un predicato di similarità. Esistono fondamentalmente due tipi di approcci:

bottom-up si parte dall'ipotesi iniziale che le regione siano costituite da un singolo pixel e, generalmente in modo ricorsivo, si accorpano regioni fino a che un certo vincolo non viene raggiunto

top-down contrariamente a quanto definito sopra, la procedura viene inizializzata con una sola regione, contentente tutti i pixel dell'immagine; si opera quindi una suddivisione, sempre ricorsivamente, fino al raggiungimento di un certo vincolo

Un esempio di approccio bottom-up è costituito dalla tecnica *region growing* mentre per quanto concerne gli algoritmi top-down, si segnala *region splitting and merging*.

Naturalmente le classficazione appena descritta costituisce solo un piccolo campionario delle tecniche esistenti, e il suo scopo infatti è meramente introduttivo. Per dovere di cronache si citano, ad esempio:

- metodi basati sull'*edge detection*
- metodi basati su *graph partitioning*
- segmentazione basata sul modello
- segmentazione multiscala
- ...

2.3 Spectral Clustering

In relazione alla formulazione del problema del raggruppamento basato su grafi, si inquadra la tecnica di *Spectral Clustering*[SM00]. L'idea è di costruire un grafo $G = (V, E)$ i cui nodi rappresentino elementi di un *feature space*, legati da archi pesati secondo una funzione di similarità $w(i, j)$, cercando poi di suddividere i vertici in modo tale che la somiglianza tra nodi appartenenti allo stesso sottoinsieme sia elevata e che al contempo vi sia un basso valore di similarità tra gruppi distinti.

La teoria dei grafi definisce il taglio come tra due sottoinsiemi, siano essi A e B, di nodi che formino una partizione per l'insieme esteso come:

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, w)$$

Sia poi

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

dove $assoc(A, V) = \sum_{u \in A, t \in V} w(u, t)$. L'obbiettivo diventa quello di partizionare i nodi del grafo in nodo da minimizzare la grandezza $Ncut$. Si dimostra che questa operazione è riconducibile alla soluzione del problema generalizzato degli autovalori per il sistema

$$(D - W)y = \lambda y$$

dove D è una matrice diagonale, quadrata, di taglia pari alla cardinalità dell'insieme dei nodi del grafo e nella quale $D_{ii} = d(i) = \sum_j w(i, j)$, W è la matrice che stiva i coefficenti $w(i, j)$, y è un vettore ricavato dalla partizione dei nodi del grafo in due sottoinsiemi e infine λ è la variabile che indica l'autovalore da cercare. In particolare, l'autovettore associato al secondo più piccolo autovalore, è legato alla soluzione ottima. I dettagli possono essere reperiti in ...

2.3.1 Approssimazione di Nystrom applicata allo Spectral Clustering

La specificazione sopra descritta, presenta lo spiacevole inconveniente di appartenere alla classe dei problemi NP-completi, rendendolo, almeno a quanto si sappia, intrattabile da un punto di vista computazionale.
Gli stessi autori hanno però sviluppato una versione polinomiale, che sia in grado di risolvere in modo approssimato il problema generalizzato degli autovalori, attraverso un'approssimazione introdotta dal matematico tedesco E.J. Nystom.
Senza entrati nei particolari, presenti in [FBCM04], basti dire che, campionando un sottoinsieme di pixel, anche molto minore rispetto al numero totale di cui è composta l'immagine, è possibile attraverso il metodo accennato sopra, approdare ad una partizione molto simile a quella garantita dalla soluzione puntuale.
Il modello prevede la formazione di un grafo $G = (V, E)$ dove i nodi sono costituiti dalla rappresentazione in un determinato spazio di colore, di ciascuno dei pixel che costituisce l'immagine.
La matrice dei pesi, la quale viene generata solo parzialmente è costituita da elementi che codifichino la similarità tra due pixel i e j e rende questa tecnica abbastanza flessibile da portare all'ottenimento di segmentazioni di varia natura: è possibile ricavare gli *edge* di un'immagine, così come separarla nelle *texture*

che la compongono, oppure semplicemente raggruppare le porzioni porzioni di immagine che presentano somiglianza di colore o vicinanza geometrica, scegliendo la metrica di confronto più opportuna.
Nel caso specifico si è puntato verso una funzione esponenziale decrescente e simmetrica come:

$$e^{-\frac{d_c(x_i,x_j)}{\sigma^2}}$$

dove x_i ed x_j sono punti in un opportuno spazio.

Capitolo 3

Stereopsi computazionale

In questo Capitolo, partendo da argomenti di carattere principalmente matematici e quindi di descrizione di un modello, ci si muoverà verso discussioni legate alla risoluzione di problemi più legati ad aspetti pratici.

3.1 Intoduzione alla stereopsi

La stereopsi computazionale è la disciplina che si occupa di evincere la struttura tridimensionale di una scena, date le immagini ricavate da una coppia di telecamere, inquadranti la scena e poste in modo opportuno.
Questo processo si prefigge lo scopo di risolvere due questioni distinte ma strettamente legate che possono essere individuate nel *calcolo delle corrispondenze* e nella *triangolazione*.
Con calcolo delle corrispondenze, ci si riferisce alla possibilità di individuare degli accoppiamenti tra punti appartenenti all'immagine di riferimento e punti giacenti sull'immagine target, grazie al fatto che le due diverse riprese della scena si discostino di poco, inducendo una similarità tra le immagini (si veda Figura 3.2); una coppia di punti di questo tipo si definisce *coniugata*. Una discussione più approfondita verrà espletata nella Sezione 3.2.
Triangolare significa, almeno in questo constesto, dedurre le profondità di un punto, dati una mappa di disparità che codifichi gli accoppiamenti di cui si è appena parlato, un parametro detto *baseline* che misura la distanza tra i centri di proiezione delle camere e la focale delle camere ovvero la distanza tra il piano immagine ed il centro di proiezione. Se aggiuntivamente si considerano le *matrici di proiezione prospettica* (MPP) che caratterizzano le camere codificandone i parametri intrinseci, diventa possibile stimare la completa geometria della scena.

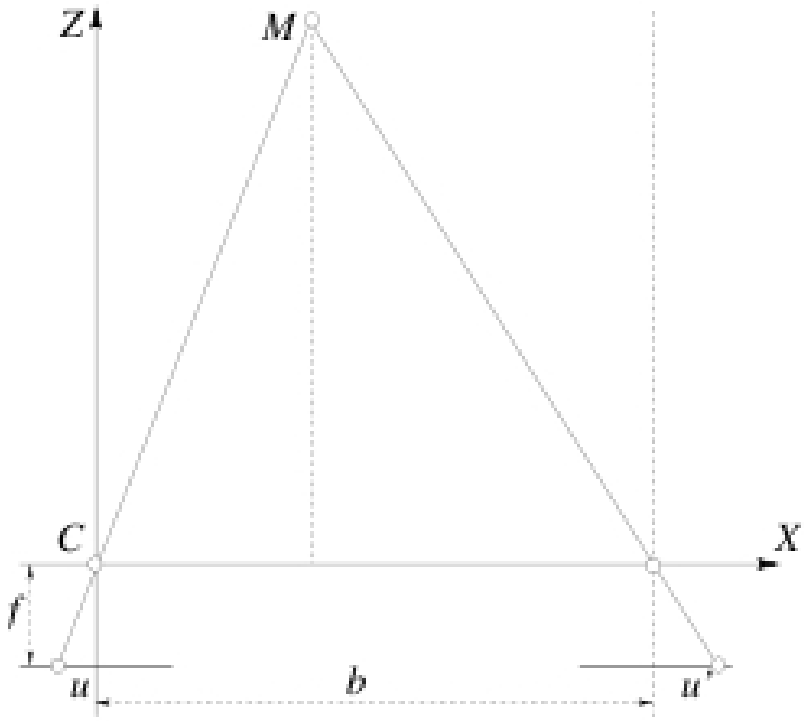


Figura 3.1: Immagine che descrive la triangolazione nel caso semplificato

Per capire come funzioni il meccanismo della triangolazione, si possono presumere i seguenti vincoli senza perdita di generalità:

- assi ottici delle due camere paralleli
- focale f uguale per ognuna delle due camere
- sistema di riferimento solidale al sistema mondo

Grazie a ciò, ed in riferimento alla situazione illustrata in figura 3.1, si ricava facilmente, grazie alle note formule legate alla similitudine tra triangoli, si può giungere alle seguenti formule:

$$\begin{cases} \frac{f}{z} = -\frac{u}{x} \\ \frac{f}{z} = -\frac{u'}{x-b} \end{cases} \tag{3.1}$$

dalle quali si ottiene

$$z(u,v) = \frac{fb}{u'-u} = \frac{fb}{d(u,v)} \tag{3.2}$$

con d valore della mappa di disparità nel punto $p(u, v)$. Si può infine mostrare, non difficilmente, che vale la seguente relazione:

$$M = Km$$

dove

$$m = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}$$

$$M = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

e

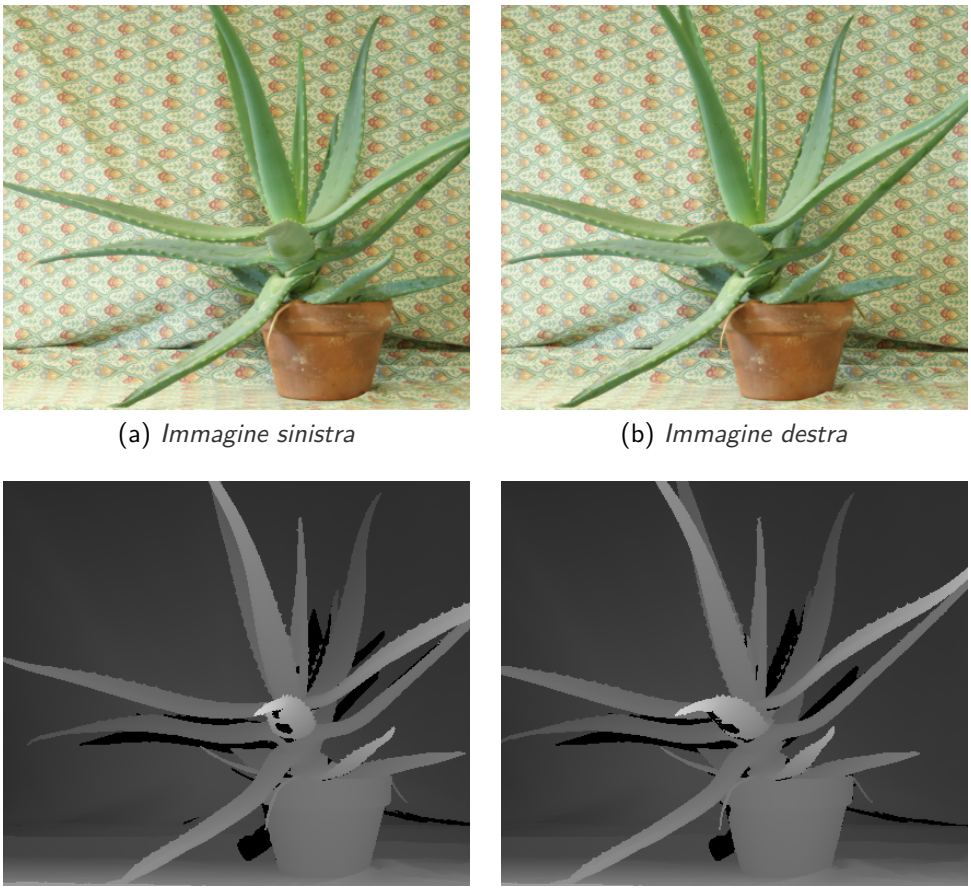
$$K = \begin{bmatrix} fk_u & 0 & u_0 \\ 0 & fk_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Nella matrice K compaiono dei termini, k_u e k_v che permettono di effetturare un cambio di unità di misura mentre u_0 e v_0 servono per compensare al fatto che il centro di proiezione non sempre concida con l'origine degli assi coordianti nello spazio mondo.
Per semplificare le cose, ed essendo sempre possibile farlo, d'ora in poi si supporrà sempre verificata la condizione per la quale le immagini trattate siano rettificate e corrette della distorsione prospettica; ciò infatti permette di ridurre il calcolo delle corrispondenze ad un problema unidimensionale e anche la triangolazione ne trae benefici. Sotto questa ipotesi, infatti, si può agire come spiegato sopra, evitando di scomodare metodi più complessi.

3.2 Calcolo delle corrispondenze

Mentre in generale la mappa di disparità è un insieme di vettori, quando si lavora su immagini rettificate, la si può ricondurre ad un insieme di scalari, giacché i valori agiscono solo su una della due coordinate, cosa che rende più agevole e veloce la ricerca delle corrispondenze giacché ne implica la ricerca su una retta monodimensionale.
Le difficoltà non sono tuttavia risolte, infatti la natura di un sistema stereoscopico e la composizione della scena pongono sul piatto alcune questioni che portano in molti casi a sbagliare la stima. Eccone alcune:

- le porzioni di immagini occluse da altri oggetti, non presentano nessuna corrispondenza valida, poiché costituite da punti che compaiono su una sola delle due immagini



(a) *Immagine sinistra* (b) *Immagine destra*

(c) *Left-Right disparity map* (d) *Right-Left disparity map*

Figura 3.2: Esempio di coppia stereo con relative disparity map

- la distorsione radiometrica causata dal fatto che gli oggetti ripresi possano essere costituiti da superfici a radianza dipendente dal punto di osservazione (superficie non lambertiane), causano gravi difficoltà per via del fatto che una particolare riflessione può, ad esempio, essere presente in una sola delle inquadrature

- la distorsione prospettica causa variazioni nella forma di alcuni oggetti (forma proiettata si intende)

- i pattern ripetitivi possono mettere in difficoltà molti algoritmi per ovvie ragioni, così come le regioni uniformi

Per fortuna, lo studio dei problemi ha portato anche alla definizione di vincoli che le corrispondenze sono tenute a rispettare:

somiglianza sembra quasi ovvio pensare che due punti coniugati presentino una somiglianza locale

vincolo epipolare si può mostrare come, dato un punto nell'immagine di riferimento, il suo coniugato debba giacere su una retta nell'immagine target

lisciezza la disparità varia poco lontano dai bordi

unicità almeno idealmente, un punto nell'immagine *reference* deve corrispondere ad un solo punto nell'immagine *target*

ordinamento monotono definendo una relazione d'ordine dei punti rispetto alla loro posizione nell'asse u, questa deve valere anche per l'insieme dei punti coniugati

3.2.1 Classificazione dei metodi per il calcolo delle corrispondenze

È possibile suddividere le tecniche impegnate nella risoluzione del problema della ricerca di corrispondenze in due macro-categorie:

metodi globali i metodi globali solitamente hanno come obbiettivo la ricerca di una configurazione di disparità che minimizzi una funzione energia opportunamente definita a seconda del modello

metodi locali solitamente cercano, mediante l'ausilio di finestre, di individuare quelle che presentino maggiore somiglianza

nel caso della seconda famiglia, la funzione utilizzata per valutare la migliore corrispondenza, viene detta *matching cost function* e le definizioni di questa possono essere le più svariate. Senza scendere nei particolari, si preferisce rimandare alla lettaratura per una catalogazione più esaustiva si vedano [SMB] che presenta un dataset di immagine ormai diventate standard nella valutazione di algoritmi stereo, oppure [SS02] e [Sze10]. Una volta realizzata una prima stima della *disparity map* può essere utile capire quali siano le corrispondenze inconsistenti effettuando il cosidetto *cross-checking*. Questa tecnica prevede di considerare due diverse mappe di disparità, una calcolata considerando l'immagine destra come riferimento (d_{RL}) e una prendendo invece l'immagine sinistra come *reference image* (d_{LR}). I punti non consistenti sono quelli per i quali è vera la seguente disuguaglianza:

$$|d_{LR}(u,v) - d_{RL}(u + d_{LR}(u,v), v) \geq T|$$

dove T è una soglia tipicamente impostata al valore 1.

3.3 Segment Support

L'uso della segmentazione a corredo del calcolo delle corrispondenze, è una pratica che ha preso un certo piede in letteratura, sotto l'ipotesi che punti appartenenti allo stesso segmento siano associati a disparità che variano in modo non repentino.
L'algoritmo proposto da Matoccia et al. in [TMS07] prevede che sia l'immagine di riferimento sia quella target, vengano segmentate per poi comporre l'informazione fornita in una funzione costo definita da

$$C(p_c, q_c) = \frac{\sum_{p_i \in W_r, q_i \in W_t} w_r(p_i, p_c) \cdot w_t(q_i, q_c) \cdot TAD(p_i, q_i)}{\sum_{p_i \in W_r, q_i \in W_t} w_r(p_i, p_c) \cdot w_t(q_i, q_c)} \tag{3.3}$$

dove

- W_r e W_t rappresentano rispettivamente la finestra di valutazione nell'immagine reference e la finestra di valutazione nell'immagine target
- p_c e q_c rappresentano i punti centrali per le finestre definite sopra
- w_r e w_t sono pesi opportunamente definiti
- $TAD(\cdot, \cdot)$ è uno stimatore robusto chiamato *Truncated Absolute Difference*

Contrariamente ad alcune procedure definite in lavori precedenti, i quali costruivano i coefficenti w_r e w_t in base a fattori legati al colore e ad una componente spaziale, *Segment Support*, questo il nome colloquialmente usato per il metodo definito, si basa maggiormente sulle informazioni estratte dalla segmentazione, portando ad una buona gestione. Nel caso particolare, la funzione peso è stata pensata come

$$w_r(p_i, p_c) = \begin{cases} 1.0 & p_i \in S_c \\ exp(-\frac{d_c(I_r(p_i), I_r(p_c))}{\sigma_c}) & altrimenti \end{cases} \tag{3.4}$$

dove S_c è il segmento a cui appartiene il punto p_c e $I_r(p_i)$ rappresenta il colore del pixel p_i. Stesso discorso vale sostituendo t ad r. La tecnica qui brevemente descritta è stata implementata sfruttando, per portare a compimento la fase di segmentazione, l'algoritmo di *Spectral Clustering*.
La decisione di puntare su questo approccio, è dettata dalla difficoltà di implementazione, relativamente bassa e dagli ottimi risultati ottenibili pur, come controparte, essendo caratterizzato da tempi di esecuzione molto lunghi.

Capitolo 4

Struttura dell'applicazione

Il presente capitolo è teso a descrivere la struttura ad alto livello dell'applicazione e della relativa libreria di supporto, ponendo al centro dell'attenzione la suddivisione in moduli funzionali e le scelte progettuali.

4.1 Struttura del framework

Pipeline funzionali

Concettualmente si può pensare all'applicazione come opportuna unione di due canali di elaborazione funzionalmente indipendenti ma cooperanti: *Segmentation Pipeline* e *Stereo Pipeline*. Questa suddivisione è figlia della necessità di indagare due problemi diversi che possono comunque essere combinati per migliorare l'accuratezza di entrambi.
Si ricorda inoltre come il risulato della segmentazione dell'insieme composto da elementi che combinano geometria e colore, vengano riposti in input, attraverso retroazione, al modulo che si occupa della stereopsi (si veda la Figura 4.1).

4.1.1 Prima caratterizzazione della struttura

sfruttare appieno lo stereo

Come già accennato in fase di introduzione al problema, la classica segmentazione di un'immagine basata esclusivamente su informazioni legate al colore, può essere unita alla disposizione geometrica dei punti descriventi la scena e addirittura ad una ulteriore componente informativa, che deriva dall'apparato di acquisizione, di tipo binoculare. Quest'ultima *feature* rappresenta la prima componente innovativa rispetto ai lavori precedentemente sviluppati (si veda [DMZC11]) e pertiene al modulo di elaborazione legato alla segmentazione (poiché tende a perseguire efficacente questo scopo), pur integrando massicciamente elementi legati alla natura stereoscopica dell'apparato di acquisizione; sembra importante

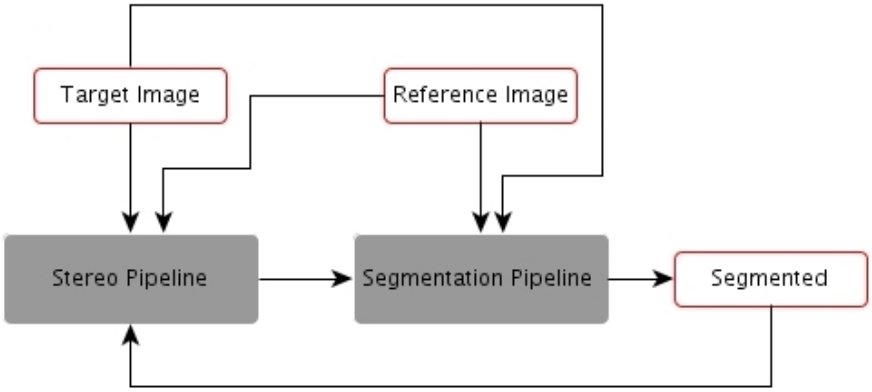


Figura 4.1: Struttura del framework

sottolineare come la componente stereoscopica sia solo uno strumento a corredo della segmentazione.

Il secondo *macro modulo*, *Stereo Pipeline*, mira alla semplice stima di *disparity map* utilizzando però immagini segmentate basandosi su ipotesi già discusso nel capitolo specifico.

4.2 Segmentation pipeline

Il successivo passaggio è utile a particolareggiare la composizione dell'applicazione, permettendo di definire quelle che dovrebbero essere le funzioni a supporto. Pur non essendoci stato pieno riscontro implementativo, l'obbiettivo era quello di realizzare un *set* di primitive in grado di gestire *input* generici, almeno nei limiti del possibile, e quindi una sorta di libreria, che potesse essere riutilizzata anche in altre situazioni. Come si può capire dallo schema a blocchi, si suppone di disporre di immagini rettificate, provenienti da due sorgenti opportunamente posizionate. I parametri estrinseci ed intrinseci del sistema binoculare, si presumono noti a priori, non sarebbbe tuttavia così complesso integrare la fase di calibrazione e rettificazione all'interno di quanto già costruito.

ipotesi sulle immagini

Adesso sono stati specificati tutti gli elementi che permettano un descrizione più accurata dei vari blocchi che compongono lo schema.

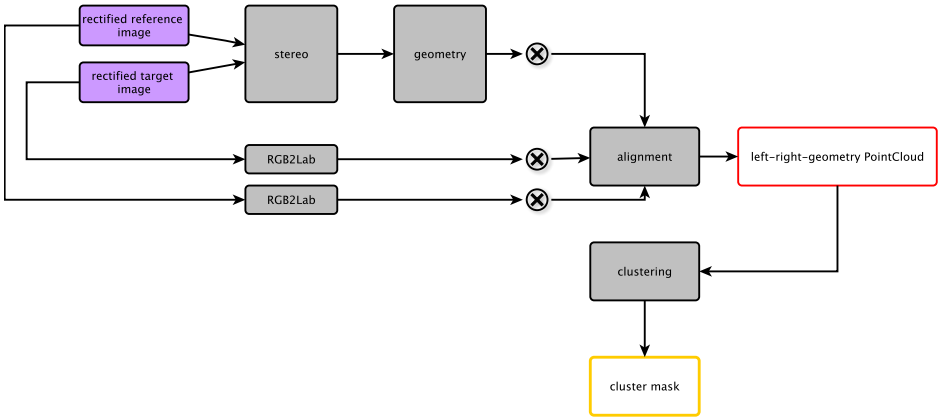


Figura 4.2: Schema a blocchi di Segmentation pipeline

4.2.1 Rappresentazione PointCloud

La necessità di fondere dati geometrici, ovvero coordinate spaziali tridimensionali (in prima approssimazione termini appartenenti a $\mathbb{R}^3$), con colori associati ai singoli punti (solitamente terne di valori ad 8 *bit*), porta naturalmente all'istituzione di una struttura dati che permetta la gestione di questa peculiarità.

Point Cloud per dati eterogenei...

Una struttura simile, chiamata *PointCloud*, rappresenta una collezione di elementi, ad esempio punti in uno spazio, ai quali associare informazioni aggiuntive (colore ripreso da due telecamere nel caso specifico, ma potenzialmente con la possibilità di trattare quasi ogni genere di informazione a corredo).

...ma anche di natura precisa

Si noti come la struttura dati qui presentata, sia comunque utilizzata come forma di interscambio dati tra vari moduli; ciò significa che, in effetti, sia usata per rappresentare anche nuvole di solo colore o di sola geometria.
Da un punto di vista formale, possiamo figurare la *PointCloud*, nella generica forma:

$$\mathcal{P} = \{p_1, p_2, \ldots, p_n\} \tag{4.1}$$

dove è utile specificare la forma dei singoli punti nel caso specifico, i quale si presentano come:

$$\widetilde{p}_i = \begin{bmatrix} L_i^* \\ a_i \\ b_i \end{bmatrix} \tag{4.2}$$

$$\hat{p}_i = \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \tag{4.3}$$

$$\bar{p}_i = \begin{bmatrix} L_r^* \\ a_r \\ b_r \\ L_t^* \\ a_t \\ b_t \\ x \\ y \\ z \end{bmatrix} \tag{4.4}$$

ad ognuno la propria nuvola

dove, come intuibile, l'espressione 4.2, stà ad indicare il vettore che descrive, nello spazio colore CIELab (scelta non casuale, infatti CIELab ha la caratteristica di essere uno spazio uniforme), le caratteristiche cromatiche del pixel $\widetilde{p}_i$, la 4.3 intende codificare le coordinate geometriche (nello spazio delle camere) del punto $\hat{p}_i$ ed infine 4.4 rappresenta, questa volta in riferimento all'elemento $\bar{p}_i$, la composizione in termini di accoppiamento tra le informazioni legate al colore e quelle di natura spaziale.
In particolare per quanto riguarda l'ultima formulazione, una spiegazione più dettagliata, può essere reperita nella sezione 4.2.4. La forma finale del vettore 9-dimensionale di cui si è appena fatta menzione, in realtà si compone in modo lievemente differente. Per rendere omogenei i dati dalla prospettiva indotta dalle unità di misura, si rende necessario moltiplicare colore e geometria per l'inverso della deviazione standard della propria componente *luminanza*, e fare lo stesso con la componente geometrica ma rispetto alla coordinata z.
Inoltre, per poter bilanciare il peso delle coordinate spaziali e di quelle cromatiche, specialmente pensando all'operazione di *clustering*, si moltiplica per un'ulteriore

normalizzazione dei dati

termine, chiamato λ, la parte xyz, pervenendo alla forma finale

$$p'_i = \begin{bmatrix} \frac{1}{\sigma_{L^*_r}} L^*_r \\ \frac{1}{\sigma_{L^*_r}} a_r \\ \frac{1}{\sigma_{L^*_r}} b_r \\ \frac{1}{\sigma_{L^*_t}} L^*_t \\ \frac{1}{\sigma_{L^*_t}} a_t \\ \frac{1}{\sigma_{L^*_t}} b_t \\ \frac{\lambda}{\sigma_z} x \\ \frac{\lambda}{\sigma_z} y \\ \frac{\lambda}{\sigma_z} z \end{bmatrix} \tag{4.5}$$

4.2.2 Mask module e Filter module

I moduli trattati in questo paragrafo lavorano in sinergia per attuare delle semplici operazioni per il filtraggio delle nuvole di punti e delle immagini.

filtraggio di una Point-Cloud

Con filtraggio si intende in questo caso, eliminazione di elementi indesiderati, magari perché frutto di imprecisione di misurazione o presenza intrinseca di valori indesiderati generati da precedenti computazioni (o per evitare il verificarsi di quest'ultima evenienza), e che pertanto necessitano di essere gestiti in modo particolare.
Le operazioni di filtraggio avvengono in due step:

- Selezione dei punti da filtrare
- Eliminazione o rimpiazzo dei punti selezionati

Il *Mask module* fà riferimento al primo tipo di azione, mentre nel *Filter module* sono raggruppate le funzioni di eliminazione e rimpiazzo. Le mask possono essere generate tramite il controllo sui valori di *disparity map*, ad esempio per evitare che valori nulli pervengano in modo indesiderato o non gestibile in modo uniforme rispetto all'architettura, alla porzione del modulo geometrico che si occupa di calcolare le terne coordinate descriventi la posizione dei punti nello spazio; un'altra modalità di selezione prevista permette di gestire valori di profondità errati, dovuti ad imprecisioni presenti nelle *disparity map*.

due punti di vista

Anche se, alla luce dell'equivalenza descritta nel Capitolo 3, la seconda funzionalità può sembrare un doppione della prima, in realtà quest'ultima risulta utile per due motivi:

- fornisce un controllo diretto e intuitivo sulla geometria

- agisce in modo nativo su *PointCloud*

Le maschere possono subire mutazioni della forma e ciò permette di aggregare le informazioni scaturite da operazioni effettuate sulla *disparity map* con quelle prodotte dal filtro geometrico.

un possibile improvement

Interessante sarebbe estendere le capacità del filtro geometrico, al fine di selezionare diverse componenti o gruppi di componenti per ogni punto.

4.2.3 Geometric module

Il modulo geometrico è molto semplice: è composto solamente dalla funzione che stima la struttura geometrica a partire dalla mappa di disparità e non richiede considerazioni particolari, almeno per quanto concerne il lavoro di progettazione; la sua esistenza ha semmai comportato riflessioni su altre componenti, come già visto nella sezione dedicata al filtraggio e alle maschere. Unica nota di un qualche interesse, il modulo geometrico dovrebbe ricevere in ingresso una *disparity map*, per poi fornire una nuvola di punti.

4.2.4 Alignment module

Operazione fondamentale, perché caratterizzante l'idea della tesi è combinare informazioni di colore provenienti da due telecamere che inquadrano una stessa scena, con una componente geometrica associabile a (quasi) ogni punto della scena stessa.

accoppiare colore e geometria

Fondamentalmente è necessario fissare un'immagine, tra le due che compongono la coppia stereo, detta *reference image*, sulla base della quale viene costruita la mappa di disparità e quindi stimata la componente geometrica, e un'immagine detta *target image* che fornirà la componente di colore secondaria da associare alla coppia di componenti *reference image-geometria*. Il primo problema che ci si presenta, posto dalla necessità di accoppiare i dati, risiede nella *non iniettività* che caratterizza la corrispondenza, attraverso la *disparity map*, *reference-target*. Ciò stà a significare che esistono punti per i quali non è possibile individuare correttamente il coniugato. Un simile problema è tuttavia intrinseco al modello matematico usato per approssimare la realtà descritta, ed esula dalla trattazione della tesi.

analisi di accoppiamenti errati o inesistenti

Più significativo, invece, considerare il fatto che alcune corrispondenze portano ad associare punti reference a punti non appartenenti al dominio target, ovvero a punti di coordinate non valide nell'immagine target; per ovviare a quest'inconveniente, sono previsti due controlli, da svolgere congiuntamente o separatamente: il primo contempla l'uso delle classiche maschere così da evitare il trattamento

di dati mascherati, mentre il secondo si tuffa direttamente nel mondo delle coordinate, gestendo valori nell'asse u negativi.
Una volta catturato il coniugato e separate corrispondenze sicuramente non buone da corrispondenze probabilmente buone, si palesa la questione dell'aggregare tutti i dati a disposizione, allineati opportunamente, per poi accomodarli all'interno della struttura *PointCloud*.
Con allineamento si intende la giustapposizione, idealmente per ogni punto rappresentato nell'immagine di riferimento, della componente colore del particolare punto $\widetilde{p}(u, v)$, della componente colore del coniugato di $\overline{p}(\widehat{u}, \widehat{v})$ determinato come $\widetilde{p}(u + sd, v)$, e della terna di coordinate che individuano il punto $\widetilde{P}(x, y, z)$, la cui proiezione rispetto allo spazio vettoriale generato dal sistema di telecamere, origina $\widetilde{p}$.

modulo allineatore

La realizzazione del modulo allineatore è stata concepita in modo molto semplice, attraverso l'uso di una*look-up table*, la quale descrive una permutazione definita sull'insieme degli interi minori o uguali rispetto alla taglia delle *PointCloud*. Si noti come, a causa della particolarità sopra descritta (non iniettività), non sempre (anzi mai nel caso specifico) risulta possibile ricondurre la funzione che descrive le corrispondenze, ad una permutazione. Prevedendo il verificarsi di questa eventualità, vanno istituiti dei punti sentinella, non validi, ad esempio con indice negativo, che segnalino che la posizione corrente mappa in un punto non valido. L'ultima considerazione riguarda l'ipotesi secondo la quale solo il vettore che identifica l'immagine *target* vada permeutato; in altre parole, il vettore relativo alla componente colore dell'immagine di riferimento e alla corrispondente descrizione geometrica, rimangono fissate, in termini di struttura, allo stato descritto nella sezione 4.2.1.
Riassumendo le idee appena esposte è possibile, anche con l'ausilio dello schema in figura 4.3, capire come si è deciso di procedere per realizzare l'allineamento:

ricerca dei coniugati si costruisce una coppia di punti, nel sistema di coordinate delle immagini, che rappresenta un punto nella *reference image* ed il suo coniugato; questo per ogni pixel dell'immagine di riferimento

generazione della pemutazione partendo dalgli accoppiamenti rilevati al passo precendente, si costruisce la funzione che porge le corrispondenze sotto forma di permutazione. Da questo passo si è in grado di rilevare quando la funzione prodotta non è biiettiva. Il risultato della computazione sarà una sorta di tabella delle corrispondenze *reference-target*, in termini questa volta di posizione intera, considerando un ordinamento dei punti di tipo *row-major* Più nel dettaglio, sia $P_t^i(\widetilde{u}, \widetilde{v})$ il pixel di coordinate $(\widetilde{u}, \widetilde{v})$ rispetto al sistema di coordinate ortoganali centrato nell'angolo in alto a destra dell'immagine target e con versori orientati rispettivamente nel verso

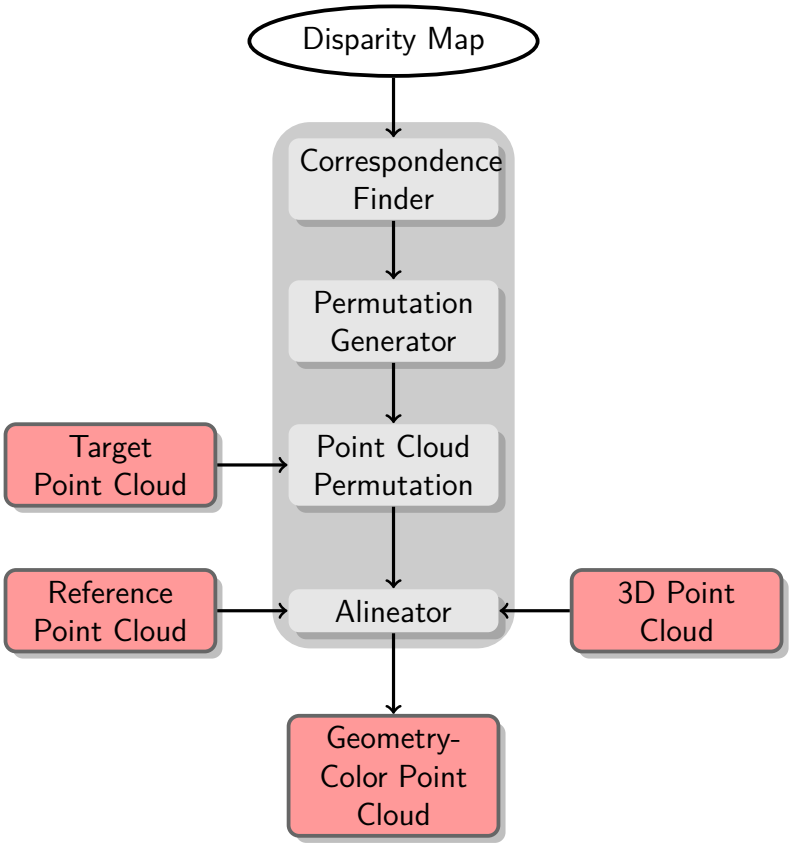


Figura 4.3: Schema di funzionamento del modulo allineatore

secondo il quale crescono l'indice di riga e colonna; sia poi $i = \widetilde{u} \cdot n_r + \widetilde{v}$ con n_r intero che quantifica il numero di righe (numero di pixel sul lato verticale) dell'immagine (target o reference). Si può infine definire $C(i) = j$, la funzione che assegna la nuova posizione j al punto i. La locazione j è calcolata sulla base della posizione del coniugato di P_t^i nell'immagine di riferimento in modo analogo a quanto spiegato appena prima.

permutazione del vettore target questo passo ridispone le componenti della *PointCloud* secondo la legge ricavata nel passo di generazione della permutazione; qualora non sia valido uno dei vincoli di iniettività o suriettività, sarà necessario riempire i buchi presenti nella nuvola mediante un'opportuna generazione di punti

associazione dei vettori nell'ultima fase si raccolgono le tre nuvole di punti, già opportunamente allineate, e si aggregano per formare la struttura finale

4.2.5 Clustering module

Il *Clustering module* rappresenta l'anello finale dello *stream* di computazione e anch'esso dev'essere pensato per lavorare con nuvole di punti piuttosto generiche. Anche in questo caso si è reso necessario un processo di *defeaturing* rispetto a quanto desiderato, a causa della elevata complessità implementativa richiesta e della, potenzialmente enorme, mole di lavoro necessaria anche nel caso si fosse trovata una soluzione sensata al problema della costruzione di una *PointCloud* adatta a quasi tutti gli scopi.
La *PointCloud* viene processata da un *middleware* che la rende adatta ad essere trattata dal modulo di *clustering*, il quale produce il vettore delle classi associate ad ogni punto che ha ricevuto in ingresso.

diverse metriche

Un'altra caratteristica interessante, prevista ma non ancora sviluppata, risiede nella possibilità di integrare un sistema che gestisca metriche differenti. La scelta di non cercare una soluzione implementativa a questo problema è, come spesso accade, dettata da mancanza di tempo ed esperienza.

il problema del campionamento

Per concludere la rapida trattazione rispetto alle idee progettuali riservate a questo modulo, và forse esplicitato il fatto che l'algoritmo utilizzato, probabilmente perché non deterministico, fornisce, a seconda della composizione dell'insieme di punti presi a campione, dei risultati spesso variabili. Per ovviare parzialmente a questo problema, peraltro rilevato tangibilmente solo in fase di implementazione, si è deciso di introdurre una procedura di campionamento alternativa a quella inizialmente prevista, la quale si limitava a scegliere punti casuali appartenenti alla *PointCloud*, pescandone con distribuzione uniforme gli indici.
Il nome affibiato alla tecnica ne lascia intuire il funzionamento: *Periodic Sam-*

pling. Grossolanamente per il momento (qualche dettaglio verrà chiarito nel Capitolo 5) si descrive l'idea che ha condotto a questa soluzione:

1. si seleziona randomicamente l'indice corrispondente ad un elemento della nuvola di punti, definito come *seed*

2. partendo dal *seed* scelto, si procede selezionando un indice ogni T elementi, in modo tale che gli indici siano equidistanti l'uno dall'altro

4.3 Stereo pipeline

inserire schema a blocchi

Con *Stereo pipeline* si intende la porzione di applicazione che si occupa di produrre la o le *disparity map* e include, oltre al sistema stereo vero e proprio, alcune funzionalità prelevate dalla *Segmentation Pipeline*. Aldilà dello schema presentato, a livello progettuale c'è ben poco da dire, la maggior parte delle problematiche sono inerenti a livelli di astrazione diversi da quello trattato in questo capitolo.

4.3.1 Stereo module

Lo *Stereo module* come facilmente intuibile, rappresenta il cuore di questa sezione.
Il blocco riceve idealmente in ingresso una coppia di immagini stereo e le corrispettive maschere, che rappresentano i cluster di appartenenza di ciascun pixel, per poi fornire la stima della mappa di disparità dell'immagine di riferimento, calcolata rispetto all'immagine target.
Ulteriore passo da effettuare prima di procedere all'elaborazione da parte del *Clusterin module*, consiste nell'operare il test di coerenza destra-sinistra o *Left-Right consistency check* per individuare discrepanze nel calcolo delle disparità. Ci si può rifare al Capitolo 3 per delucidazioni.

Capitolo 5

Implementazione

Il concretizzarsi di un'idea, espressa sotto forma di vincoli astratti ma in un qualche senso resi tangibili dalla progettazione, sfocia sperabilmente, almeno dal punto di vista di uno sviluppatore, in una implementazione reale. La sezione che segue cerca di enucleare le considerazioni a basso livello e il lavoro pratico svolto.

5.1 Tool utilizzati

La scelta degli strumenti di sviluppo è ovviamente un passo fondamentale per la realizzazione di un software in tempi utili e in modo efficiente.
I vincoli che hanno condotto verso una certa configurazione, al fine di realizzare la presente tesi, sono di varia natura, e non sempre probabilmente è stata percorsa la strada migliore.

5.1.1 OpenCV

La libreria che rappresenta lo stato dell'arte per quanto concerne la visione computazionale, si può verosimilmente identificare in *OpenCV*, inizialmente sviluppata da *Intel* e poi passata sotto l'ala protettrice dell'intraprendente e ambiziosa *WillowGarage*. Cos'è?

Tra le caratteristiche notevoli, *OpenCV* offre un ventaglio di funzionalità ampissimo, oltre a prestazioni di prima categoria, grazie anche all'integrazione di codice ottimizzato per processori che supportano le istruzioni *SSE*. Si sono sfruttate particolarmente le capacità di manipolazione di matrici, come ad esempio le classiche operazioni di prodotto, somma e differenza, nonché inversione e trasposizione. Qualora si dovesse rendere necessario, verrà fatto riferimento con maggiore dettaglio alle *facilities* impiegate. Perché?

Il punto forse debole di questa scelta risiede, probabilmente, nella scarsa capacità, a meno di adattamenti forse a tratti forzosi, di gestire nuvole di punti; è opportuno comunque sottolineare come l'abilità di un bravo programmatore possa sopperire a questa situazione.

Una possibile alternativa : PCL

Sotto buon suggerimento, l'approdo alla libreria, sempre mantenuta da *WillowGarage*, *PCL (Point Cloud Library)*, è sembrata, seppur a posteriori, un'opportunità migliore per via dell'integrazione nativa di una struttura dati che ha richiesto invece di essere definita.
La scarsa documentazione presente al momento dell'inizio dello sviluppo del software di tesi, con conseguente probabile protrarsi oltre il tempo stabilito dell'attività di apprendimento, ha portato a non prendere in considerazione l'uso di questo *tool*. La decisione presa non si è tuttavia rilevata sbalgliata.
Prima di procedere con la lettura, può non essere inutile farsi un'idea di massima su cosa offre OpenCV e sulla procedura da attuare per utilizzare la libreria in ambiente Visual Studio; tali informazioni si possono reperire all'indirizzo [OCV] oppure consultando il libro [BK08]

5.1.2 Visual Studio

Il lavoro svolto si è rivelato in parte essere il *porting*, corredato dal riadattamento per aggiungere nuove caratteristiche, di codice MATLAB in codice *C++*; un lavoro di una certa complessità, soprattutto per la mole raggiunta dal codice, richiede stumenti di sviluppo dedicati a gestire efficacemente questa peculiarità. In questo contesto è necessario focalizzarsi sui cosidetti *IDE (integrated development enviroment)* che forniscono funzionalità di supporto per tutto il ciclo produttivo di un software: Visual Studio si inquadra esattamente in qeusto contesto. Non dà meno, l'elevata diffusione del formato di progetto generato dall'ambiente, condiziona la scelta.

Le caratteristiche maggiormente sfruttate, si possono ricondurre alle funzionalità di *debugging (possibilità di impostare break point e di indagare il valore delle variabili)* e di creazione di configurazioni personalizzate a seconda della fase dello sviluppo, senza dimenticare le potenzialità messe a disposizione da *compilatore e linker*.

eventualmente inserire le opzioni di maggior interesse di compilazione e linking in un'appendice

5.1.3 Matlab

Più che un ambiente, uno standard

MATLAB si può considerare praticamente uno *standard de facto* nell'ambito

della simulazione scientifica. Questo è stato il motivo principale che ha condotto all'uso di questo ambiente di sviluppo e simulazione, grazie al quale è stato possibile studiare i problemi affrontati e capirne la portata tecnica senza curarsi eccessivamente di problematiche troppo a basso livello.
La possibilità di prototipare velocemente ha fatto il resto.

5.1.4 OpenMP

OpenMP è un API che definisce un insieme di direttive `#pragma`, e quindi gestite a livello di compilazione tramite la disposizione di particolare istruzioni aggiuntive riferite al compilatore stesso, e le cui implementazioni forniscono supporto al calcolo su sistemi multiprocessore tramite il paradigma a memoria condivisa; in particolare, permette di gestire in modo abbastanza semplice (almeno in casi in cui il codice presenti forte località) il *multithreading* sia in riferimento a problemi espressi con logica *task parallel* che *data parallel*.
Per maggiori delucidazioni (si veda [OMP])

5.2 Struttura del codice

Fatta la doverosa premessa su strumenti utilizzati, ci si può inoltrare all'interno delle questioni più squisitamente legate ad aspetti tecnici ed implementativi e sembra un buon punto di partenza la descrizione del *repository* che definito dalla struttura del codice.

Fondamentalmente si possono separare le cartelle contententi la vera e propria implementazione, da quelle invece solo di servizio, evidenziate con colore diverso; si noti che dalla trattazione verranno tralasciate considerazioni legate a elementi della struttura generati da *Visual Studio*, elementi che solitamente constano degli eseguibili dell'applicazione e di altri *file* a supporto dell'ambiente di sviluppo.
I costituenti fondamentali per capire come sia stato organizzato il codice si possono evincere direttamente dal grafico in figura 5.1.

`include` la cartella contiene tutti gli header del progetto, comprese le dichiarazioni e le implementazioni delle funzioni *template*

`src` contiene tutti i sorgenti relativi al codice vero e proprio, per intendersi, i file `*.cpp`

`test` contiene le funzioni test utilizzate per valutare alcuni aspetti del codice

`img`

`segmentation_in` contiene le immagini offerte come input alla già menzionata *Segmentation pipeline*

`segmentation_out` contiene le immagini, risultato dell'elaborazione del suddetto modulo

`stereo_in` stiva le immagini destinate al *processing* del modulo *Stereo pipeline*

`stereo_out` analogamente a quanto già visto sopra, questa cartella è popolata dalle *disparity map* generata dalla computazione dovuta al modulo di cui sopra

`log` questa locazione viene utilizzata per produrre varie informazioni su file di testo, quali tempi di computazione, contenuto di vettori o matrici e informazioni di servizio. Un modulo organico che si occupi della gestione dei log non è tuttavia stato implementato

`conf` contiene, almeno idealmente, i file di configurazione dei vari moduli che compongono l'applicazione. Anche questa feature, seppur prevista, non è presente

`docs` contiene la documentazione del codice prodotto

5.3 Descrizione dei moduli software

Come ovvio, l'implementazione deve cercare di essere fedele, dal punto di vista della struttura e delle funzionalità, e con tutti i limiti del caso, a quanto congetturato e organizzato nella fase di design.
Peraltro, ad onor del vero, le due fasi realizzative si sono condizionate a vicenda alternando fasi di progettazione e strutturazione a fasi di costruzione effettiva, passando per alcune parantesi di *prototyping* al fine di cogliere certe problematiche non palesatesi in precedenza nella mente di chi scrive.
La presente sezione, si curerà di descrivere soprattutto le questioni legate alla libreria, più complesse, mentre gli aspetti inerenti l'applicazione, mera composizione procedurale delle funzioni realizzate, vengono solo menzionati, in quanto privi di aspetti implementativi di grande rilievo.

5.3.1 Utils

Composto dai file con suffisso (esclusa l'estensione) `utils`, questo modulo fornisce tutta una serie di funzioni, tipicamente piuttosto piccole e dalle funzionalità

mirate, di appoggio a funzioni più complesse: costituisce, in sostanza, il *core* della libreria.
Il testo che segue è volto a spiegare con maggiore dettaglio quanto realizzato

array_utils

Il file `array_utils.c` e relativo *header* `array_utils.h`, contiene alcune *utility* mirate a sfruttare in vari modi strutture dati di tipo `std::vector<T>`.
Si fanno notare specialmente le funzioni che, in un modo o nell'altro sono coinvolte attivamente nella computazione, lasciando in disparte invece quelle che si occupano di mansioni di servizio (log a riga di comando del valore degli elementi).
Si è parlato in precedenza del modulo allineatore, il quale effettua tra le altre cose, operazioni che portano alla riorganizzazione degli elementi di una PointCloud; chi si occupa di fare ciò è la funzione

```
void vectorPerm( const vector<int>& corresp_idxs,
                 const Mat& mask,
                 const vector<Vec3f>& src_vec,
                 vector<Vec3f>& dst_vec );
```

`Mat` e `Vec3f` sono rispettivamente la struttura dati utilizzata per trattare matrici generiche e il tipo per la specifica gestione di vettori tridimensionali `float`, forniti da *OpenCV*; `vectorPerm(...)` richiede tre ingressi:

- un vettore di interi che rappresenta la funzione di permutazione da applicare. Le celle che indicano uno scambio non valido, contengono valori negativi.
- una maschera binaria utile a filtrare ulteriormente eventuali scambi non validi
- il vettore da permutare

e un'uscita, il risultato dell'elaborazione. La natura dell'implementazione, prevede che `dst_vec` abbia le stesse dimensioni del vettore sorgente e questo pone una questione relativa alla possibile presenza di elementi non inizializzati in uscita; in questo caso è consigliabile impostare `dst_vec` già inizializzato, facendo diventare quest'ultimo, di fatto, un "tappa-buchi"; solo gli elementi corrispondenti ad un indice valido nel vettore `corresp_idxs` verranno modificati.
Ultima funzione di un certo interesse del modulo, è quella che genera il vettore di indici *random*, utilizzato specialmente nell'implementazione dell'algoritmo di *clustering*.

```
void getRandomVec( const int sample_num ,
                   const int max_val ,
                   int type,
                   vector<int>& sample_vec ,
                   const int min_val = 0)
```

dove

- `sample_num` è un parametro che quantifica la taglia del vettore di interi da generare
- `max_val` rappresenta invece il limite superiore al valore degli indici generati
- `type` è un intero, che prevede due valori validi secondo quanto definito all'interno dell `enum` anonima

  ```
  enum RANDOM_SAMPLING , PERIODIC_SAMPLING ;,
  ```

  e che permettono di condizionare la tipologia di campionamento, in accordo con quanto esposto nel Capitolo 4
- `min_val`, posto di default a zero, ha lo stesso signicato di `max_val`, ma imponendo un limite inferiore

in uscita viene prodotto il vettore di interi desiderato. Sembra utile infine esplicitare il codice che realizza il campionamento periodico

```
1  void getRandomVec( const int sample_num ,
2                     const int max_val ,
3                     int type,
4                     vector<int>& sample_vec ,
5                     const int min_val )
6  {
7    ...
8    RNG rand_gen( getTickCount() );
9    int random_elem = min_val;
10   ...
11   if( type == PERIODIC_SAMPLING ){
12
13     random_elem = rand_gen.uniform( min_val , max_val );
14     int delta = cvFloor((double)(max_val/sample_num));
15
16     for( int i = 0 ; i < sample_num ; i++ ){
17       try{
18         sample_vec.push_back(( random_elem+delta*i )%max_val);
19       }catch(...){}
20     }
```

```
21    }
22 }
```

Listing 5.1: Implementazione del random sampling

debug_utlis

Il codice in questione, ovvero il file `debug_utils.h` è un semplice *header* contenente alcune *macro* utili per investigare alcuni aspetti del codice attraverso log a riga di comando; le funzionalità risultano attiva solo qualora la *flag* `_DEBUG` sia attiva.

img_utils

La coppia di file `img_utils.cpp` e `img_utils.h`, fornisce alcune funzioni che permettono di aprire immagini, con tanto di controllo per verificare l'esistenza delle stesse, e funzioni utili a costruire delle maschere atte a visualizzare i risultati, ad esempio, della segmentazione.

math_utils

Il contenuto dei file `math_utils.cpp` e relativa intestazione si può considerare formato da utilità legate, come suggerito dal nome, a finalità matematiche: calcolo di deviazioni standard e di coefficenti di normalizzazione, nonché di *TAD*(*Truncated Absolute Difference*). Le funzioni che richiedono maggiore attenzione sono invece quelle imputate al computo della radice quadrata di una matrice. Il problema è intrinsecamente privo di soluzione unica, almeno nel caso generale, ma si dirime una volta posto il vincolo che la matrice coinvolta sia simmetrica (o meglio hermitiana; nel qual caso si rende tuttavia necessaria la gestione di valori nel dominio dei complessi) e definita positiva. In questo caso si può procedere mediante ortogonalizzazione (tramite decomposizione agli autovalori o *SVD*, numericamente più stabile) e computazione della radice quadrata degli autovalori ottenuti. Per capire quale fosse la strada più consona da percorrere, la decisione di implementare una versione per ogni tipo di decomposizione, è sembrata una scelta sensata; si è verificato tuttavia il permanere in entrambi i casi di errori di approssimazione anche non banali, che comunque non hanno compromesso, almeno nel caso specifico, il buon fine della procedura nella quale il calcolo doveva essere impiegato.

matrix_utils

In questo caso può essere utile commentare le funzioni

```
void computeAllPairDistMat( const Mat& point_mat ,
                            Mat& all_pair_mat ,
                            float sigma ,
                            int dist_type = EXP_EUCLID_DIST );

void computeSampledToOtherDist( const Mat& sampled_point_mat ,
                                const Mat& other_point_mat ,
                                Mat& sampled_to_other_dist_mat ,
                                float sigma ,
                                int dist_type = EXP_EUCLID_DIST);
```

la prima calcola la matrice delle distanze tra i punti contenuti in `point_mat`, posizionati uno per ogni colonna e le cui componenti siano in prossimità dell'indice di riga. Può gestire sia valori `float` che `double`, e il contenuto della matrice di uscita, `all_pair_mat`, alla posizione localizzata dall'indice i per le righe e da j per le colonne, si può esprimere come

$$\text{all\_pair\_mat}(i,j) = \exp(-d_c(\text{point\_mat\_col}(i), \text{point\_mat\_col}(j))/2\sigma^2)$$

dove d_c rappresenta la distanza euclidea tra i vettori rappresentati dalle colonne della matrice di ingresso.

La seconda funzione invece si occupa di produrre una matrice di distanze tra ogni punto appartenente ad un certo insieme ogni punto appartenente ad un insieme diverso dal primo, usando la stessa metrica espressa appena sopra. In questo caso la questione dell'allocazione della componente di uscita risulta invece più complessa, non perché l'operazione comporti complicazioni in sé , ma perché si è deciso di adottare un procedimento più sofisticato che potesse essere utile al fine di migliorare le prestazioni di alcune porzioni della libreria.
L'allocazione viene gestita come da copione in tutti i casi classici, ovvero in caso di matrice d'uscita non allocata oppure nel caso la stessa abbia taglia compatibile con la taglia corretta d'uscita ed infine nel caso in cui sia di dimensioni comunque incompatibili con la taglia corretta d'uscita, pari a `other_point_mat.cols` $\times$ `sampled_point_mat.cols`, dove `cols` rappresenta una variabile interna alla classe `Mat` di *OpenCV* che quantifica il numero di colonne della matrice in questione. Nella situazione in cui, invece, l'elemento `sampled_to_other_dist_mat` sia stato pre-allocato con numero di righe pari alla somma tra il numero di colonne di `other_point_mat` e di `sample_point_mat` e cardinalità dell'insieme delle colonne uguale al numero di colonne di `sampled_point_mat`.

Grazie a dio quest'estenuante trattazione volge al termine.

5.3.2 Alignment

La sezione 4.2.4, ha introdotto, con alcune considerazioni ad altro livello il blocco chiamato *Alignment module*, di cui ora vengono forniti alcuni particolari implementativi.
A rappresentare le coppie di punti coniugati, indipendentemente dal fatto che possano essere accoppiamenti validi o meno, ci pensa la `struct`

```
typedef struct PointCorrespondence{

  Point2i reference;
  Point2i target;

}PointCorrespondences;
```

mentre la funzione

```
void findPointCorrespondence(const cv::Mat& disparity_map,
  std::vector<PointCorrespondence>& correspondence_index,
  int dmax = 0,
  int type = 0 );
```

si occupa di popolare il vettore di corrispondenze per ogni pixel dell'immagine di riferimento; il fatto che quest'ultima possa essere riferita all'inquadratura destra oppure sinistra, ha condotto ad introdurre un parametro che discrimini tra le due situazioni, l'intero `type` infine, `dmax` è utile per lavorare su immagini che abbiano subito crop di colonne.

Chi invece si accolla l'onere di calcolare la funzione di permutazione da applicare alla nuvola di punti al fine di poter allineare i dati ricavati dalle immagini e dallo stereo, è

```
void pointCorrespToVectorCorresp(
  const std::vector<PointCorrespondence>& corresp_idx,
  const int num_row,
  const int num_cols,
  Mat& mask,
  std::vector<int>& vector_corresp_idx);
```

che fornisce anche un'informazione aggiuntiva formata da una maschera, la quale si occupa di segnalare le posizioni nelle quali sono stati rilevati accoppiamenti impossibili, ovvero caratterizzati dal fatto che il punto coniugato abbia coordinata u negativa.

Le funzioni implementate all'interno di `windowing.cpp` permettono la gestione di maschere che rappresentino zone di interesse, anche nel caso in cui queste ultime sforino parzialemente rispetto ai limiti dell'immagine. Una soluzione forse più semplice a questo problema è stata comunque individuata nell'uso della primitiva `copyMakeBorder` fornita da *OpenCV*, la quale forma un bordo attorno all'immagine e consentendo quindi di portare agevolmente a compimento quanto voluto.

5.3.3 Clustering

I file `clustering.cpp` e `clustering.h` contengono definizione ed implementazione dell'algoritmo di *Spectral Clustering* con approssimazione di Nystrom che, nel particolare, si richiama attraverso la seguente funzione:

```
void spectralClusteringNystrom1( Mat& point_matrix ,
                                 const int sample_number ,
                                 const int num_cluster ,
                                 float sigma ,
                                 Mat& class_vec );
```

Il parametro `point_matrix` è la matrice dei *feature-point*, posti uno per colonna, mentre gli altri parametri numerici sono utilizzati per la configurazione dell'algoritmo; `class_vec` contiene, sempre uno per colonna, le classi di appartenenza prodotto della segmentazione.
L'implementazione per come realizzata, non permette purtroppo l'uso agevole di metriche alternativa a quella implementata per costruire la matrice dei pesi del grafo da trattare; in questo senso, potrebbe essere interessante rivedere la realizzazione di questa porzione ai fini di renderla più interessante per un uso generico.

5.3.4 Filter

Composto dalle funzioni

```
void disparityMask(const Mat& disparity_map,
                   Mat& disparity_mask,
                   int dmin=0,
                   int dmax=255);

void zfilter(const Mat& cloud3d,
             Mat& cloud3d_mask,
```

```
            float z_near,
            float z_far);

void zfilter(const vector<Vec3f> cloud3d,
            Mat& cloud3d_mask,
            float z_near,
            float z_far);

void getFilteredMatrix(const Mat& src_mat,
                       const Mat& mat_mask,
                       Mat& filtered_mat);

void getFilteredPointCloud(const vector<Vec3f>& src_point_cloud,
                           const Mat& mat_mask,
                           vector<Vec3f>& filtered_point_cloud);
```

questo raggruppamento tende a realizzare le specifiche definite da *Filter module*, che ricordiamo prevedere una serie di funzionalità per generazione di maschere ed eliminazione di *outliers*.
Le prime tre definizioni, identificano le funzioni del primo tipo (generazione maschere) mentre le ultime due sono utili per ottenere strutture `Mat` oppure *PointCloud*, private degli elementi selezionati attraverso le impostazioni di filtraggio.
Entrando maggiormente nel dettaglio, `disparityMask(...)` è pensata per scandire i valori di una matrice formata da valori `CV_8U` (ovvero `unsigned char`) alla ricerca di elementi che sforino un limite inferiore e uno superiore; il verificarsi di questa evenienza in posizione (i, j) porta ad impostare a `false` la corrispondente locazione all'interno della maschera.

Similmente agisce `zfilter(...)`, che può essere invocata sia su `Mat` che su `vector<Vec3f>`, ponendo questa volta il controllo sulla terza coordinata spaziale di ogni elemento, secondo i vincoli imposti dai parametri `z_near` e `z_far`. Qualora si decidesse di lavorare su matrici, il non rispetto del vincolo che prevede che queste siano formate da 3 righe, una per la x, una per la y e una per la z, è punito con il lancio di un'eccezione.

Ciò che resta da analizzare, ovvero `getFiltered*(...)` non fà altro che iterate sulla struttura posta in ingresso controllando la corrispondente colonna della maschera; a seconda del fatto che quest'ultima presenti valore logico vero o falso per la locazione selezionata, copia o meno il contenuto della riga della matrice di ingresso o dell'opportuno elemento della nuvola di punti, nella prima posizione libera di una struttura di uscita. Si noti quindi che, generalmente, la taglia dell'output sarà diversa da quella dell'input.

5.3.5 Geometry

Geometry è il termine scelto per raccogliere le funzioni relative a quello che nel precedente Capitolo è stato definito come *Geometric module*. Comprende una sola funzione sovraccarica, situata all'interno del file `geometry_estimator.cpp`, con definizione in `geometry_estimator.h` la cui firma viene riportata sotto:

```
void geometryEstimator3D( const cv::Mat& intrinsic_params ,
                          const float baseline ,
                          const cv::Mat& disparity ,
                          const cv::Mat& disparity_mask ,
                          cv::Mat& cloud3d );

void geometryEstimator3D( const cv::Mat& intrinsic_params ,
                          const float baseline ,
                          const cv::Mat& disparity ,
                          const cv::Mat& disparity_mask ,
                          vector<cv::Vec3f>& cloud3d );
```

l'unica differenza tra le due versioni sopra, risiede nel fatto che, mentre la prima pone in uscita una matrice contenente le coordinate tridimensionali dei punti di una scena, la seconda fornisce invece una *PointCloud*, ovvero un `vector<Vec3f>` come formato di uscita dei dati.
Data per assodata la nota appena scritta, si precisa il fatto che nel seguito, aderentemente alla trattazione di questo specifico modulo software, ci si riferirà indistintamente alla generica funzione `geometryEstimator3D`. Fondamentamente, basandosi sui parametri estrinseci, codificati all'interno della struttura `intrinsic_params`, sul valore della baseline e sui valori di disparità contenuti in `disparity`, è possibile calcolare attraverso `geometryEstimator3D` la nuvola di punti tridimensionali associati alla scena; `disparity_mask` è invece una maschera che ha lo scopo di risparmiare il calcolo delle coordinate di punti non validi e permette di gestire la situazione per la quale si possono presentare valori della *disparity map* nulli, situazione che condurrebbe ad un risultato fondamentalmente non determinato (valori infiniti) della terna coinvolta.
Ultima nota implementativa, `intrinsic_params` dev'essere non singolare; qualora si dovesse verificare quest'evenienza, è previsto il lancio di un'eccezione.

5.3.6 Stereo

Stereo, omologo software di *Stereo module*, si occupa di questioni inerenti la stima di mappe di disparità e fornisce qualche funzionalità di supporto al trattamento di queste, in particolare la funzione che consente di operare il controllo di

consistenza conosciuto come *Left-Right cross- check*.
Il modulo consiste di due file sorgente + header i quali sono stati chiamati `disparity.cpp` e `windowing.cpp`, il secondo dei quali contiene un paio di funzioni di appoggio a Segment Support.

Si procede ora, alla descrizione di massima di qualche procedura per così dire interessante. Il cross-check è implementato da

```
void crossCheck( const Mat& ref_disp ,
                 const Mat& tar_disp ,
                 Mat& cross_checked ,
                 int type = LR_DISPARITY ,
                 int threshold = 1 );
```

e supporta il check dei valori sia nel caso in cui l'immagine di riferimento sia la destra sia nel caso sia la sinistra.

La porzione più importante nello sviluppo del modulo stereoscopico è certamente stata l'implementazione dell'algoritmo Segment Support, la cui dichiarazione è:

```
void segmentSupport( const Mat& reference_img ,
                     const Mat& target_img ,
                     const Mat& reference_seg ,
                     const Mat& target_seg ,
                     const SegmentSupportParams& params ,
                     int type ,
                     Mat & disparity_map );
```

dove sembra utile scrivere un piccolo approfondimento rispetto al significato di `SegmentSupportParams`, struttura che contiene i parametri da di inizializzazione legati all'algoritmo e `type` che ammette i valori

```
enum{ LR_DISPARITY = 1 , RL_DISPARITY = 0 }
```

e che consente di calcolare disparity da destra a sinistra e viceversa.

5.3.7 Test Functions

Una porzione alquanto importante, sia in termini di utilità ai fini tecnici sia in termini di lavoro profuso, è rappresentata dalle funzioni test, utilizzate per valutare in modo più o meno esaustivo il corretto comportamento del codice.
La struttura generale di queste è abbastanza semplice e incompleta, nel senso che non considera tutte le questioni che solitamente fanno capo al comparto test

di un software professionale.
Addentrandosi ora in qualche dettaglio, è possibile spiegare come sono state costruite le procedure in questione:

- le funzioni non prendono parametri in ingresso e ritornano valori `void` e gli scenari richiesti, ovvero i dati da porre in pasto alle funzioni da testare, sono costruiti all'interno degli stessi test. Questo ha forzato in qualche modo a generare moduli piuttosto piccoli e semplici, facili da testare, rendendo maggiormente agevole il lavoro.

- le intestazioni delle funzioni test sono tutte del tipo:

  ```
  testNomeFunzioneDaTestare()
  ```

  ad esempio, per testare la funzione che decide se un vettore di interi è una permutazione, chiamata `isPerm(...)`, si usa:

  ```
  void testIsPerm()
  ```

- nella loro costruzione, le funzioni test sono state tendenzialmente pensate per capire le reazioni delle procedure sottoposte a prova se sollecitate da input di varia natura, cercando dunque di individuare tipologie di ingresso non valide nel senso del tipo di dato immesso, oppure ingressi che presentassero tipi validi ma che potessero condurre a conclusioni errate e quindi per dedurre eventuali errori logici nella scrittura dei componenti dei moduli.

- per validare le prove al punto precedente si sono usati semplici `print` a console, oppure i più sofisticati meccanismi di lancio di eccezione forniti dal linguaggio o, ancora, costrutti `assert(...)`, presenti nativamente in linguaggio C, piuttosto che le dedicate `CV_Assert(...)` fornite da *OpenCV*.

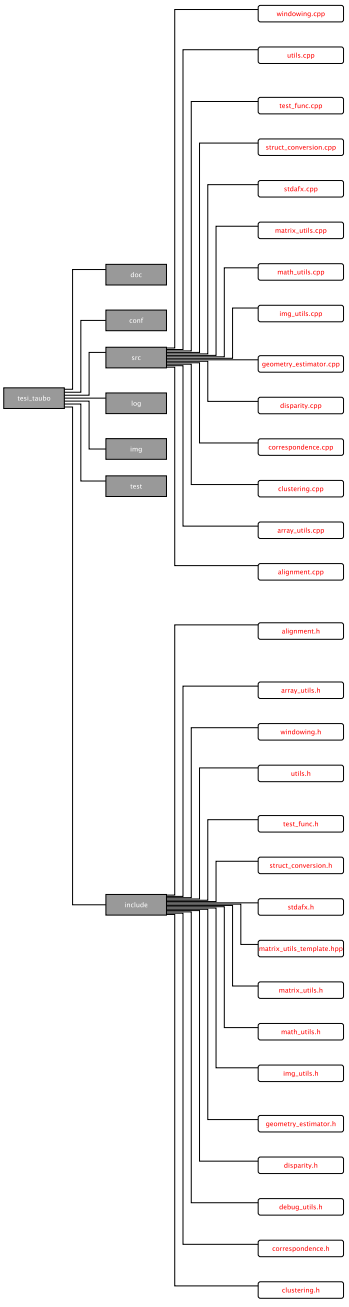


Figura 5.1: struttura del codice

Capitolo 6

Test ed analisi dei risultati

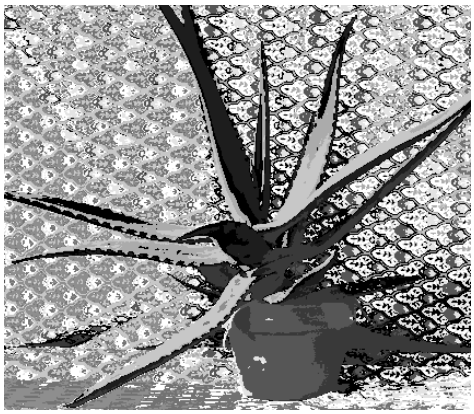
Questo capitolo riassume i risultati ottenuti dalle prove effettuate, sulla sola immagine Aloe, reperibile all'indirizzo [SMB] e cerca di porre qualche riflessione per eventuali sviluppi futuri.

6.1 Segmentazione

Per quanto riguarda la segmentazione effettuata sul vettore 9-dimensionale contenente sia geometria, sia colore, sono state effettuate delle prove al variare del numero dei cluster desiderati, e provando diversi valori del parametro λ, il quale influisce sulla geometria. Per effettuare tutte le prove, si è fatto uso del dataset reperibile su [SMB]. La combinazioni delle due informazioni, permette di compensare ai difetti derivanti dell'uso di solo colore, se per esempio la scena presenta porzioni omogenee cromaticamente ma distinguibili una volta disponibile la profondità, e con un ragionamento simile, si possono immaginare i vantaggi della fusione rispetto all'utilizzo di sola geometria. In Figura 6.1 si possono notare i risultati ottenuti segmentando l'immagine attraverso diversi insiemi di nuvole di punti. La terza immagine, risultato del processo di segmentazione effettuato su colore e geometria, è stato ottenuto impostando a 5 il numero di cluster, con valore λ pari a 7.

Pur notandosi una migliore separazione degli oggetti, le maschere prodotte dalla segmentazione risulatano poco pulite; sarebbe sufficiente per ottenere un risultato più apprezzabile, inserire all'interno della pipeline un modulo in grado di individuare le componenti connesse e riempire opportunamente quelle di cardinalità inferiore ad un certo limite.

(a) *Immagine originale*



(b) *Immagine segmentata basandosi su solo colore*

(c) *Immagine segmentata colore+geometria*

Figura 6.1: Risultati della segmentazione

6.2 Stereo

Le prove effettuate per capire l'evolversi del calcolo della *disparity map*, all'aumentare delle iterazioni sono riportate in Figura.
L'immagine ha dimensione, in pixel, pari a 427×370 con range di disparità $[0 \div 85]$; i parametri utilizzati sono:

Per il clustering

- numero di campioni $=$ 64
- numero di cluster $=$ 16
- $\sigma = 4.0$
- segmentazione effettuata sulla nuvola di punti rappresentante l'accoppiamento *reference-target* fornito dallo stereo, senza considerare la componente strettamente geometrica

Per l'algoritmo stereo

- dimensione della finestra $=$ 19
- $\sigma_c = 22.0$
- k_TAD $=$ 80

inoltre, durante la fase di test si è verificato come l'utilizzo dello spazio colore *RGB* a sostituzione di *CIELab*, abbia fornito, almeno apparentemente un miglioramento delle qualità della mappa generata, evidenziando una maggiore precisione, specie in prossimità dei bordi. Oltre a questo, si è reso necessario suddividere la tipologia di output della segmentazione per offrire un opportuno ingresso all'algoritmo stereo; si è reso infatti necessario, contrariamente a quanto accaduto Nella Tabella 6.1 sono riportati alcuni dati che descrivono il comportamento dell'algortimo stereo, all'aumentare delle *run*, ovvero all'aumentare del numero di iterazioni della procedura; sono stati inclusi RMSE (root mean square error), calcolato rispetto ad una mappa di riferimento, detta *ground-truth*, e viene menzionato inoltre il numero di pixel per i quali è stata rilevata inconsistenza attraverso la procedura di controllo della consistenza.
Le prove sono state effettuate generando una maschera, applicata sia nella mappa *ground-truth* sia a quella ricavata dalla computazione, che rilevasse i punti a disparità sconosciuta (ovvero a disparità nulla) uniti a quelli per i quali il controllo di consistenza desse responso negativo (inconsistenza tra mappa destra e mappa sinistra per lo specifico pixel) e l'errore è stato calcolato ignorando i punti per i quali la maschera fosse falsa.
Le Figure da 6.2 a 6.6 mostrano l'evoluzione delle *disparity map* destra-sinistra con il procedere delle iterazioni.

#run	RMSE	#bad pixel	%bad pixel
0	1.70871		
1	1.56725	36969	23.4%
2	1.56384	36590	23.2%
3	1.54935	36903	23.4%
4	1.54240	36516	23.1%

Tabella 6.1: Tabella che riassume i risultati conseguiti nei test

6.3 Breve analisi sui risultati e sviluppi futuri

La quantità di prove effettuate, con particolare riferimento alla porzione dedicata all'affinamento dello stereo, non è sicuramente sufficiente a tratte delle conclusioni scientifiche: si rende infatti necessario variare il dataset di immagini sul quale effettuare le misure e, come insegna la letteratura [SS02], sarebbe certamente stato utile provare la qualità delle mappe di disparità ottenute operando su aspetti come la capacità di gestire opportunamente le occlusioni.
Nonostante ciò, sembra intuirsi la possibilità di ottenere dei risultati interessanti, alla luce del fatto che i pochi dati ottenuti sono confortanti.

Oltre a queste considerazioni, non bisogna dimenticare come la stima dei parametri da utilizzare, e questa considerazione vale sia per la porzione di calcolo stereo sia per la pipeline di segmentazione, sia ancora effettuata grazie a prove empiriche e per nulla automatizzata; capire come e se sia possibile individuare una procedura algoritmica in grado di adempiere a questa attività, potrebbe essere uno sviluppo interessante.
Altro aspetto assolutamente fondamentale e che quindi si ritiene opportuno ridabire riguarda l'indagine più approfondita rispetto alla propagazione dell'errore iterando il calcolo delle corrispondenze, o alla sua reiezione, capendo quali siano gli scenari per i quali si presenta un situazione piuttosto che un'altra.
Il clustering richiede, da parte sua, uno studio sulle metriche da utilizzare, anche se la questione potrebbe essere superflua ha rappresentato comunque un motivo di interesse per l'autore. Non da meno, la scelta del numero di segmenti da ricavare nonché uno studio accurato sulla stima del coefficente che pesa il contributo della geometria rispetto al colore.
Oltre all'aspetto scientifico, le questioni implementative possono diventare rilevanti: la possibilità di parallelizzare il codice, magari per venire incontro all'uso, sempre più massiccio nell'ambito dell'elaborazione delle immagini, di unità grafiche; una ristrutturazione del codice prodotto, al fine di renderlo più organico e magari più generico, a il passaggio, almeno per le porzioni che meglio si addicono,

alla libreria *PCL* la quale dovrebbe inoltre migliorare l'integrazione con *OpenCV*, potrebbe costituire un'alternativa interessante ed adeguata al contesto operativo.

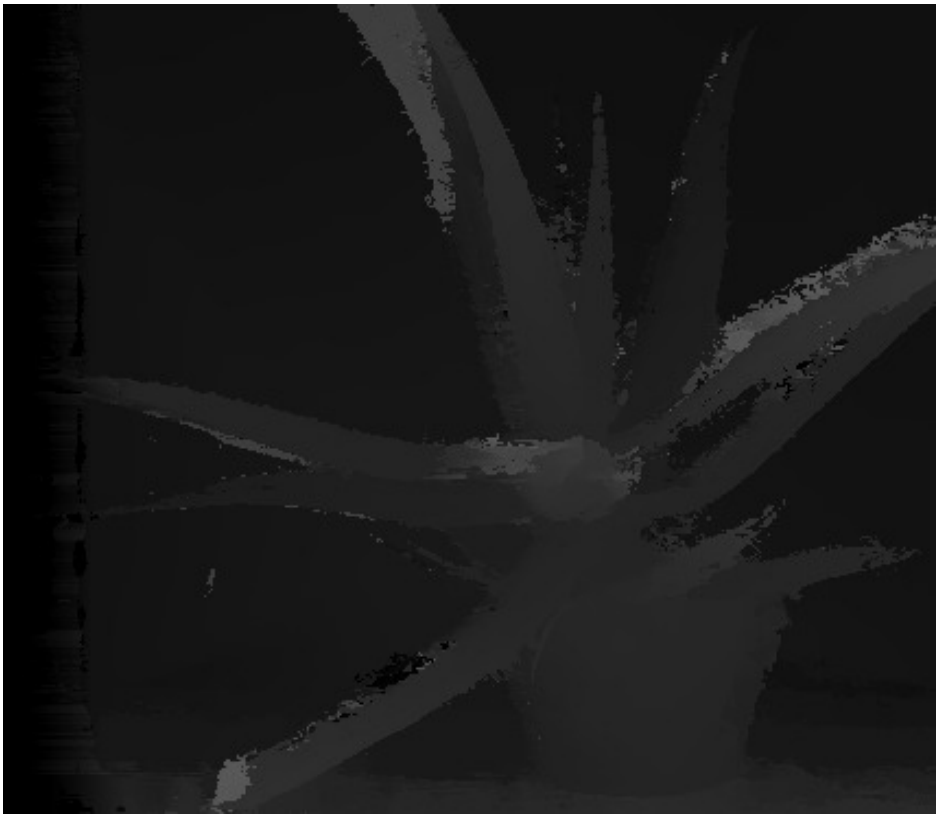


Figura 6.2: LR disparity map run 0

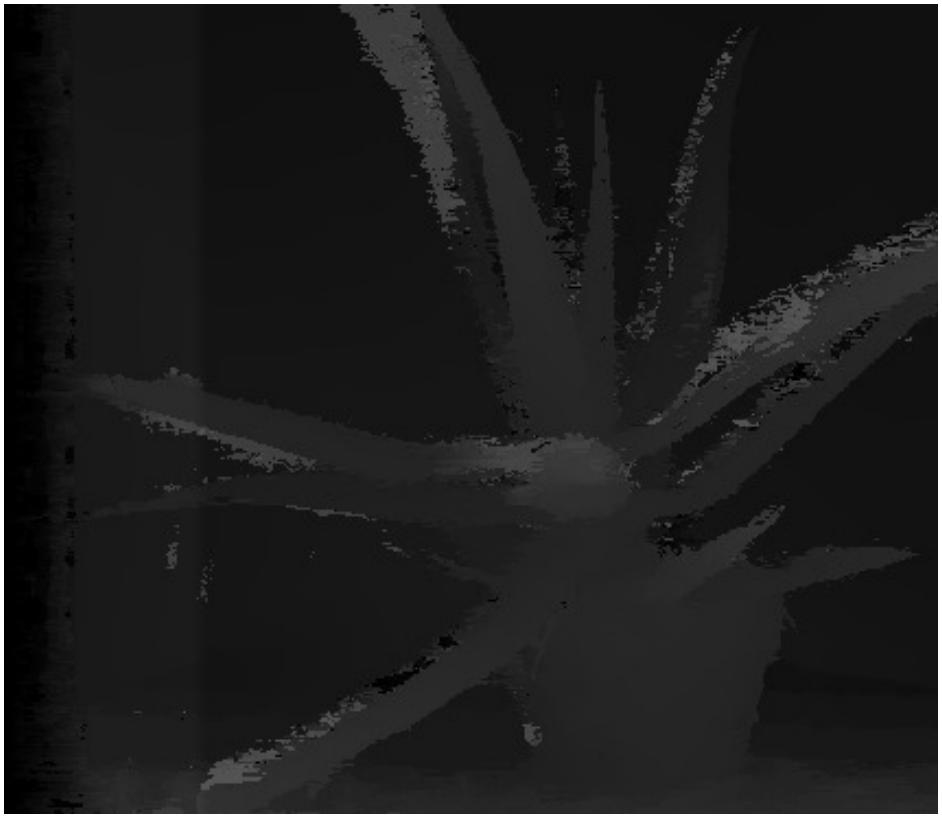


Figura 6.3: LR disparity map run 1

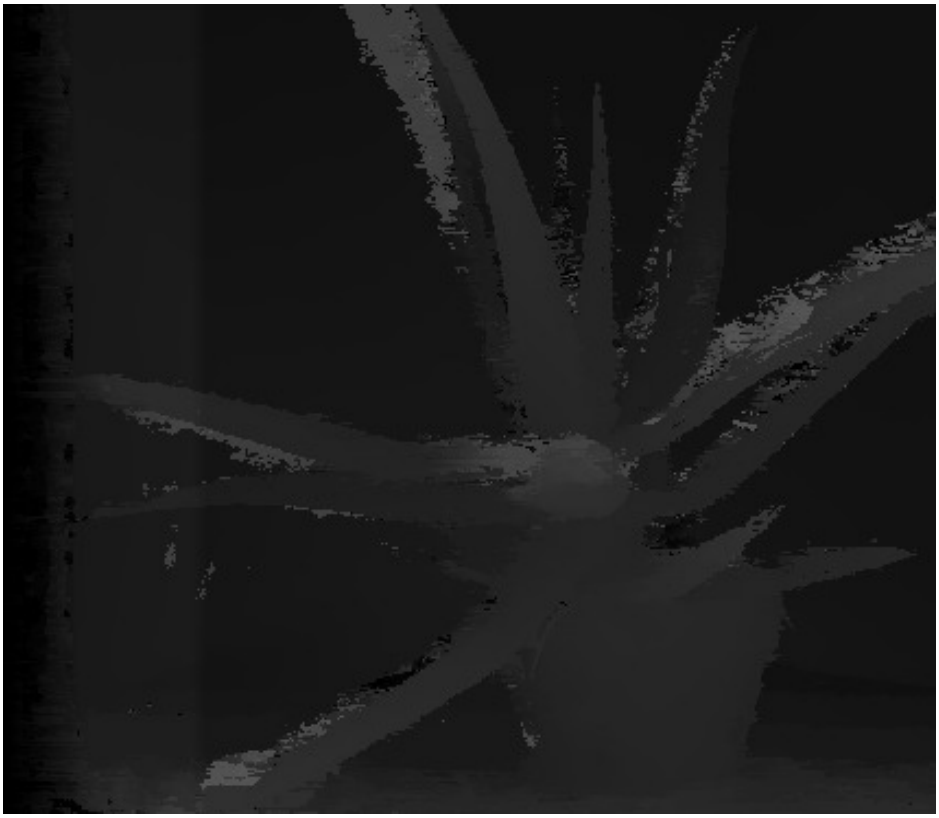


Figura 6.4: LR disparity map run 2

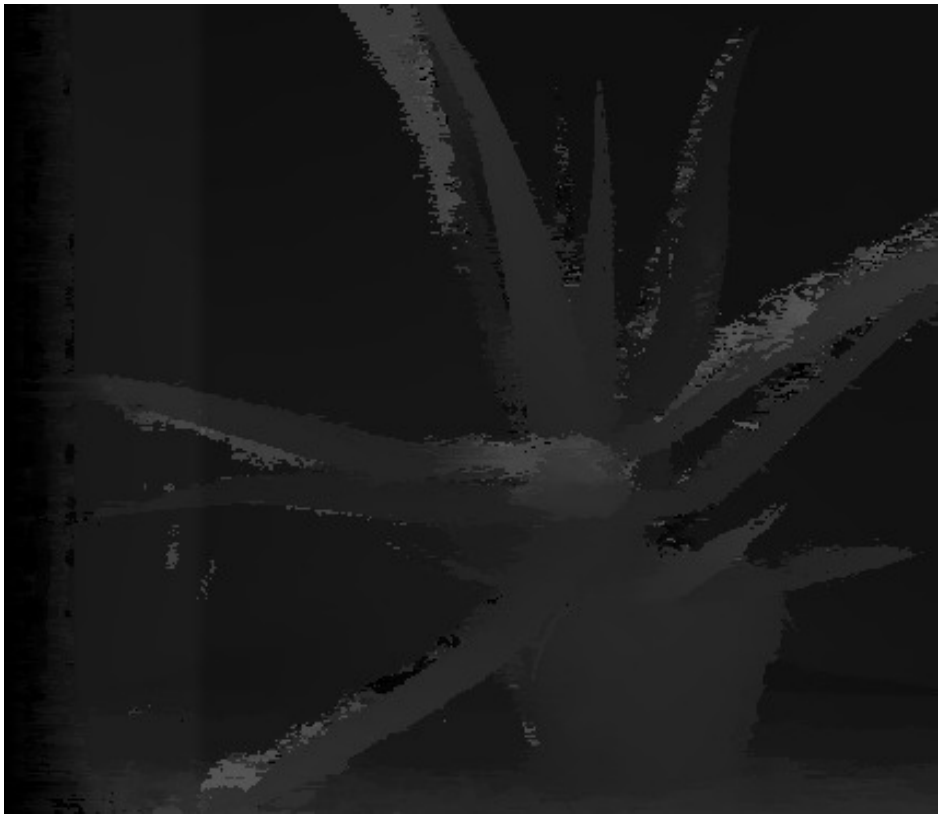


Figura 6.5: LR disparity map run 3

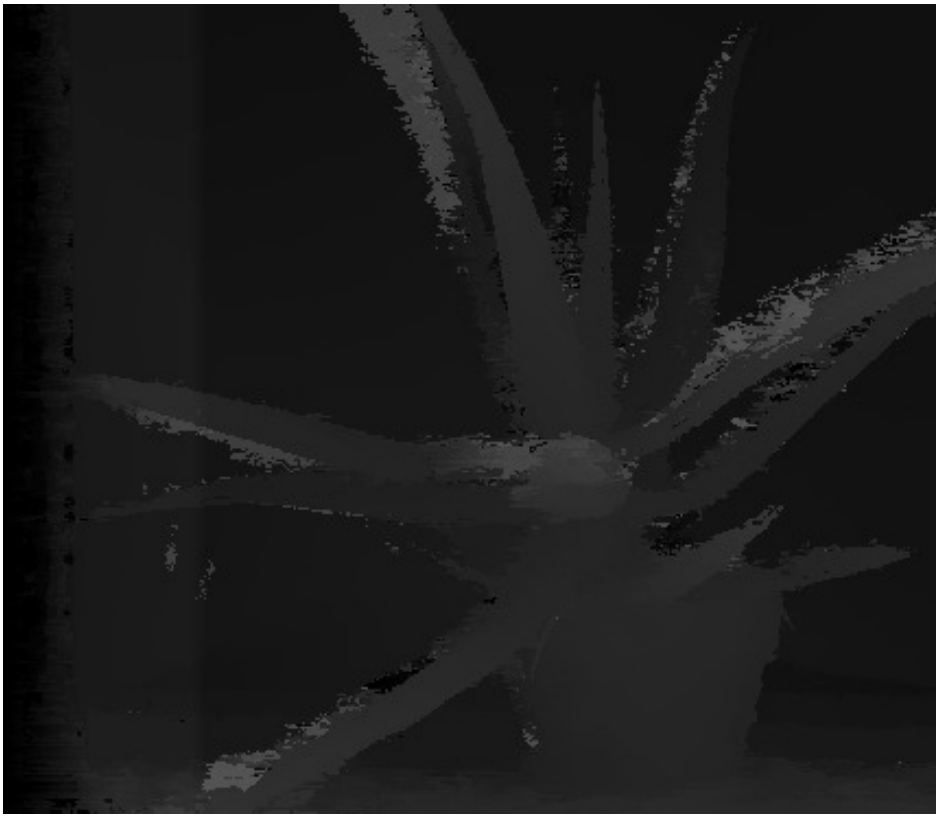


Figura 6.6: LR disparity map run 4

Conclusioni

Come già anticipato, le prove effettuate non sono da ritenersi abbastanza approfondite ed in quantità tale da poter portare ad una conclusione oltre ogni ragionevole dubbio. Tuttavia alcuni elementi lasciano intuire possibili interessanti orizzonti che richiedono comunque un ulteriore lavoro di indagine e sviluppo, nonché riscrittura.

Bibliografia

[BK08] G. Bradski and A. Kaehler. *Learning OpenCV: computer vision with the OpenCV library*. O'Reilly Series. O'Reilly, 2008.

[DMZC11] Carlo Dal Mutto, Stefano Mattoccia, Pietro Zanuttigh, and Guido Maria Cortelazzo. Scene segmentation from stereo data. In *3DIMPVT*, Hangzhou, China, May 2011.

[FBCM04] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the nystrom method. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(2):214 –225, feb. 2004.

[OCV] OpenCV Wiki web page. `http://opencv.willowgarage.com/wiki/`.

[OMP] OpenMP api specification official web site. `http://openmp.org/wp/`.

[SM00] Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(8):888 –905, aug 2000.

[SMB] Middlebury Stereo web page. `http://vision.middlebury.edu`.

[SS02] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47:7–42, April 2002.

[Sze10] R. Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer, 2010.

[TMS07] Federico Tombari, Stefano Mattoccia, and Luigi Di Stefano. Segmentation-based adaptive support for accurate stereo correspondence. Technical report, 2007.