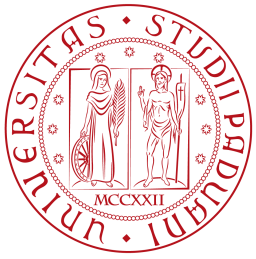


Università degli Studi di Padova
Facoltà di Ingegneria
Laurea Magistrale in Ingegneria delle Telecomunicazioni

Approccio iterativo per la soluzione congiunta dei problemi della segmentazione e della visione stereoscopica

Enrico Claudio Deambrosis

Relatore:

Prof. Pietro Zanuttigh

Correlatore:

Ing. Giampaolo Pagnutti

Luglio 2014

Ai miei genitori, che mi hanno accompagnato attraverso i molti insuccessi e gioito con me delle vittorie. Spero un giorno di riuscire a dimostrare di aver meritato tutto il vostro supporto, grazie.

Sommario

Questa tesi ha come obbiettivo il miglioramento reciproco di due aspetti correlati della visione computazionale: la segmentazione e la stereopsi computazionale. Si cercherà pertanto di effettuare una segmentazione di immagini avvalendosi anche di informazioni provenienti dalla geometria della scena e di calcolare mappe di disparità con un algoritmo stereo basato sulla segmentazione, con l'ausilio delle immagini segmentate ricavate al passo precedente, iterando il processo per cercare di ottenere risultati sempre migliori per entrambe le tecniche. L'unione di questi due aspetti riflette infatti l'intento di avvicinarsi al metodo naturale con cui l'uomo riconosce visivamente gli oggetti: tramite il colore e la geometria; mentre iterando il processo si vuole provare a dotare, in un certo senso, la macchina di una sorta di esperienza della scena per cercare di migliorare i suoi risultati sulla base di dati precedentemente trovati.

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Segmentazione e Visione Stereoscopica Computazionale

Il sistema ottico umano non ha difficoltà ad elaborare e comprendere un'immagine bidimensionale separando e distinguendo gli oggetti che la compongono e al tempo stesso riconoscere la forma, la profondità e la struttura tridimensionale dei vari elementi. Tutto questo grazie a varie informazioni, provenienti dai due occhi, che implicitamente elaboriamo e uniamo nel cervello come il colore, la luce e le ombre, i riflessi ma anche attraverso l'esperinza che abbiamo di un oggetto o di una situazione nota.
Lo scenario è totalmente diverso se cerchiamo di ricreare la stessa profonda comprensione di un'immagine attraverso una macchina e questo per l'estrema difficoltà di riuscire a convertire, in numeri e formule, tutte le informazioni che invece noi assimiliamo con un semplice sguardo.
Due campi ampliamente studiati della visione computazionale che hanno questo obbiettivo sono la *segmentazione* e la *visione stereoscopica computazionale*. L'idea portata avanti in questa tesi è che potrebbe non essere proficuo, l'analisi delle immagini considerando separatamente segmentazione e stereopsi computazionale; ognuna delle due tecniche può infatti aiutare e migliorare l'altra nella comprensione di qualsiasi scena presa in considerazione.

1.1 Segmentazione

La segmentazione si pone il problema di riconoscere e separare le regioni che compongono un'immagine bidimensionale basandosi sull'informazioni di colore, intensità o trama ricavate dalla stessa; più precisamente raggruppa i pixel dell'immagine etichettandoli in modo tale che se condividono una determinata etichetta (in generale si tratta di un numero as-

sociato a un colore o spesso a una tonalità di grigio) hanno in comune certe caratteristiche visive e appartengono allo stesso gruppo, detto anche *cluster*, che nel caso ideale corrisponde a un oggetto della scena (fig 1.1).

(a) Immagine a colori (b) Immagine segmentata

Figura 1.1: Esempio segmentazione

Le tecniche e gli algoritmi sviluppati negl'anni sono molteplici, ampiamente studiati e basano la loro metrica di separazione delle regioni su fattori come il colore, l'intensità, la spazialità, i contorni, la probabilità o combinazioni di questi, ma nonostante questo la perfetta segmentazione di un'immagine rimane ancora, in un caso generale, un obbiettivo lontano a causa di situazioni ambigue che la segmentazione da sola non può risolvere; per esempio due oggetti dello stesso colore ma su piani differenti non sono riconosciuti come distinti da una tecnica di segmentazione classica.

1.1.1 Spectral Clustering

In questa tesi si è affrontato il problema del raggruppamento dei pixel, al fine della segmentazione, utilizzando lo *Spectral Clustering*, tecnica che ha le sue fondamenta nella segmentazione basata sui grafi; rappresenta infatti gli elementi da raggruppare come vertici v di un grafo $G = (V, E)$, dove gli archi pesati $w(v, u)$ uniscono i vertici adiacenti secondo un criterio di similarità prestabilito. Nella teoria dei grafi si definisce *taglio* la somma del peso degli archi rimossi per creare una partizione disgiunta del grafo in due insiemi A e B tale che $A \cup B = V$ e $A \cap B = \emptyset$:

$$cut(A, B) = \sum_{v \in A, u \in B} w(v, u) \tag{1.1}$$

Nel caso della segmentazione con spectral clustering l'obbiettivo è dividere il grafo in n sottografi e minimizzare il taglio massimo che li partiziona. Studi precedenti dimostrano come il taglio definito dall'equazione 1.1 porta verso una partizione che separa dal contesto piccoli gruppi di nodi isolati, non arrivando ad ottenere la partizione ottima; per ovviare a ciò Shi e Malik [1] propongono una nuova misura per partizionare i grafi che chiamano

taglio normalizzato:

$$Ncut(A,B) = \frac{cut(A,B)}{assoc(A,V)} + \frac{cut(A,B)}{assoc(B,V)} \tag{1.2}$$

dove $assoc(A,V) = \sum_{v \in A, t \in V} w(v,t)$ è la somma dei pesi di tutti gli archi che collegano tutti i vertici in A con tutti i vertici del grafo, inserendo così un parametro di similarità non solo tra nodi ma tra insiemi di nodi, considerando il problema ad un livello più alto. Purtroppo questa scelta ha l'inconveniente che voler minimizzare il taglio normalizzato rientra tra i problemi NP-completi, e quindi richiede un'approssimazione.
Lo spectral clustering basa i suoi metodi di risoluzione sugli autovettori e autovalori di una matrice NxN, dove N è il numero di pixel dell'immagine, derivata dalla matrice che descrive l'affinità tra i cluster. Shi e Malik arrivano a una soluzione approssimata per il caso di due cluster (estendendolo poi ricorsivamente per i casi con più di due insiemi di nodi) facendo una stima dell'autovettore corrispondende al secondo più piccolo autovalore del Laplaciano normalizzato L, definito come:

$$L = D^{-1/2}(D - W)D^{-1/2} \tag{1.3}$$

dove D è la matrice diagonale con valori sulla diagonale pari al grado dei nodi e $W \in \mathbb{R}^{NxN}$ è la matrice contenente i valori dei pesi degl'archi del grafo considerato.
Risolvere questo tipo di problemi di autovalori per tutti gli autovettori porta a una complessità computazionale $O(N^3)$, impraticabile quando il numero di nodi sono i pixel di un'immagine; perciò gli autori dell'articolo [1] fanno tre ipotesi fondamentali per la riuscita della loro approssimazione:

1. I grafi sono spesso solo localmente connessi e il sistema di autovettori è in questo caso molto sparso.

2. Solo i primi pochi autovettori sono necessari per la partizione del grafo.

3. La precisone richiesta per gli autovettori è bassa.

Il fine ultimo di queste ipotesi è abbassare il più possibile i valori diversi da zero nella matrice W, la quale considera tutti gli archi pesati da ogni nodo verso ogni altro nodo: in un'imagine creata da una fotocamera digitale con risoluzione di qualche megapixel comporta l'elaborazione di numeri come 10^{12} archi pesati. L'approssimazione invece considerata in questa tesi è quella fornita dal metodo di Nystrom [2] che si basa sul trovare una soluzione completa al problema della partizione dell'immagine, considerando solo un campione di pixel molto minore del totale ed estrapolando da esso le informazioni globali. Invece di considerare l'affinità di tutti i pixel dell'immagine con una lista di centri dei cluster, il metodo di Nystrom li compara a un piccolo insieme di punti scelti arbitrariamente

e così facendo la complessità dell'operazione, dato un determinato numero di campioni, scala linearmente con la risoluzione dell'immagine; al contempo, prendendo in esame un numero di campioni pari circa all'1% del numero di pixel dell'immagine, i risultati ottenuti con questo metodo sono comparabili con quelli ottenuti considerando una matrice W densa.

1.2 Visione stereoscopica computazionale

La stereopsi computazionale è il processo che si prefigge di ottenere la struttura tridimensionale sfruttando le informazioni ricavate da due o più immagini bidimensionali, con angolazioni diverse, di una stessa scena. Considerando il caso in cui si prendono in considerazione due immagini, quello che si vuole ricreare è lo stesso principio che porta il sistema di visione umana a distinguere la profondità degli oggetti osservati, cercando di convertire in formule le problematiche di questo processo. Quello che si vuole ottenere sono, date una coppia di immagini chiamate immagine *referece* e *target*, delle *disparity map*: immagini che associano ad ogni pixel una determinata tonalità di grigio associata alla distanza del pixel dalle fotocamere(fig1.2).

(a) Immagine a colori (b) Disparity map

Figura 1.2: Esempio visione stereoscopica computazionale

1.2.1 Calcolo delle corrispondenze

A differenza del cervello umano che elabora all'unisono l'immagine proveniente dall'occhio sinistro e destro all'istante, un primo problema che si incontra nell'elaborazione delle immagini da parte di un computer è il *calcolo delle corrispondenze* tra i punti delle due immagini (fig.1.3). Far ciò vuol dire riconoscere che due punti su due immagini distinte sono proiezione dello stesso punto della scena sfruttando determinati vincoli che rendono il problema trattabile; il più importante fra questi è il vincolo epipolare, il quale afferma che il corrispondente di un punto in un'immagine può trovarsi solo su una retta (retta

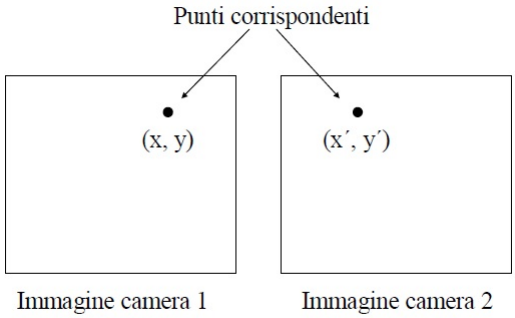


Figura 1.3: Corrispondenza di punti in una coppia stereo di immagini

epipolare) nell'altra immagine, e che quindi rende la ricerca delle corrispondenze unidimensionale invece che bidimensionale. Una tale coppia di punti è detta coppia coniugata e la differenza(inteso come vettore) tra i due punti immaginando di sovrapporre le immagini è detta disparità. La ricerca delle corrispondenze equivale anche al calcolo della disparità per tutti punti di un'immagine rispetto all'altra, ottenendo così una mappa di disparità, sinonimo anche di mappa di profondità.
Altri vincoli, oltre al vincolo epipolare, che possono essere utili nel calcolo delle corrispondenze sono, seguendo la lista fatta da Fusiello[3]:

- **vincolo di somiglianza**: derivante dal fatto che le immagini stereo sono molto simili e se queste sono anche rettificate, il loro cambiamento è solo una traslazione laterale rispetto alla scena; è naturale ritrovare all'interno delle due immagini, particolari simili.

- **vincolo di liscezza**: si suppone in questo caso che se gli oggetti inquadrati hanno superfici piane, la differenza di profondità fra due punti vicini, appartenenti alla stessa superficie in questione, sia minima se non nulla.

- **unicità**: Questo vincolo esprime la univocità che ogni punto di un'immagine dovrebbe avere nell'accopiamento con il coniugato nell'altra; nella pratica della ricerca però si vedrà che spesso si trovano accoppiamenti non univoci tra i punti delle immagini e ci si porrà il problema della scelta del giusto accoppiamento.

- **ordinamento monotono**: l'ultimo vincolo esprime il concetto secondo cui, in generale, se un oggetto si trova alla destra o sinistra di un altro in un'mmagine, continuerà a giacere alla sua destra o sinistra anche nell'immagine corrispondente; detto ciò, esiste però un cono d'ombra fra le due immagini in cui questo vincolo fallisce se non è presente in corrispondenza una superficie opaca e coesa.

Fondamentale inoltre nel calcolo delle corrispondenze è l'assunzione che le immagini non siano troppo diverse l'una dall'altra ma nonostante quest'ipotesi e i vincoli, un grosso probema nella risoluzione degli accoppamenti sono i falsi accoppiamenti cioè la non univocità delle corrispondenze fra punti delle due immagini; altri problemi ancora sono le occlusioni, che causano che porzioni della scena siano presenti solo in una delle immagini, la diversa intensità della luce percepita dalle due fotocamere e la distorsione prospettica che fa apparire un oggetto diverso nelle due proiezioni sulle immagini. Fatto interessante è che tutti questi problemi si aggravano tanto più quanto più le fotocamere sono distanti, ma d'altra parte, per avere una disparità significativa, le fotocamere devono essere ben separate tra loro.
I metodi presenti in letteratura per il calcolo delle corrispondenze cercano di accoppiare i pixel delle due immagini sfruttando alcuni dei vincoli citati in precedenza dividendosi in due macrofamiglie:

- **metodi globali**: impongono vincoli a tutta la retta orizzontale passante per il pixel in esame o addiritura a tutta l'immagine; sfruttano i vincoli disponibili in modo non locale per ridurre la sensibilità a regioni per le quali l'accoppiamento fallisce, come per le regioni uniformi o a causa delle occlusioni. Ne risulta però un problema di ottimizzazione che tipicamente ha un costo computazionale maggiore rispetto ai metodi locali che sono quasi sempre preferiti ai globali.

- **metodi locali**: impongono vincoli solo a un piccolo numero di pixel che circondano i punti da accoppiare. Questi metodi si basano principalmente sull'accoppiamento tra finestre di pixel nelle due immagini: considerata una finestra che circonda il pixel che si vuole accoppiare nell'immagine reference, si cerca nell'immagine target la finestra della stessa dimensione che più c'assomigia secondo una determinata metrica di giudizio. Sfruttando la geometria epipolare, le finestre sono da ricercare sulla retta epipolare del pixel in esame e le metriche che si possono scegliere sono svariate e le principoali sono basate sulla correlazione, sulla differenza di intensità o su operatori di rango. Per questi medoti la scelta della grandezza della finestra è cruciale e porta ad dover affrontare il compromesso fra accuratezza e affidabilità; infatti per finestre troppo piccole il rapporto tra la variazione d'intensità (segnale) e rumore è basso e la disparità che si ottiene è poco affidabile, ma viceversa, con finestre troppo grandi, si rischia che la finestra copra porzioni della scena in cui varia la profondità, producendo un sicuro errore nel calcolo della disparità, dal momento che non esisterebbe una disparità attribuibile a tutta la finestra.

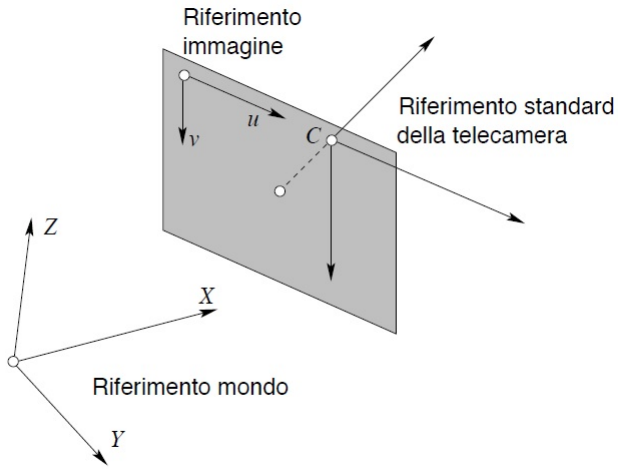


Figura 1.4: Sistemi di riferimento

1.2.2 Triangolazione

Noti gli accoppiamenti tra i punti delle due immagini e nota la posizione reciproca delle fotocamere è possibile ricostruire la posizione nella scena dei punti che sono proiettati sulle due immagini. Questo processo di triangolazione necessita della calibrazione dell'apparato stereo, ovvero del calcolo dei *parametri intrinseci*: parametri che tengono conto della traslazione del centro ottico, della riscalatura indipendente degli assi u e v e di un eventuale angolo (diverso da 90°) tra gli assi u e v; inoltre della conoscenza della posizione reciproca delle fotocamere, ovvero dei *parametri estrinseci*, per tenere conto del fatto che, in generale, il sistema di riferimento mondo non coincide con il sistema di riferimento standard della fotocamera ed è quindi necessario introdurre una trasformazione rigida che lega i due sistemi di riferimento(fig. 1.4).
Con queste informazioni è possibile calibrare, attraverso passaggi noti in letteratura, il sistema delle due fotocamere in modo tale da ricondursi a un caso semplificato del problema equivalente ad avere le fotocamere parallele ed alineate. Con queste semplificazioni ci troviamo nella situazione della figura 1.5, in cui la disparità è puramente orizzontale. Fissato il riferimento mondo solidale con la fotocamera di sinistra, possiamo scrivere, mediante semplici considerazioni sulla similitudine dei tringoli, le seguenti equazioni di *proiezione prospettica*:

$$\begin{cases} \frac{f}{z} = \frac{-u}{x} \\ \frac{f}{z} = \frac{-u'}{x-b} \end{cases} \tag{1.4}$$

da cui si ottiene:

$$z = \frac{bf}{u' - u} \tag{1.5}$$

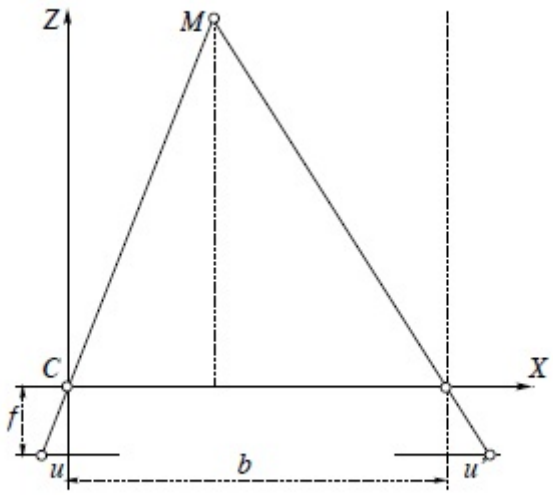


Figura 1.5: Triangolazione stereoscopica, caso semplificato con fotocamere parallele e allineate

La precedente equazione 1.5 ci suggerisce che è possibile ricavare la terza coordinata z, noti la geometria del sistema stereo ,ovvero la lunghezza della *linea di base* b e della *focale* f in questo caso, e la disparità $u^{'} - u$. Si capisce dopo questa osservazione come il calcolo di una disparity map, basato sul calcolo di $u^{'} - u$, avendo a disposizione i valori di b e f, sia equivalente al calcolo di una depth map, basato sul calcolo di z.
Si vede anche che b si comporta come un fattore di scala: la disparità associata ad un punto della scena fissato, dipende in modo diretto da b e inoltre se il parametro b è incognito è possibile la ricostruzione della struttura tridimensionale a meno di un fattore di scala.

1.3 Segment Support

La segmentazione può aiutare indubbiamente il processo del calcolo delle corrispondenze, evitando in molti casi, con l'informazione proveniente dal colore e dall'appartenenza a un cluster, l'errore dei falsi accoppiamenti e degli accoppiamenti non univoci. In letteratura sono presenti diversi metodi che si avvalgono di algoritmi stereo basati sulla segmentazione però, prima dell'articolo di Tombari et al. [4], la segmentazione era stata effetuata solo sull'immagine reference e le finestre usate per il calcolo delle corrispondenze utilizzavano metriche di accoppiamento fisse.
Yoon e Kweon [5] introducono nel loro articolo l'idea di estrarre un costo adattativo

per ogni possibile corrispondenza assegnando un peso a ogni pixel che appartiene alla finestra di correlazione considerata W_r nell'immagine *reference* e , corrispondentemente, nella finestra di correlazione W_t nell'immagine *target*. Se p_c e q_c sono i pixel centrali delle finestre W_r e W_t, per i quali si sta valutando la corrispondenza, il valore puntuale di accoppiamento tra $p_i \in W_r$ e $q_i \in W_t$ è la *Truncated Absolute Difference* (TAD), pesato però secondo il coefficiente $w_r(p_i, p_c)$ e corrispondentemente secondo $w_t(q_i, q_c)$. In questo modo il costo totale della corrispondenza (p_c, q_c) è dato dalla somma di tutti i valori della TAD pesati appartenenti alle finestre di correlazione e normalizzati per la somma dei pesi:

$$C(p_c, q_c) = \frac{\sum_{p_i \in W_r, q_i \in W_t} w_r(p_i, p_c) \cdot w_t(q_i, q_c) \cdot TAD(p_i, q_i)}{\sum_{p_i \in W_r, q_i \in W_t} w_r(p_i, p_c) \cdot w_t(q_i, q_c)} \tag{1.6}$$

dove il peso dell'accopiamento $w_r(p_i, p_c)$ e il suo corrispettivo per q_i, è calcolato tenendo conto della distanza spaziale e della distanza nello spazio colore CIELAB, rispetto al pixel centrale della finestra:

$$w_r(p_i, p_c) = exp\left(- \frac{d_p(p_i, p_c)}{\gamma_p} - \frac{d_c(p_i, p_c)}{\gamma_c} \right) \tag{1.7}$$

dove d_c e d_p sono rispettivamente la distanza euclidea tra due triplette di valori nello spazio CIELAB e la distanza euclidea tra due di punti di uno spazio 2D; mentre γ_p e γ_c sono due parametri dell'algoritmo.
Tombari et al. [4], a differenza dei predecessori, effettuano una segmentazione su entrambe le immagini reference e target allo scopo di sfruttare la simmetria intrinseca nel calcolo delle corrispondenze, e introducono inoltre un nuovo coefficiente di peso per la corrispondenza di punti basato sul'ipotesi iniziale per cui tutti i pixel appartenenti allo stesso cluster del pixel centrale della finestra di correlazione devono avere un valore simile di disparità, e quindi il loro peso deve essere il massimo possibile, cioè 1; propongono perciò di modificare il coefficiente del peso di Yoon e Kweon con:

$$w_r^{'}(p_i, p_c) = \begin{cases} 1 & p_i \in S_c \\ -\frac{d_c(p_i, p_c)}{\gamma_c} & altrimenti \end{cases} \tag{1.8}$$

dove S_c è il cluster a cui appartiene p_c; in questo caso per tutti i pixel fuori dal cluster S_c, è stato eliminato il termine che teneva conto della distanza dei pixel dal centro della finestra e quindi ogni pixel della finestra di correlazione ha la stessa importanza indipendentemente dalla loro lontananza dal centro; hanno per cui sostituito il criterio della prossimità spaziale con quello della prossimità di colore nello spazio CIELAB, ovvero con l'informazione derivata dalla segmentazione.

Capitolo 2

Algoritmo di segmentazione e ricostruzione 3D

In questo capitolo si cercherà di spiegare il funzionamento dei 2 moduli che costituiscono l'applicazione, chiamati *Segmentation pipeline* e *Stereo pipeline*, descrivendo i moduli con cui lavorano e i tipi di dati con cui si avrà a che fare.
Per riuscire ad ottenere i risultati prefissati bisogna riuscire ad elaborare i dati provenienti dalla segmentazione e dalla visione stereoscopica, introducendo nel processo di entrambi, elementi che dipendono dall'altro, combinandoli opportunamente al fine di migliorare vicendevolmente le due tecniche. Ogni processo è indipendente e lavora su un canale adibito ad esso, costituito da una serie di moduli con sottofunzioni più specifiche. Infine i due canali vengono collegati in modo tale che il prodotto di un canale sia parte fondamentale nel processo dell'altro, rendendo così il processo iterativo.

2.1 Stereo Pipeline

Si suppone innanzitutto di dispore di immagini rettificate, selezionate in questa tesi dal Middlebury dataset del 2006 [6], e di conoscere i parametri intrinseci ed estrinseci del sistema stereo con cui sono state scattate queste immagini, in modo tale da tralasciare la calibrazione del sistema e concentrarsi su altre questioni.
Il fulcro di questo canale è costituito dallo *Stereo module* che ha l'obiettivo di creare le disparity map avendo a diposizione una coppia di immagini a colori, con le caratteristiche sovracitate, e dalle loro corrispondenti immagini segmentate; a tale scopo in questa tesi si è usato l'algoritmo SegmentSupport, descritto in 1.3, che integra le informazioni geome-

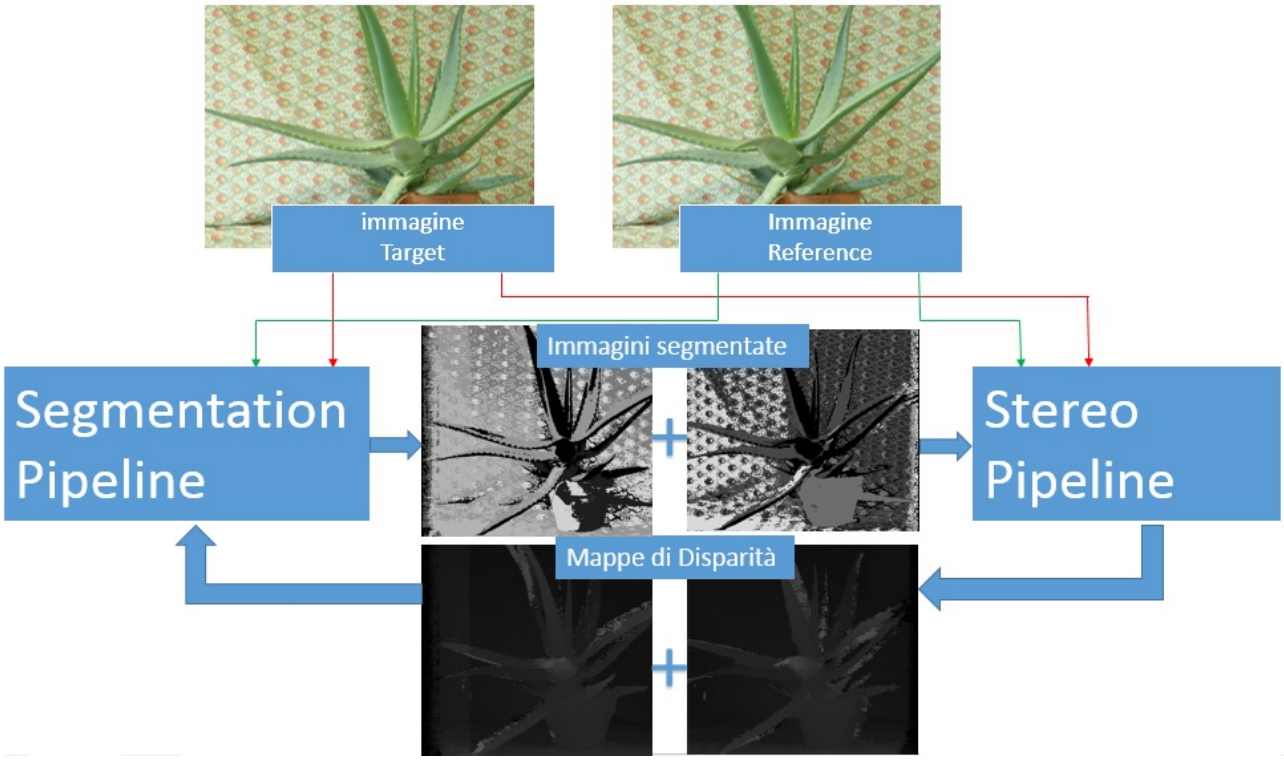


Figura 2.1: Struttura dell'applicazione

triche con quelle provenienti dalla segmentazione; l'aspetto innovativo è utilizzarlo in un processo ciclico che crea all'interno di esso la segmentazione da fornire in ingresso all'algoritmo. Un parametro che si andrà a scegliere ad ogni ciclo dell'algoritmo è la grandezza della finestra di correlazione, spiegata brevemente in 1.2.1, che influenzerà il *tradeoff* tra accuratezza e pixel errati della diparity map uscente.

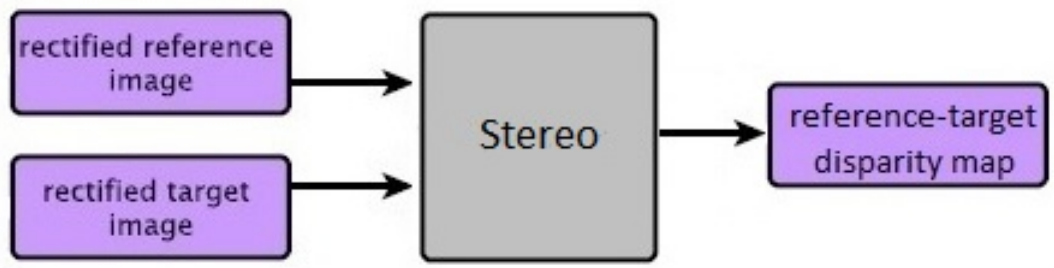


Figura 2.2: Schema a blocchi della Stereo Pipeline

Inizialmente, al primo ciclo del processo, si segmentano le due immagini basandosi solo sul colore e si forniscono all'algoritmo per il calcolo delle disparity map, dopodichè si fissa una delle due immagini stereo come immagine reference, sulla quale si calcola la disparity map rispetto all'immagine target; successivamente i ruoli tra immagine reference e target si possono scambiare, essendo il processo simmetrico, trovando in questo modo due disparity map distinte che descrivono la geometria delle due immagini a colori iniziali. A corredo di questo modulo è presente anche un ulteriore passo che controlla la consistenza del calcolo delle disparity map, chiamato *left-right* (o right-letf a seconda della disparity map considerata) *consistency check* che serve in particolare per individuare e correggere le discrepanze che possono esserci in caso di non univocità della corrispondenza tra le coppie coniugate.

2.2 Segmentation Pipeline

La Segmentation Pipeline è composta principalmente da tre moduli:

- **Geometry module**: si occupa di ricavare dalle disparity map le componenti geometriche per ogni pixel dell'immagine.

- **Alignment module**:unisce in una struttura omogenea i dati provenienti dal colore e dalla geometria.
- **Clustering module**:associa ogni pixel ad un segmento nell'immagine finale risultante.

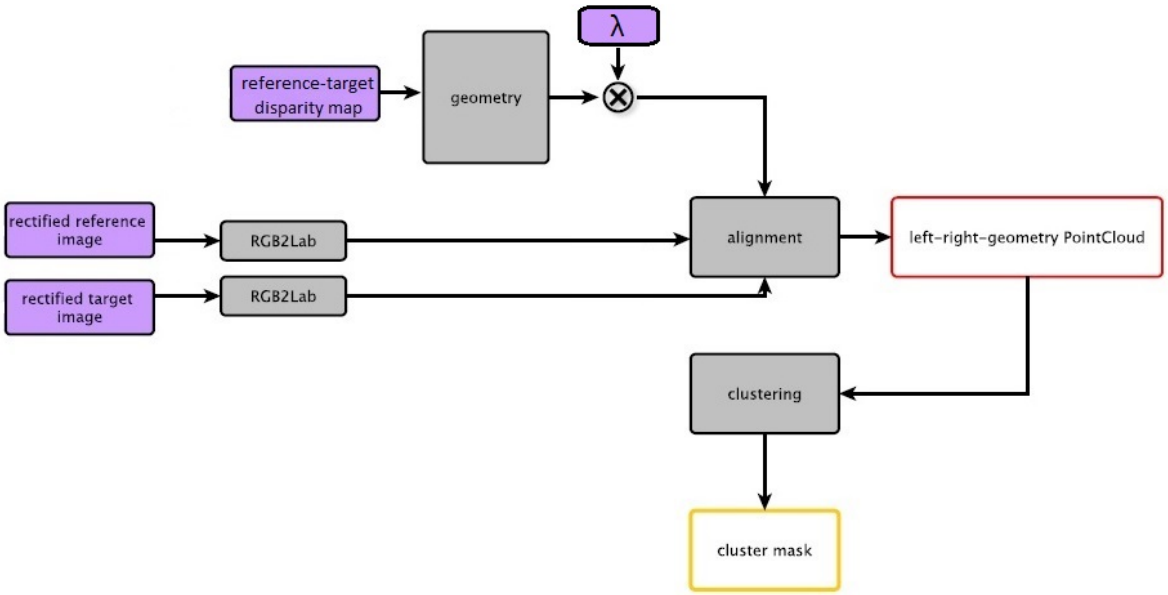


Figura 2.3: Schema a blocchi della Segmentation Pipeline

2.3 Geometry module

Come mostrato in figura 2.3, il *Geometry module* è il primo dei moduli della Segmentation pipeline e prende in ingresso la disparity map reference-target dell'immagine e da questa, per ogni pixel, stima le tre componenti geometriche dell'immagine che verrano passate all'Alignment module. A corredo di questa funzionalità ci sono una serie di passaggi in cui i dati vengono filtrati, eliminati o rimpiazzati per evitare situazioni non controllabili dal processo. Ad esempio un primo filtraggio dei valori provenienti dalla disparity map è necessario per evitare che valori nulli vengano inseriti nella pipeline in modo indesiderato o non gestibile dall'applicazione.

2.4 Alignment module

Questo secondo modulo, in serie al precedente, prende in ingresso, le tre coordinate geometriche $(x, y, z) \in \mathbb{R}^3$, provenienti dal Geometric module e le componenti cromatiche di

ogni pixel, di entrambe le immagini a colori, per creare una nuova struttura dati.

2.4.1 Introduzione alla PointCloud

Si è deciso a questo scopo di adottare una struttura chiamata *PointCloud*, ovvero una raccolta di punti con caratteristiche analoghe, ognuno dei quali, nella forma finale utile per la tesi , corrisponde di fatto a un vettore formato da 9 elementi, ognuno associato a un pixel dell'immagine:

$$p_i = \left[L_r, a_r, b_r, L_t, a_t, b_t, x, y, z\right] \tag{2.1}$$

I primi sei elementi del vettore descrivono le caratteristiche di colore nello spazio CIELAB, rispettivamente dell'immagine reference e della target; la scelta di questo spazio è stata dettata dal fatto che ha la pecularietà di essere uno spazio uniforme; mentre gli ultimi tre elementi sono le coordinate geometriche (in riferimento allo spazio delle fotocamere), prese nel caso generale nello spazio $\mathbb{R}^3$. Si fanno a questo vettore 9-dimensionale delle ulteriori operazioni per rendere omogenei i dati rispetto alle unità di misura: si dividono le componenti associate al colore per la deviazione standard della propria componente della *luminanza*, mentre si dividono le componenti geometriche per la deviazione standard della componente z. Per finire si moltiplicano le tre coordinate spaziali per un parametro λ che ha la funzione di pesare l'importanza da attribuire alle coordinate geometriche rispetto a quelle cromatiche, arrivando quindi alla rappresentazione finale di un punto della PointCloud:

$$p_i^{'} = \left[\frac{L_r}{\sigma_{L_r}}, \frac{a_r}{\sigma_{L_r}}, \frac{b_r}{\sigma_{L_r}}, \frac{L_t}{\sigma_{L_t}}, \frac{a_t}{\sigma_{L_t}}, \frac{b_t}{\sigma_{L_t}}, \frac{\lambda}{\sigma_z}x, \frac{\lambda}{\sigma_z}y, \frac{\lambda}{\sigma_z}z\right] \tag{2.2}$$

Il parametro λ ha il ruolo fondamentale in questa tesi di attribuire alle coordinate spaziali, a seconda del suo valore, più o meno importanza nel processo che porterà alla segmentazione dell'immagini: un valore troppo basso comporterà poca influenza delle caratteristiche tridimensionali della scena mentre un valore troppo alto farà in modo che quest'ultime siano considerate maggiormente, tralasciando le informazioni cromatiche. Un punto fondamentale nel processo di segmentazione in ogni iterazione è stato trovare il valore ottimo di λ che fornisse una buona segmentazione e allo stesso tempo un buon ingresso per la iterazione successiva della Stereo Pipeline. A questo scopo si è utilizzato come base di parametro di scelta la funzione $Q(I, D, S)$ introdotta nel suo articolo da Dal Mutto et al. [7]:

$$Q(I, D, S) = Q^{color}(I, S) + 3 \cdot Q^{depth}(D, S) \tag{2.3}$$

Dove $Q^{color}(I, S)$ indica la bontà della segmentazone (S) rispetto all'imagine a colori (I) (considerata questa volta nello spazio RGB) e $Q^{depth}(D, S)$ inceve rispetto alla disparity map (D); inoltre la moltiplicazione per il valore 3 serve per equilibrare il peso dei due

termini dal momento che il primo tiene conto dei tre canali RGB di cui si compone l'immagine a colori, mentre il secondo deriva da un unico canale. Nell'articolo si dimostra come il valore ottimo di λ per la segmentazione di un'immagine si trova, al variare dello stesso λ, in concomitanza con il massimo valore di $Q(I, D, S)$.
Nel caso specifico per i propositi espressi in questa tesi però, un cambiamento da effettuare è l'eliminazione del secondo termine di valutazione della segmentazione, $Q^{depth}(D, S)$ che considera la segmentazione in relazione alla disparity map, la quale in questo caso è un dato da ricercare, interno al processo stesso e che non si ha a disposizione a priori dalla prima itarazione. Non si può in ogni caso, anche una volta calcolata la prima disparity map, inserire come metrica di valutazione della segmentazione la stessa disparity map da cui deriva, questo porterebbe a risultati devianti. Perciò si è deciso di tenere in considerazione, come paramentro di scelta possibile, solo il λ ottimo che massimizza la funzione $Q^{color}(I, S)$.

2.4.2 Realizzazione dell'allineamento dati

Come primo passo dell'allineamento bisogna trovare le coppie di punti coniugati nell'immagini reference e target, compito non sempre facile a causa della non iniettività della corrispondenza, che porta ad avere punti per cui non esiste un coniugato sull'altra immagine, o anche a causa di accoppiamenti di punti dell'immagine reference con punti dell'immagine target fuori dal dominio o con coordinate negative. Se per il primo problema non esiste soluzione dal momento che non dipende da come si gestiscono i dati ma da come sono creati: infatti in un modello stereo è ovvio che ci siano parti della scena presenti solo in un immagine e non nell'altra; d'altra parte, il secondo problema è risolvibile con l'utilizzo di maschere che isolano determinati punti scomodi o, se possibile, li rimpiazzano con altri valori accettabili. Nel caso generale comunque è un operazione facile grazie ai vincoli epipolari che legano le coordinate dei punti coniugati su assi orizzontali, essendo le immagini rettificate.
Trovate le corrispondenze, si passa all'allineamento, cioè all'affiancamento dei dati provenienti da tre Point Cloud differenti:

- La **Point Cloud dell'immagine target**
- La **Point Cloud dell'immagine reference**
- La **Point Cloud della disparity map**

Per creare la Point Cloud finale, chiamata *Geometry-Color Point Cloud.*

2.5 Clustering module

L'ultimo modulo della Segmentation Pipeline è pensato per ricevere la Point Cloud, generata dal modulo precedente debitamente elaborata, e restituire un vettore che indica per ogni pixel dell'immagine reference, a quale cluster deve appartenere, definendo di fatto la segmentazione dell'immagine. Il numero di cluster da utilizzare per la segmentazione in questa tesi è ancora stabilito manualmente, ma un possibile sviluppo futuro potrebbe essere la determinazione automatica del numero di cluster adatto a segmentare la scena. E' all'interno di questo modulo che opera l'algoritmo basato sull'approssimazione di Nystrom citata in 1.1.1, usato qui però con una piccola modifica introdotta all'unico scopo di rendere ripetibile il processo di segmentazione: si è cambiato perciò il metodo di campionamento dei punti dell'immagine; nel metodo di Nystrom classico quest'ultimo è totalmente casuale e perciò i risultati cambierebbero ad ogni prova mentre il campionamento scelto per la realizzazione del progetto in questa tesi è periodico e, anche se resta casuale la scelta del campione da cui cominciare il campionamento, i risultati diventano deterministici a parità di parametri. Determinata la segmentazione si comincia una nuova iterazione del processo segmentazione-visione stereoscopica che sfrutta la segmentazione appena trovata come ingresso, ricominciando tutto dal pricipio.

Capitolo 3

Implementazione dell'algoritmo

Questo capitolo vuole dare una parnoramica di come effettivamente sono stati implementati i moduli descritti nel capitolo precedente, implementati nella tesi di Tubiana [8] e dare un'idea degli strumenti utilizzati nella pratica come supporto a questo lavoro. Il codice è interamente scritto in linguaggio C++ e si appogggia all'IDE (*integrated development enviroment*) fornito da Visual Studio. La maggior parte dei dati trattati essendo matrici sono stati ben maneggiati con l'ausilio della libreria *OpenCV* e in parte dalla libreria *Eigen*, sopratutto in relazione all'implemantazione della funzione di clustering.

3.1 Implementazione del canale Stereo Pipeline

3.1.1 Stero module

Lo Stereo Pipeline è caratterizzato dal solo modulo Stereo che si occupa di creare le mappe di disparità, la funzione per far ciò è il SegmentSupport la cui dichiarazione è:

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

L'algoritmo prende in ingresso le immagini stereo a colori e le relative immagini segmentate e restituisce una disparity map per l'immagine reference; in seguito invertendo reference e target si ottiene la disparity map anche per l'altra immagine. Qua come nel resto dell'implementazioni le immagini, le immagine segmentate (dati d'ingresso in questo

caso) e le disparity map sono gestite dalla struttura per gestire le matrici, chiamata *Mat*, messa a disposizione da OpenCV. In questa funzione i parametri per la creazione della disparity map sono passati grazie a una struttura dati apposita per questo scopo, chiamata *SegmentSupportParams* che include il valore della grandezza della finestra di correlazione e il coefficiente TAD; inoltre contempla anche un valore *type* che serve per indicare se si sa facendo la disparity da destra a sinistra o viceversa.

3.2 Implementazione canale Segmentation Pipeline

3.2.1 Geometry module

Il modulo si occupa di estrapolare le informazioni tridimensionali da una disparity map (controllando la validità dei punti grazie a una *disparity mask*) conoscendo i parametri intrinseci del sistema e la lunghezza della *baseline*;la funzione è chiamata *geometryEstimator3D* e restituisce una matrice con le coordinate 3D associate a ogni pixel della matrice associata alla disparity map d'entrata: utilizza prima la formula 1.5 per calcolare la coordinata z di ogni punto e poi ricava le coordinate tridimensionali dei punti grazie alla relazione $M = Km$ che lega i punti tridimensionali M con i punti dell'immagine m attraverso la matrice dei parametri intrinseci K.
La sua dichiarazione è la seguente:

```
void geometryEstimator3D( const Mat& intrinsic_params,
 const float baseline,
 const Mat& disparity,
 const Mat& disparity_mask,
 const Mat& params,
 Mat& cloud3d)
```

3.2.2 Alignment module

Il modulo si occupa inizialmente degli accoppiamenti fra punti dell'immagine reference con quella target e lo fa attraverso una struttura apposita *PointCorrespondence* che raccoglie una coppia di punti 2D:

```
typedef struct PointCorrespondence{
 Point2i reference
 Point2i target
```

```
}PointCorrespondences
```

mentre chi ha il compito di trovare i due punti da accoppiare è la funzione *findPointCorrespondence* che sfrutta il valore della disparità di ogni punto della disparity map per trovare il punto target corrispondente nell'altra immagine; grazie al vincolo epipolare questo è possibile semplicemente sottraendo o aggiungendo (a seconda di qual'è l'immagine reference) la disparità alla coordinata x del punto reference.
La dichiarazione della funzione è:

```
void findPointCorrespondence(const Mat& disparity_map,
 vector<PointCorrespondence>& corrispondence_index,
 int dmax=0,
 int type=0,)
```

quest'ultima restituisce un vettore con le corrispondenze dei punti dell'immagine reference; come in altre funzioni, anche qui è stato inserito un parametro aggiuntivo type che indica quale delle due immagini sia presa come reference.
La funzione infine che si occupa di allineare il tutto si chiama *pointCorrespToVectorCorresp*:

```
void pointCorrespToVectorCorresp( const vector<PointCorrespondence>& corrisponden
    const int num_row,
 const int num_cols,
 Mat& mask,
 vector<int>& vector_corresp_idx)
```

che prende in ingresso il vettore precedentemente trovato dalla funzione *findPointCorrespondence* e ne restituisce un altro con i dati allineati, avvalendosi anche di una maschera che indica i punti con accoppiamenti non validi.

3.2.3 Clustering module

Il cuore del modulo è la funzione che implementa l'algoritmo di Spectral Clustering con approssimazione di Nystrom, chiamato appunto *spectralClusteringNystrom*:

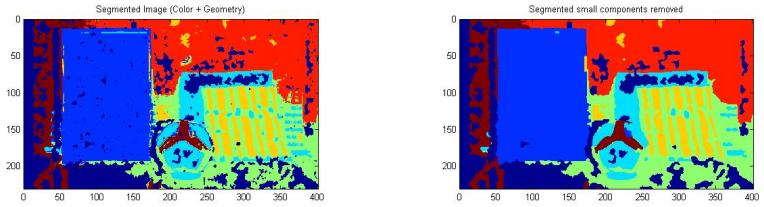
```
void spectralClusteringNystrom( Mat& point_matrix,
 const int sample_number,
 const int num_cluster,
```

```
float sigma,
Mat& class_vec,
vector<int>& clusterIndex)
```

il quale restituisce un vettore indicante il cluster d'appartenenza di ogni punto della point cloud *point_matrix* inserita in ingresso; *sample_number*, *num_cluster* e *sigma* sono i parametri che caratterizzano l'algoritmo. Il primo indica il numero di campioni da scegliere per l'approsimazione di Nystrom, il secondo il numero di cluster in cui si vuole dividere l'immagine e il terzo è un fattore nella metrica di aggegazione fra pixel dell'immagine.
Elemento nuovo di questa versione dell'applicazione è invece la funzione *RemoveSmallComponent*, trascritta dal linguaggio Matlab in C++, che affina l'immagine risultante dalla segmentazione:

```
void RemoveSmallComponent( int minArea,
 Mat& imgCluster,
 int numCluster)
```

Prende appunto in ingresso l'immagine segmentata e il numero di cluster con cui è stata fatta l'operazione, e si occupa di cercare gli aglomerati di pixel, grandi al più come il numero indicato da *minArea*, sostituendo il valore di quei pixel con il valore del cluster che li circonda, eliminando in questo modo tutti i pixel isolati all'interno delle aree più grandi di cluster.



(a) (b)

Figura 3.1: Esempio del risultato della funzione *RemoveSmallComponent*

Capitolo 4

Test e Risultati

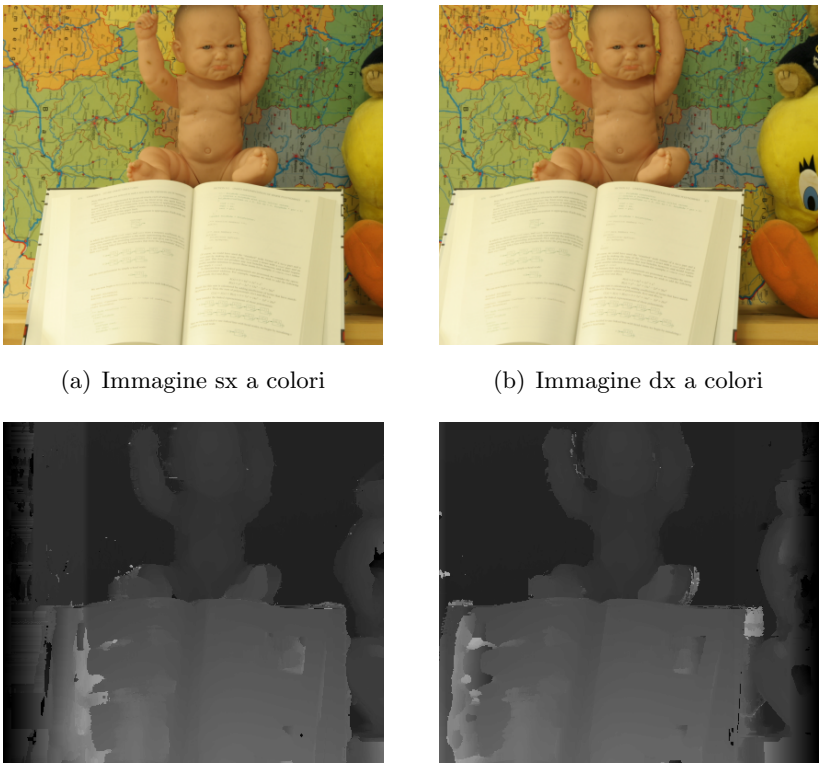
In questo capitolo si andrà ad esaminare la segmentazione e le disparity map di 3 differenti immagini stereo scelte dal *Middlebury data set*, in particolare l'evoluzione nel corso delle iterazioni, che chiameremo *run*, dove si susseguiranno alternativamente le operazioni di segmentazione e calcolo delle disparity map; si esamineranno i dati di ogni run, cercando di trovare dei tratti comuni negl'andamenti delle funzioni utilizzate per valutare la qualità delle immagini risultanti, affidandosi a volte anche al solo giudizio visivo e se rispecchia quello che ci si aspetterebbe dalla segmentazione di un'immagine. In particolare, le funzioni per valutare la segmentazione sono tre: la $Q(I, D, S)$, prendendo come riferimento la disparity map, chiamata *ground_truth*, messa a disposizione dal data set, la $Q^{color}(I, S)$, parte della funzione precedente, che valuta la segmentazione solo rispetto all'immagine a colori e la $Q^{depth}(D, S)$, analoga alla precedente che valuta la segmentazione solo rispetto alla disparity map . La $Q^{color}(I, S)$ è stata anche usata come base di scelta del lambda ottimo delle immagini segmentate da inserire nell'iterazione successiva nel calcolo delle disparity map ma in ogni caso è la funzione $Q(I, D, S)$ che bisogna prendere come riferimento per valutare in assoluto i risultati della segmentazione.
Per le disparity map invece si è scelto di considerare l'errore quadratico medio (RMSE: *root mean square error*), calcolato sempre rispetto alla ground_truth, e il numero di pixel tra le due disparity map trovati inconsistenti, ovvero senza coniugato, tramite la funzione *crossCheck*.
Si è cercato in tutto ciò un approccio sistematico all'analisi del problema variando alcuni parametri e cercando di selezionare ad ogni iterazione il migliore: per la segmentazione in ogni run si è variato il paramentro λ nell'intervallo $[0.1, 9]$, selezionando alla fine il λ ottimo che massimizzava la funzione $Q^{color}(I, S)$, mentre si sono mantenuti fissi i valori della funzione di segmentazione *spectralClusteringNystrom*, ovvero il numero di cluster utilizzato per la segmentazione, il valore sigma e il numero di campioni utilizzato per l'approssimazione di Nystrom, rispettivamente pari a 8, 5 e 64. Per le disparity map

invece si è variata la grandezza della finestra di correlazione dell'algoritmo *SegmentSupport*, partendo per i primi run da una finestra più grande e diminuendola per i successivi, cercando in questo modo ad affinare la precisione della disparity map all'aumentare dei run.

4.1 Immagine *Baby2*

Run 0

Andiamo ora a considerare i run eseguiti sull'immagine chiamata *Baby2*. Si comincia da una segmentazione delle due immagini stereo basata solo sul colore, con gli stessi parametri detti in precedenza, e si prosegue con il calcolo della prima disparity map, in questo caso partendo con una finestra di dimensione 19 pixel e il risultato è nella fig. 4.1. Il RMSE



(a) Immagine sx a colori

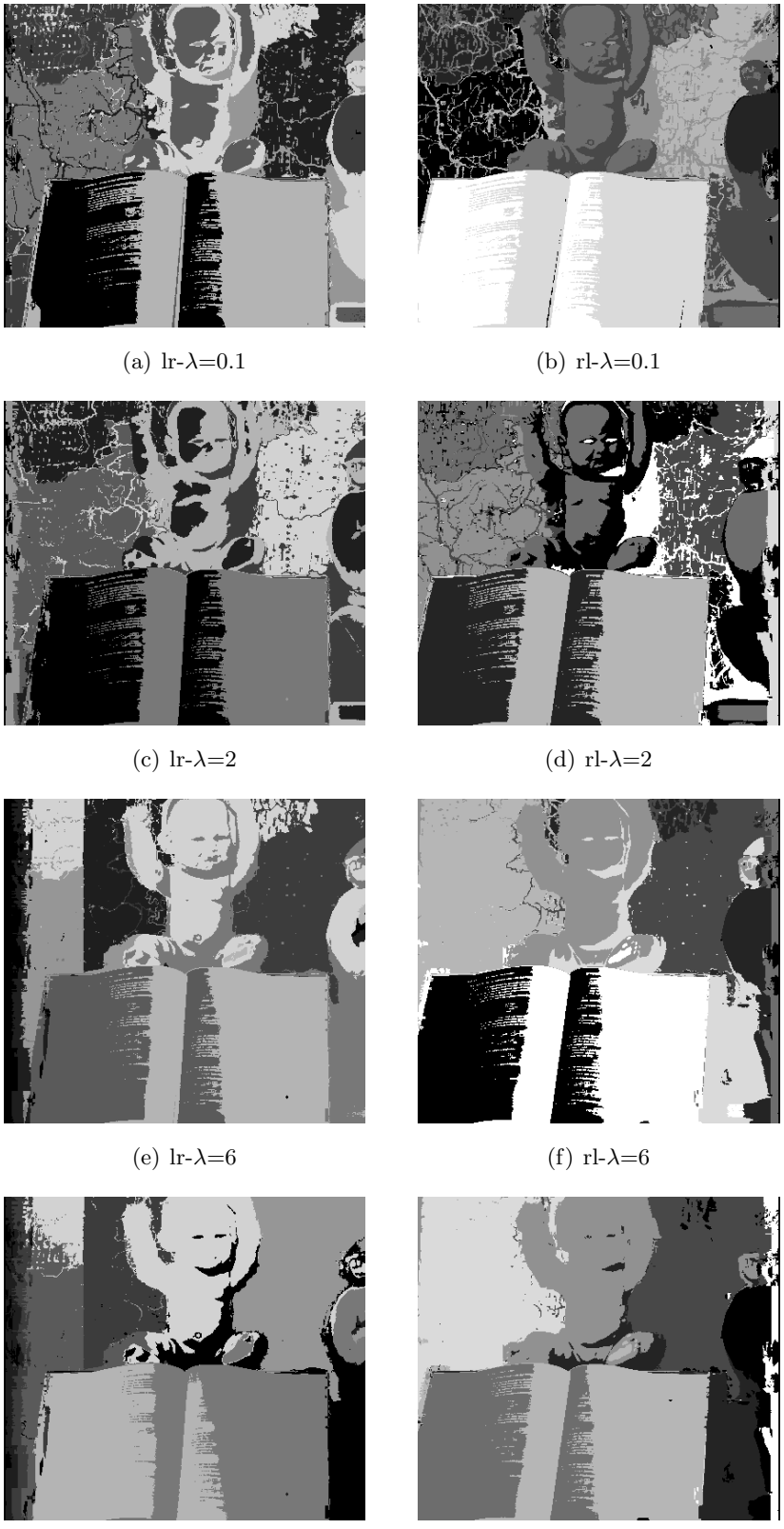
(b) Immagine dx a colori

(c) Disparity map lr run0

(d) Disparity map rl run0

Figura 4.1: Immagini a colori e disparity map al run0

delle due disparity map risulta pari a 6.26268, mentre i pixel inconsistenti sono il 24.7%. Queste due disparity map sono state fornite all'ingresso della *Segmentation Pipeline* e si sono calcolate dodici immagini al variare di λ, di cui alcuni esempi sono mostrati nella fig 4.2 Si noti come all'aumentare del λ la segmentazione prenda sempre più i tratti della



(a) lr-λ=0.1 (b) rl-λ=0.1

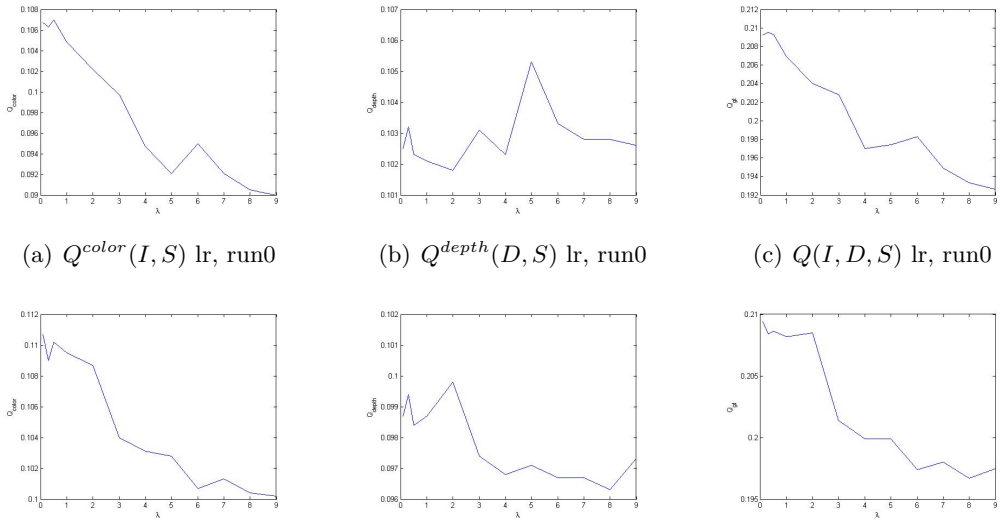
(c) lr-λ=2 (d) rl-λ=2

(e) lr-λ=6 (f) rl-λ=6

(g) lr-λ=9 (h) rl-λ=9

Figura 4.2: Esempi segmentazione al run0

disparity map e già dal valore $\lambda = 4$ in poi, la trama della mappa sullo sfondo tenda a scomparire e risalti anche il difetto della disparity map sulla parte sinistra delle immagini lr.
I dati raccolti dalla segmentazione forniscono i grafici delle funzioni $Q(I, D, S)$, chiamata Q_{gt}, e $Q^{color}(I, S)$, chiamata Q_{color}, e $Q^{depth}(D, S)$, chiamata Q_{depth}, tutte al variare di λ 4.3.



(a) $Q^{color}(I, S)$ lr, run0 (b) $Q^{depth}(D, S)$ lr, run0 (c) $Q(I, D, S)$ lr, run0

(d) $Q^{color}(I, S)$ rl, run0 (e) $Q^{depth}(D, S)$ rl, run0 (f) $Q(I, D, S)$ rl, run0

Figura 4.3: grafici delle funzioni $Q^{color}(I, S)$, $Q^{depth}(D, S)$ e $Q(I, D, S)$, run0

Si noti come le funzioni, sia per le immagini lr, sia per le rl, abbiano certamente massimi diversi ma counque andamenti simili che tendono a privilegiare una scelta del valore di lambda basso, orientando maggiornemte il processo verso una segmentazone più fedele al colore. I valori medi delle due funzioni graficate sono nella tabella 4.1:

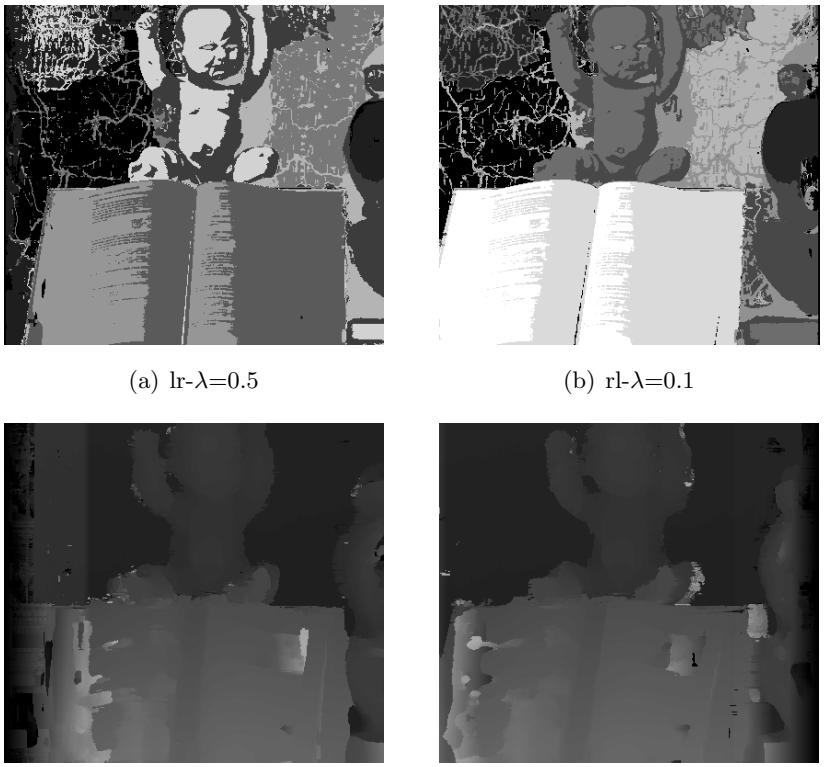
Tabella 4.1: Run0 *Baby2*

Img	$Q^{color}(I, S)$	$Q^{depth}(D, S)$	$Q(I, D, S)$
lr	0.0984	0.1028	0.2012
rl	0.1050	0.0978	0.2028

Run 1

Per la seconda iterazione si sono scelti quindi di inserire nel processo della *Stereo Pipeline* le immagini segmentate corrispondenti ai valori di λ che massimizzavano la funzione

$Q^{color}(I, S)$, quindi per l'immagine lr, $\lambda = 0.5$, e per l'immagine rl, $\lambda = 0.1$.
Da queste si è giunti, utilizzando anche in questo caso una finestra di dimensione 19 pixel, a una nuova coppia di disparity map con un errore quadratico medio nettamente inferiore: RMSE=1.9021 e con numero di pixel inconsistenti al 24.6% (fig. 4.4).



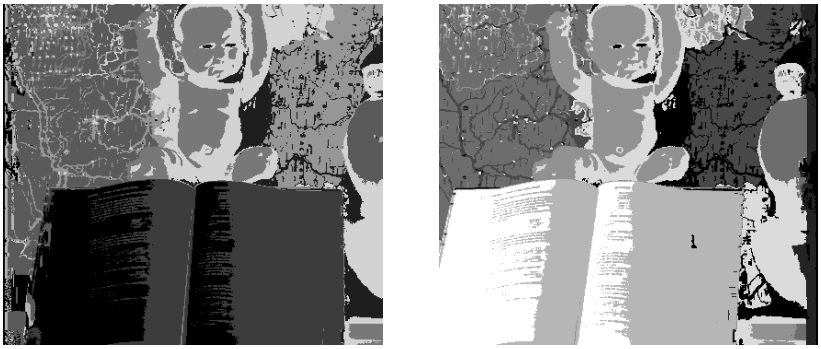
(a) lr-λ=0.5 (b) rl-λ=0.1

(c) Disparity map lr run1 (d) Disparity map rl run1

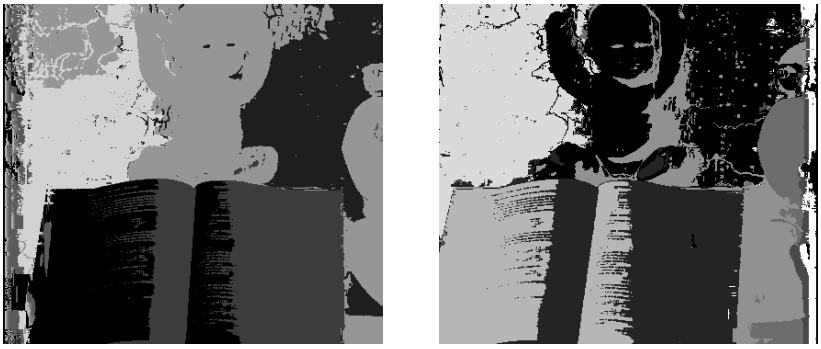
Figura 4.4: Immagini segmentate scelte dal run0 e disparity map del run1

Degli esempi di immagini segmentate ottenute da queste nuove disparity map sono mostrate in figura 4.5: Visivamente rispetto al run0 sembrerebbe che gli oggetti della scena sono segmentati meglio e si noti meno l'errore sulla parte sinistra delle immagini lr, anche per lambda elevati; però nonostante il netto miglioramento del dato di RMSE, i dati relativi alla segmentazione sono stabili per le immagini rl e peggiori per le immagini lr (fig.4.6): I grafici sono molto frastagliati rispetto al run0 e le immagini con picchi negativi potrebbero essere un motivo della diminuzione del valore medio delle due funzioni; nelle immagini rl si nota un picco molto negativo per tutti i grafici per l'immagine corrispondente al valore $\lambda = 5$ (fig 4.5); dovuto al fatto che la segmentazione in quel caso non distingue elementi del bambolotto e dell'orso di pezza, dallo sfondo. Più in dettaglio, i valori medi risultanti dalla segmentazione al run 1 sono in tabella 4.2.

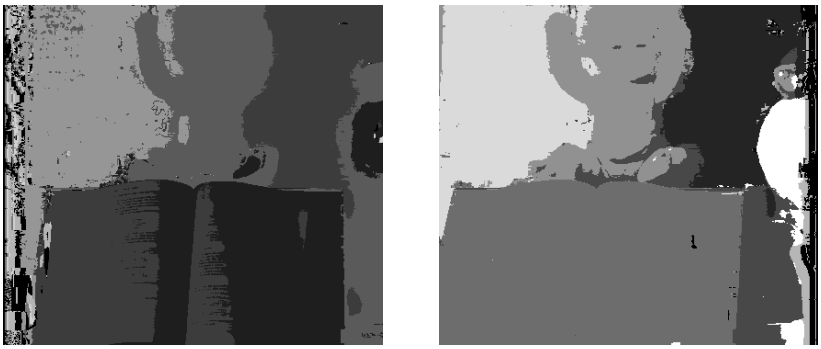
(a) lr-λ=0.1 (b) rl-λ=0.1



(c) lr-λ=2 (d) rl-λ=2

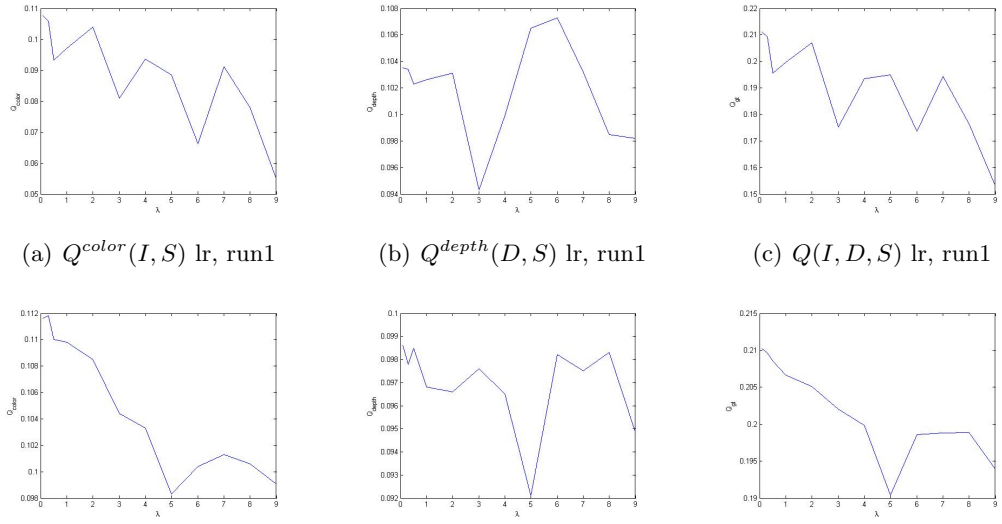


(e) lr-λ=5 (f) rl-λ=5



(g) lr-λ=9 (h) rl-λ=9

Figura 4.5: Esempi segmentazione al run1



(a) $Q^{color}(I,S)$ lr, run1 (b) $Q^{depth}(D,S)$ lr, run1 (c) $Q(I,D,S)$ lr, run1

(d) $Q^{color}(I,S)$ rl, run1 (e) $Q^{depth}(D,S)$ rl, run1 (f) $Q(I,D,S)$ rl, run1

Figura 4.6: grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$, run1

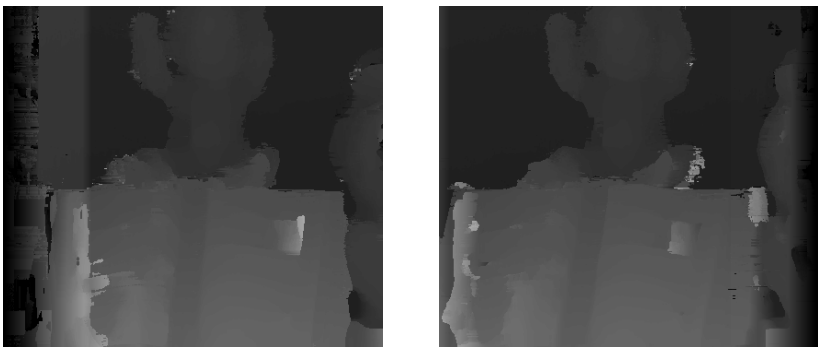
Tabella 4.2: Run1 *Baby2*

Img	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
lr	0.0885	0.1028	0.1904
rl	0.1050	0.1019	0.2019

Run 2

La scelta per l'iterazione succesiva è caduta sulle due immagini segmentate corrispondenti a $\lambda = 0.3$ per l'immagine rl e $\lambda = 0.1$ per l'immagine lr; però mantenendo costante la grandezza della finestra, l'errore quadratico medio risultava maggiore rispetto al precedente: RMSE=1.92839, con pixel inconsistenti pari al 24.4%. Per questa immagine la riduzione della grandezza della finestra non comportava un affinamento della disparity map e quindi un miglioramento in termini di RMSE e al contrario, si otteneva un peggioramento. Viceversa, un ampliamento della finestra ha comportato un miglioramento in termini di RMSE e si sono ottenute così le due nuove disparity map con RMSE=1.8224 utilizzando una finestra di grandezza 25 pixel che ha comportato però un aumento dei pixel inconsistenti, in questo caso pari al 32.2% (fig. 4.7). Queste due disparity map si sono inserite nel processo di segmentazione ottenendo le immagini di fig. 4.8: In questo run ritorna più accentuato l'errore della disparity map lr mentre nell'immagine rl, senza errori, si giunge a una buona segmentazione, confermata anche dai dati, nonostante

(a) lr-λ=0.1 (b) rl-λ=0.3



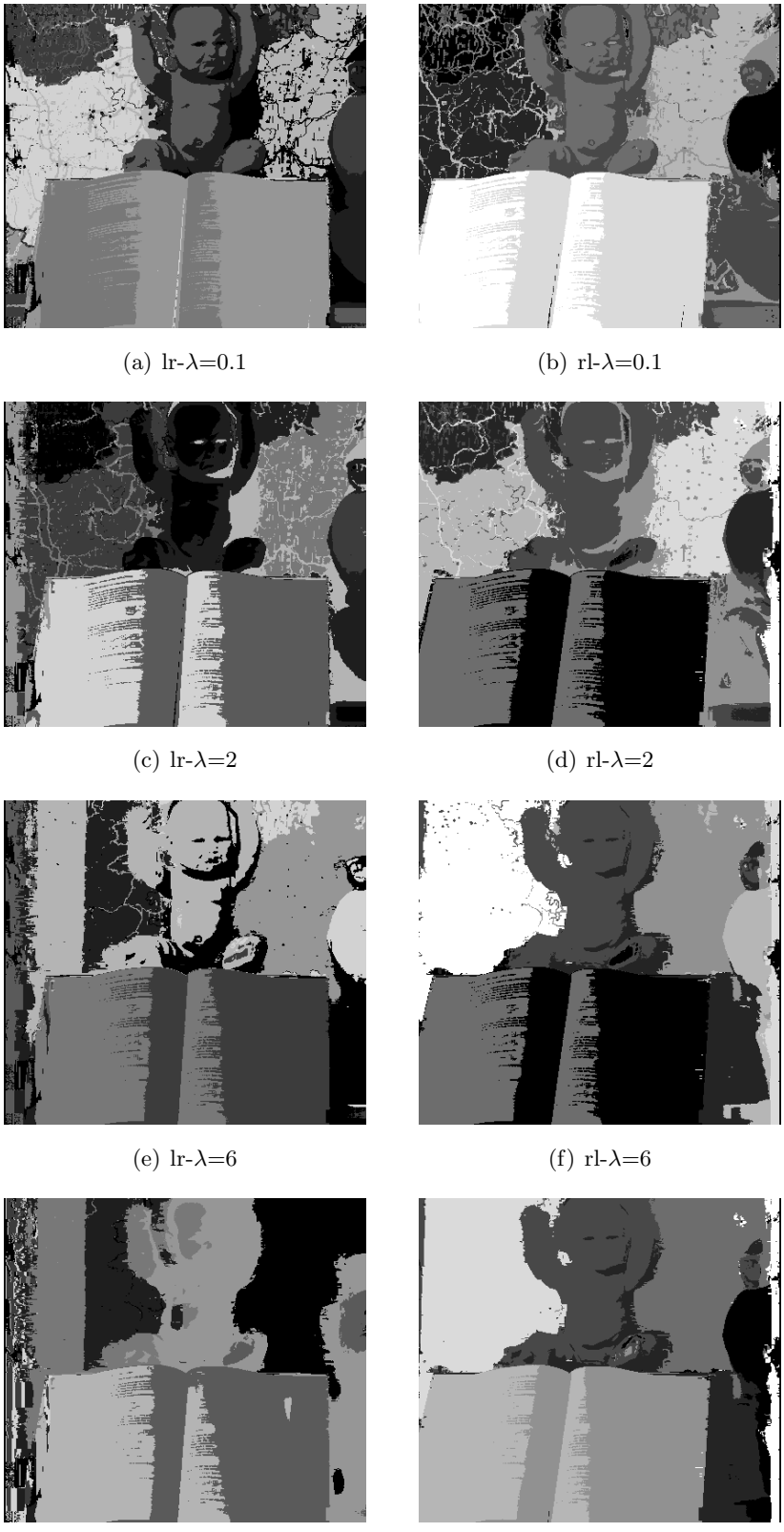
(c) Disparity map lr run2 (d) Disparity map rl run2

Figura 4.7: Immagini segmentate scelte dal run1 e disparity map del run2

l'incremento maggiore in media si riscontri nei risultati relativi all'immagine lr. I grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$ sono in figura fig. 4.9. I valori medi riscontrati nelle due funzioni sono nella tabella 4.3:

Tabella 4.3: Run2 *Baby2*

Img	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
lr	0.0972	0.1015	0.1987
rl	0.1070	0.0974	0.2043



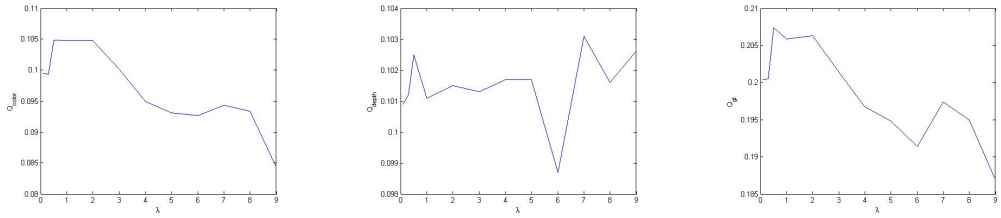
(a) lr-λ=0.1 (b) rl-λ=0.1

(c) lr-λ=2 (d) rl-λ=2

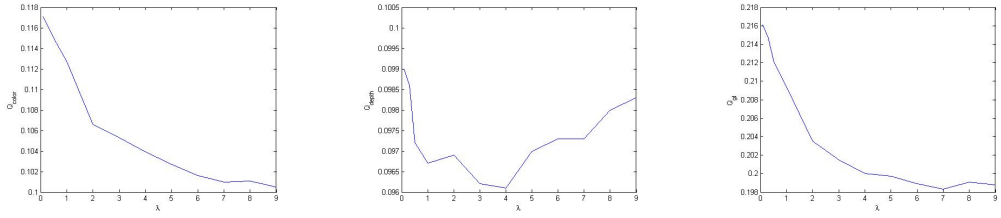
(e) lr-λ=6 (f) rl-λ=6

(g) lr-λ=9 (h) rl-λ=9

Figura 4.8: Esempi segmentazione al run2



(a) $Q^{color}(I,S)$ lr, run2 (b) $Q^{depth}(D,S)$ lr, run2 (c) $Q(I,D,S)$ lr, run2

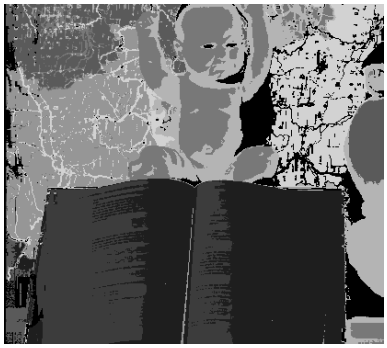


(d) $Q^{color}(I,S)$ rl, run2 (e) $Q^{depth}(D,S)$ rl, run2 (f) $Q(I,D,S)$ rl, run2

Figura 4.9: grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$, run2

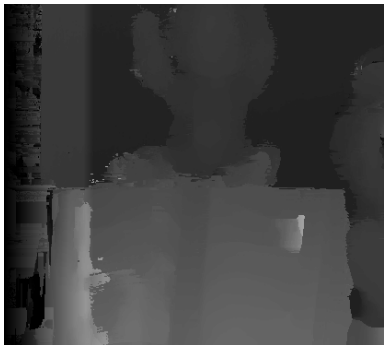
Run 3

L'iterazione è stata compiuta per emtrambe con le due immagini segmentate corrispondenti a $\lambda = 0.3$; anche in questo caso il mantenimento della grandezza della finestra portava un miglioramento minimo in termini di errore quadratico medio: RMSE=1.82194, con pixel inconsistenti 32.3%, e quindi si è ampliata ancora di più la finestra di correlazione, portandola a 27 pixel. In questo modo si è ottenuto un RMSE=1.82028, con pixel inconsistenti al 32.6%, di molto poco inferiore al precedente, sintomo forse dell'avvicinars del limite del miglioramento; inoltre a causa dei tempi molto lunghi che comporta l'algoritmo *SegmentSupport* che sono proporzionai alla grandezza della finestra, si è deciso di prendere questo dato come buono e non andare oltre (fig. 4.10). Queste due disparity map si sono inserite nel processo di segmentazione ottenendo le immagini seguenti,fig. 4.11: In questo run, a differenza dei precedenti si segmenta molto meglio il libro: già dal $\lambda = 2$ per rl non si vedono più i riflessi di luce e le parole scritte su du esso. I grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$ sono nella fig. 4.12.

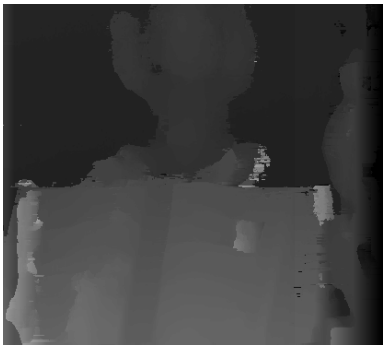


(a) lr-λ=0.3

(b) rl-λ=0.3



(c) Disparity map lr run3



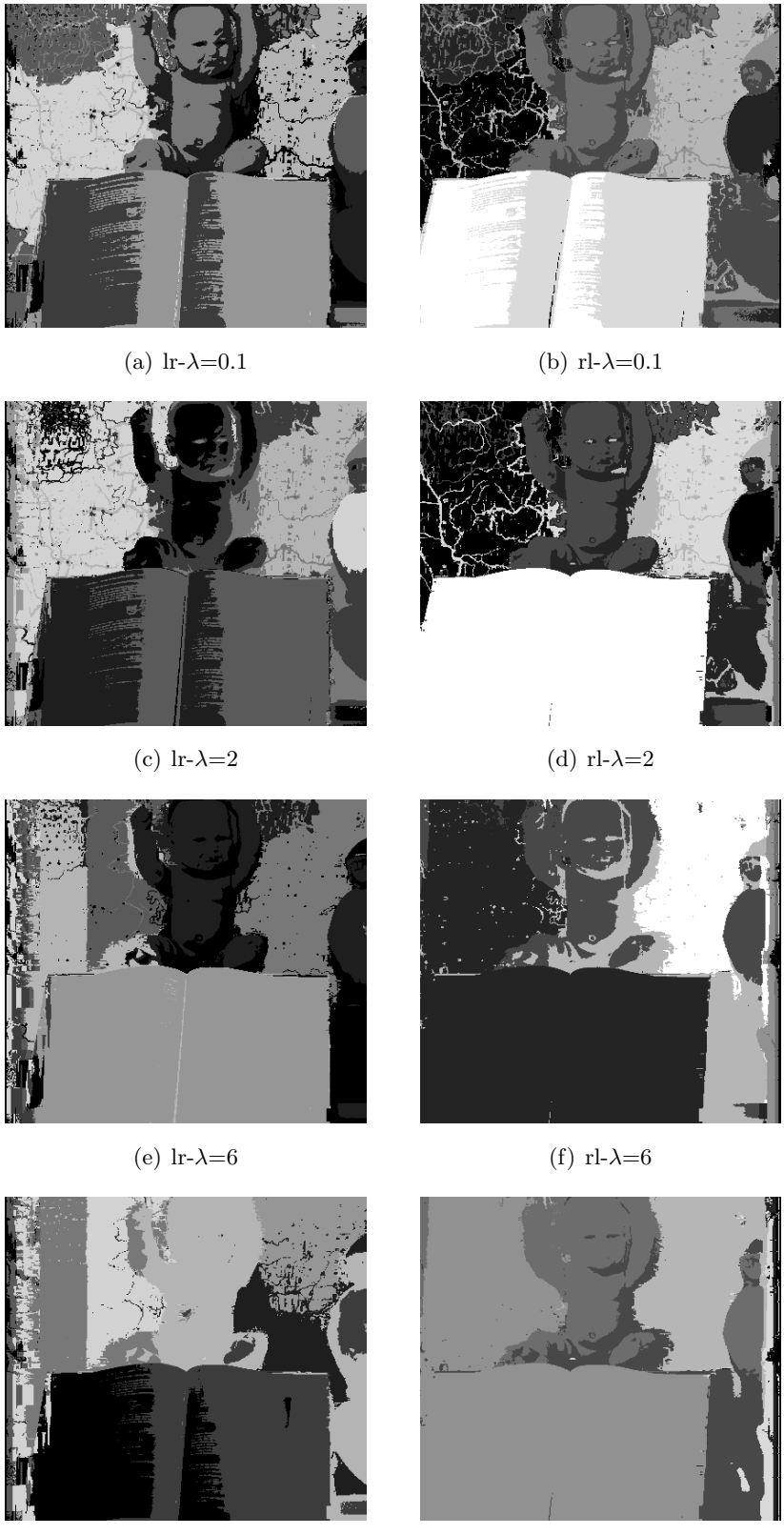
(d) Disparity map rl run3

Figura 4.10: Immagini segmentate scelte dal run2 e disparity map del run3

I valori medi riscontrati nelle due funzioni sono nella tabella 4.4.

Tabella 4.4: Run3 *Baby2*

Img	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
lr	0.0975	0.1017	0.1992
rl	0.1078	0.0828	0.1905



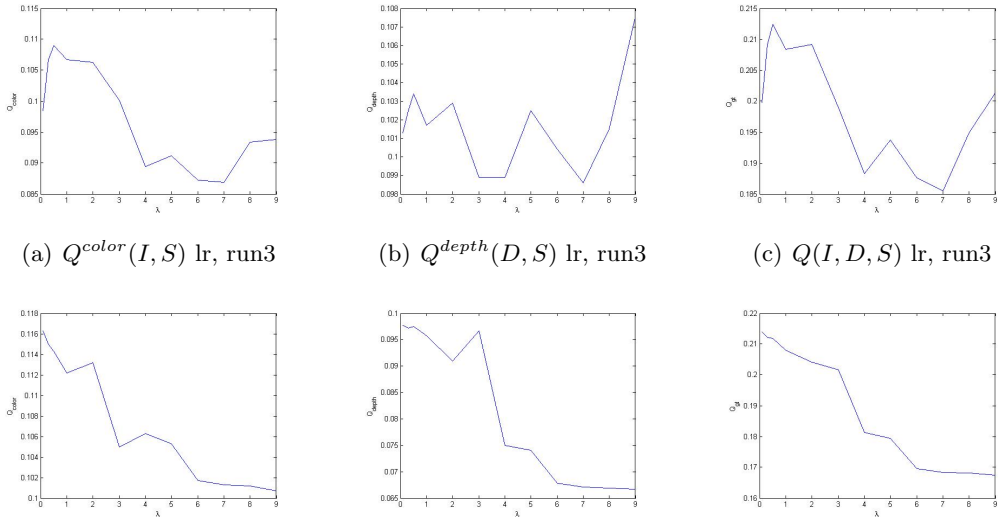
(a) lr-λ=0.1 (b) rl-λ=0.1

(c) lr-λ=2 (d) rl-λ=2

(e) lr-λ=6 (f) rl-λ=6

(g) lr-λ=9 (h) rl-λ=9

Figura 4.11: Esempi segmentazione al run3



(a) $Q^{color}(I,S)$ lr, run3 (b) $Q^{depth}(D,S)$ lr, run3 (c) $Q(I,D,S)$ lr, run3

(d) $Q^{color}(I,S)$ rl, run3 (e) $Q^{depth}(D,S)$ rl, run3 (f) $Q(I,D,S)$ rl, run3

Figura 4.12: grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$, run3

Per concludere ecco delle tabelle (4.5,4.6,4.7) che riassumono i dati trovati, per ogni run, per le funzioni considerate.

Tabella 4.5: Immagine *Baby2*

Run	RMSE	finestra	pixel inconsistenti
0	6.26268	19	24.7%
1	1.9021	19	24.6%
2	1.8224	25	32.2%
3	1.82028	27	32.6%

Dato abastanza significativo in quest'immagine è come un netto miglioramento dell'erroe quadratico medio dal run0 al run1, accompagnato da un abbassamento (anche se minimo) del numero dei pixel inconsistenti, non comporti un altrettanto miglioramento nei valori della segmentazione che abbassano la loro media per entrambe le immagini.

Tabella 4.6: Immagine lr-*Baby2*

Run	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
0	0.0984	0.1028	0.2012
1	0.0885	0.1028	0.1904
2	0.0972	0.1015	0.1987
3	0.0975	0.1017	0.1992

Tabella 4.7: Immagine rl-*Baby2*

Run	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
0	0.1050	0.0978	0.2028
1	0.1050	0.1019	0.2019
2	0.1070	0.0974	0.2043
3	0.1078	0.0828	0.1905

4.2 Immagine *Baby1*

Run 0

Si considerano ora le immagini chiamate *Baby1* del data set, molto simile alla precedente come scena ma con risultati nei processi molto diversi. Si inizia con la segmentazione delle immagini considerado solo il colore e si ottengono le prime diparity map con una finestra di grandezza 19 pixel (fig. 4.13). Il dato relativo al RMSE è 2.79529 e i pixel inconsistenti sono il 22.6 %. Rilevante in queste disparity map è l'errore che si viene a creare su un lato del cubo sotto il bambolotto che compromette la segmentazione soprattutto per le immagini con λ alti che rispechiano maggiormente l'errore della disparity map. Ecco alcuni esempi della segmentazione risultanti da queste disparity map (fig. 4.14); già da lambda relativamente bassi si veda l'errore introdotto dalla disparity map. I grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$ sono nella fig. 4.15.
Come si vede bene nei grafici dell'mmagine lr, spesso picchi di massimo o minimo relativo in una funzione, non corrispondano a un massimo o minimo nell'altra; in questo caso, un picco di massimo in corrispondenza di $\lambda = 3$ nella funzione $Q^{color}(I,S)$ corrisponde a un minimo nella funzione $Q(I,D,S)$, mentre il minimo che si riscontra per $\lambda = 7$, corrisponde a un valore nella media nell'altra funzione; viceversa nell'immagine rl, i picchi di minimo e massimo relativo corrispondono nelle due funzioni: si osservi i valori per $\lambda = 7$ e $\lambda = 3$. Fortunatamente riguardo la nostra scelta di considerare per l'iterazione successiva solo la

(a) Immagine sx a colori

(b) Immagine dx a colori

(c) Disparity map lr run0

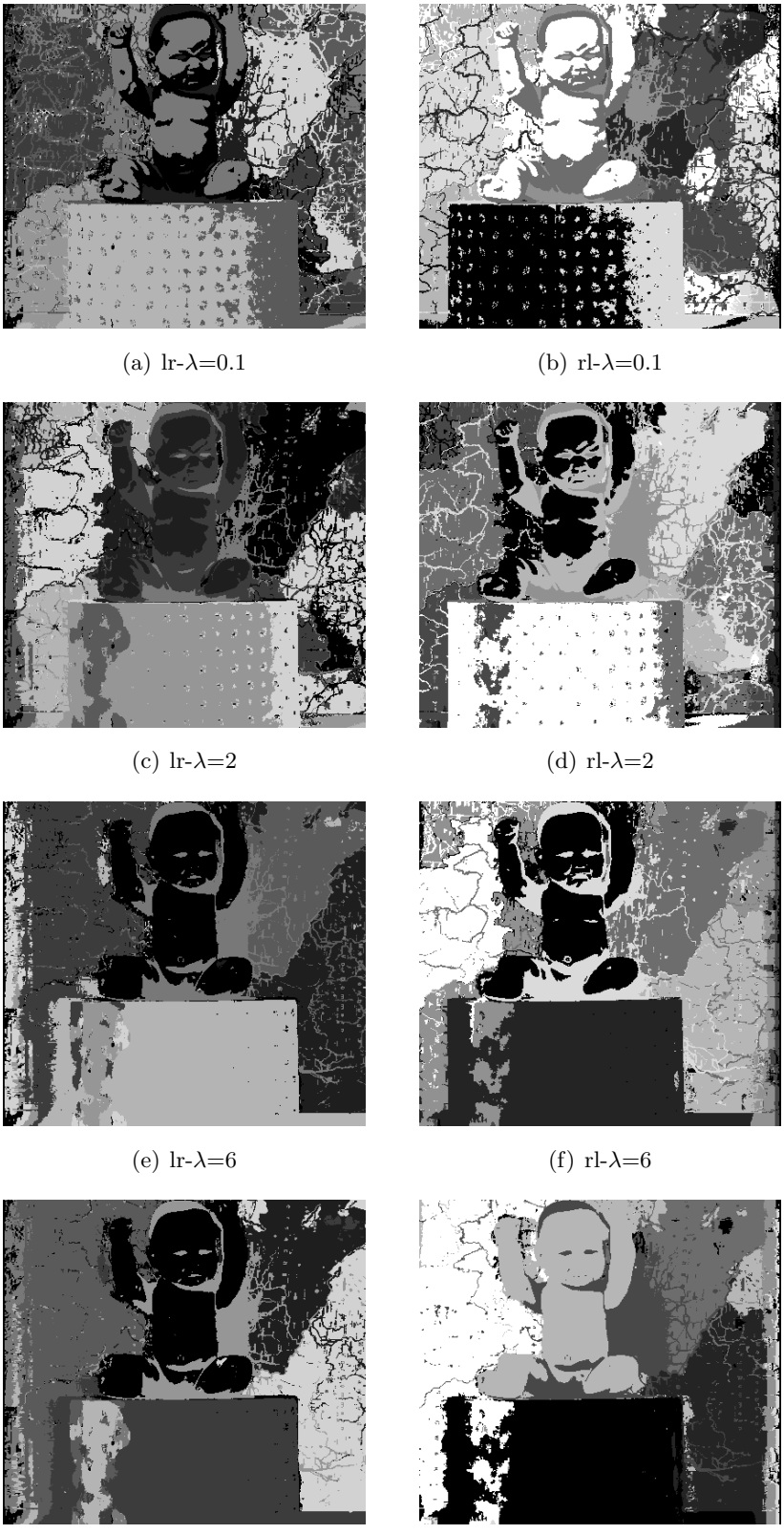
(d) Disparity map rl run0

Figura 4.13: Immagini a colori e disparity map al run0

$Q^{color}(I, S)$, scegliamo dei λ molto buoni per entrambe le funzioni. I valori medi riscontrati nelle tre funzioni sono nella tabella 4.8

Tabella 4.8: Run0 *Baby1*

Img	$Q^{color}(I, S)$	$Q^{depth}(D, S)$	$Q(I, D, S)$
lr	0.0550	0.0723	0.1272
rl	0.0534	0.0719	0.1253



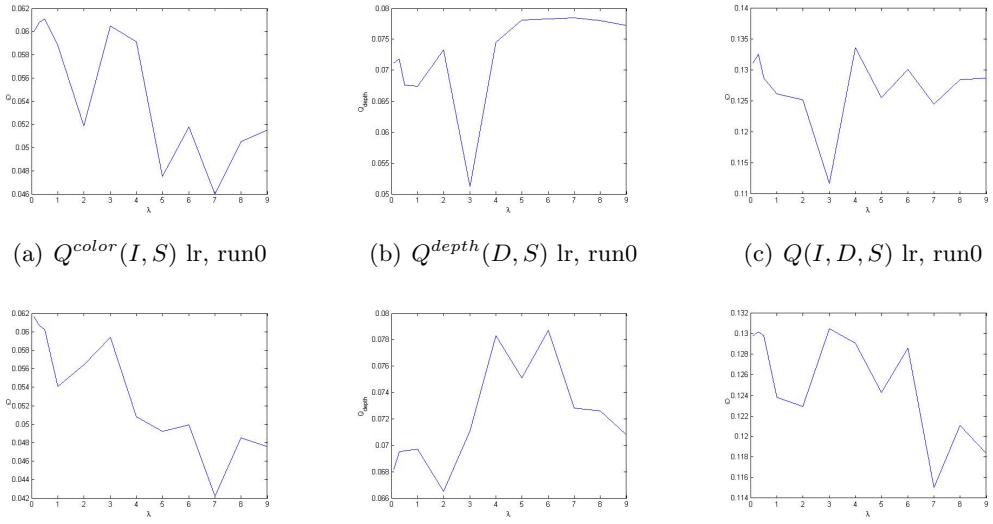
(a) lr-λ=0.1 (b) rl-λ=0.1

(c) lr-λ=2 (d) rl-λ=2

(e) lr-λ=6 (f) rl-λ=6

(g) lr-λ=9 (h) rl-λ=9

Figura 4.14: Esempi segmentazione al run0



(a) $Q^{color}(I,S)$ lr, run0 (b) $Q^{depth}(D,S)$ lr, run0 (c) $Q(I,D,S)$ lr, run0

(d) $Q^{color}(I,S)$ rl, run0 (e) $Q^{depth}(D,S)$ rl, run0 (f) $Q(I,D,S)$ rl, run0

Figura 4.15: grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$, run0

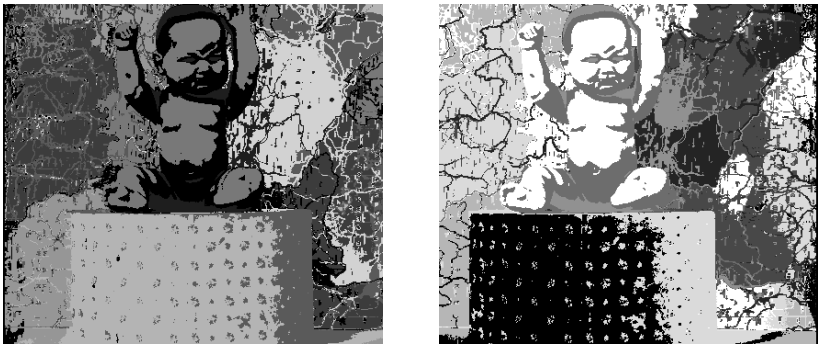
Run 1

Le disparity map ottenute derivano dalle immagni segmentate con $\lambda = 0.5$ per l'immagine lr e $\lambda = 0.1$ per rl; si è mantenuta costante a 19 pixel la grandezza della finestra e si è ottenuto un RMSE=2.71411 e numero di pixel inconsistenti pari al 21.2% (fig. 4.16). L'errore è ancora presente ma sicuramente diminuito, soprattutto nella disparity map rl. Da queste nuove disparity map si è segmentato ed ecco gli esempi nella fig. 4.17. In tutte le immagini, anche per λ alti, lo sfondo della mappa è ancora visibile per questo run e non si vede un sostanziale miglioramento nella segmentazione.
I grafici delle funzioni che valutano la segmentazione sono in fig. 4.18. I valori medi riscontrati nelle tre funzioni sono nella tabella 4.9: Se i valori delle funzioni per l'immagine

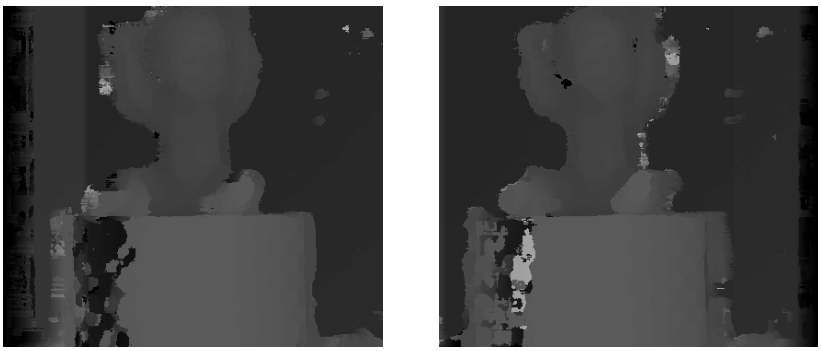
Tabella 4.9: Run1 *Baby1*

Img	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
lr	0.0523	0.0718	0.1240
rl	0.0547	0.0725	0.1271

rl aumentano entrambi, non è così per l'immagine lr; anche questa volta però i λ che massimizzano $Q^{color}(I,S)$ sono valori molto buoni anche per la funzione di riferimento.



(a) lr-λ=0.5 (b) rl-λ=0.1



(c) Disparity map lr run1 (d) Disparity map rl run1

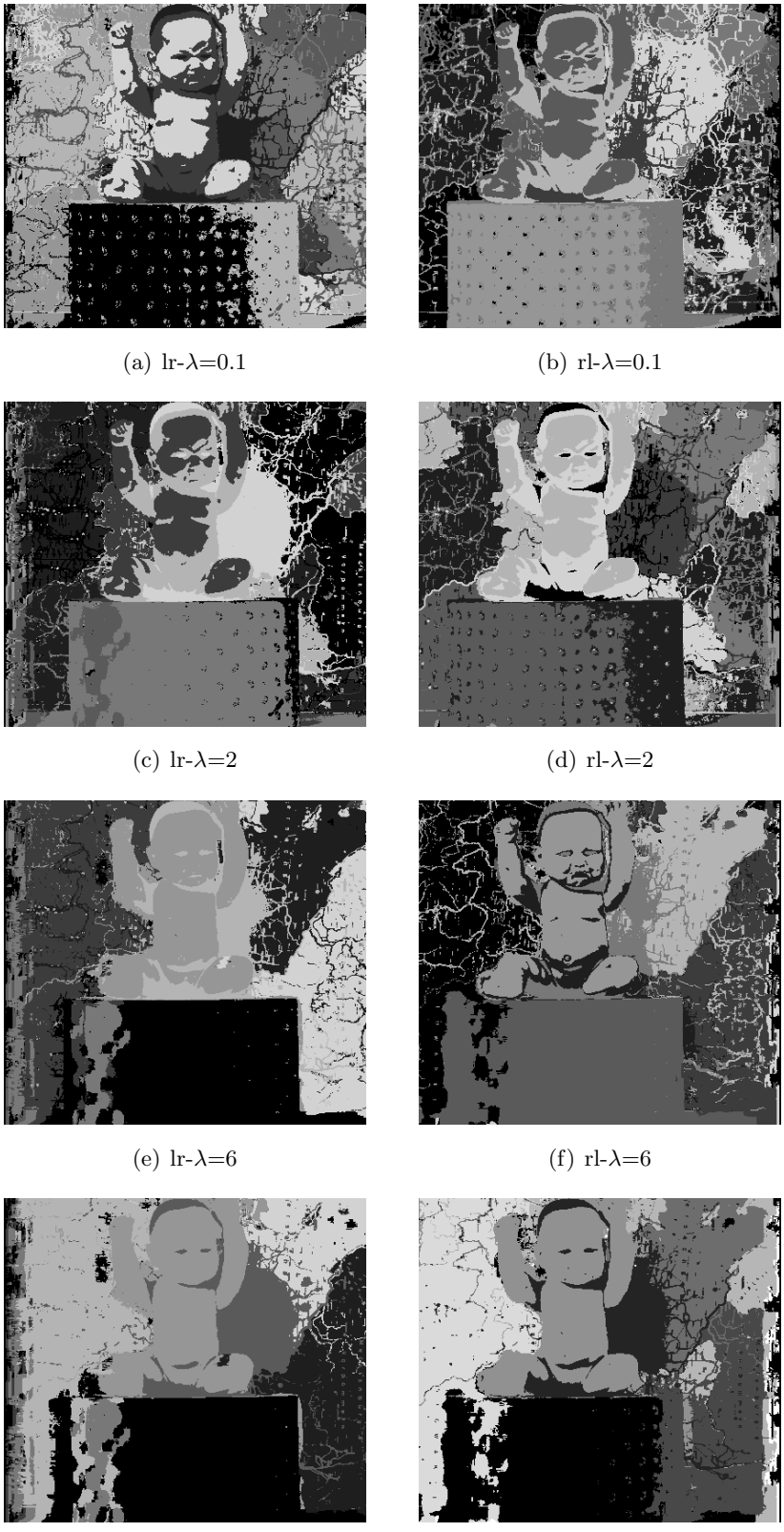
Figura 4.16: Immagini segmentate scelte dal run0 e disparity map del run1

Run 2

I parametri scelti per il nuovo run sono per lr $\lambda = 0.1$ e per rl $\lambda = 0.5$; abbassando questa volta la finestra a 15 pixel, dato che la finestra di grandezza 19 aveva dato RMSE=2.71678. Con la finestra invece di grandezza 15 si trova RMSE=2.56136, con pixel inconsistenti al 32% (fig. 4.19). Esempi della segmentazione con queste disparity map sono in fig. 4.20. L'errore della disparity map per questo run è un po' attenuato, in particolare per l'immagine rl ma la presenza del pattern di fondo è costante.
I grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$ sono in figura fig. 4.21. I valori medi riscontrati nelle due funzioni sono nella tabella 4.10: In questo run a scendere

Tabella 4.10: Run2 *Baby1*

Img	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
lr	0.0522	0.0720	0.1242
rl	0.0532	0.0692	0.1224



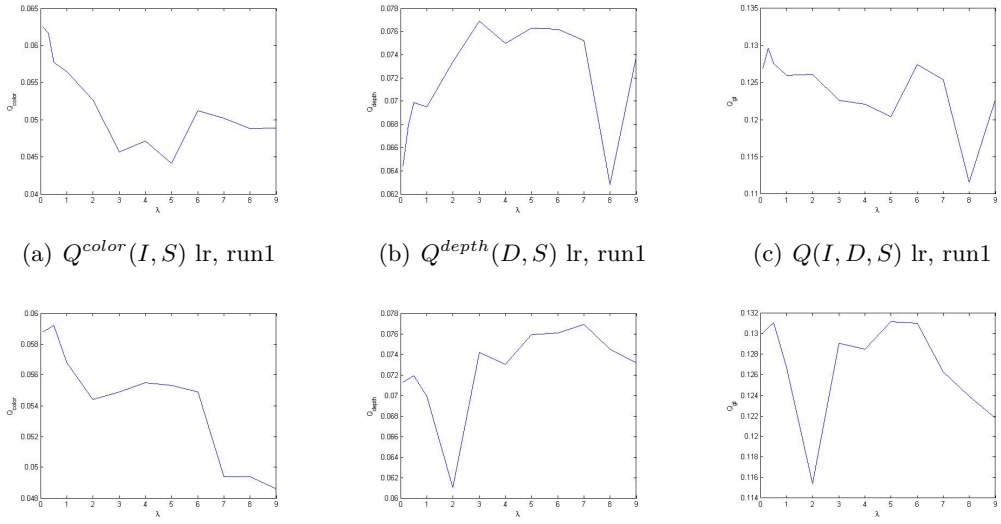
(a) lr-λ=0.1 (b) rl-λ=0.1

(c) lr-λ=2 (d) rl-λ=2

(e) lr-λ=6 (f) rl-λ=6

(g) lr-λ=9 (h) rl-λ=9

Figura 4.17: Esempi segmentazione al run1

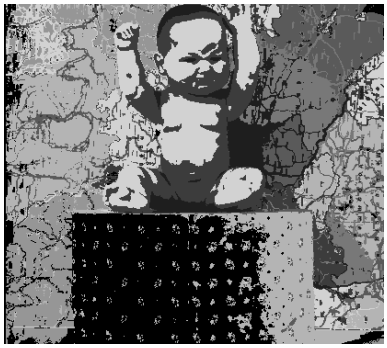


(a) $Q^{color}(I,S)$ lr, run1 (b) $Q^{depth}(D,S)$ lr, run1 (c) $Q(I,D,S)$ lr, run1

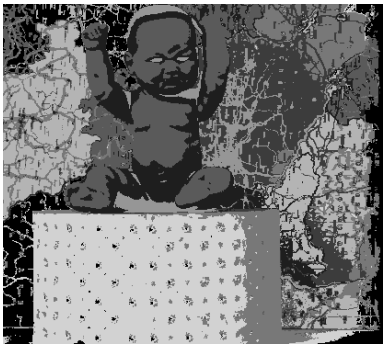
(d) $Q^{color}(I,S)$ rl, run1 (e) $Q^{depth}(D,S)$ rl, run1 (f) $Q(I,D,S)$ rl, run1

Figura 4.18: grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$, run1

sono le medie delle funzioni di rl, mentre rimangono stabili le funzioni di lr.

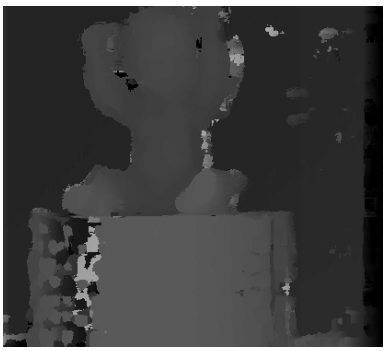


(a) lr-λ=0.1



(b) rl-λ=0.5

(c) Disparity map lr run2

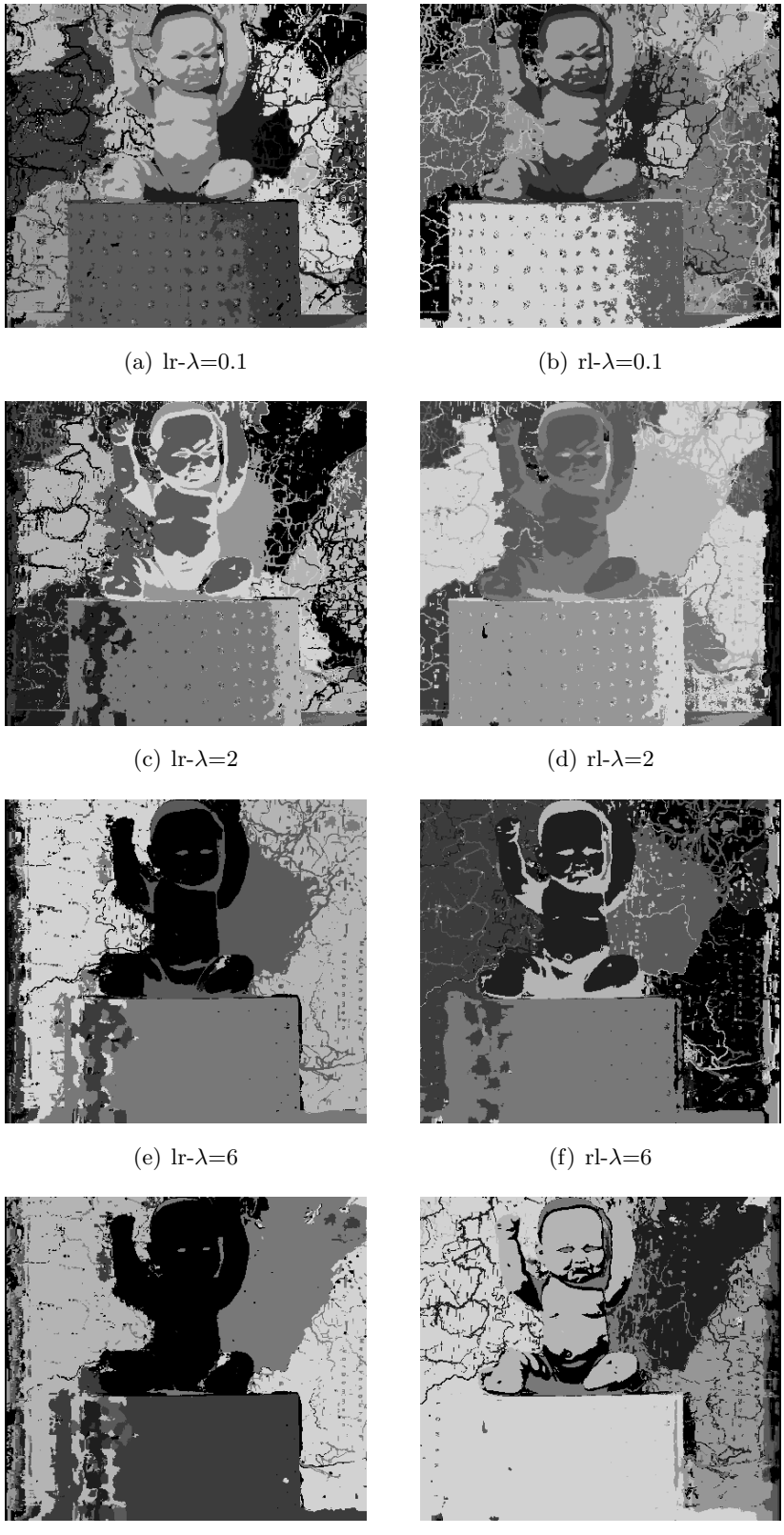


(d) Disparity map rl run2

Figura 4.19: Immagini segmentate scelte dal run1 e disparity map del run2

Run 3

Per il run 3 si è scelto il parametro $\lambda = 0.3$ per entrambe le figure; si è partiti per calcolare le disparity map da una finestra di grandezza 19 pixel che ha portato un risultato di RMSE=2.73427, si è abbassavo il valore della grandezza della finestra a 15 e si è ottenuto un RMSE=2.5823 e quindi si è abbassato ancora il valore a 11 trovando finalemte un miglioramento con un RMSE=2.4828 e pixel inconsistenti al 34.4% (fig. 4.22), dato molto superiore rispetto alla prima finestra la cui percentuale era 21.1%, compromesso da accettare se si vuole migliorare l'errore quadratico medio. Esempi della segmentazione con queste disparity map sono in fig. 4.23. Anche in questo run la mappa geografica sullo sfondo è un problema nella segmentazione e risulta ancora troppo rilevante nelle immagini; inoltre ritorna più accentuato l'errore legato alla disparity map, probabilmente a causa dell'alto numero di pixel inconsistenti e questo si nota in particolare per l'immagine rl con $\lambda = 9$ che nel run precedente invece lo presentava in minima parte.
I grafici delle funzioni $Q^{color}(I, S)$, $Q^{depth}(D, S)$ e $Q(I, D, S)$ sono in figura fig. 4.24.



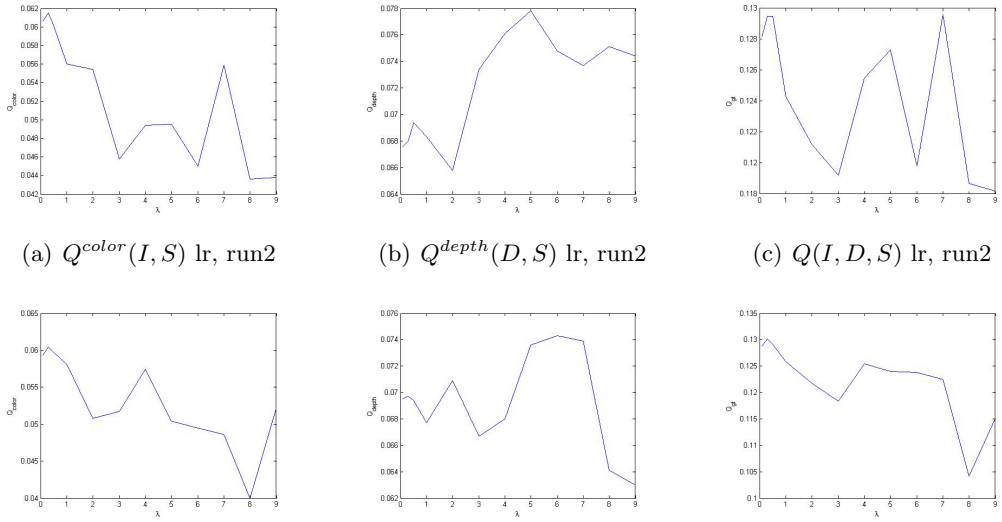
(a) lr-λ=0.1 (b) rl-λ=0.1

(c) lr-λ=2 (d) rl-λ=2

(e) lr-λ=6 (f) rl-λ=6

(g) lr-λ=9 (h) rl-λ=9

Figura 4.20: Esempi segmentazione al run2



(a) $Q^{color}(I,S)$ lr, run2 (b) $Q^{depth}(D,S)$ lr, run2 (c) $Q(I,D,S)$ lr, run2

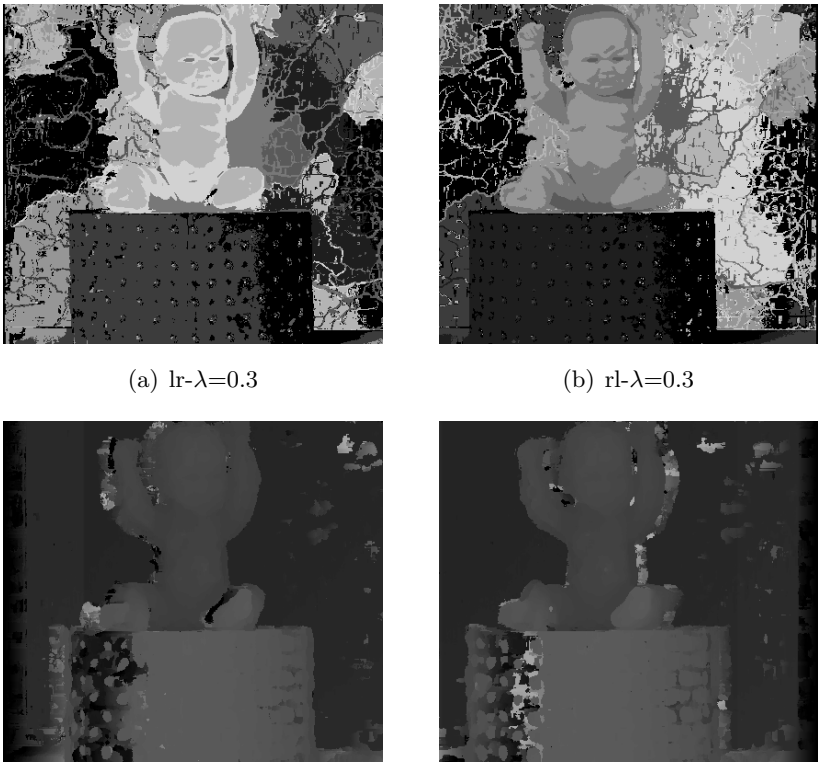
(d) $Q^{color}(I,S)$ rl, run2 (e) $Q^{depth}(D,S)$ rl, run2 (f) $Q(I,D,S)$ rl, run2

Figura 4.21: grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$, run2

I valori medi riscontrati nelle due funzioni sono nella tabella 4.11. Le medie mi abbassano per entrambe le funzioni e immagini, frutto forse del ritorno più accentuato dell'errore.

Tabella 4.11: Run3 *Baby1*

Img	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
lr	0.0514	0.0698	0.1212
rl	0.0509	0.0696	0.1205



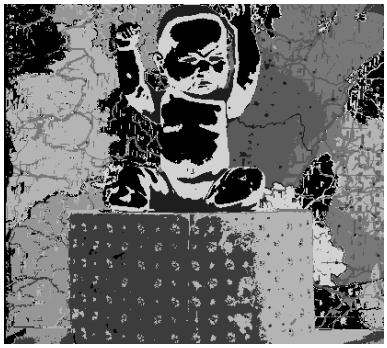
(a) lr-λ=0.3 (b) rl-λ=0.3

(c) Disparity map lr run3 (d) Disparity map rl run3

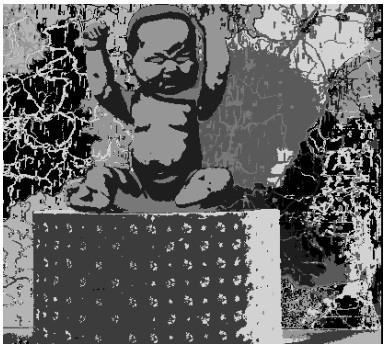
Figura 4.22: Immagini segmentate scelte dal run2 e disparity map del run3

4.2.1 Run 4

L'ultimo run per questa immagine è stato fatto scegliendo le immagini segmentate col fattore $\lambda = 0.1$ per entrambe (immagini già presenti entrambe in fig. 4.23); per ottenere le due nuove disparity map si sono dovuti fare più tentativi con finestre di grandezza diversa infatti con una finestra di 15 pixel si riscontrava un RMSE=2.59078, con 11 un RMSE=2.49787 e neppure con 9 si è ottenuto un miglioramento con un RMSE=2.56823, per cui si opta per scegliere le disparity map ottenute con una finestra di 11 che hanno un numero di pixel inconsistenti al 34.5% (fig. 4.25). Esempi della segmentazione con queste disparity map sono in fig. 4.26. Anche dopo quattro run non si è riusciti a correggere l'errore scaturito dall'algoritmo del calcolo di disparity map, simbolo questo che è fondamentale avere una disparity map iniziale il più possibile senza buchi perchè questi non possono essere riempiti con l'aumentare dei run.
I grafici delle funzioni $Q^{color}(I, S)$, $Q^{depth}(D, S)$ e $Q(I, D, S)$ sono in figura fig. 4.27. I valori medi riscontrati nelle tre funzioni sono nella tabella 4.11. Si chiude l'analisi dei run per quest'immagine sostanzialmente con un saldo positivo per quanto riguarda le funzioni per la valutazione della segmentazione anche se nell'ultimo run non si è riusciti



(a) lr-λ=0.1



(b) rl-λ=0.1

(c) lr-λ=2

(d) rl-λ=2

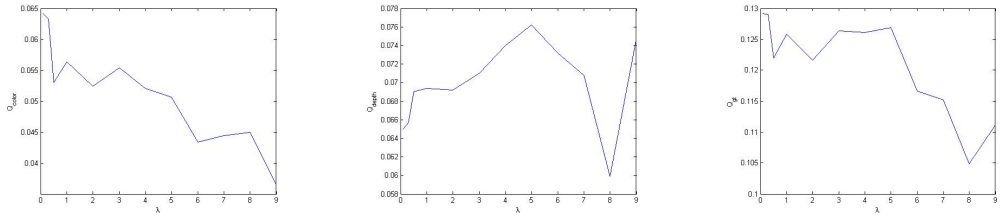
(e) lr-λ=6

(f) rl-λ=6

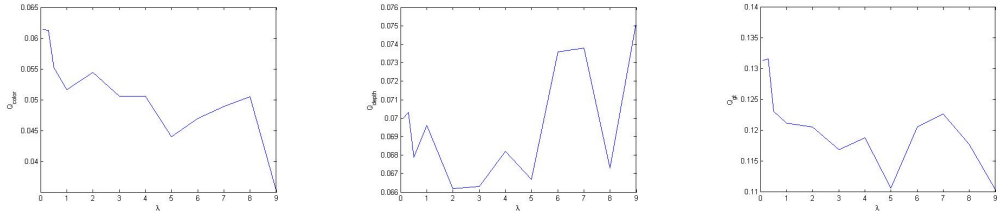
(g) lr-λ=9

(h) rl-λ=9

Figura 4.23: Esempi segmentazione al run3

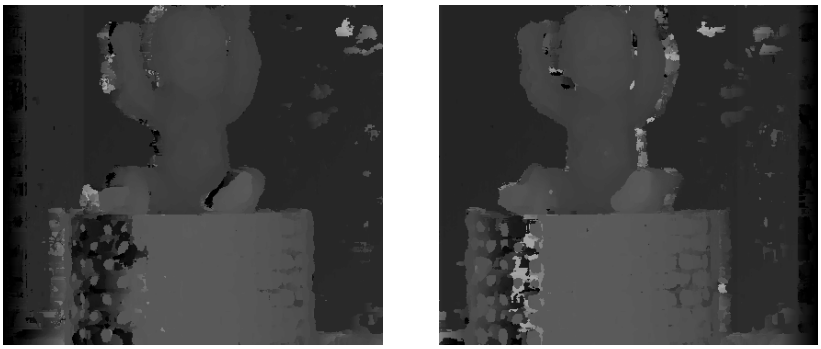


(a) $Q^{color}(I,S)$ lr, run3 (b) $Q^{depth}(D,S)$ lr, run3 (c) $Q(I,D,S)$ lr, run3



(d) $Q^{color}(I,S)$ rl, run3 (e) $Q^{depth}(D,S)$ rl, run3 (f) $Q(I,D,S)$ rl, run3

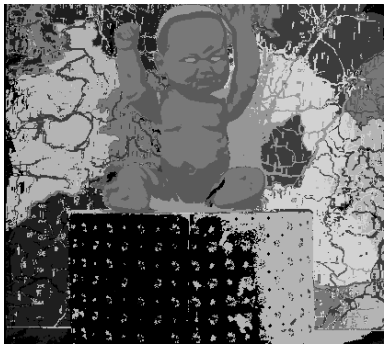
Figura 4.24: grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$, run3



(a) Disparity map lr run4 (b) Disparity map rl run4

Figura 4.25: Disparity map del run4

a migliorare il dato relativo all'errore quadratico medio. Per concludere le tabelle 4.13, 4.14, 4.7 che riassumono i dati trovati per ogni run per le funzioni considerate:



(a) lr-λ=0.1

(b) rl-λ=0.1

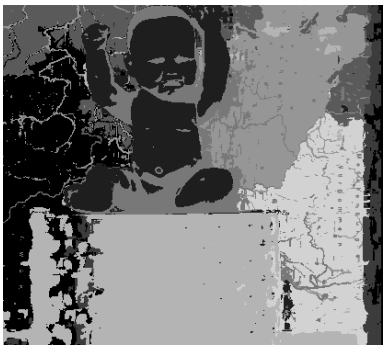
(c) lr-λ=2

(d) rl-λ=2

(e) lr-λ=6

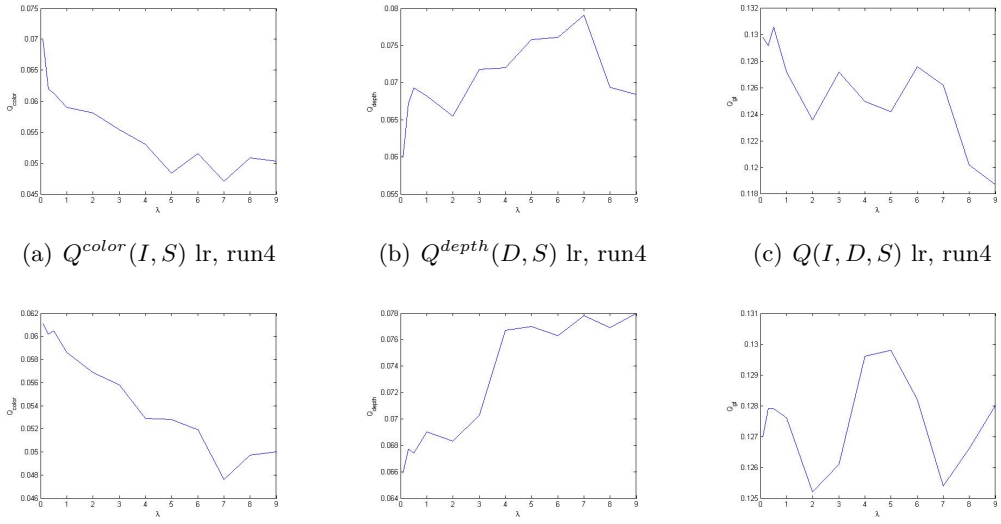
(f) rl-λ=6

(g) lr-λ=9



(h) rl-λ=9

Figura 4.26: Esempi segmentazione al run4



(a) $Q^{color}(I,S)$ lr, run4 (b) $Q^{depth}(D,S)$ lr, run4 (c) $Q(I,D,S)$ lr, run4

(d) $Q^{color}(I,S)$ rl, run4 (e) $Q^{depth}(D,S)$ rl, run4 (f) $Q(I,D,S)$ rl, run4

Figura 4.27: grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$, run4

Tabella 4.12: Run4 *Baby1*

Img	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
lr	0.0556	0.0702	0.1258
rl	0.0548	0.0726	0.1274

4.3 Immagine Bowling2

Run 0

Si analizza ora un'immagine più complessa delle precedenti perchè composta da più oggetti su più piani diversi e posizionati anche sui bordi delle immagini, punti sempre critici per il calcolo della disparity map; l'immagine è chiamata *Bowling2* e si comincia il calcolo delle disparity map, come sempre, dalle due immagini segmentate considerando solo il colore e usando una finestra di correlazione per l'algoritmo *SegmentSupport* pari a 19 pixel. In figura 4.28 sono mostrate le immagini a colori e le prime disparity map. Si ottiene un RMSE=2.31872 e una percentuale di pixel inconsisteni al 30.5%, alta per essere il primo run. Esempi della segmentazione sono nella fig. 4.29. Da notare che nessuna segmentazione riconosce la palla centrale come un elemento unico ma la divivono tutte almeno in due o tre zone; inoltre un altro punto critico del processo è la distinzione della testa del birillo con la palla, effettuata solo per segmentazioni con λ bassi. In queste disparity per λ alti invece è rilevante l'errore introdotto sia nella parte sinistra

Tabella 4.13: Immagine *Baby1*

Run	RMSE	finestra	pixel inconsistenti
0	2.79529	19	22.6%
1	2.71411	19	21.2%
2	2.56136	15	32%
3	2.4828	11	34.4%
4	2.49787	11	34.5%

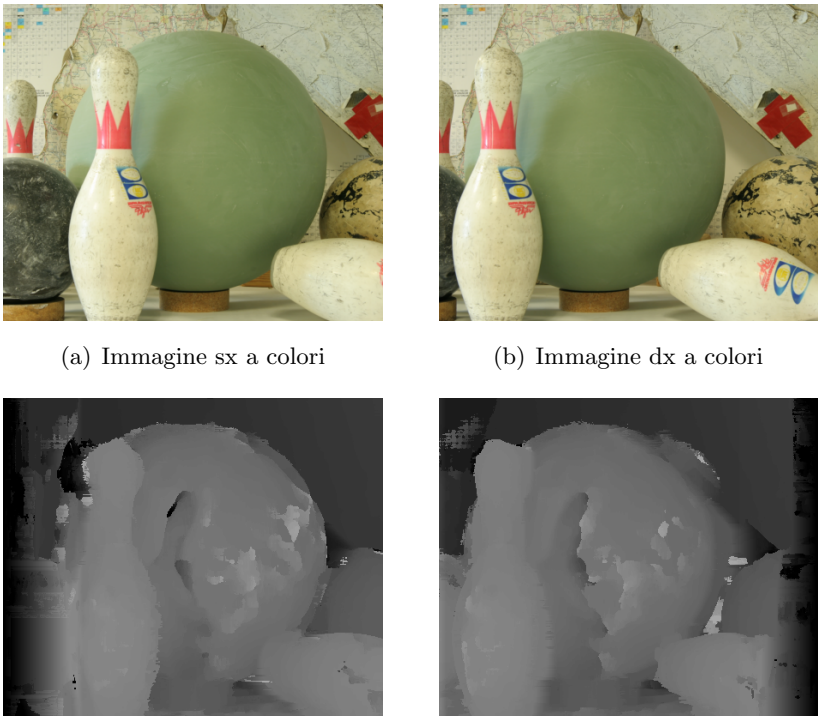
Tabella 4.14: Immagine lr-*Baby1*

Run	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
0	0.0550	0.0723	0.1272
1	0.0523	0.0718	0.1240
2	0.0522	0.0720	0.1242
3	0.0514	0.0698	0.1212
4	0.0556	0.0702	0.1258

dell'immagine lr, sia in quella destra dell'immagine rl, compromettendo la precisione della segmentazione ai lati dell'immagini.
I grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$ sono in figura fig. 4.30. Si evidenzia in questo caso come la scelta della funzione $Q^{color}(I,S)$ come parametro di giudizio per la selezione delle immagine segmentate corrispondenti al λ ottimo non è sempre la migliore: nello specifico i grafici delle due funzioni hanno andamento opposto; $Q(I,D,S)$ evidenzia una migliore segmentazione per i valori alti di λ, mentre noi andremo a scegliere valori bassi, basandoci sull'altra funzione.
I valori medi riscontrati nelle tre funzioni sono nella tabella 4.16 Nonostante la complessità dell'immagine si ottengono comunque dei buoni valori medi in relazione alle altre immagini analizzate.

Tabella 4.15: Immagine rl-*Baby1*

Run	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
0	0.0550	0.0719	0.1253
1	0.0547	0.0725	0.1271
2	0.0532	0.0692	0.1224
3	0.0509	0.0696	0.1205
4	0.0548	0.0726	0.1274



(a) Immagine sx a colori

(b) Immagine dx a colori

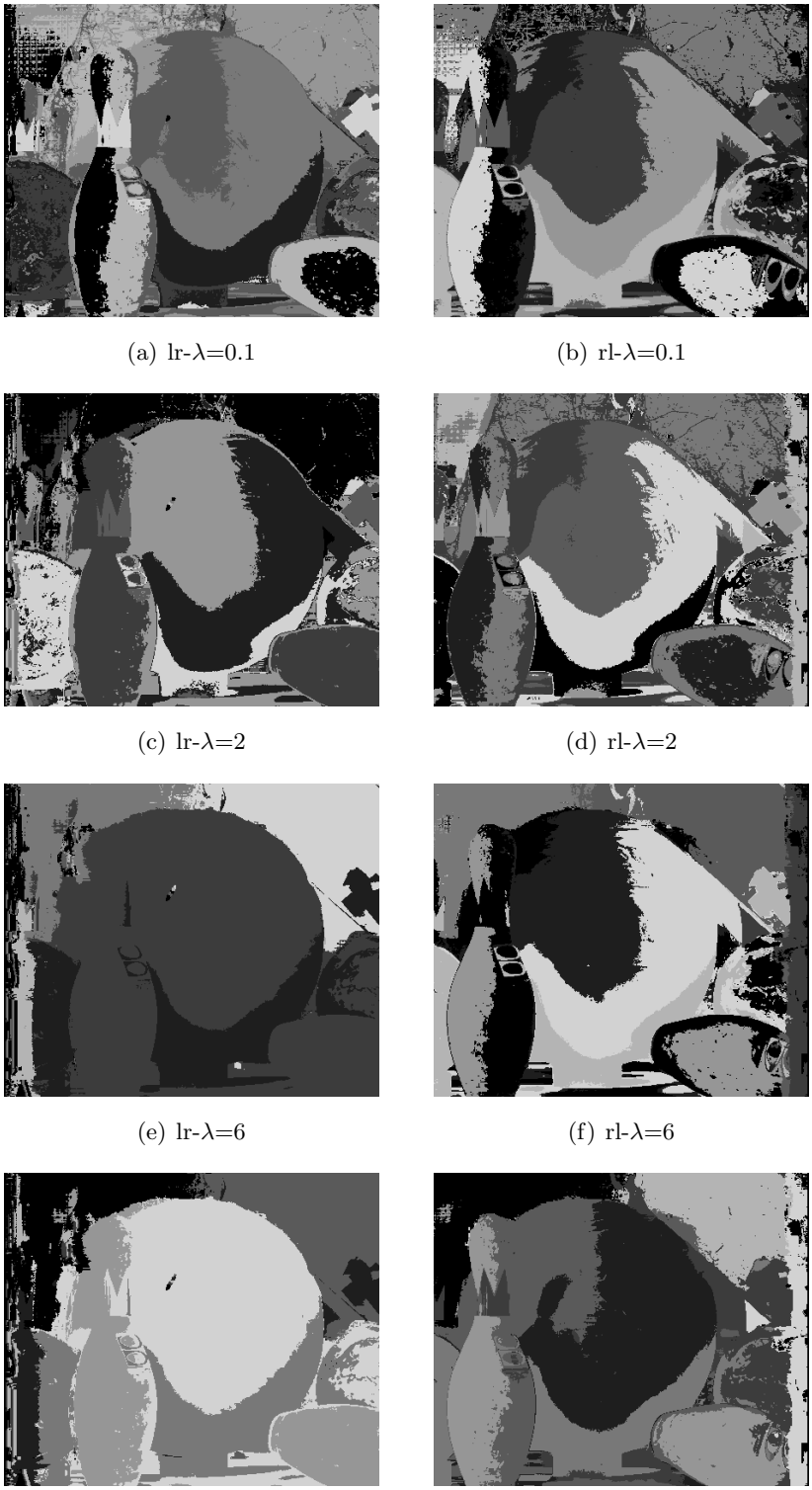
(c) Disparity map lr run0

(d) Disparity map rl run0

Figura 4.28: Immagini a colori e disparity map al run0

Run 1

Per cui, seguendo la nostra scelta, scegliamo le immagini segmentate con $\lambda = 0.3$ da inserire nel processo del calcolo della disparity map. Mantenendo la grandezza del run precedente non si ottiene un peggioramento del RMSE e diminuendo la finestra si ottiene inizialmente un leggero migioramento e poi un nuovo peggioramento; si ottiene il miglioramento sperato invece ingrandendo la finestra a 21 pixel, analogamente a quanto era successo con l'immagine *Baby2*: in questo caso RMSE=2.2983 con pixel inconsistenti al



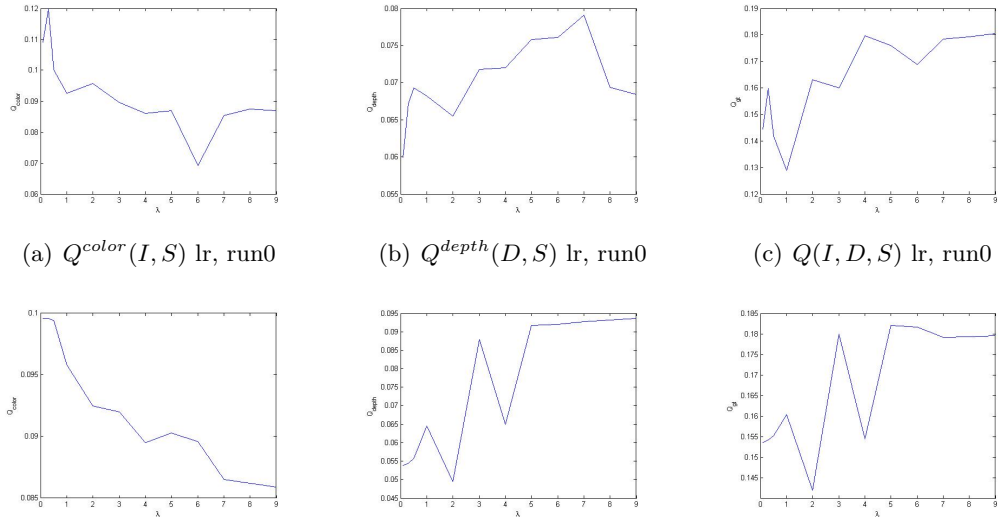
(a) lr-λ=0.1 (b) rl-λ=0.1

(c) lr-λ=2 (d) rl-λ=2

(e) lr-λ=6 (f) rl-λ=6

(g) lr-λ=9 (h) rl-λ=9

Figura 4.29: Esempi segmentazione al run0



(a) $Q^{color}(I,S)$ lr, run0 (b) $Q^{depth}(D,S)$ lr, run0 (c) $Q(I,D,S)$ lr, run0

(d) $Q^{color}(I,S)$ rl, run0 (e) $Q^{depth}(D,S)$ rl, run0 (f) $Q(I,D,S)$ rl, run0

Figura 4.30: grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$, run0

Tabella 4.16: Run0 *Bowling2*

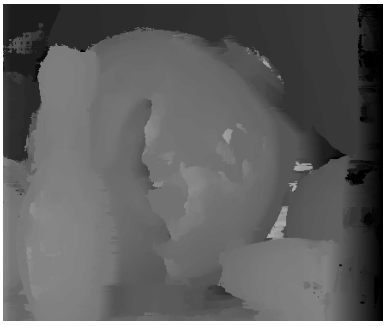
Img	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
lr	0.0924	0.0702	0.1634
rl	0.0922	0.0745	0.1667

30.4 % (fig. 4.31). Esempi delle segmentazioni risultanti sono in fig. 4.32 I problemi della segmentazione della palla e birillo non sono risolti però è riconosciuto meglio lo sfondo come entità unica rispetto alle immagini precedenti.
I grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$ sono in figura fig. 4.33.
Anche in questo caso si andranno a scegliere due valori di λ lontati dagl'ottimi ideali per la segmentazione ma comunque si otterrà un miglioramento del RMSE.

(a) lr-λ=0.3

(b) rl-λ=0.3

(c) Disparity map lr run1



(d) Disparity map rl run1

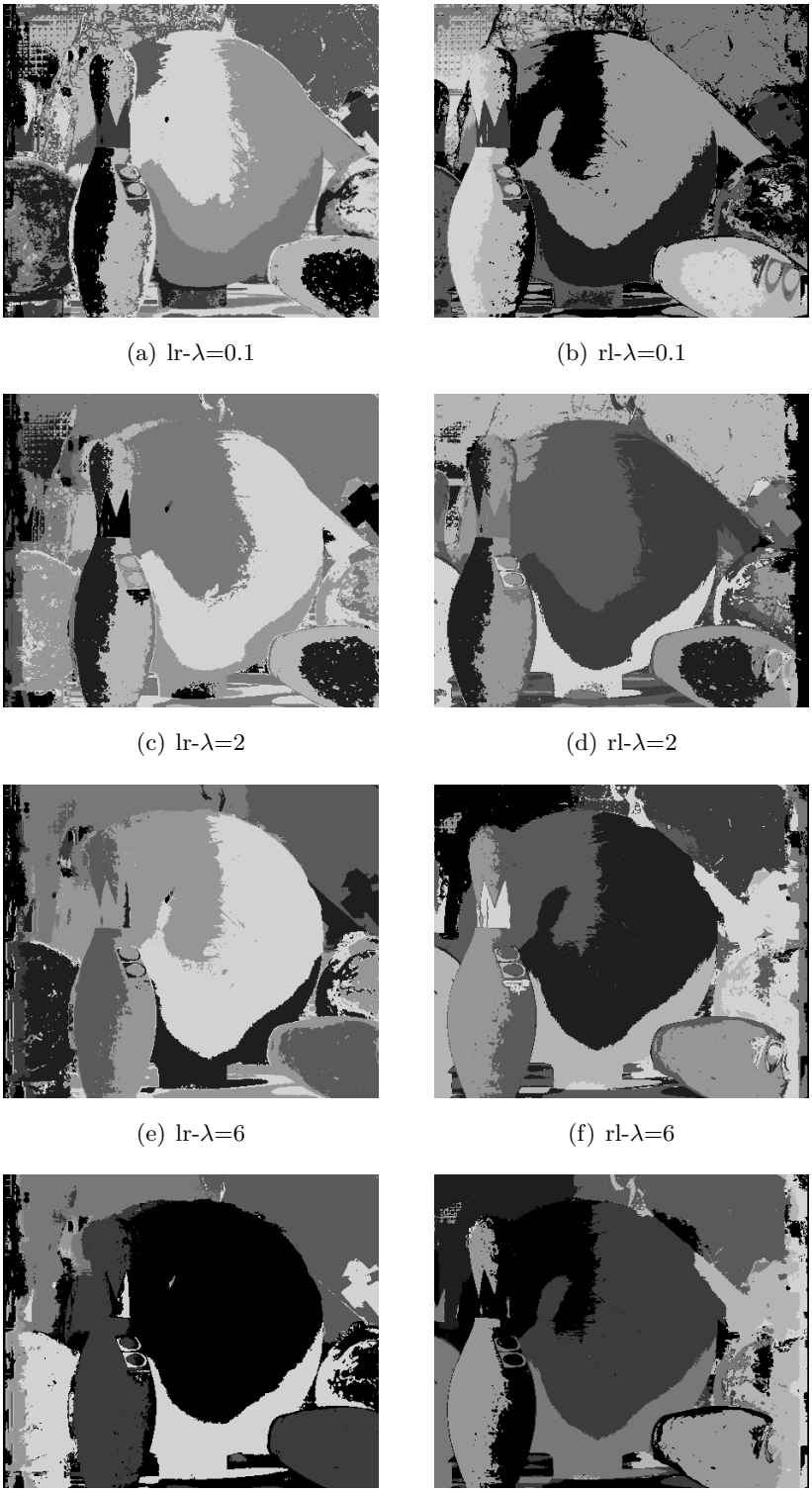
Figura 4.31: Immagini segmentate scelte dal run0 e disparity map del run1

I valori medi riscontrati nelle tre funzioni sono nella tabella 4.17. Netto peggioramento

Tabella 4.17: Run1 *Bowling2*

Img	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
lr	0.0887	0.0632	0.1519
rl	0.0943	0.0748	0.1691

del valore della funzione lr-$Q(I,D,S)$.



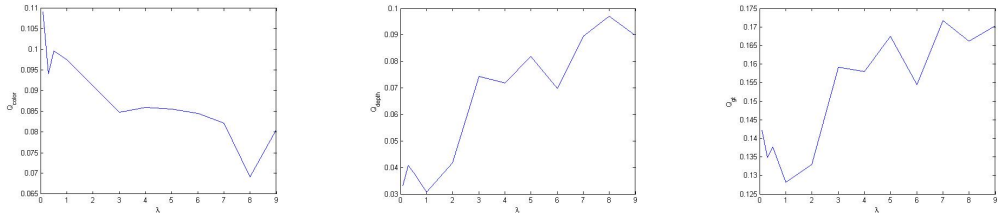
(a) lr-λ=0.1 (b) rl-λ=0.1

(c) lr-λ=2 (d) rl-λ=2

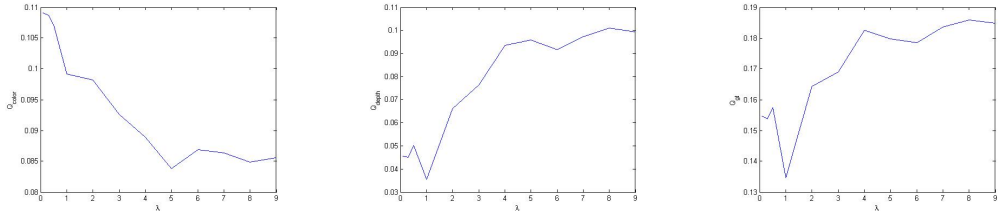
(e) lr-λ=6 (f) rl-λ=6

(g) lr-λ=9 (h) rl-λ=9

Figura 4.32: Esempi segmentazione al run1



(a) $Q^{color}(I,S)$ lr, run1 (b) $Q^{depth}(D,S)$ lr, run1 (c) $Q(I,D,S)$ lr, run1

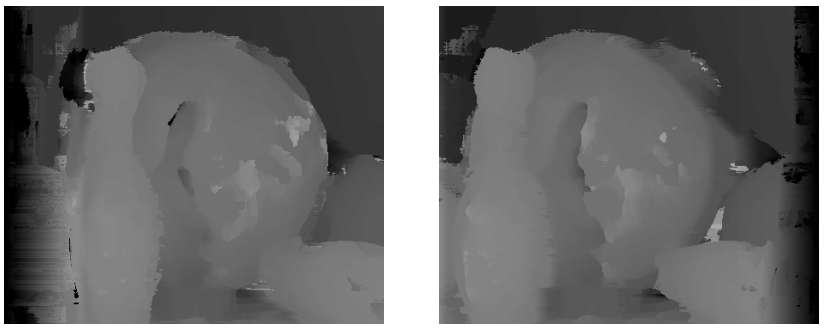


(d) $Q^{color}(I,S)$ rl, run1 (e) $Q^{depth}(D,S)$ rl, run1 (f) $Q(I,D,S)$ rl, run1

Figura 4.33: grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$, run1

Run2

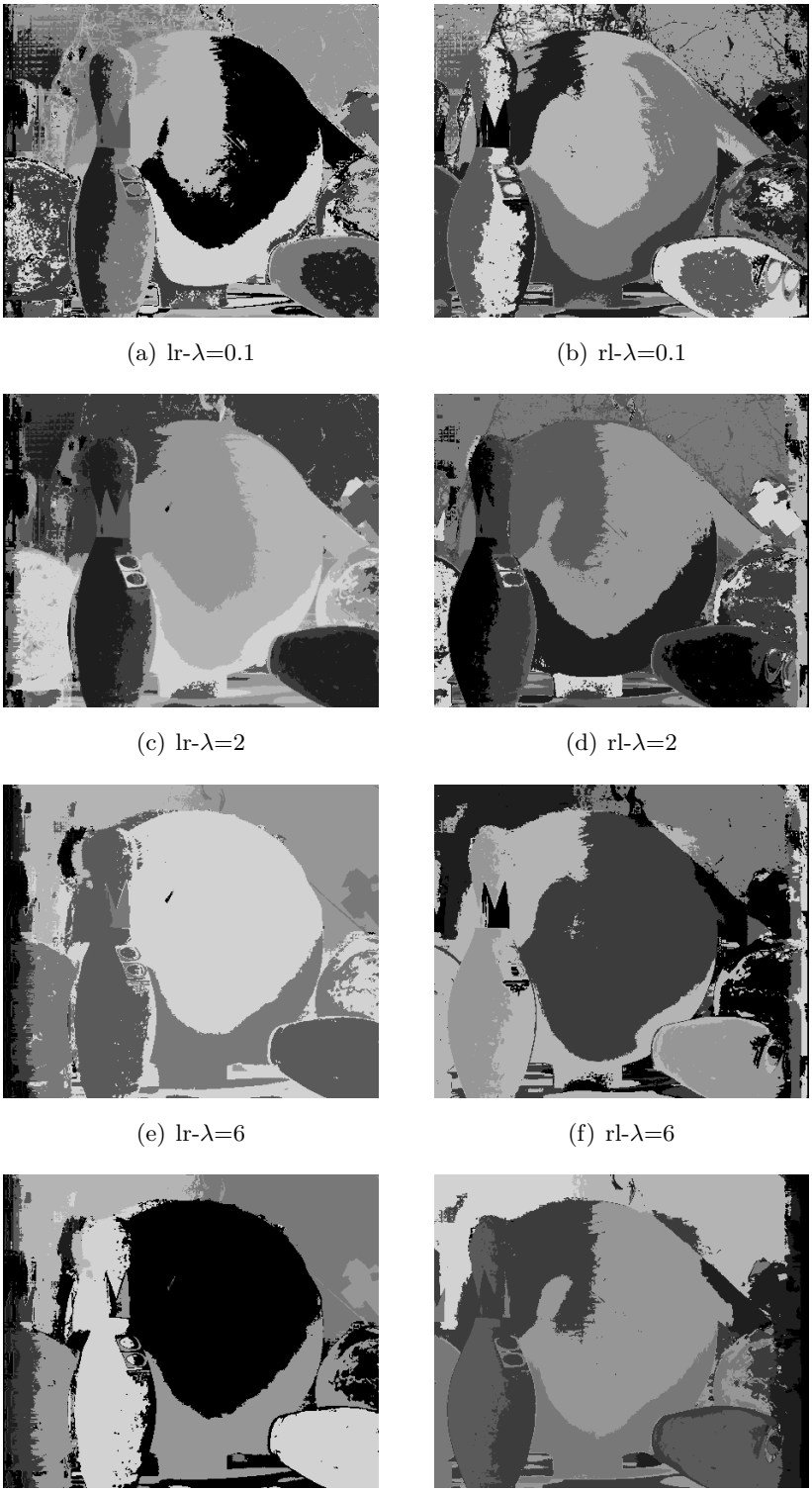
Per il run 2 si sono scelte per entrambe le immagini la segmentazione con $\lambda = 0.1$ (già presenti in fig. 4.32), ottenendo un miglioramento dell'errore quadratico medio aumentando ancora la finestra a 23 pixel: RMSE=2.17091 con pixel inconsistenti al 33% (fig.) Pur non avendo scelto dei valori ottimi per la segmentazione delle immagini da inserire



(a) Disparity map lr run2 (b) Disparity map rl run2

Figura 4.34: Immagini segmentate scelte dal run1 e disparity map del run2

nel processo, si ottiene comunque un miglioramento, anche se minimo, nel calcolo delle disparity map, mantenendo anche invariato il numero di pixel inconsistenti. Esempi della segmentazione di questo run in fig. 4.35. I grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$ sono in figura fig. 4.36.



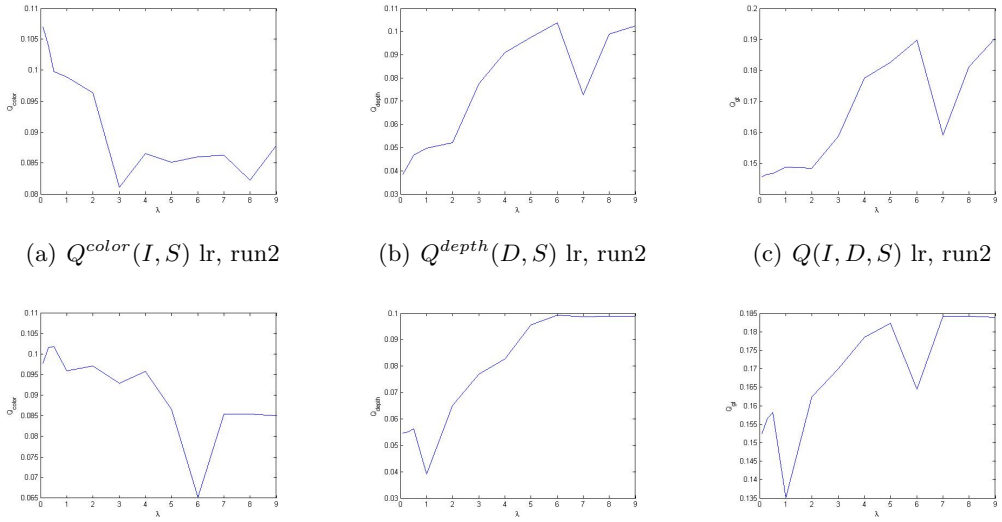
(a) lr-λ=0.1 (b) rl-λ=0.1

(c) lr-λ=2 (d) rl-λ=2

(e) lr-λ=6 (f) rl-λ=6

(g) lr-λ=9 (h) rl-λ=9

Figura 4.35: Esempi segmentazione al run2



(a) $Q^{color}(I,S)$ lr, run2 (b) $Q^{depth}(D,S)$ lr, run2 (c) $Q(I,D,S)$ lr, run2

(d) $Q^{color}(I,S)$ rl, run2 (e) $Q^{depth}(D,S)$ rl, run2 (f) $Q(I,D,S)$ rl, run2

Figura 4.36: grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$, run2

Anche in questo run è confermato il fatto che la segmentazione è migliore per λ alti. I valori medi riscontrati nelle tre funzioni sono nella tabella 4.18. Si riscontra nei run

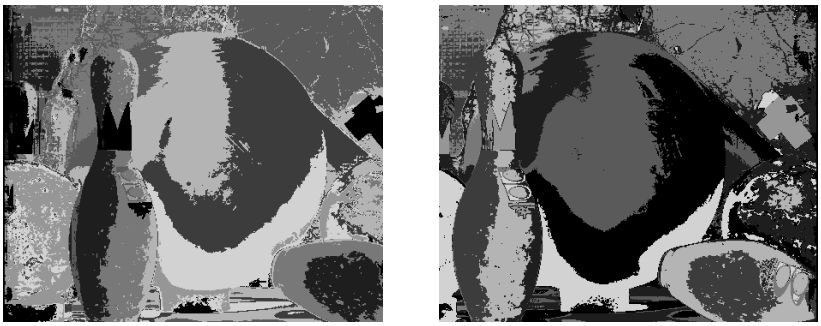
Tabella 4.18: Run2 *Bowling2*

Img	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
lr	0.0918	0.0728	0.1645
rl	0.0909	0.0768	0.1677

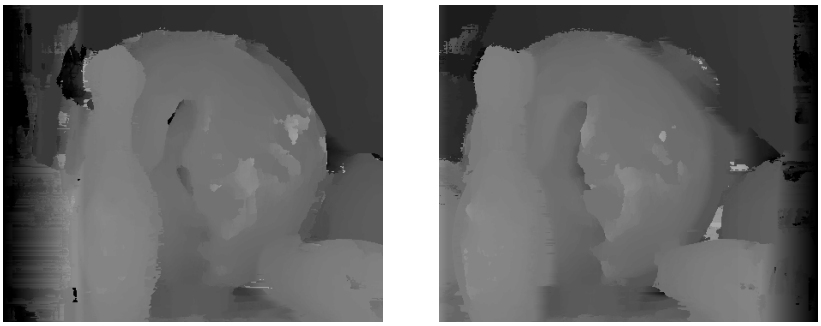
di quest'immagine un andamento altalenante dei valori delle funzioni per valutare la segmentazione: nel run1 i valori dell'immagine rl erano scesi di molto, mentre quelli di lr erano saliti; al secondo run accade il contrario sebbene non si notino differenze sostanziali nelle disparity map, rispetto ai run precedenti, da giustificare questo andamento.

Run 3

Per il run successivo si selezionano quindi per entrambe le immagini, le segmentazioni ottenute con il valore $\lambda = 0.3$, anche in questo caso, non una scelta ottima o vicino all'ottimo. Si è ottenuto un miglioramento mantenendo la finestra di correlazione al valore 23 pixel, ottenendo un RMSE=2.16017 con pixel inconsistenti al 33.1%, un leggero aumento rispetto al run precedente (fig. 4.37). Esempi delle immagini segmentate risultanti sono



(a) lr-λ=0.3 (b) rl-λ=0.3



(c) Disparity map lr run3 (d) Disparity map rl run3

Figura 4.37: Immagini segmentate scelte dal run2 e disparity map del run3

in fig. 4.38 L'aumento della finestra ha comportato però contorni meno definiti e netti. I grafici delle funzioni $Q^{color}(I, S)$, $Q^{depth}(D, S)$ e $Q(I, D, S)$ sono in figura fig. 4.39. I valori relativi all'immagine lr continuano la loro risalita dopo la brusca discesa del run1, mentre quelli per lr scendono ancora nonostante anche la funzione rl-$Q^{color}(I, S)$ aumenti. I valori medi riscontrati nelle tre funzioni sono nella tabella 4.19.

(a) lr-λ=0.1

(b) rl-λ=0.1

(c) lr-λ=2

(d) rl-λ=2

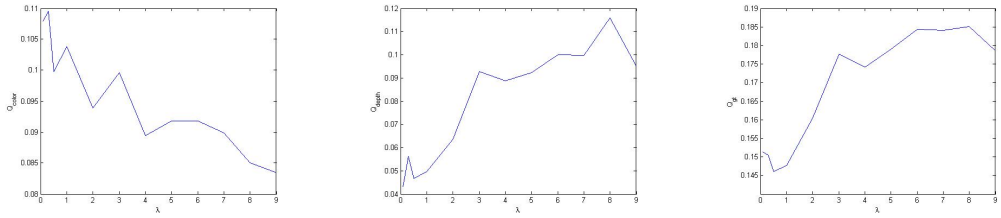
(e) lr-λ=6

(f) rl-λ=6

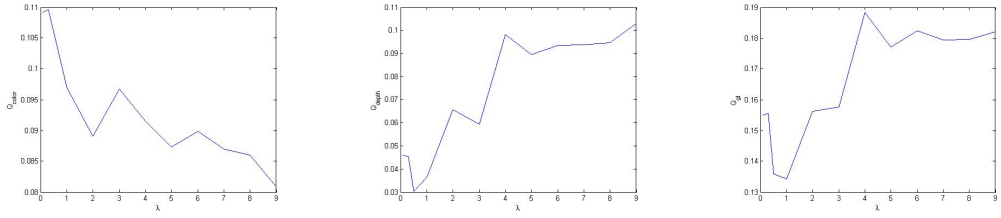
(g) lr-λ=9

(h) rl-λ=9

Figura 4.38: Esempi segmentazione al run3



(a) $Q^{color}(I,S)$ lr, run3 (b) $Q^{depth}(D,S)$ lr, run3 (c) $Q(I,D,S)$ lr, run3



(d) $Q^{color}(I,S)$ rl, run3 (e) $Q^{depth}(D,S)$ rl, run3 (f) $Q(I,D,S)$ rl, run3

Figura 4.39: grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$, run3

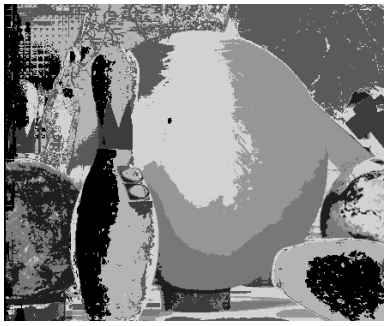
Tabella 4.19: Run3 *Bowling2*

Img	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
lr	0.0955	0.0726	0.1682
rl	0.0942	0.0711	0.1653

Run 4

Per l'ultimo run si sono scelte le immagine segmentate corrispondenti al valore di $\lambda = 0.1$ per lr e $\lambda = 0.5$ per rl, provando inizialmente ad ottenere un miglioramento mantenendo la grandezza della finestra del run precedende, mentre il miglioramento voluto si ottiene per una finestra di grandezza 25 pixel: RMSE=2.1078 con pixel inconsistenti al 33.5% (fig.4.40).
Esempi delle immagini segmentate risultanti sono in fig. 4.41
I grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$ sono in figura fig. 4.42. I valori relativi all'immagine lr continuano la loro risalita dopo la brusca discesa del run1, mentre quelli per lr scendono ancora nonostante anche la funzione rl-$Q^{color}(I,S)$ aumenti. I valori medi riscontrati nelle tre funzioni sono nella tabella 4.20.

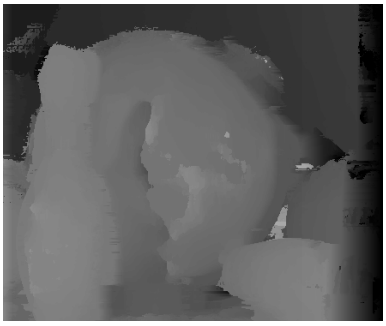
Per concludere le tabelle 4.21, 4.22, 4.23 che riassumono i dati trovati per ogni run per le funzioni considerate. Gli andamenti delle due funzioni risultano quasi complementari: lr-$Q(I,D,S)$ dopo un brusco calo al run1, aumenta ad ogni iterazione e termina con un



(a) lr-λ=0.1

(b) rl-λ=0.5

(c) Disparity map lr run4



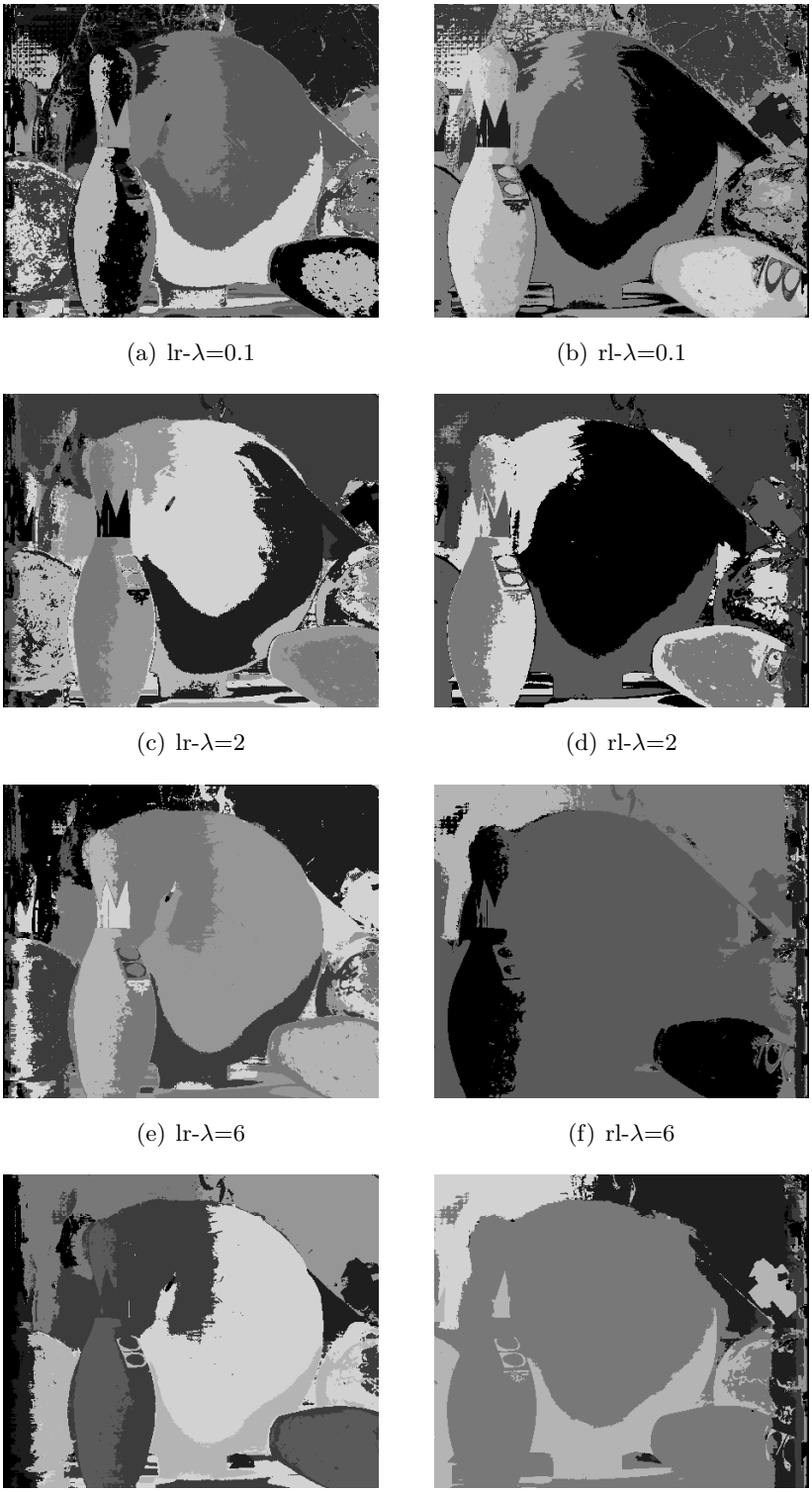
(d) Disparity map rl run4

Figura 4.40: Immagini segmentate scelte dal run3 e disparity map del run4

Tabella 4.20: Run4 *Bowling2*

Img	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
lr	0.0986	0.0728	0.1714
rl	0.0796	0.0802	0.1596

saldo nettamente positivo rispetto al run0; la corrispettiva funzione di rl invece ha un aumento al run1 e dopo un continuo calo che termina con valori molto più bassi rispetto al run0.



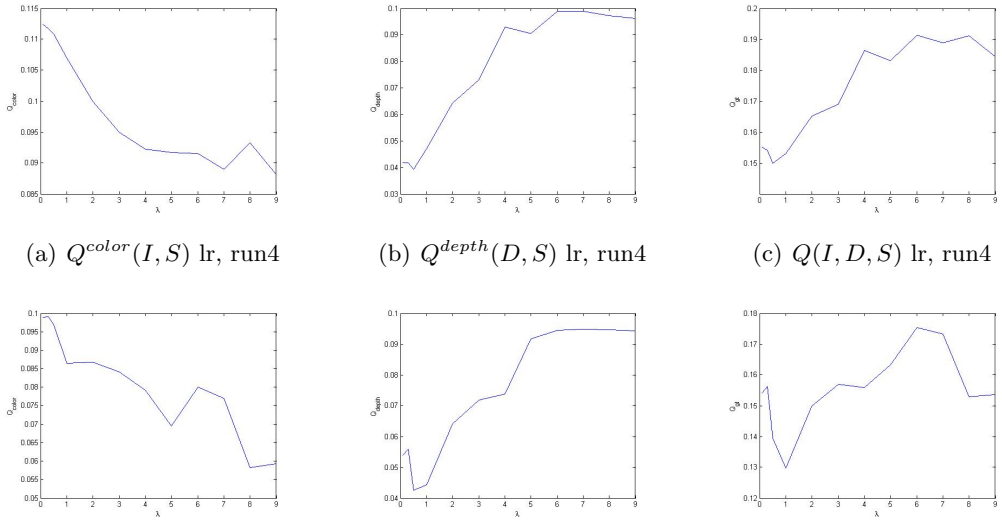
(a) lr-λ=0.1 (b) rl-λ=0.1

(c) lr-λ=2 (d) rl-λ=2

(e) lr-λ=6 (f) rl-λ=6

(g) lr-λ=9 (h) rl-λ=9

Figura 4.41: Esempi segmentazione al run4



(a) $Q^{color}(I,S)$ lr, run4 (b) $Q^{depth}(D,S)$ lr, run4 (c) $Q(I,D,S)$ lr, run4

(d) $Q^{color}(I,S)$ rl, run4 (e) $Q^{depth}(D,S)$ rl, run4 (f) $Q(I,D,S)$ rl, run4

Figura 4.42: grafici delle funzioni $Q^{color}(I,S)$, $Q^{depth}(D,S)$ e $Q(I,D,S)$, run4

Tabella 4.21: Immagine *Bowling2*

Run	RMSE	finestra	pixel inconsistenti
0	2.31872	19	30.5%
1	2.2983	21	30.4%
2	2.17091	23	33%
3	2.16017	23	33.1%
4	2.1078	25	33.5%

4.4 Confronto finale

Si raccolgono in questa sezione gli andamenti dell'errore quadratico medio e delle funzioni $Q(I,D,S)$ per le tre immagini, confrontando i risultati ottenuti nei tre casi i esame.
Si osservi come nell'immagine *Baby2* il dato RMSE migliora nettamente dal run0 al run1 e i pixel inconsistenti diminuiscano, mentre il dato relativo alla segmentazione peggiora per entrambe le immagini; al contrario nell'immagine *Baby1* il dato RMSE dal run3 al run4 peggiora leggermente e aumentano di poco i pixel inconsistenti, ma i dati della segmentazione invece migliorano per entrambe le immagini. In altri casi ancora un miglioramento o un peggioramento RMSE ha portato ad un corrispondente miglioramento o peggioramento, non evidenziando un legame stretto tra le due funzioni, come mostrato in figura 4.43. Si nota, soprattutto nell'immagine *Bowling2*, che l'andamento della funzione

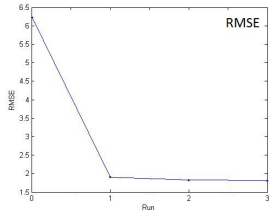
Tabella 4.22: Immagine lr-*Bowling2*

Run	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
0	0.0924	0.0702	0.1634
1	0.0887	0.0632	0.1519
2	0.0918	0.0728	0.1645
3	0.0955	0.0726	0.1682
4	0.0986	0.0728	0.1714

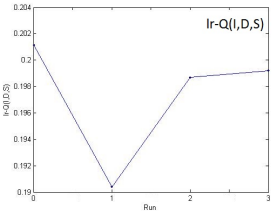
Tabella 4.23: Immagine rl-*Bowling2*

Run	$Q^{color}(I,S)$	$Q^{depth}(D,S)$	$Q(I,D,S)$
0	0.0922	0.0745	0.1667
1	0.0943	0.0748	0.1691
2	0.0909	0.0768	0.1677
3	0.0942	0.0711	0.1653
4	0.0796	0.0802	0.1596

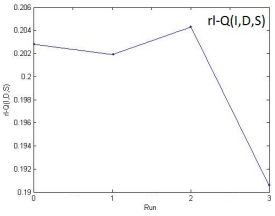
$Q^{color}(I,S)$ potrebbe non essere lo stesso di $Q(I,D,S)$, che valuta l'effettiva qualità della segmentazione. Per le due immagini *Baby1* e *Baby1* però gli andamenti delle due funzioni erano simili e la scelta delle immagini corrispondenti ai λ ottimi, era ottima o molto vicino all'ottimi anche per la funzione $Q(I,D,S)$ e ciò giustifica in parte la scelta di utilizzare quella funzione come parametro di scelta.



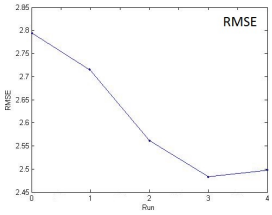
(a) RMSE immagine *Baby2*



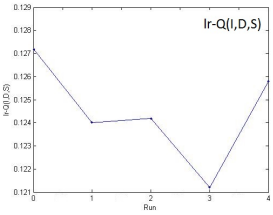
(b) $Q(I, D, S)$ lr, *Baby2*



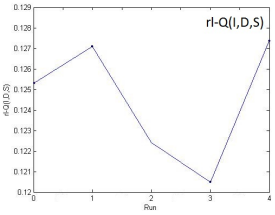
(c) $Q(I, D, S)$ rl, *Baby2*



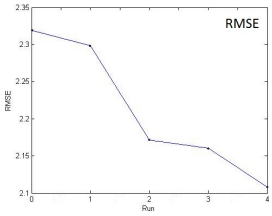
(d) RMSE immagine *Baby1*



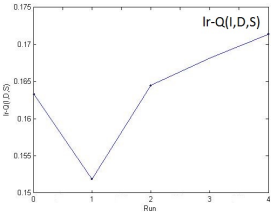
(e) $Q(I, D, S)$ lr, *Baby1*



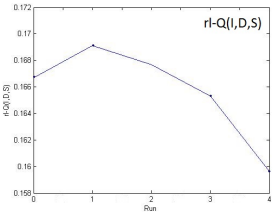
(f) $Q(I, D, S)$ rl, *Baby1*



(g) RMSE immagine *Bow*



(h) $Q(I, D, S)$ lr, *Bow*



(i) $Q(I, D, S)$ rl, *Bow*

Figura 4.43: grafici di RMSE, $Q(I, D, S)$ lr e $Q(I, D, S)$ rl al variare dei run per le immagini considerate

Capitolo 5

Conclusioni

L'obbiettivo di questa tesi è stato migliorare due aspetti connessi della visione computazionale: la segmentazione e la stereopsi computazionale, inserendo informazioni ricavate dalla geometria della scena nell'algoritmo di segmentazione e informazioni ricavate dal colore dell'immagine nell'algoritmo di calcolo delle disparity map.
Si è cercato sempre, ove possibile, come punto fondamentale per iniziare il run successivo, di abbassare il dato relativo all'errore quadratico medio delle disparity map, riuscendo sempre, ad eccezione dell'ultimo run dell'immagine Baby1, ad ottenere dei miglioramenti, sacrificando a volte la nitidezza dei contorni o il numero di pixel consistenti dell'immagine; per cui da questo punto di vista il sistema messo in atto è sicuramente valido, seppur ancora con parti da correggere come l'errore che deriva dall'algoritmo del SegmentSupport nella parte sinistra delle disparity map. D'altra parte però un miglioramento, anche se netto dell'errore quadratico medio, anche a parità di pixel inconsistenti, non ha sempre comportato un conseguente miglioramento dei dati relativi alla segmentazione.
Sembra che non ci sia una correlazione diretta tra miglioramento dell'errore quadratico medio e miglioramento della segmentazione, bisogna però approfondire la ricerca variando parametri ed effettuando ancora più prove di quanto non abbia fatto in questa tesi nel tempo prestabilito, ricercando in particolare parametri comuni nel miglioramento o peggioramento della segmentazione da un run all'altro.
Da migliorare inoltre è la scelta delle due immagini segmentate, corrispondenti ai λ ottimi della funzione considerata, da inserire nel run successivo; in ogni caso però anche con scelte lontane dall'ottimo si è comunque riusciti ad ottenere un qualche migliorameto del dato RMSE cambiando al più un solo parametro (la grandezza della finestra di correlzione), risultato che conferma come la segmentazione può effettivamente aiutare il calcolo delle disparity map.
Uno sviluppo futuro interessante è sicuramente legato all'automazione della scelta dei parametri ideali da inserire per ogni run, lasciando al minimo l'interazione con l'utente

e facendogli risparmiare tempo evitando soprattutto di ricalcolare più volte la disparity map, processo con il tempo di calcolo più lungo, in cerca di quella ottima. Per far ciò però bisognerebbe essere in grado di stimare la disparity map a priori e decidere i parametri migliori su quella stima, cosa non facile.
In conlcusione si può rimanere soddisfatti dell'abbassamento dell'errore quadratico medio all'aumentare dei run, risultato prefisso ad inizio tesi e rimandare ad analisi future lo studio del miglioramento della segmentazione all'aumentare delle iterazioni.

Bibliografia

[1] J. Shi and J. Malik, "Normalized cuts and image segmentation," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 22, no. 8, pp. 888–905, August 2007.

[2] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, "Spectral grouping using the nystrom method," *Pattern Analysis and Machine Intelligencem, IEEE Transactions on*, vol. 26, no. 2, pp. 214–225, February 2004.

[3] A. Fusiello, *Visione Computazionale,Appunti delle lezioni*, 2009.

[4] F. Tombari, S. Mattoccia, and L. Di Stefano, "Segmentation-based adaptive support for accurate stereo correspondence," *Proceedings of IEEE Pacific-Rim Symposium on Image and Video Technology*, pp. 427–438, April 2007.

[5] K. Yoon and I. Kweon, "Adaptive support-weight approach for correspondence search," *Pattern Analysis and Machine Intelligencem, IEEE Transactions on*, vol. 28, no. 4, pp. 650–656, April 2006.

[6] B. Hiebert-Treuer, S. Al Nashashibi, and D. Scharstein, "Stereo datasets with ground truth," 2006. [Online]. Available: http://vision.middlebury.edu/stereo/

[7] C. Dal Mutto, P. Zanuttigh, and G. Cortelazzo, "Fusion of geometry and color information for scene segmentation," *IEEE Journal of selected topics in signal processing*, vol. 6, no. 5, pp. 505–521, September 2012.

[8] M. Tubiana, "Segmentazione congiunta di immagini ed informazione geometrica e suo utilizzo all'inerno di algoritmi di visione stereoscopica," 2010/2011.