



**UNIVERSITÀ DEGLI STUDI DI PADOVA**

---

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

*Corso di Laurea in Ingegneria dell'Informazione*

**MISURE DI CONFIDENZA PER ALGORITMI DI  
STEREO VISION**

**CONFIDENCE MEASURES FOR STEREO VISION ALGORITHMS**

*Laureando*

**Nicola Dal Lago**

*Relatore*

**Prof. Pietro Zanuttigh**

*Correlatore*

**Dott. Giulio Marin**



# Abstract

Scopo di questa tesi è il confronto di varie tecniche di stima della confidenza a partire dai dati acquisiti da una coppia di fotocamere stereo. Vengono quindi utilizzate più metriche fisse, con parametri variabili e combinazioni di esse. Si confrontano poi i risultati con la mappa di disparità *ground truth* disponibile nei dataset citati. Per il calcolo delle disparità viene usato un algoritmo di semi *Semi-global block matching (SGBM)*, opportunamente modificato, presente nelle librerie *OpenCV*. Le misure di confidenza e i risultati sono invece scritti con il linguaggio di programmazione *MATLAB*, il cui codice viene riportato nelle appendici.



# Indice

<b>Abstract</b>	<b>ii</b>
<b>1 Introduzione</b>	<b>2</b>
1.1 Stereopsi . . . . .	2
1.2 Calcolo delle corrispondenze . . . . .	3
<b>2 Misure di confidenza</b>	<b>6</b>
2.1 Implementazione utilizzata . . . . .	6
2.2 Proprietà locali della curva . . . . .	7
2.3 Minimo locale della curva . . . . .	8
2.4 Intera curva . . . . .	8
2.5 Consistenza fra disparità di destra e sinistra . . . . .	9
<b>3 Risultati</b>	<b>12</b>
3.1 Variazione parametri . . . . .	13
3.2 Confronto delle misure . . . . .	15
<b>4 Combinazione di misure</b>	<b>18</b>
4.1 Risultati . . . . .	18
<b>Appendici</b>	<b>19</b>
<b>A Codice MATLAB utilizzato</b>	<b>20</b>
A.1 compute_confidence.m . . . . .	20
A.2 left_right_confidence.m . . . . .	23
A.3 result.m . . . . .	25
<b>Bibliografia</b>	<b>30</b>



# Capitolo 1

## Introduzione

La visione stereo è stata un'area attiva della ricerca per decenni. Negli ultimi anni, gli algoritmi di stereo vision sono maturati a tal punto da essere applicati in un vasto scenario, dalla automazione industriale, al gaming fino alla guida assistita [2].

### 1.1 Stereopsi

*La stereopsi è la capacità percettiva che consente di unire le immagini provenienti dai due occhi, che a causa del loro diverso posizionamento strutturale, presentano uno spostamento laterale. Questa disparità viene sfruttata dal cervello per trarre informazioni sulla profondità e sulla posizione spaziale dell'oggetto mirato. Di conseguenza la stereopsi permette di generare la visione tridimensionale.*<sup>1</sup>

Si possono quindi identificare due problemi: calcolo delle corrispondenze e triangolazione [1].

Il primo consiste nell'accoppiare punti delle due immagini, detti punti coniugati, che sono proiezione dello stesso punto nella scena. Il calcolo delle corrispondenze è un problema possibile in quanto le due immagini differiscono di poco, quindi un punto della scena deve apparire simile nei punti coniugati delle due immagini. Basandosi solo su questo però, sono possibili molti accoppiamenti sbagliati; le due immagini vengono quindi rettificate prima del calcolo delle corrispondenze, in modo che due punti coniugati si trovino sulla stessa retta (detta retta epipolare). Questo si ottiene ruotando le immagini originali attorno ai loro centri ottici finché i piani focali non diventano co-planari (e quindi anche i piani immagine).

Per triangolazione si intende il calcolo della distanza tra un punto della scena e il piano formato dalle due fotocamere. Nel caso di due fotocamere parallele ed allineate ci si può facilmente ricondurre alla figura 1.1.

Fissato come riferimento la fotocamera di sinistra si possono scrivere le equazioni di proiezione prospettica:

---

<sup>1</sup>da <https://it.wikipedia.org/wiki/Stereopsi>

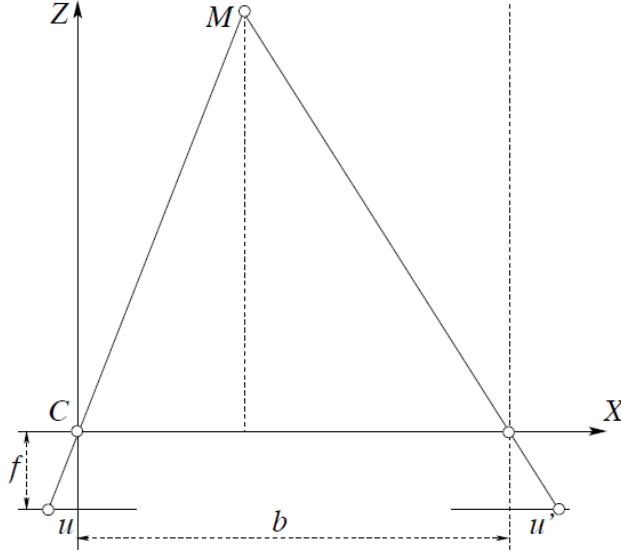


Figura 1.1: Triangolazione stereoscopica.

$$\begin{cases} \frac{f}{z} = \frac{-u}{x} \\ \frac{f}{z} = \frac{-u'}{x - b} \end{cases} \quad (1.1)$$

e risolvendo si ottiene:

$$z = \frac{bf}{u' - u} \quad (1.2)$$

dove  $b$  è la distanza tra le due fotocamere,  $f$  la focale delle fotocamere e  $u' - u$  la distanza fra i due centri ottici.

## 1.2 Calcolo delle corrispondenze

Il calcolo delle corrispondenze o della disparità è il problema principale della stereo vision.

La disparità è la differenza tra due punti coniugati (vettore), immaginando di sovrapporre le due immagini. Il calcolo delle corrispondenze non è altro che il calcolo della disparità per ogni pixel delle due immagini [1]. Si ottiene quindi una mappa di disparità del tipo di figura 1.2.

Gli approcci tipici per il calcolo della disparità sono basati su correlazione [3] o *semi-global matching* (SGM) [4]. In questa tesi viene utilizzato un algoritmo di tipo SGM, questo tipo di algoritmi usa una regione dell'immagine al posto del singolo pixel per identificare i punti coniugati. Ogni punto viene confrontato con tutti i punti nella retta epipolare, fino ad ottenere il punto coniugato e quindi la disparità. La disparità però

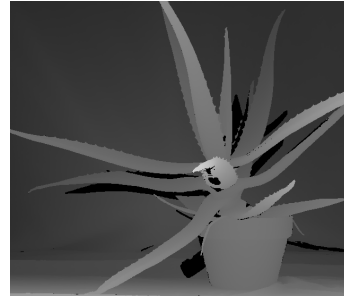




(a) Fotocamera di destra.



(b) Fotocamera di sinistra.



(c) Mappa di disparità.

Figura 1.2: Mappa di disparità con immagine di destra come riferimento, immagine presa da <http://vision.middlebury.edu/stereo/data/>.

non è una funzione esatta, in quanto la misurazione può essere soggetta da rumore (scarsa o eccessiva illuminazione ecc...), oppure l'algoritmo di per se può sbagliare. Viene quindi generata una funzione detta "funzione di costo", la quale ha per ascissa tutti i possibili valori di disparità, e per ordinata la probabilità che quel valore di disparità sia esatto. In realtà, matematicamente parlando, non si ha una probabilità, ed è per questo che si parla di costo.



## Capitolo 2

### Misure di confidenza

Come accennato nella sezione 1.2, ad ogni pixel dell'immagine di riferimento (destra o sinistra), viene assegnata una funzione costo, la quale identifica quanto una determinata disparità sia esatta per quel pixel.

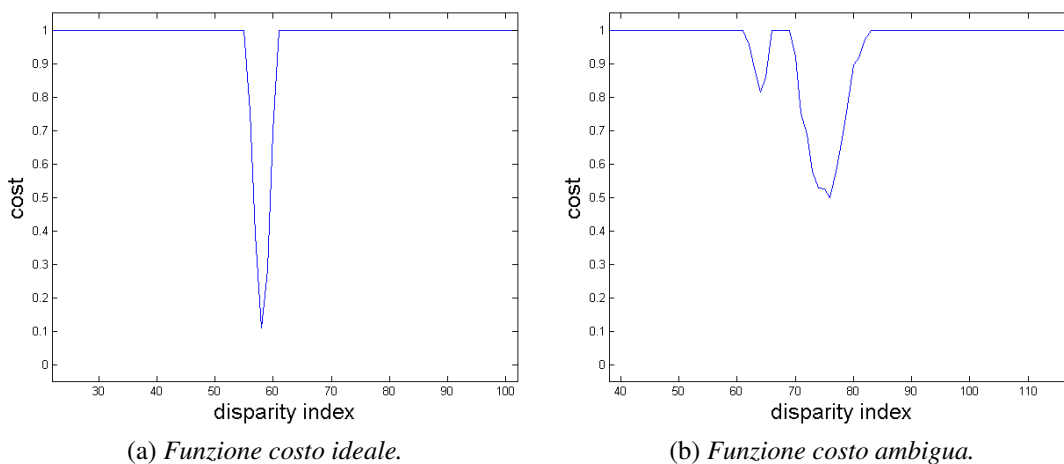


Figura 2.1: Due possibili funzioni costo.

Due possibili funzioni costo sono riportate in figura 2.1. E' chiaro che nel primo caso la disparità è individuata da quell'unico picco; nel secondo caso invece, la funzione è ambigua, in quanto vi sono diversi picchi, e non è quindi più così banale trovare l'esatta disparità. Si rende quindi necessario lo studio di varie tecniche per misurare la confidenza con la quale viene dato un determinato valore.

#### 2.1 Implementazione utilizzata

Per il calcolo della funzione costo di ciascun pixel, è stato usato un algoritmo di tipo *semi-global matching*, presente nelle librerie open source di visione computazio-

nale *OpenCV* [9]. Questo algoritmo, scritto nel linguaggio C++, di per se produrrebbe in output solo la mappa di disparità. L'algoritmo è stato quindi opportunamente modificato per poter estrarre la funzione costo di ciascun punto. Si ha quindi in output anche una matrice di formato *.dat*, delle stesse dimensioni dell'immagine originale, ma che ha per ciascuna cella la funzione costo. Successivamente la matrice viene convertita in formato *.mat*, per rendere il tutto più facile da utilizzare con il linguaggio di programmazione *MATLAB*. Poi, con script *MATLAB*, vengono calcolate le confidenze con diverse metriche. Le immagini utilizzate sono state prese dal dataset Middlebury [7, 8] disponibili su <http://vision.middlebury.edu/stereo/data/>. Prima di proseguire però, si rende necessario dare alcune definizioni:

- $c(d)$ : il valore del costo assegnato ad una ipotetica mappa di disparità  $d$ , per un pixel di coordinate  $(x, y)$ , è denotato con  $c(x, y, d)$  o  $c(d)$ , se le coordinate del pixel non sono ambigue;
- $c_1$ : il minimo valore del costo di un pixel è definito con  $c_1$  e il corrispondente valore di disparità da  $d_1$ ;  $c_1 = c(d_1) = \min\{c(d)\}$ ;
- $c_2$ : con  $c_2$  viene denotato il secondo valore più piccolo in  $d_2$ , mentre con  $c_{2m}$  si denota il secondo minimo locale più piccolo.

Seguono quindi tutte le varie tecniche utilizzate, raggruppate secondo l'aspetto del costo che considerano [6].

## 2.2 Proprietà locali della curva

In questo genere di metriche, si sfrutta il fatto che la forma della curva di costo intorno al minimo (la nitidezza o la planarità) è indice di certezza nella partita.

**Curvature (CUR)** E' largamente utilizzata nella letteratura [6], ed è definita come

$$C_{CUR} = \frac{-2c(d_1) + c(d_1 - 1) + c(d_1 + 1)}{2} \quad (2.1)$$

se  $d_1 - 1$  o  $d_1 + 1$  sono fuori dal range di disparità, il punto minimo  $c(d_1)$  viene usato due volte. Il tutto viene diviso per 2 per garantire valori compresi tra zero e uno.

**Local Curve (LC)** Molto simile alla misura *CUR*, la *Local Curve* è descritta da

$$C_{LC} = \frac{\max\{c(d_1 - 1), c(d_1 + 1)\} - c_1}{\gamma} \quad (2.2)$$

il parametro  $\gamma$  verrà poi scelto tale da garantire una distribuzione del costo più uniforme possibile tra zero e uno.

## 2.3 Minimo locale della curva

Si basa sul concetto che la presenza di altri candidati è un'indicazione di incertezza, mentre la loro assenza di certezza.

**Peak Ratio Naive (PKRN)** A differenza del *Peak Ratio (PKR)*, che calcola il costo con la formula 2.3

$$C_{PKR} = \frac{c_{2m}}{c_1} \quad (2.3)$$

il *PKRN* non richiede che il numeratore sia un minimo locale. Inoltre la formula è leggermente diversa da quella proposta in letteratura [2]

$$C_{PKRN} = \frac{c_2 + \epsilon}{c_1 + \epsilon} - 1 \quad (2.4)$$

*PKRN* può essere visto come una combinazione del *PKR* e *CUR*, che assegna bassa confidenza per le corrispondenze con minimi piatti o concorrenti forti. Anche se le modifiche al *PKRN* violano leggermente la metrica originale, ha i seguenti vantaggi rispetto alla controparte originale:

- le rare singolarità in cui il denominatore sia nullo non sono più presenti;
- piccole variazioni nei costi dovuti al rumore ai livelli bassi del costo, non hanno un forte impatto nella metrica;
- potendo scegliere il valore di  $\epsilon$ , il range dei possibili valori può essere adattato, al fine di omogeneizzare la misura tra zero e uno.

**Maximum Margin (MMN)** il margine tra  $c_1$  e  $c_2$  è anch'esso indicazione di confidenza. *MMN* è definito dalla formula

$$C_{MMN} = c_2 - c_1 \quad (2.5)$$

**Nonlinear Margin (NLM)** è definito come

$$C_{NLM} = e^{\frac{c_2 - c_1}{2\sigma_{NLM}^2}} - 1 \quad (2.6)$$

le variazioni del parametro  $\sigma_{NLM}$  verranno poi discusse nella sezione 3.1.

## 2.4 Intera curva

Questi metodi convertono la funzione costo in una distribuzione di probabilità sulla disparità.

**Maximum Likelihood Metric (MLM)** Insieme a *PKRN* è una delle metriche più promettenti. Entrambe hanno ottenuto risultati sopra la media sia su immagini indoor che outdoor [6]. La *Maximum Likelihood Metric* è definita come

$$C_{MLM} = \frac{e^{-\frac{c_1}{2\sigma_{MLM}^2}}}{\sum_d e^{-\frac{c(d)}{2\sigma_{MLM}^2}}} \quad (2.7)$$

In questo caso  $\sigma_{MLM}$  rappresenta l'incertezza della disparità, e anche per questo caso la sua variazione verrà discussa nella sezione 3.1.

**Attainable Maximun Likelihood (AML)** è una variante di *MLM* che modella il costo per un particolare pixel usando una distribuzione Gaussiana centrata nel minimo valore di costo che è stato finora calcolato per quel pixel ( $c_1$  nella nostra notazione).

$$C_{AML} = \frac{e^{-\frac{(c_1 - c_1)^2}{2\sigma_{AML}^2}}}{\sum_d e^{-\frac{(c(d) - c_1)^2}{2\sigma_{AML}^2}}} = \frac{1}{\sum_d e^{-\frac{(c(d) - c_1)^2}{2\sigma_{AML}^2}}} \quad (2.8)$$

**Winner Margin Naive (WMNN)** esiste il *Winner Margin*, che richiede che al numeratore vi sia un minimo locale, ma come per il *PKRN* definiamo la versione naive come

$$C_{WMNN} = \frac{c_2 - c_1}{\sum_d c(d)} \quad (2.9)$$

## 2.5 Consistenza fra disparità di destra e sinistra

Questa tipologia di misure consiste nel fatto che, idealmente, un punto nella mappa di disparità destra dovrebbe essere lo stesso nella mappa di disparità sinistra. Fin ora abbiamo denotato la disparità con  $c(x, y, d)$ , ma per chiarezza chiameremo  $c_R(x_R, y, d_R)$  la disparità ottenuta tenendo come riferimento l'immagine di destra. Questo genere di misure risulterà più complicata delle precedenti, in quanto prima di calcolare la funzione costo, si rende necessario calcolare prima la disparità sia per l'immagine di destra che quella di sinistra.

**Left Right Consistency (LRC)** *Left Right Consistency (LRC)* viene calcolato prendendo il valore disparità calcolato in una immagine, e proiettandolo nell'altra immagine. Se la differenza nei valori è inferiore a una determinata soglia, allora il pixel è occluso. Questo procedimento viene poi ripetuto anche al contrario, cioè proiettando la seconda immagine nella prima. Inoltre il costo di *LRC* viene calcolato come segue:

$$C_{LRC}(x, y) = |d_1 - D_R(x - d_1, y)| \quad (2.10)$$

con  $d_1 = \arg \min_d \{c(x, y, d)\}$  e  $D_R(x - d_1, y) = \arg \min_{d_R} \{c_R(x - d_1, y, d_R)\}$ .

**Left Right Difference (LRD)** *Left Right Difference (LRD)* è una misura proposta per la prima volta in [5]. Essa considera sia i due minimi della disparità di sinistra, ma anche il minimo di quella di destra. E' definita come

$$C_{LRD}(x, y) = \frac{c_2 - c_1}{|c_1 - \min\{c_R(x - d_1, y, d_R)\}|} \quad (2.11)$$

L'idea è che finestre di pixel corrispondenti dovrebbero risultare in valore di costo molto simili, e quindi piccoli valori al denominatore. Questa misura dovrebbe salvaguardare da due possibili errori:

- se il margine  $c_2 - c_1$  è grande, ma il pixel non ha una corrispondenza errata, il denominatore sarà grande, e la confidenza bassa;
- se il margine è piccolo, la misura è possibile che sia ambigua, in questo caso se il denominatore è piccolo denota che è stata creata con successo una corrispondenza fra due pixel





# Capitolo 3

## Risultati

Tutte le misure effettuate vengono confrontate con una mappa di disparità chiamata *ground truth*. Questa mappa è sempre disponibile nei dataset utilizzati [7, 8], per tutte le immagini e per tutte le risoluzioni. La disparità *ground truth* viene calcolata utilizzando sempre una coppia di videocamere, ma aggiungendo anche uno o più proiettori che illuminano la scena con dei pattern specifici. Ogni telecamera utilizza quindi i pattern per determinare un codice univoco per ciascun pixel. Trovare quindi le corrispondenze si traduce banalmente nel verificare quali pixel delle due immagini hanno lo stesso codice [10].



Figura 3.1: Configurazione del proiettore e della coppia di videocamere per il calcolo della disparità *ground truth*; si può notare come il proiettore illumina la scena con dei pattern diversi per facilitare il compito delle videocamere.

Prima di continuare definiamo l'errore di disparità come

$$e_d = |d_{GT} - d| \quad (3.1)$$

Dove  $d_{GT}$  è la disparità *ground truth* e  $d$  sono i valori di disparità ottenuti con l'algoritmo *SGM*.

Per valutare la capacità delle misure di confidenza di predire dove una disparità è corretta, è stato creato uno script *MATLAB* che ordina prima l'errore di disparità in ordine crescente, e poi ordina l'errore per ogni misura in ordine decrescente rispetto la sua confidenza. Successivamente si calcola l'errore globale selezionando prima il 5%, in due modi:

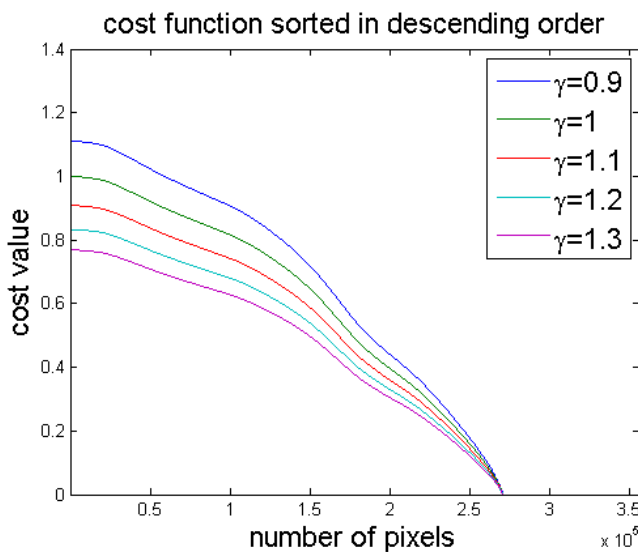
1. nel primo metodo si misura la percentuale di pixel che ha un errore di disparità maggiore di uno, questo si fa perché ovviamente se l'errore è minore di uno, non vi è alcun pixel di differenza fra le due;
2. nel secondo si misura semplicemente l'errore medio.

Si ripete poi con il 10% e così via sino al 100%. Prima di confrontare le varie metriche si è però trovato il parametro ottimale per quelle con parametri variabili.

### 3.1 Variazione parametri

In questa sezione, prima di confrontare le misure, verificheremo quali parametri sono più adatti per quelle che dipendono da tali. Consideriamo quindi la *Local Curve*, *Peak Ratio Naive*, *Nonlinear Margin* e *Maximum Likelihood metric*.

**Variazione parametro  $\gamma$  per LC** Ora indagheremo quali effetti produce la variazione del parametro  $\gamma$ . Come prima cosa si è osservato per quale valore si ha una



distribuzione più uniforme tra 0 e 1. Come si può vedere dalla figura 3.2, si ha una migliore distribuzione per  $\gamma = 1$ . Per quanto riguarda l'errore medio invece, non si ottiene nessun cambiamento significativo sia con valori di  $\gamma$  molto alti, sia con valori molto bassi. Come valore finale si è quindi deciso per un  $\gamma$  pari a 1; facendo ciò non sarà anche necessario riscalarla la funzione costo, in quanto è già compresa tra zero e uno.

Figura 3.2: Distribuzioni diverse con valori di  $\gamma$  diversi.

**Variazione parametro  $\epsilon$  per *PKRN*** Come prima, verifichiamo come si comporta la distribuzione della funzione al cambiare del parametro  $\epsilon$ . Si nota che per valori intorno a uno, la distribuzione non supera il massimo di uno, con valori minori di uno invece

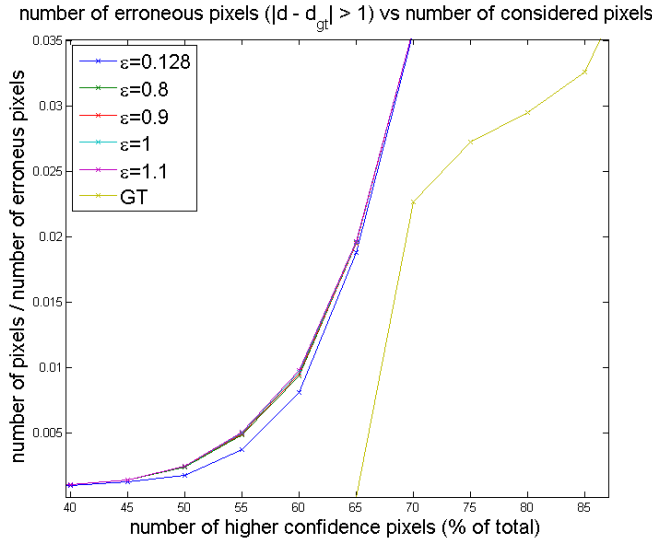


Figura 3.3: Dettaglio della curva dell'errore della metrica *PKRN*, notare che con  $\epsilon$  più piccolo si hanno dei lievi ma visibili miglioramenti; in giallo il valore di *ground truth*

lo si supera. Per quanto riguardagli errori rispetto la *ground truth*, si notano dei lievi miglioramenti per valori di  $\epsilon$  molto piccoli, intorno a un decimo; lo si può notare in figura 3.3. Si è notato inoltre che dopo il valore di 0,128 non si hanno cambiamenti. Si sceglie quindi  $\epsilon = 0.128$ . Notare che in questo modo è necessario riscalare il valore della metrica, in quanto il suo massimo è abbondantemente sopra l'uno; per farlo è sufficiente dividere ogni singolo costo per il costo massimo.

### Variazione parametro $\sigma$ per *NLM*

Ripetiamo ancora una volta il procedimento, i risultati sono pressoché simili al *LC*, cioè grosse variazioni di distribuzione e trascurabili variazioni di errore per differenti valori di  $\sigma$ . Dopo qualche tentativo con diversi parametri, si arriva alla conclusione che i risultati migliori si ottengono con  $\sigma = 0,85$ . Si fa notare però, che con valori di  $\sigma$  molto piccoli, l'errore può peggiorare di molto, mentre con valori molto grandi l'errore rimane pressoché inalterato.

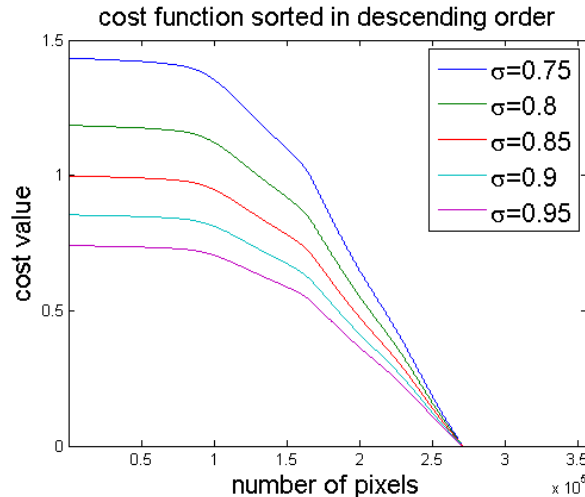


Figura 3.4: Diversi valori di distribuzione per diversi valori di sigma

### Variazione parametro $\sigma$ per *MLM*

A differenza delle altre misure, per essere utilizzata al meglio la si deve per forza riscalare. Infatti i valori che permetterebbero un costo compreso tra zero e uno, sarebbero completamente sballati per quanto riguarda gli errori. Per garantire quindi una certa uniformità e un errore non esagerato, si è quindi scelto un valore di  $\sigma$  pari a 0,3.

**Variazione parametro  $\sigma$  per *AML*** Il suo comportamento è molto simile a *MLM*, i migliori risultati si ottengono con  $\sigma$  del valore di 0,4.

## 3.2 Confronto delle misure

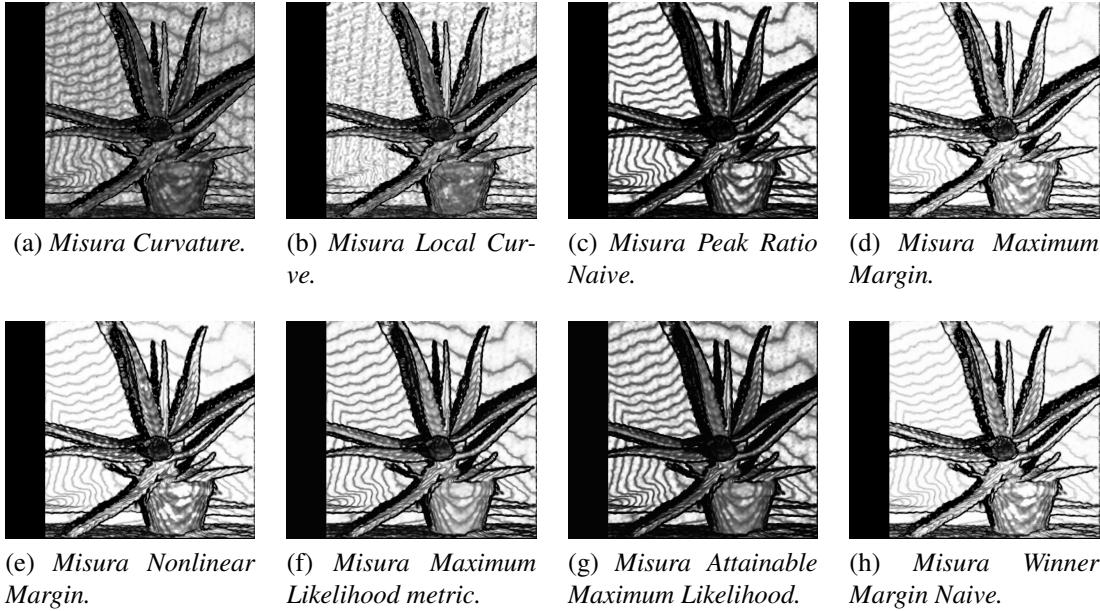
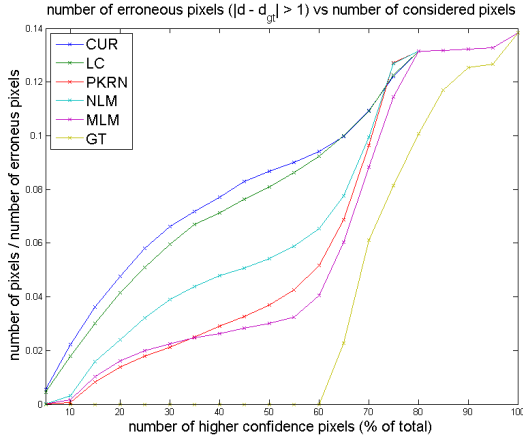


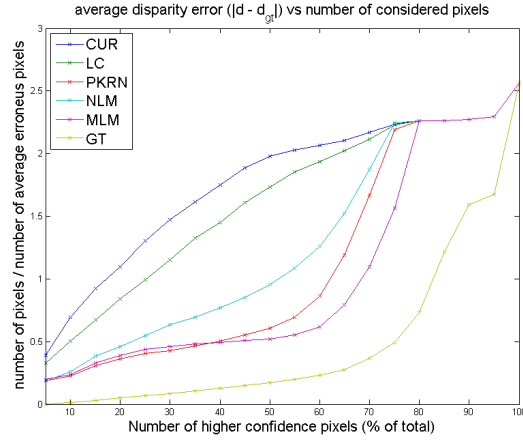
Figura 3.5: Diverse misure di confidenza.

Cominciamo dal dire cosa aspettarci dai due confronti descritti prima. Per il primo grafico ci aspettiamo di trovare una curva che resta a zero per una determinata percentuale di confidenze, e che poi tende a crescere in maniera pressoché lineare per le successive. Per il secondo confronto ci aspetteremo che l'errore di riferimento sia monotono crescente, mentre le misure possono assumere andamenti più casuali, al patto che abbiano errore sempre maggiore di quello di riferimento. Ovviamente la misura migliore sarà quella con la curva il più possibile vicino all'errore di *ground truth*.

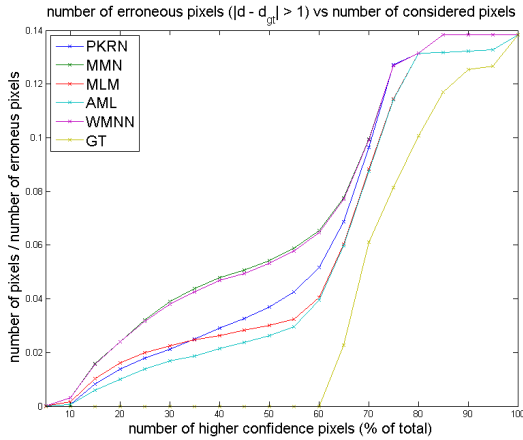
Passando quindi al confronto, si nota che le diverse metriche si comportano in maniera differente a seconda dell'immagine. Nel complesso però, le due che sembrano avere risultati migliori sono la *MLM* e *AML*. Questo è chiaramente visibile nei grafici riportati in figura 3.6.



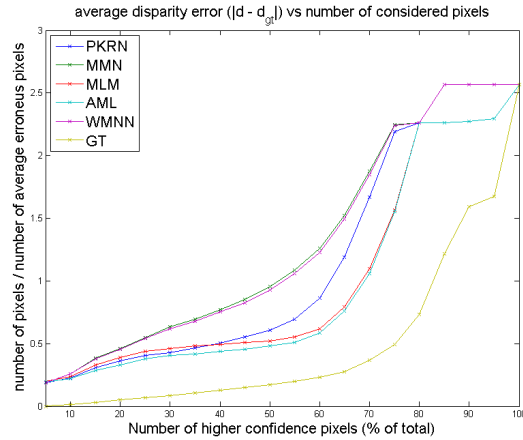
(a) Numero di pixel con errore di disparità maggiore di uno.



(b) Errore di disparità medio.



(c) Numero di pixel con errore di disparità maggiore di uno.



(d) Errore di disparità medio.

Figura 3.6: Grafici dei risultati utilizzando l'immagine *Bowling2* dal dataset *Middlebury*

Questo è il risultato dove differenzia di più tutte le misure; questo vuol dire che in certe immagini le misure si equivalgono molto, mentre in altre sono molto diverse. Non succede mai però che *MLM* e *AML* siano nettamente peggiori, succede invece che a volte una delle due sia meglio di un'altra e viceversa. Altro caso, è quando i grafici si incrociano, questo vuol dire che per una certa percentuale di confidenze è meglio una, mentre poi è meglio un'altra. Questo succede ad esempio nella figura 3.7,

dove la metrica *NonLinear Margin* è la migliore di tutte per quasi il 40% dei pixel considerati, venendo poi superata, ancora una volta, da *MLM* e *PKRN*. Come previsto, anche la metrica *PKRN* ottiene buoni risultati, superando tutte soprattutto nella prima percentuale di pixel considerati. Per quanto riguarda le misure del tipo consistenza fra disparità di destra e sinistra, la migliore delle due sembra essere *LRD*. Come verificato in [5], questa misura è migliore di *LRC* in praticamente tutte le immagini considerate e per quasi tutte le percentuali di pixel. Entrambe comunque si comportano molto bene anche rispetto le misure descritte nel capitolo 2; infatti la curva che indica il numero di pixel con errore di disparità maggiore di uno, rimane piatta fino a circa il 50% dei pixel considerati (il *ground truth* arriva al 65%) mentre anche nel migliore dei casi, le altre misure non riescono a superare il 30%.

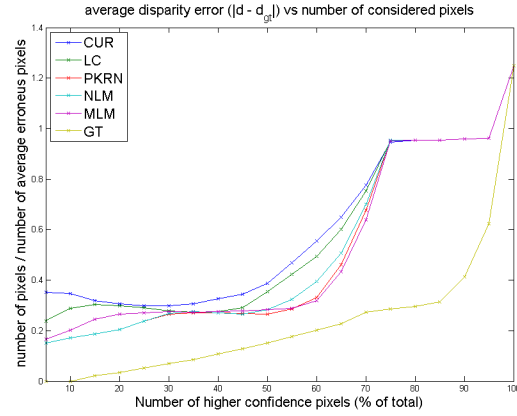


Figura 3.7: Errore di disparità medio nella figura *Aloe*.

# Capitolo 4

## Combinazione di misure

Per combinazione di misure, si intende il combinarne più insieme al fine di migliorarne i risultati. Considerando le misure di confidenza come probabilità, la cosa più semplice da fare sarebbe quella di dire che, essendo indipendenti, la loro combinazione non è altro che il loro prodotto. In realtà però, non è del tutto corretto definirle indipendenti, in quanto tutte dipendono dalla stessa “matrice dei costi”, calcolata con il medesimo algoritmo (in questo caso *SGM*). In questa tesi però non ne terremo conto, calcolando la combinazione di misure solo come semplice moltiplicazione di alcune di esse. Come combinazione sono state usate:

- combinazione di *AML* e *MLM*  $C_{comb1} = C_{AML} \cdot C_{MLM}$ ;
- combinazione delle tre migliori  $C_{comb2} = C_{AML} \cdot C_{MLM} \cdot C_{PKRN}$ ;
- combinazione fra *NLM* e *MLM*, infatti la metrica *NLM* è risultata competitiva in certi dataset per circa il primo 40% dei pixel, per poi essere superata da *MLM*  $C_{comb3} = C_{NLM} \cdot C_{MLM}$ .

Ovviamente prima di moltiplicarle, bisogna verificare che la loro distribuzione sia compresa tra zero e uno.

### 4.1 Risultati

La migliore combinazione risulta essere la prima, poi la seconda e infine la terza. La prima combinazione riesce sempre ad essere migliore di *MLM*, mentre solo a volte migliore di *AML*. La seconda migliora, e non di poco, la metrica *PKRN*, mentre la terza è peggiore delle singole misure per i primi 50% di pixel, ma in quelli successivi risulta vincente. Nel complesso quindi, questo genere di combinazioni, può riuscire a migliorare i risultati di qualcosa, ma un uso sbagliato può riuscire a peggiorarli di molto.

Per riuscire a migliorare questi risultati, si potrebbe pensare a combinazioni più complicate

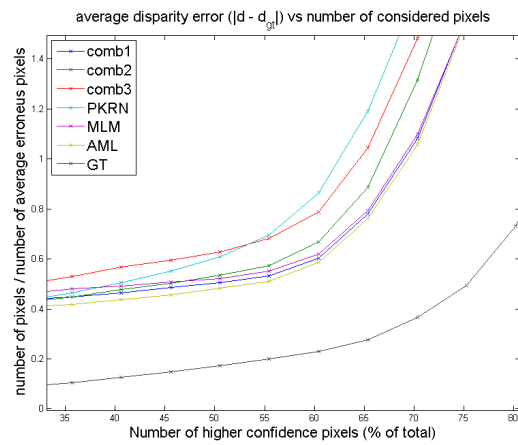


Figura 4.1: Dettaglio dell'errore medio per la combinazione di diverse misure, in questo caso la migliore resta comunque *AML*, per poi essere raggiunta dalla sua combinazione con *MLM*.



# Appendice A

## Codice MATLAB utilizzato

In questa appendice viene riportato solo il codice *MATLAB* che calcola le mappe di confidenza con le varie tecniche descritte nel paragrafo 2 e nel paragrafo 4; si riporta inoltre il codice utilizzato per analizzare i risultati ottenuti.

### A.1 compute\_confidence.m

Questo script computa la mappa di disparità per *CUR*, *LC*, *PKRN*, *MMN*, *NLM*, *MLM*, *AML* e *WMNN*. Calcola inoltre la combinazione delle confidenze.

```
1 % This script compute confidence with different metrics
2
3 % @author: Giulio Marin (giulio.marin@me.com), Nicola Dal Lago
4 % @date: 25/08/2014
5 % @version: 1.0
6
7
8 %% clear workspace
9 clear; close all; clc;
10
11 dataset = 'Bowling2'; %it must be 'Aloe', 'Bowling2', 'Flowerpots'
12
13 rows = 555;
14 if(strcmp(dataset, 'Aloe'))
15     cols = 641;
16 elseif(strcmp(dataset, 'Bowling2'))
17     cols = 665;
18 elseif(strcmp(dataset, 'Flowerpots'))
19     cols = 656;
20 end
21
22 load(['.././../C/data/Images/Middlebury/' dataset '/SGM/cost.mat'])
23
24 % Smallest costs and indexes
25 [c1, I1] = min(C,[],3);
26
27 % Second smallest costs and indexes
28 c2 = 32767*ones(size(c1));
29 I2 = ones(size(c1));
30 for d = 1:diff(disparity)
31     index = (squeeze(C(:, :, d)) >= c1);
32     index = index & (squeeze(C(:, :, d)) < c2);
```

```

33     index = index & abs(I1 - d) > 1;
34     if any(index(:))
35         tmp = squeeze(C(:, :, d));
36         c2(index) = tmp(index);
37         I2(index) = d;
38     end
39 end
40
41
42 %% Curvature (CUR) and Local Curve (LC)
43 Im = I1 - 1; Im(Im == 0) = 1;
44 Ip = I1 + 1; Ip(Ip == 81) = 80;
45
46 Cm = zeros(size(c1));
47 Cp = zeros(size(c1));
48
49 for r=1 : rows
50     for c=1 : cols
51         Cm(r,c) = C(r,c,Im(r,c));
52         Cp(r,c) = C(r,c,Ip(r,c));
53     end
54 end
55
56 % Curvature (CUR)
57 C_CUR = (-2*c1 + Cm + Cp) / 2;
58 figure; imshow(C_CUR); title('Curvature')
59
60 % Local Curve (LC)
61 gamma = 1;
62
63 C_LC = (max(Cm,Cp)-c1) ./ gamma;
64 figure; imshow(C_LC); title('Local Curve')
65
66
67 %% Peak Ratio Naive (PKRN)
68 epsilon = 0.128;
69
70 C_PKRN = (c2+epsilon) ./ (c1+epsilon) - 1;
71 C_PKRN = C_PKRN ./ max(C_PKRN(:));
72 figure; imshow(C_PKRN); title('Peak Ratio Naive')
73
74
75 %% Maximum Margin (MMN)
76 C_MMN = c2 - c1;
77 figure; imshow(C_MMN); title('Maximum Margin')
78
79
80 %% Nonlinear Margin (NLM)
81 sigma_NLM = 0.85;
82
83 C_NLM = exp((c2 - c1) ./ (2*sigma_NLM^2)) - 1;
84 figure; imshow(C_NLM); title('Nonlinear Margin')
85
86
87 %% Maximum Likelihood metric (MLM)
88 sigma_MLM = 0.3;
89
90 sum = 0;
91 for d = 1:diff(disparity)
92     sum = sum + exp(-squeeze(C(:, :, d)) ./ (2*sigma_MLM^2));
93 end
94
95 C_MLM = exp(-c1 ./ (2*sigma_MLM^2)) ./ sum;
96 C_MLM = C_MLM ./ max(C_MLM(:));
97 figure; imshow(C_MLM); title('Maximum Likelihood Metric')

```

```

98
99
100 %% Attainable Maximun Likelihood (AML)
101 sigma_AML = 0.4;
102
103 sum = 0;
104 for d = 1 : diff(disparity)
105     sum = sum + exp(-(squeeze(C(:, :, d)) - c1) ./ (2 * sigma_AML ^ 2));
106 end
107
108 C_AML = 1 ./ sum;
109 C_AML = C_AML ./ max(C_AML(:));
110 figure; imshow(C_AML); title('Attainable Maximun Likelihood')
111
112
113 %% Winner Margin Naive (WMNN)
114 sum = 0;
115 for d = 1 : diff(disparity)
116     sum = sum + squeeze(C(:, :, d));
117 end
118
119 C_WMNN = (c2 - c1) ./ sum;
120 C_WMNN = C_WMNN ./ max(C_WMNN(:));
121 figure; imshow(C_WMNN); title('Winner Margin Naive')
122
123
124 %% Save confidence
125
126 save(['./save_confidence/' dataset '/confidences.mat'], 'C_CUR', 'C_LC', 'C_PKRN', '
    C_MMN', 'C_NLM', 'C_MLM', 'C_AML', 'C_WMNN');
127
128 imwrite(C_CUR, ['./save_confidence/' dataset '/CUR.png'], 'png');
129 imwrite(C_LC, ['./save_confidence/' dataset '/LC.png'], 'png');
130 imwrite(C_PKRN, ['./save_confidence/' dataset '/PKRN.png'], 'png');
131 imwrite(C_NLM, ['./save_confidence/' dataset '/MMN.png'], 'png');
132 imwrite(C_NLM, ['./save_confidence/' dataset '/NLM.png'], 'png');
133 imwrite(C_MLM, ['./save_confidence/' dataset '/MLM.png'], 'png');
134 imwrite(C_NLM, ['./save_confidence/' dataset '/AML.png'], 'png');
135 imwrite(C_NLM, ['./save_confidence/' dataset '/wmnn.png'], 'png');
136
137
138 %% Combination one
139 C_AML = C_AML ./ max(C_AML(:));
140 C_MLM = C_MLM ./ max(C_MLM(:));
141
142 C_comb1 = C_AML .* C_MLM;
143 figure; imshow(C_comb1); title('Combination C_{AML} \cdot C_{MLM}')
144
145
146 %% Combination two
147 C_PKRN = C_PKRN ./ max(C_PKRN(:));
148
149 C_comb2 = C_AML .* C_MLM .* C_PKRN;
150 figure; imshow(C_comb2); title('Combination C_{AML} \cdot C_{MLM} \cdot C_{PKRN}')
151
152
153 %% Combination three
154 C_NLM = C_NLM ./ max(C_NLM(:));
155
156 C_comb3 = C_NLM .* C_MLM;
157 figure; imshow(C_comb3); title('Combination C_{NLM} \cdot C_{MLM}')
158
159
160 %% save combination
161 save(['./save_confidence/' dataset '/confidences_combination.mat'], 'C_comb1', '

```

```

162         C_comb2', 'C_comb3');
163 imwrite(C_comb1, ['./save_confidence/' dataset '/comb1.png'] , 'png');
164 imwrite(C_comb2, ['./save_confidence/' dataset '/comb2.png'] , 'png');
165 imwrite(C_comb3, ['./save_confidence/' dataset '/comb3.png'] , 'png');

```

## A.2 left\_right\_confidence.m

```

1 % This script compute confidence with left and right cost functions
2
3 % @author: Nicola Dal Lago
4 % @date: 15/09/2014
5 % @version: 1.0
6
7
8 %% clear workspace
9 clear; close all; clc;
10
11 dataset = 'Aloe';      %it must be 'Aloe', 'Bowling2', 'Flowerpots'
12
13 path_SGM_disparity = ['./../../C/data/Images/Middlebury/' dataset '/SGM/
    disparity_sgm.dat'];
14 path_SGM_occluded_pixels = ['./../../C/data/Images/Middlebury/' dataset '/SGM/
    disparity_sgm.png'];
15 path_SGM_occluded_pixels_right = ['./../../C/data/Images/Middlebury/' dataset '/SGM/
    disparity_sgm.png_right.png'];
16
17 rows = 555;
18 if(strcmp(dataset, 'Aloe'))
19     cols = 641;
20 elseif(strcmp(dataset, 'Bowling2'))
21     cols = 665;
22 elseif(strcmp(dataset, 'Flowerpots'))
23     cols = 656;
24 end
25
26 load(['./../../C/data/Images/Middlebury/' dataset '/SGM/cost.mat'])
27 load(['./../../C/data/Images/Middlebury/' dataset '/SGM/cost_right.mat'])
28
29 % load occluded pixels' maps
30 tmp = double(imread(path_SGM_occluded_pixels));
31 SGM_occluded_matrix = tmp(:,:,3); % r,g,b      255 means pixel occluded
32 tmp = double(imread(path_SGM_occluded_pixels_right));
33 SGM_occluded_matrix_right = tmp(:, :, 3); %r,g,b
34
35 % Smallest costs and indexes
36 [c1, I1] = min(C,[],3);
37 [c1_right, I1_right] = min(C_right, [], 3);
38
39 % Second smallest costs and indexes
40 c2 = 32767*ones(size(c1));
41 I2 = ones(size(c1));
42 for d = 1:diff(disparity)
43     index = (squeeze(C(:, :, d)) >= c1);
44     index = index & (squeeze(C(:, :, d)) < c2);
45     index = index & abs(I1 - d) > 1;
46     if any(index(:))
47         tmp = squeeze(C(:, :, d));
48         c2(index) = tmp(index);
49         I2(index) = d;
50     end

```

```

51 end
52
53 % load SGM disparity
54 id = fopen(path_SGM_disparity);
55 D_SGM_line = fread(id, rows * cols, 'float');
56 fclose(id);
57
58 D_SGM = zeros(rows, cols);
59 for x = 1 : 1 : rows
60     for y = 1 : 1 : cols
61         current_position = ((x - 1) * cols) + y;
62         D_SGM(x, y) = D_SGM_line(current_position);
63     end
64 end
65
66 %figure; imshow(D_SGM ./ max(D_SGM(:))); title('SGM')
67 %figure; imshow(SGM_occluded_matrix ./ max(SGM_occluded_matrix(:))); title('occluded
    matrix')
68
69 %% Left Right Consistency
70 C_LRC = ones(rows, cols);
71
72 for x = 1 : 1 : rows
73     for y = 1 : 1 : cols
74         if (SGM_occluded_matrix(x, y) == 255) || (SGM_occluded_matrix_right(x, y) ==
            255)
75             % C_LRC(x, y) = 1; % occluded pixel
76             else
77
78                 right_position = round(x - D_SGM(x, y));
79                 if right_position < 1
80                     right_position = 1;
81                 end
82                 C_LRC(x, y) = abs(c1(x, y) - c1_right(right_position, y));
83             end
84         end
85     end
86
87 C_LRC = 1 - C_LRC;
88 figure; imshow(C_LRC); title('Left Right Consistency')
89
90
91 %% left Right Difference
92 C_LRD = zeros(rows, cols);
93
94 for x = 1 : 1 : rows
95     for y = 1 : 1 : cols
96         if (SGM_occluded_matrix(x, y) == 255) || (SGM_occluded_matrix_right(x, y) ==
            255)
97             C_LRD(x, y) = 0; % occluded pixel
98             else
99
100                 right_position = round(x - c1(x, y));
101                 if right_position < 1
102                     right_position = 1;
103                 end
104                 C_LRD(x, y) = (c2(x, y) - c1(x, y)) / abs(c1(x, y) - c1_right(
                    right_position, y));
105             end
106         end
107     end
108 end
109
110 figure; imshow(C_LRD); title('Left Right Difference')
111

```

```

112
113 %% save
114 save(['../save_confidence/' dataset '/confidences_left_right.mat'], 'C_LRC', 'C_LRD');
115 imwrite(C_LRC, ['../save_confidence/' dataset '/LRC.png'], 'png');
116 imwrite(C_LRD, ['../save_confidence/' dataset '/LRD.png'], 'png');

```

## A.3 result.m

```

1 % This script compare 5 different cost function with ground truth
2 % disparity. Also excludes from the comparison occluded pixels.
3
4 % @author: Nicola Dal Lago
5 % @date: 09/09/2014
6 % @version: 2.0
7
8
9 %% clear workspace
10 clear; close all; clc;
11
12 % parameters
13 dataset = 'Bowling2'; % it must be 'Aloe', 'Bowling2', 'Flowerpots'
14
15 path_confidence = ['../stereo/confidence/save_confidence/' dataset '/confidences.mat'
16 ];
17 path_ground_truth = ['../../../../C/data/Images/Middlebury/' dataset '/displ.png'];
18 path_SGM_disparity = ['../../../../C/data/Images/Middlebury/' dataset '/SGM/disp_sgm.
19 dat'];
20 path_SGM_occluded_pixels = ['../../../../C/data/Images/Middlebury/' dataset '/SGM/
21 disparity_sgm.png'];
22 path_confidence_combination = ['../stereo/confidence/save_confidence/' dataset '/
23 confidences_combination.mat'];
24
25 % different images have different size
26 rows = 555;
27 if(strcmp(dataset, 'Aloe'))
28     cols = 641;
29 elseif(strcmp(dataset, 'Bowling2'))
30     cols = 665;
31 elseif(strcmp(dataset, 'Flowerpots'))
32     cols = 656;
33 end
34
35 number_of_costs = 6;
36 cost_functions = {'comb1' 'comb2' 'comb3' 'PKRN' 'MLM' 'AML'};
37
38 % load confidence
39 cost = zeros(rows, cols, number_of_costs);
40 for i = 1 : 1 : number_of_costs - 3
41     cost(:, :, i) = cell2mat(struct2cell(load(path_confidence_combination, ['C_'
42 cell2mat(cost_functions(i))]))));
43 end
44
45 for i = number_of_costs - 2 : 1 : number_of_costs
46     cost(:, :, i) = cell2mat(struct2cell(load(path_confidence, ['C_' cell2mat(
47 cost_functions(i))]))));
48 end
49
50 % load ground truth disparity and SGM disparity
51 D_GT = double(imread(path_ground_truth));
52 D_GT = D_GT ./ 2; % cause the half size of the image
53

```

```

48 tmp = double(imread(path_SGM_occluded_pixels));
49 SGM_occluded_matrix = tmp(:,:,3); % r,g,b      255 means pixel occluded
50
51 id = fopen(path_SGM_disparity);
52 D_SGM_line = fread(id, rows * cols, 'float');
53 fclose(id);
54
55 D_SGM = zeros(rows, cols);
56 for x = 1 : 1 : rows
57     for y = 1 : 1 : cols
58         current_position = ((x - 1) * cols) + y;
59         D_SGM(x, y) = D_SGM_line(current_position);
60     end
61 end
62
63 clearvars tmp D_SGM_line id % save memory
64
65 %% compute error vector
66 error_vector = zeros(rows * cols, 2); % error column, occluded column (0 if occluded,
67     1 if not occlude)
68
69 for x = 1 : 1 : rows
70     for y = 1 : 1 : cols
71         current_position = ((x - 1) * cols) + y;
72         error_vector(current_position, 1) = abs(D_GT(x, y) - D_SGM(x, y));
73
74         if (SGM_occluded_matrix(x, y) == 255) || (D_GT(x, y) == 0) % occluded pixel
75             error_vector(current_position, 2) = 0;
76         else % not occluded pixel
77             error_vector(current_position, 2) = 1;
78         end
79     end
80 end
81
82 %% order pixels according to their confidence in decreasing order
83 confidence_sorted = zeros(rows * cols, 3, 5); % (confidence column, error column,
84     occluded column, number of cost function)
85
86 for i = 1 : 1 : number_of_costs
87     for x = 1 : 1 : rows
88         for y = 1 : 1 : cols
89
90             current_position = ((x - 1) * cols) + y;
91             confidence_sorted(current_position, 1, i) = cost(x, y, i);
92
93         end
94     end
95
96     confidence_sorted(:, 2, i) = error_vector(:, 1); % save error
97     confidence_sorted(:, 3, i) = error_vector(:, 2); % save occluded column
98
99     % sort by first column
100     tmp = confidence_sorted(:, :, i);
101     confidence_sorted(:, :, i) = - sortrows(- tmp, 1);
102 end
103
104 clearvars cost % save memory
105
106 %% order the error vector in decreasing order
107 error_vector = sortrows(error_vector, 1);
108
109
110

```

```

111
112 %% parameter of plots
113 step = 20; % if step = 20 increasing comparison 5% at each time
114
115
116 %% number of erroneous pixels ( $|d_{gt} - d| > 1$ ) vs number of considered pixels
117 error_curves = zeros(step, number_of_costs + 1);
118
119 for s = 1 : 1 : step
120
121     percent_of_pixels = 100 - ((step - s) * (100 / step)); % 5 : 10 : 15 ... 95 : 100
122     number_of_considered_pixels = round(((rows * cols) / 100) * percent_of_pixels);
123
124     for i = 1 : 1 : number_of_costs
125         number_of_erroneus_pixels = 0; % with error bigger than 1
126         number_of_occluded_pixels = 0;
127
128         for j = 1 : 1 : number_of_considered_pixels % sum
129
130             if confidence_sorted(j, 3, i) == 0
131                 number_of_occluded_pixels = number_of_occluded_pixels + 1;
132             elseif confidence_sorted(j, 2, i) > 1
133                 number_of_erroneus_pixels = number_of_erroneus_pixels + 1; % pixel
134                 with bigger than 1 error
135             end
136
137         end
138         error_curves(s, i) = number_of_erroneus_pixels / (number_of_considered_pixels
139             - number_of_occluded_pixels);
140
141     end
142
143     number_of_erroneus_pixels = 0;
144     number_of_occluded_pixels = 0;
145     for j = 1 : 1 : number_of_considered_pixels % sum
146
147         if error_vector(j, 2) == 0
148             number_of_occluded_pixels = number_of_occluded_pixels + 1;
149         elseif error_vector(j, 1) > 1
150             number_of_erroneus_pixels = number_of_erroneus_pixels + 1;
151         end
152
153     end
154     error_curves(s, number_of_costs + 1) = number_of_erroneus_pixels / (
155         number_of_considered_pixels - number_of_occluded_pixels);
156
157 end
158
159 %% average disparity error ( $|d - d_{gt}|$ ) vs number of considered pixels
160 average_error_curves = zeros(step, number_of_costs + 1);
161
162 for s = 1 : 1 : step
163
164     percent_of_pixels = 100 - ((step - s) * (100 / step)); % 5 : 10 : 15 ... 95 : 100
165     number_of_considered_pixels = round(((rows * cols) / 100) * percent_of_pixels);
166
167     for i = 1 : 1 : number_of_costs
168         number_of_occluded_pixels = 0;
169         to_sum = 0;
170
171         for j = 1 : 1 : number_of_considered_pixels
172
173             if confidence_sorted(j, 3, i) == 0
174                 number_of_occluded_pixels = number_of_occluded_pixels + 1;
175             else

```



```

173         to_sum = to_sum + confidence_sorted(j, 2, i);
174     end
175
176     end
177
178     average_error_curves(s, i) = to_sum / (number_of_considered_pixels -
        number_of_occluded_pixels);
179 end
180
181 number_of_occluded_pixels = 0;
182 to_sum = 0;
183
184 for j = 1 : 1 : number_of_considered_pixels
185     if error_vector(j, 2) == 0
186         number_of_occluded_pixels = number_of_occluded_pixels + 1;
187     else
188         to_sum = to_sum + error_vector(j, 1);
189     end
190 end
191
192 average_error_curves(s, number_of_costs + 1) = to_sum / (
    number_of_considered_pixels - number_of_occluded_pixels);
193 end
194
195
196 %% plot
197 legend_string = cell(number_of_costs + 3, 1);
198 for i = 1 : 1 : number_of_costs
199     legend_string(i) = cost_functions(i);
200 end
201 legend_string(number_of_costs + 1) = {'GT'};
202 legend_string(number_of_costs + 2) = {'Location'};
203 legend_string(number_of_costs + 3) = {'NorthWest'};
204
205 font_size = 16;
206
207 figure('units','normalized','outerposition',[0 0 2/3 1])
208 plot(linspace(number_of_costs, 100, step), error_curves, '-x')
209 h_title = title('number of erroneous pixels ( $|d - d_{gt}| > 1$ ) vs number of
    considered pixels');
210 set(h_title, 'FontSize', font_size);
211 h_xlabel = xlabel('number of higher confidence pixels (% of total)');
212 set(h_xlabel, 'FontSize', font_size);
213 h_ylabel = ylabel('number of pixels / number of erroneous pixels');
214 set(h_ylabel, 'FontSize', font_size);
215 h_legend = legend(legend_string{:});
216 set(h_legend, 'FontSize', font_size);
217 xlim([5 100])
218
219 figure('units','normalized','outerposition',[1/3 0 2/3 1])
220 plot(linspace(number_of_costs, 100, step), average_error_curves, '-x')
221 h_title = title('average disparity error ( $|d - d_{gt}|$ ) vs number of considered
    pixels');
222 set(h_title, 'FontSize', font_size);
223 h_xlabel = xlabel('Number of higher confidence pixels (% of total)');
224 set(h_xlabel, 'FontSize', font_size);
225 h_ylabel = ylabel('number of pixels / number of average erroneous pixels');
226 set(h_ylabel, 'FontSize', font_size);
227 h_legend = legend(legend_string{:});
228 set(h_legend, 'FontSize', font_size);
229 xlim([5 100])

```



# Bibliografia

- [1] A. Fusiello, *Visione Computazionale, appunti delle lezioni*, <http://profs.sci.univr.it/~fusiello>, 2008. 2, 3
- [2] D. Pfeiffer, S. Gehrig, N. Schneider, *Exploiting the Power of Stereo Confidences*, IEEE Conference on Computer Vision and Pattern Recognition, 2013. 2, 8
- [3] D. Scharstein, R. Szeliski, *A taxonomy and evaluation of dense two-frame stereo correspondence algorithms*, IJCV, 47(1-3):7–42, 2002. 3
- [4] H. Hirschmüller, *Accurate and efficient stereo processing by semi-global matching and mutual information*, IEEE CVPR, pages 807–814, San Diego, USA, June 2005. 3
- [5] X. Hu, P. Mordohai, *A Quantitative Evaluation of Confidence Measures for Stereo Vision*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2012. 10, 17
- [6] X. Hu, P. Mordohai, *Evaluation of Stereo Confidence Indoors and Outdoors*, IEEE Conference on Computer Vision and Pattern Recognition (CVPR), San Francisco, USA, June 2010. 7, 9
- [7] D. Scharstein, C. Pal, *Learning conditional random fields for stereo*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), Minneapolis, MN, June 2007. 7, 12
- [8] H. Hirschmüller, D. Scharstein, *Evaluation of cost functions for stereo matching*, IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2007), Minneapolis, MN, June 2007. 7, 12
- [9] *OpenCV library 2.4.9*, <http://opencv.org/>. 7
- [10] D. Scharstein, R. Szeliski, *High-Accuracy Stereo Depth Maps Using Structured Light*, Proc. CVPR, volume I, pages 195–202, 2003. 12