ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica – Scienza e Ingegneria

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

VALUTAZIONE SPERIMENTALE DI ALGORTIMI DI STEREO VISIONE FINALIZZATI AD UNA SUCCESSIVA IMPLEMENTAZIONE SU FPGA

CANDIDATO
Andrea Bombino

RELATORE
Prof. Stefano Mattoccia

CORRELATORE
Matteo Poggi

Anno Accademico 2015/2016

Sessione II

Ringraziamenti

Si chiude così il primo ciclo della mia carriera universitaria, che mi ha portato a raggiungere le prime soddisfazioni e ad affrontare le prime difficoltà della vita adulta. Le persone che conosco, che ho conosciuto e che mi sono state accanto in questo percorso sono tante e a tutte quante vorrei ringraziare.

In primo luogo ringrazio il professore Stefano Mattoccia e il Dottore Matteo Poggi, per il supporto, la fiducia e la disponibilità mostrata. Ringrazio tutta la mia numerosa famiglia, per aver sempre creduto in me, per avermi spronato e supportato in tutte le mie decisioni e le mie strade. Ringrazio i miei genitori e mia sorella per l'amore che mi hanno sempre dato e per i sacrifici che hanno fatto e ancora faranno per permettermi di soddisfare i miei sogni. Ringrazio di cuore la mia ragazza Francesca, per tutto l'aiuto datomi nei momenti difficili, per aver sempre trovato le parole per starmi vicino, e per sopportarmi soprattutto con lo studio. Senza di te sarebbe stato tutto più difficile, grazie di essere come sei, UNICA. Ringrazio i miei amici di Università Gianna, Buro, Simo, Tab, Fucio, Albe per aver condiviso con me obiettivi, pensieri, passioni ma soprattutto ansie e per avermi accompagnato quotidianamente in questi tre anni, rappresentando inconsciamente uno stimolo continuo per un mio miglioramento personale. Ringrazio i miei migliori amici di Monteguiduccio Brosio, Lele, Lupo, Serra, Taribo per aver riempito le mie giornate ed aver reso la mia vita più bella in un'infinità di modi anche tutt'ora che ci vediamo poco a causa del mio studio. Ringrazio i miei migliori amici di Cagli Peco, Ale, Vecchio, Giovi e Uomoz per rendermi ogni weekend impegnativo e diverso cercandomi di farmi star sempre il meno possibile sopra i libri, d'altronde "sa quei de cai non te sbai". Ringrazio i miei compagni ed amici di corso, per avermi reso il percorso più leggero, per avermi fatto affrontare le giornate sempre con positività ma soprattutto per avermi sopportato per gli esami; in particolare ringrazio Luca, Erika, Balda, Ventu e Alessio. Ringrazio anche i miei due coinquilini che ogni giorno mi sopportano all'interno della casa e che hanno visto in me l'ansia in persona. Ringrazio tutti i miei amici e tutti coloro che hanno sempre creduto in me e che continueranno a farlo.

Andrea

Sommario

Capitolo 1 – Introduzione

L'argomento della tesi riguarda il problema della 3D Vision e degli algoritmi che permettono di ricavare dati dall'ambiente circostante e sulla natura degli oggetti di cui esso è composto.

La visione stereo cerca di risolvere il problema utilizzando un dispositivo formato da due sensori (stereo camera) in modo da ricavare la distanza di in un oggetto in base alla posizione che va ad occupare nelle due immagini catturate dai sensori. Proprio come il cervello esegue in maniera automatica elaborando la scena "ripresa" dagli occhi. Le due immagini scattate in ogni momento dai nostri occhi sono molto simili tra loro ma non identiche poiché prese da posizioni diverse. Il cervello le sovrappone e, in base allo scostamento che compare, elabora un'unica immagine che è quella che vediamo noi realmente. Il principio di base è lo stesso presente in una stereo camera.

A questo punto il problema più arduo diventa determinare la corrispondenza tra i punti delle due immagini: un punto posizionato in un'immagine ha un suo corrispondente nell'altra immagine?

Alla base di questo concetto, in un sistema stereo, c'è la geometria epipolare [8]:

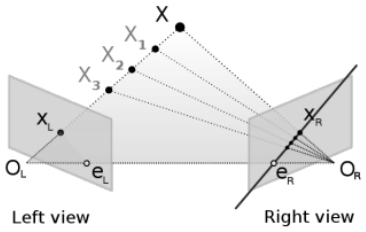


Figura 1.1 Schema di Stereo Camera

date due immagini raffiguranti una scena comune e X un punto osservato da entrambe, si ha che nella vista di sinistra tale punto è mappato nello spazio a due dimensioni in X_L e dall'insieme dei punti che in R^3 portano da X a X_L. Tale spazio è visto come un unico punto rispetto al centro ottico di sinistra O_L, mentre nella seconda immagine l'insieme è visto come una linea retta (linea epipolare) [7].

Da qui un importante vincolo: qualsiasi punto X_i nella seconda immagine, che corrisponde al punto X visto dalla prima, risiede lungo tale retta e di conseguenza è giusto pensare che esista un sistema di mappatura tra punti delle due immagini.

Quindi attraverso la corrispondenza dei punti è possibile ricavare la mappa di disparità: un'immagine in cui la profondità (depth) è codificata attraverso l'intensità del colore; tipicamente a valori più alti di intensità corrisponde una distanza minore.

Esistono numerosi algoritmi che permettono di risolvere il problema della corrispondenza (*correspondence problem*) appena descritto.

Visto che questi algoritmi richiedono operazioni computazionalmente onerose, una implementazione su dispositivi hardware a elevate prestazioni come Field Programmabile Gate Array (FPGA) sarebbe molto vantaggiosa al fine di ridurre consumi e aumentare il frame-rate. A tal proposito, in questa tesi, sono state valutate, mediante algoritmi implementati in software su PC, le prestazioni di metodi a una successiva implementazione su FPGA per determinare quale potessero essere i più idonei.

Questa valutazione via software è stata effettuata per applicare successivamente l'algoritmo migliore su FPGA poiché l'implementazione diretta su FPGA è molto costosa in termini di tempi di progettazione.

Capitolo 2 - Visione Stereo

La teoria della Visione Stereo è basata sull'utilizzo di due o più camere che inquadrano una stessa scena da differenti punti di vista. Nel caso ideale di Fig.2.1 le due camere sono perfettamente allineate e poste alla stessa altezza, a una determinata distanza T tra di loro, e osservano entrambe lo stesso punto P rispettivamente dai centri di proiezione Ol e Or. Per ciascuna di esse, P è proiettato sul rispettivo piano di proiezione nei punti pl e pr, aventi medesime coordinate verticali [4].

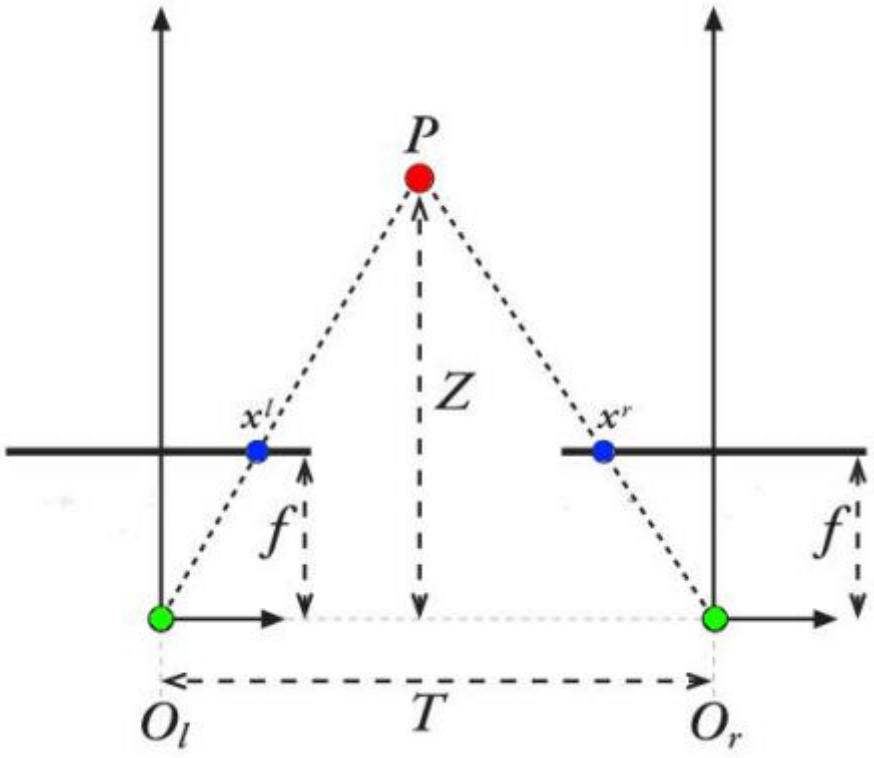


Figura 2.1 - Calcolo della profondità mediante triangolazione (sistema stereo ideale)

Dal confronto della coppia di immagini ottenuta da un sistema in questa configurazione è possibile ottenere informazioni sulla profondità della scena stessa. Infatti, sfruttando la particolare geometria del sistema stereoscopico (geometria epipolare), è possibile valutare lo scostamento orizzontale tra i pixel omologhi delle due immagini[6].

Chiamando xl e xr le coordinate orizzontali di p l e pr sui rispettivi piani, tale scostamento xl - xr (denominato disparità) permette di determinare, tramite una procedura di triangolazione e la conoscenza di opportuni parametri del sistema stereoscopico, la posizione del punto nello spazio: più i due punti sono lontani, minore è la disparità e viceversa.

L'esatta misura di profondità Z è calcolata mediante la formula seguente [2]:

$$\frac{T - (x^l - x^r)}{Z - f} = \frac{T}{Z} \Rightarrow Z = \frac{fT}{(x^l - x^r)}$$

Dove T è la distanza tra i due centri di proiezione e f è la lunghezza focale delle due camere.

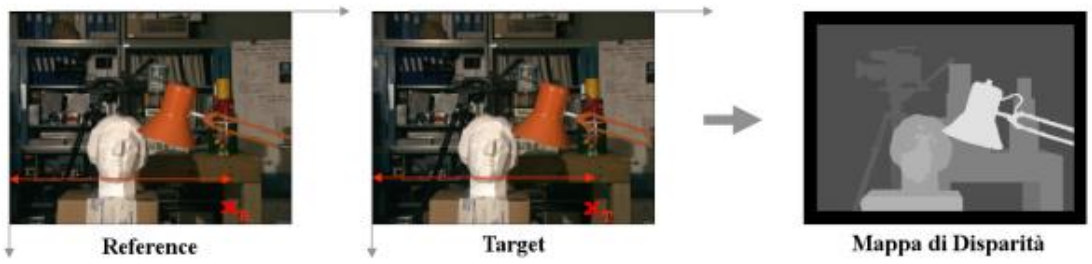


Figura 2.2 - Un esempio di mappa di disparità

Individuando le coppie di pixel omologhi delle immagini (matching stereo) e iterando il calcolo della disparità per tutte le coppie di pixel ottenute, si ottiene la cosiddetta mappa di disparità, ossia un immagine in scala di grigi dove l'intensità di ciascun pixel è proporzionale alla disparità in quel punto.

Come già sottolineato, la visione stereo è in grado di generare mappe di disparità attraverso la corrispondenza dei punti e di conseguenze dei punti, tra immagini corrispondenti (correspondence problem).

Un modo per affrontare questo problema è scomporlo in due fasi, che verranno analizzate di seguito: rettificazione e stereo matching.

2.1 Rettificazione

In linea di principio, dato un punto A in un'immagine, per trovare il punto corrispondente B nella seconda immagine è necessario effettuare una ricerca in due dimensioni lungo tutta la seconda immagine[1].

Per semplificare la ricerca si procede con la rettificazione: processo che consente di rimuovere la distorsione delle lenti e di modificare l'immagine in ingresso in maniera che punti corrispondenti si trovino sulla stessa linea retta orizzontale.

Lo scopo di questa operazione è di rendere le due camere virtualmente parallele e perfettamente allineate.

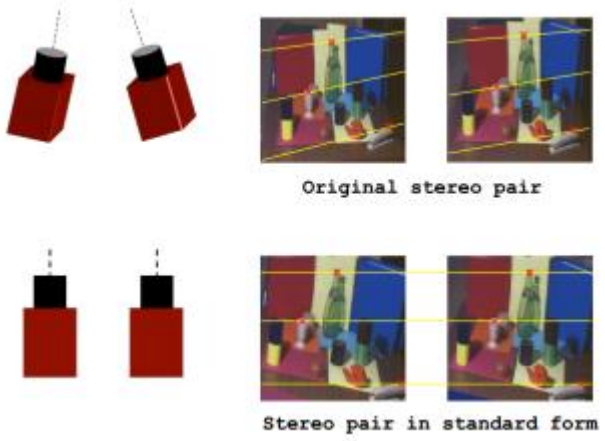


Figura 2.3 Rettificazione perfetta

Quindi la rettificazione costituisce una fase preliminare che consente di ridurre drasticamente le elaborazioni necessarie per affrontare il problema delle corrispondenze.

Il modulo che si occuperà di eseguire la rettificazione può essere visto come un filtro che, date in ingresso le immagini distorte produce in uscita le immagini rettificate sulle quali poi sarà effettuato il matching dei punti corrispondenti.

2.2 Algoritmi per la corrispondenza stereo

Il secondo stadio, una volta superata la fase di rettificazione, si occupa di trovare per ogni punto dell'immagine di riferimento (tipicamente l'immagine di sinistra left) il punto corrispondente nell'immagine target (immagine di destra right) [6].

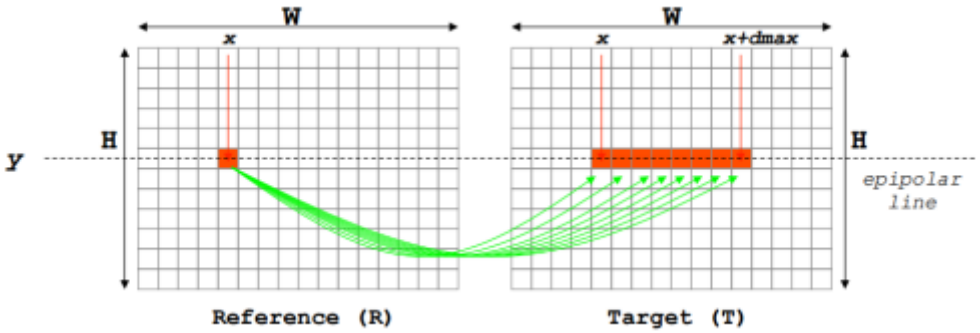


Figura 2.4 Algoritmo di matching basico

Questa ricerca è all'interno della stessa riga grazie alla rettificazione e al vincolo epipolare. Una volta trovato il punto corrispondente diventa facilmente ricavabile la mappa di disparità, poiché a ogni punto dell'immagine di riferimento è associato un valore di intensità, *disparità,* che indica la distanza del punto corrispondente: quanto più i punti corrispondenti sono lontani e tanto più saranno vicini alla camera e la disparità sarà alta.

Per limitare le risorse e aumentare le prestazioni la disparità non viene calcolata per tutti i valori fisicamente possibili ma all'interno di un range ristretto che in seguito sarà indicato come: [DISPARITY_MIN, DISPARITY_MAX] [1].

Si può dedurre che al diminuire di DISPARITY_MIN l'algoritmo sarà in grado di calcolare correttamente la disparità per punti sempre più lontani, mentre all'aumentare di DISPARITY_MAX verranno rilevati correttamente punti più vicini alla camera.

La scelta dell'algoritmo di matching è fondamentale ed influenza ogni aspetto della progettazione hardware utilizzate e la qualità della mappa generata ed esistono due categorie di algoritmi stereo[2]:

- **Algoritmi locali:** la scelta del punto corrispondente è effettuata considerando solo le vicinanze del punto in esame ed è di tipo "Winner Takes All"; l'algoritmo più semplice è quello che, preso un punto dell'immagine di riferimento, effettua il confronto in N punti dell'immagine target (nella tessa riga) e sceglie quello che minimizza la differenza in valore assoluto (cioè quello più simile al punto di riferimento). Altri algoritmi poco più evoluti effettuano la ricerca in una finestra di matching aggregando i costi trovati. Gli algoritmi locali sono prevalentemente performanti e utilizzano poche risorse, però in alcuni casi la mappa di disparità generata può non essere di buona qualità, soprattutto in regioni uniformi.

La scelta di aggregare i costi di una finestra porta inoltre ad ottenere un'ulteriore perdita di qualità in prossimità dei bordi, in quanto punti appartenenti a regioni di spazio ben distinte vengono utilizzati assieme.

- **Algoritmi globali:** la scelta del punto corrispondente è effettuata considerando tutta l'immagine, in particolare viene cercato un punto che minimizzi una determinata funzione di costo lungo tutta l'immagine.

Esiste poi una categoria intermedia di algoritmi detti **semi-globali** in cui la funzione di costo viene minimizzata all'interno di una porzione dell'immagine.

Di quest'ultima categoria appartiene l'algoritmo analizzato successivamente in questa tesi, SGM, e le sue varianti.

2.3 Filtering delle immagini in input

Tra i moduli di rettificazione ed SGM sono presenti altri due moduli di pre-filtraggio: uno che implementa la trasformata Census e uno che implementa il Box-Filtering.

La trasformata Census viene applicata in un'immagine usando una finestra di pixel (*census windows)* il cui centro coincide con il pixel in esame e traduce tale finestra in una stringa di bit.

Tale funzione di controllo utilizzata per ogni pixel è la stessa ed è semplicemente:

$$census(i) = 1 \;\; if \;\; I\big(p(i)\big) > I\,(p_c)\; else\; 0$$

dove I è l'intensità del pixel, $p(i)$ è l'i-esimo pixel della finestra e p_c è il pixel centrale.

Per questo motivo viene detta anche trasformata binaria, come i due valori possibili che può assumere.

Considerando una finestra quadrata, come tipicamente viene scelta, la trasformata Census per ogni pixel produce in uscita una stringa di NxN – 1 bit, dove N è la larghezza della finestra.

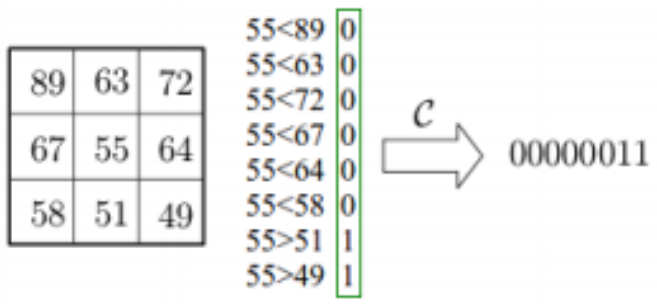


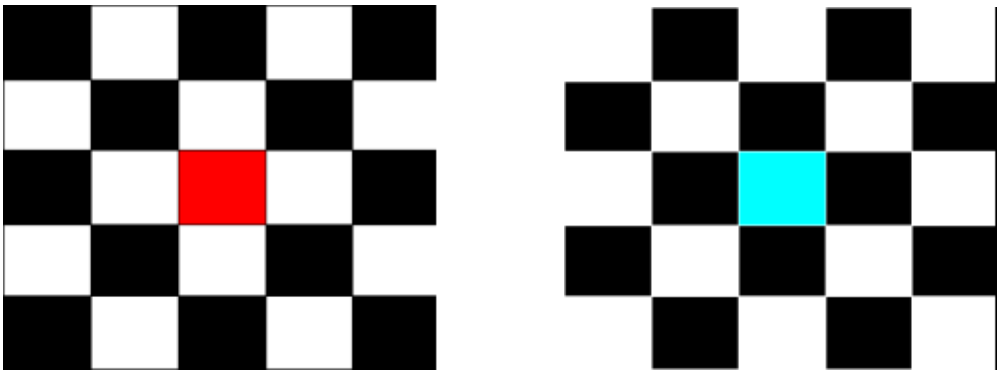
Figura 2.5 Esempio trasformata Census

La Census quindi è una trasformata locale e porta dei vantaggi come tolleranza al rumore e a trasformazioni monotone dell'intensità, in cambio di un costo computazionalmente basso: i confronti fatti possono essere eseguiti in parallelo e

con poche risorse, e il risultato in uscita resta comunque una stringa di bit di dimensioni modesta.

La trasformata Census implementata nel software utilizza una finestra 5x5 ma si ha la possibilità di cambiare le dimensioni e presenta due variazioni:

- La trasformata non viene calcolata su tutti i punti della finestra ma in un intervallo più ristretto cosiddetto a scacchiera; in questo modo si perdono informazioni ma si ottiene una stringa di bit più piccola (12 confronti contro 24) che potrà essere gestita in maniera più efficiente da SGM. A sua volta questo tipo di trasformata può esser considerata in due casi distinti, ovvero, a scacchiera pari e a scacchiera dispari.



Scacchiera pari Scacchiera Dispari

- Viene utilizzata una variante della Census binaria, chiamata Census ternaria la cui funzione è:

$$census\ ternaria(i) = \begin{cases} a & if \quad -\epsilon \leq I(p(i) - I(p_c)| \leq \epsilon \\ b & if \quad I\big(p(i)\big) - I(p_c) > \epsilon \\ c & if \quad I\big(p(i)\big) - I(p_c) < -\epsilon \end{cases}$$

dove a, b e c possono essere codificati a piacere.

Questa variante permette di introdurre un intervallo di incertezza (quello centrale, in quanto non c'è stata una variazione di intensità) che potrà essere opportunamente gestito da SGM.

Lo svantaggio è soltanto in termini di risorse, in quanto ad ogni confronto sono associati 2 bit anziché 1, quindi la stringa di uscita ritorna ai 24 bit originari.

Anche in questo caso possiamo distinguere due casi differenti ovvero scacchiera pari e dispari.

A questo punto SGM non riceverà più in ingresso direttamente i pixel ma la stringa generata dalle due trasformate Census (left e right); la funzione che calcolerà il costo locale dovrà quindi tenerlo in considerazione.

Capitolo 3 - Ambiente di Sviluppo e Organizzazione

Lo scopo della tesi è quello di capire quale algoritmo di visione stereo applicare e con quali parametri di configurazioni risultasse migliore ovvero quello che abbia un indice di errore minore, per poi essere implementato su FPGA.

Questi algoritmi di stereo vision sono stati applicati a un dataset di immagini, KITTI_2015, all'interno del quale sono presenti 200 immagini che rappresentano diverse scene ognuna delle quali ha subito una rettificazione perfetta.

Per effettuare questa analisi è stato realizzato un progetto software utilizzando come linguaggio di programmazione C con il supporto della libreria OpenCV, concepita per agevolare le implementazioni di algoritmi Computer Vision. Infatti l'utilizzo di questa libreria consente un'efficace manipolazione dei dati multimediali, partendo dalla visualizzazione di immagini fino all'utilizzo dei singoli pixel.

Il progetto è organizzato da tre passi eseguiti in pipeline:

1) Calcolo del costo puntuale;
2) Aggregazione dei costi;
3) SGM con le sue varianti.

Alla fine di queste fasi di esecuzione, avremmo salvati all'interno di un'apposita struttura dati tuti i costi che permettono di generare la mappa di disparità.

Attraverso questa mappa e ad un immagine di riferimento possiamo calcolare l'indice di errore in percentuale che rappresenta, per quella determinata immagine il rapporto fra pixel corretti e pixel totali.

L'immagine di riferimento è chiamata *ground-truth* e rappresenta i valori ideali di disparità ottenuta tramite un sensore Lidar.

In questa immagine, i pixel a zero sono pixel per i quali non abbiamo i valori di ground-truth perché il sensore non è in grado di acquisire i valori per tutti i pixel (essendo un laser che rimbalza sulla scena), ed è per questo motivo che consideriamo come pixel reali quelli che hanno valore diverso da zero.

Alcuni pixel reali però possono essere dotati ancora di errore ed è per questo che vengono definiti i pixel corretti ovvero quei pixel dove viene garantita assenza di errore.

Per ogni immagine viene salvato su un file Excel l'indice di errore e il numero di pixel reali.

3.1 Calcolo dei costi locali

La funzione che si occupa di calcolare i costi locali deve comparare le stringhe di bit generate dalla trasformata Census e stabilire quando queste sono più o meno simili.

Ciò porta immediatamente ad utilizzare la distanza di Hamming come metro di somiglianza [1].

La distanza di Hamming fra due stringe di bit si ottiene contando il numero di bit diversi in posizioni corrispondenti:

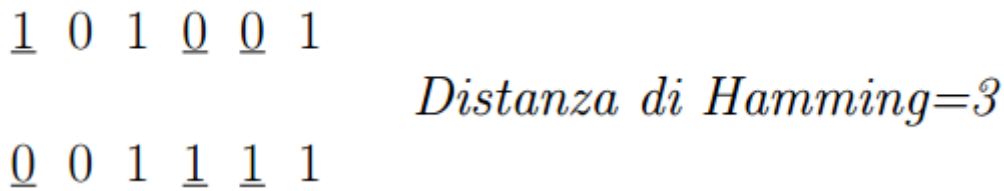


Figura 3.1 Esempio distanza di Hamming

Utilizzando la Census ternaria, e avendo quindi tre possibili valori in uscita è possibile scegliere quando considerare simili due stringhe di bit. In base all'analisi delle immagini generate successivamente, si è scelto di definire simili due stringhe di due bit soltanto quanto queste sono identiche tra di loro; in questo modo i pixel che nella Census erano stati considerati di fasce diverse verranno qui considerati diversi.

L’utilizzo della distanza di Hamming porta alcuni vantaggi e svantaggi:

- Il valore massimo che è possibile ottenere dalla distanza di Hamming è 12 ovvero quando le stringhe di bit sono ritenute simili. Quindi è possibile codificare i costi locali con soltanto 4 bit.
- Per natura, la distanza di Hamming, è un’operazione sequenziale (in quanto è necessario incrementare un contatore ogni volta che si verifica una somiglianza tra strignhe). Per sfruttare il parallelismo ed aumentare le prestazioni il calcolo è stato diviso in più moduli, ognuno dei quali opera all’interno di una sotto regione della stringa totale; i contatori di ogni modulo engono infine sommati e costituiscono il costo locale.

3.2 Aggregazione dei costi

Il secondo passo della pipeline prevede non più di calcolare le disparità basandosi soltanto sui valori dei costi calcolati pixel per pixel ma di avere una finestra detta finestra di aggregazione dei costi [2].

Così facendo, per il calcolo del costo viene effettuata la somma di tutti i pixel vicini al pixel centrale.

Per fare ciò è adottata una tecnica detta Fixed Window (FW): ovvero l’aggregazione dei costi in una finestra con dimensione fissa.

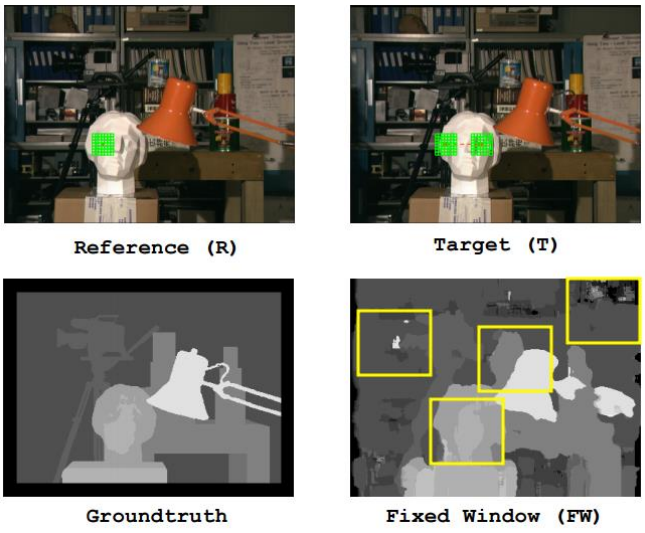


Figura 3.2 Fixed Windows

Ma ha diversi aspetti negativi:

- ❖ Assume implicitamente che le superfici siano frontali e parallele
- ❖ Ignora le discontinuità di disparità
- ❖ Non si occupa esplicitamente delle aree uniformi
- ❖ Non si occupa esplicitamente di schemi ripetitivi

Per quanto concerne FW, diminuendo la dimensione della finestra contribuisce a ridurre il problema della localizzazione dei bordi. Però questa scelta rende più ambiguo il problema della corrispondenza dei punti (specialmente quando si tratta di regioni uniformi).

In pratica, per l'approccio a finestra fissa la scelta della dimensione della finestra ottimale è fatta empiricamente.

Abbastanza sorprendentemente, nonostante le sue limitazioni, FW è ampiamente utilizzato. Si tenga conto che è l'algoritmo più usato frequentemente nelle applicazioni, poiché:

- È semplice e veloce da implementare grazie a schemi di calcolo incrementali
- Viene eseguito in real-time sui processori standard
- Non occupa molto spazio a livello di memoria
- L'implementazione hardware (FPGA) esegue in real-time con un'energia di consumo limitato (<1W).

All'interno della del progetto è stata utilizzata un'ottimizzazione di FW chiamata Box-Filtering (BF).

3.3 Algoritmo SGM

Questa tipologia di algoritmo, come detto in precedenza, appartiene alla categoria degli algoritmi semiglobale che effettua il calcolo dell'energia lungo 8 percorsi, chiamati *scanline*, a 45° l'uno dall'altro. Le scanline verranno numerate da 0 a 7 in senso orario a partire dalla scanline a sinistra/ovest, inoltre le scanline e le relative funzioni costo sono indipendenti fra di loro [3].

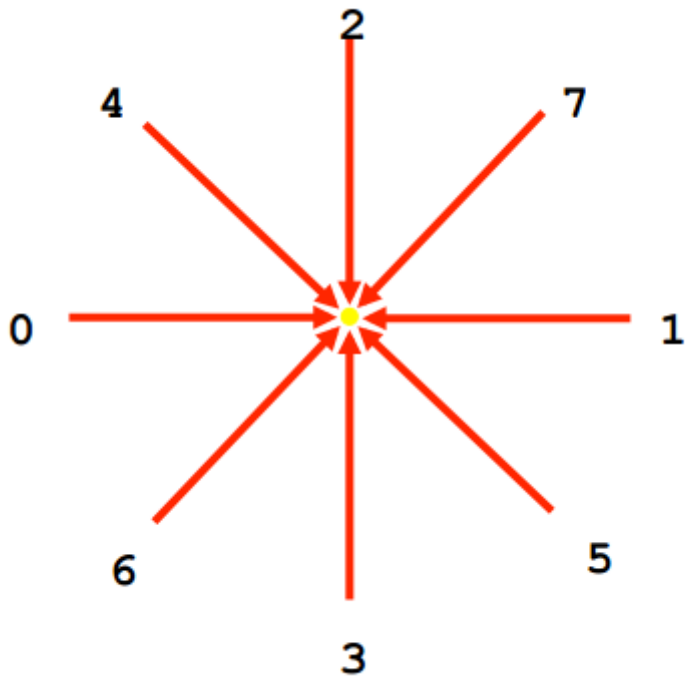


Figura 3.3 - Le otto scanline di SGM

La funzione di costo è espressa: $E(d) = E_{data}(d) + E_{smooth}(d)$ dove $E_{data}(d)$ misura la corrispondenza diretta tra i punti in esame (implementata come una semplice funzione di costo tra pixel) e $E_{smooth}(d)$ è un termine che penalizza le variazioni di disparità.

Possiamo ricavare delle considerazioni basate sulla definizione di E_{smooth}:

- In regioni uniformi il termine E_{smooth} sarà sufficientemente alto da impedire variazioni considerevoli di disparità, consentendo quindi di ridurre notevolmente il rumore.
- Nelle regioni di discontinuità come i bordi vi sarà una fase transitoria in cui inizialmente E_{smooth} impedirà la variazione di disparità. Nel momento in cui tale variazione si paleserà sempre più, entrambi i termini verranno ridotti e consentiranno di effettuare il cambio di disparità.

L'algoritmo utilizzato nel nostro caso è SGM con anche alcune sue varianti che vedremo successivamente.

SGM [5] è un algoritmo semi-globale che esegue il calcolo della funzione di costo lungo 8 percorsi chiamati *scanline*, a 45° l'uno dall'altro (le scanline saranno numerate da 0 a 7 in senso orario a partire dalla scanline a sinistra/ovest).

Inoltre, le scanline e le relative funzioni di costo sono indipendenti fra di loro.

La funzione di costo è calcolata lungo gli 8 percorsi all'interno dell'intervallo di disparità :$d \in [\, DISPARITY\_MIN\, , DISPARITY\_MAX\,]$.
Indicando con x e y le coordinate del punto analizzato, la formula della funzione costo è la seguente (riportata per il primo percorso ovvero la scanline 0):

$$L(x,y,d) = C(x.y,d) + \min\{L(x-1,y,d), L(x-1,y,d-1) \\ + P1, L(x-1,y,d+1)\, , L(x-1,y,i)\ + P2\,\}$$

Come si può notare la funzione di costo è ben suddivisa nelle due componenti E_{data} ed E_{smooth}:

- $E_{data} = C(x,y,d)$ è il risultato di una funzione costo locale applicata ai punti analizzati che è indipendente dal resto dall'algoritmo (è quindi possibile applicare funzioni costo locali differenti ottenendo output di maggiore o minore qualità); il termine $C(x,y,d)$ è indipendente dallo

specifico percorso in quanto calcolata sempre negli stessi punti con la stessa disparità.

- E_{smooth} è realizzato in maniera ricorsiva, consente di tener traccia della storia passata lungo lo specifico percorso e tende a penalizzare i cambi di disparità attraverso l'utilizzo di due pesi P1 e P2 (con P1 < P2): il cambio di disparità si verifica quando un punto immediatamente a destra o a sinistra riesce a vincere la penalità P1 detta piccola penalità, oppure quando un punto più lontano riesce a vincere P2 grande penalità.

  Per evitare il tendere all'infinito della funzione di costo è sottratto il termine $minL(x-1, y, i)$.

Lo scopo di P1 e P2 (detti anche smoothness penalities) è quello di penalizzare i cambi di disparità tra punti vicini, garantendo la continuità (smoothness) dell'immagine e permettendo quindi all'algoritmo di adattarsi alle superfici curve o inclinate grazie a P1 (negli approcci locali, i pixel appartenenti a superfici inclinate spesso sono assegnati alla stessa ipotesi di disparità, pur essendo a distanze diverse), ma anche di preservare eventuali discontinuità presenti nella scena grazie alla penalità maggiore P2. Un eventuale cambio di disparità molto accentuato, infatti, verrebbe considerato valido nel caso in cui il costo in quel punto sia talmente basso da essere il minimo nonostante la penalità introdotta.

Una volta calcolate le funzioni di costo lungo le 8 scanline, queste vengono aggregate e ne viene infine calcolato il minimo. L'indice d del minimo rappresenta la vera disparità calcolata da SGM.

3.4 Algoritmo eSGM

L'algoritmo SGM si presta molto bene all'implementazione su FPGA in quanto composto soltanto da operazioni tra interi e produce una mappa di disparità di qualità tipicamente superiore rispetto agli algoritmi locali [3].

Nonostante ciò utilizza per sua natura molte risorse di calcolo (dipendenti linearmente dalla larghezza dell'intervallo di disparità) e soprattutto di memoria (più precisamente richiede una ampia larghezza di banda nei trasferimenti con la memoria).

Nell'algoritmo SGM la disparità è determinata come indice a cui corrisponde il costo minimo tra i pixel. Tuttavia, ciascuno degli otto percorsi che contribuiscono alla funzione di costo, espone il proprio candidato di minimo.

La figura sottostante mostra i costi delle otto scanline da direzioni di verse nel pixel p per tutte le disparità.

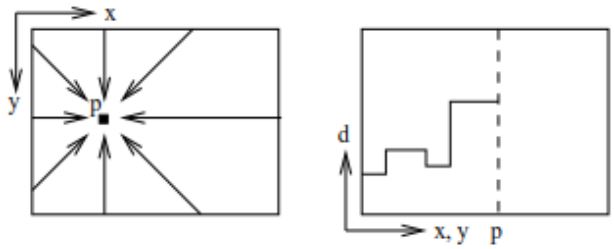


Figura 3.4 - Algoritmo SGM

Idealmente la posizione del minimo degli otto percorsi coincide con la disparità, però bisogna considerare che i percorsi possono essere "disturbati", per esempio in profondità e discontinuità. Ma comunque almeno un percorso deve prevedere la disparità correttamente.

L'idea è quindi quella di concentrarsi sono nelle posizioni di S, dove le otto scanline hanno minimo, quindi, al massimo possiamo avere otto posizioni distinte.

Se il costo minimo è in una posizione diversa rispetto a quello considerato dagli 8 percorsi? La risposta è che questa situazione è improbabile e se accade è, in molti casi e come verificabile sperimentalmente, per puro caso.

Pertanto, il nuovo metodo eSGM nel caso peggiore dovrebbe produrre risultati molto simili a quelli del metodo SGM originale.

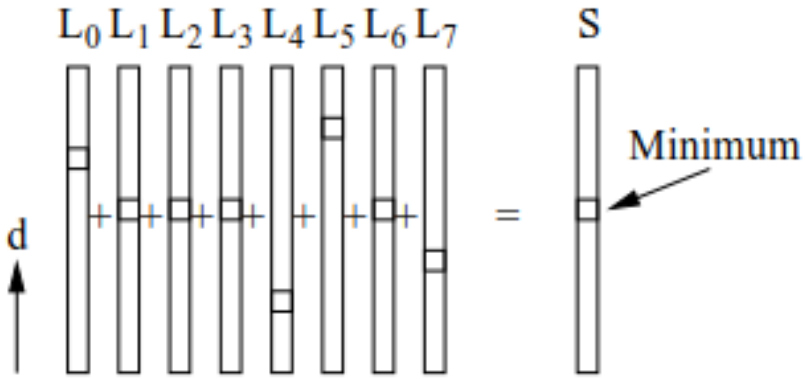


Figura 3.5 Algoritmo - eSGM

Il nuovo metodo eSGM può essere implementato con tre passaggi [3]:

1. Il primo passaggio agisce come nel metodo originale, tranne per memorizzare il risultato. Infatti invece di memorizzare i costi per tutte le disparità, per ciascun pixel vengono considerati solo 4 percorsi, dall'alto al basso, e sono memorizzati solo i costi per queste quattro disparità.

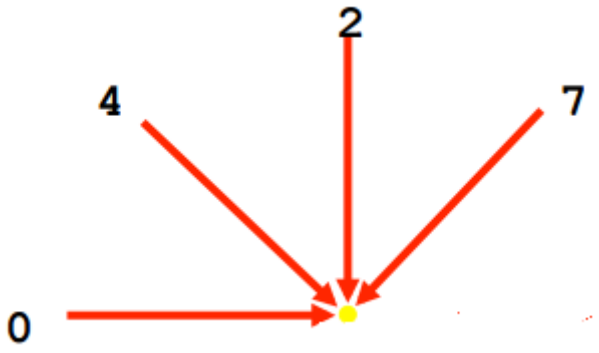


Figura 3.6 Quattro scanline superiori - Maschera 149

Il secondo passaggio calcola i costi per le 4 scanline rimanenti ovvero dal basso all'alto.

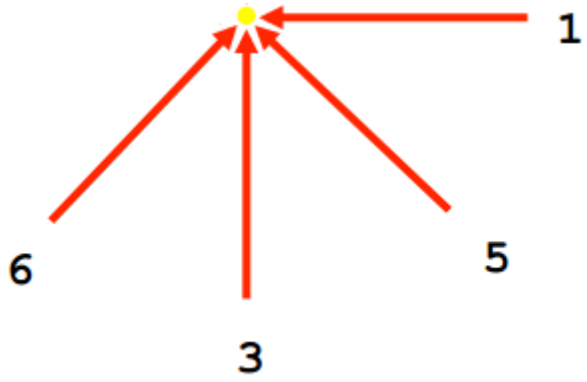


Figura 3.7 Quattro scanline inferiori - Maschera 106

Il vantaggio di utilizzare solo quattro percorsi consente di calcolare un risultato intermedio, scegliendo il costo più basso tra i quattro costi del primo passaggio cosi permettendo di liberare la memoria che verrà utilizzata per memorizzare i costi a quattro minimi del secondo.

2. L'aggregazione è completata da un terzo passaggio che calcola gli stessi percorsi come nel primo passaggio come nel primo ma aggiunge i costi minimi laddove sono stati identificati i minimi nel secondo passaggio.

Il minimo finale di ogni pixel può essere selezionato tra questi quattro minimi e il risultato intermedio del secondo passaggio.

Il ragionamento alla base di questo nuovo metodo permette di derivare la *confidence-matching* come il numero di percorsi che hanno disparità minima coincidente con la disparità finale.

Ovviamente se i percorsi di tutte le direzioni hanno lo stesso minimo allora la confidenza è molto alta.

Quindi il metodo eSGM migliora l'utilizzo della memoria notevolmente ma aumenta il tempo di calcolo del 50%. Il beneficio di eSGM è ancora maggiore su FPGA siccome la dimensione della memoria è tipicamente più limitata su questi dispositivi.

3.5 Varianti di eSGM

All'interno del progetto sono state adottate anche tre varianti dell'algoritmo eSGM per cercar di analizzare altre possibili implementazioni in modo da poter scegliere quale fosse la più conveniente in termini di tempo, memoria e risorse.

I tre algoritmi, hanno come riferimento l'algoritmo eSGM ma portano differenti modifiche sul calcolo del minimo.

L'idea di base è la seguente: dato che eSGM standard per ogni direzione (scanline), in ogni pixel può avere un massimo di otto vincitori, ovvero otto minimi tutti diversi (e questo accade se ogni direzione vota uno diverso), non consideriamo solo questi ma anche il vincitore sommando le quattro scanline superiori e il vincitore delle quattro inferiori.

Quindi la struttura di appoggio (un array) non sarà più solo di otto elementi ma di dieci elementi.

Per questo motivo le varianti applicate a eSGM sono:

- **eSGM_ten_winners:** il quale si occupa di implementare l'idea di base descritta in precedenza.
- **eSGM_ten_winners_2_scans:** somma i costi delle prime quattro scanline alla prima passata e delle rimanenti alla seconda. Non esegue la terza passata (detto eSGM a doppia passata).
- **eSGM_sums:** mentre in quest'ultima implementazione si mantiene in memoria solo due vincitori, ottenuti dalla somma delle scanline di ogni passata.

Capitolo 4 - Ricerca dei parametri ottimali

La pipeline finale realizzata, come detto sopra, è quindi formata dalle seguenti parti:

- La prima parte che comprende il calcolo della trasformata Census (con griglia completa, binaria, binaria pari/dispari, ternaria, ternaria pari/dispari) quindi il calcolo del costo locale con l'aggiunta dei costi di aggregazione dovuti al box-filtering.

- La seconda parte è composta da algoritmi semiglobali (SGM, eSGM e sue varianti) che presi in ingresso i costi aggregati produce in uscita la disparità.

  Quest'ultima parte si occupa essenzialmente di calcolare il minimo dei costi aggregati e in particolare il suo indice che altro non è che la disparità.

  Cosicché alla fine avremo una mappa di disparità attraverso la quale, effettuando un confronto pixel per pixel con l'immagine di ground-truth, calcoleremo l'indice di errore percentuale.

4.1 File di configurazione

All'interno della pipeline come appena descritto possiamo aver diversi algoritmi per entrambe le parti da cui essa è composta. Ognuno di questi algoritmi a sua volta ha dei parametri di configurazione che possono variare.

Quindi il progetto prevede un file di configurazione avente la seguente organizzazione:

1. *Parte riguardante al file system:*

```
#define IMAGE_PATH_LEFT "/path/KITTI_2015/image_0"
#define IMAGE_PATH_RIGHT "/path/Scrivania/KITTI_2015/image_1"
#define IMAGE_PATH_GROUND_TRUTH "/path/KITTI_2015/disp_occ"
#define FILE_OUTPUT_PATH "/path/Scrivania/SenzaMedia"
```

Figura 4.1 Path - File di configurazione

- ***#define IMAGE_PATH_LEFT*:** che corrisponde al path (percorso) nel file system delle immagini riprese dalla camera di sinistra del dataset in considerazione.
- ***#define IMAGE_PATH_RIGHT*:** che corrisponde al percorso nel file system delle immagini riprese dalla camera di destra.
- ***#define IMAGE_PATH_GROUND_TRUTH*:** corrisponde al percorso nel file system delle immagini di ground-truth a cui facciamo riferimento.
- ***#define FILE_OUTPUT_PATH***: consiste nel percorso e in quale file vogliamo andare a salvare i nostri file.

2. *Parametri box-filtering:* come detto in precedenza all'interno del programma, dopo aver effettuato la trasformata di Census, abbiamo l'aggregazione locale dei costi a finestra fissa ovvero il box-filtering.

   Per quanto riguarda questo modulo abbiamo i seguenti parametri:

   - ***#define SIZE_BOXFILTERING N*:** in cui il valore N rappresenta le dimensioni della finestra.
   - ***#define AREA N***
   - ***#define AVERAGE X*:** dove il valore X può assumere il valore 0 o 1.

     Il valore di X è settato a 1 implica che all'aggregazione dei costi locale viene applicata una media puntuale, ovvero ogni costo di un pixel viene diviso per l'area della finestra.

     Così facendo viene ridotto il valore massimo del costo da *COSTO_MAX_SINGOLO_PIXEL * AREA* a *COSTO_MAX_SINGOLO_PIXEL.*

     Questo permette di ridurre il numero di bit da impiegare, ed è molto comodo soprattutto quando vengono utilizzate finestre di

dimensione elevata, in cui il numero di bit potrebbe crescere eccessivamente.

Per esempio: se consideriamo una trasformata Census binaria completa con dimensione della finestra 5x5 il costo massimo che il pixel potrà assumere è 24. Se aggreghiamo in una finestra delle stesse dimensioni, 5x5 senza considerare la media, il valore massimo aggregato diventa 600, mentre con la media rimane 24.

Quindi senza media servirebbero 10 bit per memorizzare il costo di ogni pixel, mentre con la media ne bastano solo 5.

3. *Maschere di algoritmi:* rappresentano gli algoritmi che verranno processati all'interno della pipeline.

```
int SET_CENSUS_ALGORITM [] {1,1,0,0,0,0};
int SET_SGM_ALGORITM [] {1,1,1,1,1};
```

Figura 4.2 Maschere di algoritmi in considerazione

Identificati da un array di interi, verranno eseguiti solamente quelli con valore 1 mentre gli altri verranno automaticamente scartati.

Per quanto riguarda ***SET_CENSUS_ALGORITM*** rappresenta uno tra i seguenti possibili algoritmi di costi locali:

- Posizione 0: trasformata Census binaria a griglia completa;
- Posizione 1: trasformata Census binaria a griglia pari;
- Posizione 2: trasformata Census binaria a griglia dispari;
- Posizione 3: trasformata Census ternaria a griglia completa;
- Posizione 4: trasformata Census ternaria a griglia pari;
- Posizione 5: trasformata Census ternaria a griglia dispari;

Mentre per quanto riguarda ***SET_SGM_ALGORITM*** altro non è che l'insieme di algoritmi semiglobali:

- Posizione 0: SGM classico;
- Posizione 1: eSGM classico;
- Posizione 2: eSGM_ten_winners;
- Posizione 3: eSGM_ten_winners_2_scans;

- Posizione 4: eSGM_sums;

4. *Parametri di configurazione del set di algoritmi semiglobali:* tutti gli algoritmi il cui valore è settato a 1 in ***SET_SGM_ALGORITM*** vengono eseguiti con stessi parametri di configurazione:

   (t_DSI inputDSI, t_DSI *outputDSI, int dMax, float P1, float P2, int mask)*

   - **t_DSI * inputDSI:** un puntatore ad una struttura contenente l'aggregazione dei costi calcolata dal box-filtering.
   - **t_DSI * output:** puntatore alla struttura di output alla quale verranno assegnati i costi dell'algoritmo in considerazione.
   - **dMax:** valore di disparità massimo rilevato dall'immagine di ground-truth;
   - **P1, P2:** parametri fondamentali che, come espresso sopra, identificano le penalità all'interno dell'algoritmo semiglobale.
   - **mask:** indica la configurazione degli 8 percorsi (scanline) considerata. Per esempio se mask assumesse il valore 255 vuol dire che nell'algoritmo selezionato vengono considerate tutte otto le direzioni.

   Questi ultimi parametri P1, P2 e mask sono configurati all'interno del file di configurazione.

   Il parametro mask è semplicemente un intero: ***#define MASK valore***.

Mentre le penalità P1 e P2 sono configurate in modo tale che quando viene considerato un algoritmo dei costi locali possono variare ovvero:

```
#define P1_MIN  2
#define P1_MAX 10
#define P1_STEP 2
#define P2_MIN  30
#define P2_MAX 50
#define P2_STEP 5
```

Figura 4.3 Configurazione penalità

- Pi_MIN: identifica il valore minimo di Pi da cui partire.
- Pi_MAX: identifica il valore massimo che può assumere la penalità Pi.
- Pi_STEP: identifica di quanto viene incrementato il valore di Pi ogni iterazione.

Se consideriamo un box-filtering senza media ovviamente a questi algoritmi dovremmo passare le due penalità (P1, P2) moltiplicate per l'area della finestra di aggregazione.

4.2 Esecuzione del programma

Dopo aver settato i vari parametri nel file di configurazione è possibile eseguire il programma software.

Per prima cosa vengono caricate le immagini:

- Left immagine rilevata dalla camera di sinistra;
- Right immagine rilevata dalla camera di destra;
- Ground-truth immagine di riferimento.

L'immagine Left permette di crearne una nuova con le stesse caratteristiche strutturali ma in bianco e nero:

IplImage DisparityL = cvCreateImage(cvGetSize(L), 8, 1);*

Considerando un'immagine alla volta applichiamo gli algoritmi il cui valore è 1 in SET_CENSUS_ALGORITM.

Per ognuno di questi, in base a quale algoritmo è settato, vengono calcolati i costi locali attraverso la trasformata Census impostata dall'algoritmo.

```
switch(ALGORITM_GRID){
    case 0 :
        printf(" - - - Running Algoritm With All Grid Binary - - -\n");
        HammingDistanceCost(L, R, dMax, CENSUS_WINDOWS, DSI);
        break;
    case 1 :
        printf(" - - - Runnging Algoritm With Grid Binary Peer - - -\n");
        HammingDistanceCostBinaryGrid(L,R,dMax,CENSUS_WINDOWS,DSI,1);
        break;
    case 2:
        printf(" - - - Runnging Algoritm With Grid Binary Odd - - -\n");
        HammingDistanceCostBinaryGrid(L,R,dMax,CENSUS_WINDOWS,DSI,0);
        break;
    case 3:
        printf(" - - - Runnging Algoritm With All Grid Ternary - - - \n");
        HammingDistanceCostTernary(L, R, dMax, CENSUS_WINDOWS, DSI);
        break;
    case 4:
        printf(" - - - Runnging Algiritm With Grid Ternary Peer - - - \n");
        HammingDistanceCostTernaryGrid(L, R, dMax, CENSUS_WINDOWS, DSI,1);
        break;
    case 5:
        printf(" - - - Runnging Algiritm With Grid Ternary Odd - - - \n");
        HammingDistanceCostTernaryGrid(L, R, dMax, CENSUS_WINDOWS, DSI,0);
        break;
}
```

Figura 4.4 Algoritmi Trasformata Census

Come si può vedere tutti gli algoritmi accettano come parametri l'immagine Left,Right il valore dMax, la dimensione della girglia (CENSUS_WINDOW) e una struttura DSI. Struttura che rappresenta una matrice dove ciascun elemento rappresenta il costo della corrispondenza tra il pixel in posizione (x,y) dell'immagine Reference ed il pixel in posizione (x,y) dell'immagine Target alla disparità d,

Dopo aver effettuato la trasformata Census viene eseguita l'aggregazione dei costi locali attraverso il boxfiltering.

Il passaggio successivo consiste nell'esecuzione di tutti gli algoritmi semiglobali settati a 1 in SET_SGM_ALGORITM.

L'esecuzione di tutti questi viene effettuata per ogni algoritmo di costo locale facendo variare, come descritto sopra, le penalità P1 e P2.

In caso il parametro AVERAGE è settato a 1 gli algoritmi semi-globali viene considerata l'aggregazione dei costi sia con media puntuale sia senza, altrimenti vengono eseguiti senza considerare box-filtering con media.

Infine, viene creata la mappa di disparità attraverso la struttura dati DSI_OUTPUT e l'immagine DisparityL:

disparity_map(DSI_OUTPUT, DisparityL, false);

Attraverso questa mappa e all'immagine di Ground-truth calcoliamo l'indice di errore, come descritto in precedenza, per quella determinata immagine con quella precisa configurazione di parametri e algoritmi.

Il risultato dell'indice appena calcolato, l'immagine i-esima e il contatore dei pixel reali vengono salvati su file ai fini di capire quale sia l'algoritmo migliore.

00000_10	9.37	88975
00001_10	6.38	107175
00002_10	13.87	100480
00003_10	13.18	94608
00004_10	20.41	97007
00005_10	15.42	92921
00006_10	21.85	111326
00007_10	5.31	109354
00008_10	4.65	101836
00009_10	5.32	95997

Figura 4.5 Struttura del file di output

Per ogni configurazione il file di output sarà rinominato in questa maniera:

CENSUS_WINDOWS.ALGORITM_GRID.SIZE_BOXFILTERING.P1.P2.A.SGM_ALGORIT.MASK.csv

Ovvero è rappresentato dai parametri attuali presenti in ogni esecuzione del programma.

Capitolo 5 - Risultati Sperimentali

Per valutare l'efficacia dei diversi algoritmi per ogni configurazione applicata ad essi, viene calcolato l'errore medio percentuale per poi essere confrontato con gli altri risultati ottenuti.

5.1 Calcolo dell'errore medio

Alla fine di ogni esecuzione avremo tanti file, generati dal programma, uno per ogni configurazione stabilita.

Per ogni file viene aggiunta una nuova colonna che rappresenta la moltiplicazione tra il contatore dei pixel reali e l'indice di errore per quella determinata immagine.

00000_10	9.37	88975	833695.8
00001_10	6.38	107175	683776.5
00002_10	13.87	100480	1393658
00003_10	13.18	94608	1246933
00004_10	20.41	97007	1979913
00005_10	15.42	92921	1432842
00006_10	21.85	111326	2432473
00007_10	5.31	109354	580669.7
00008_10	4.65	101836	473537.4
00009_10	5.32	95997	510704

Figura 5.1 File di output - Calcolo Media Ponderata

Questa colonna viene aggiunta per il calcolo dell'errore medio totale ovvero la media ponderata dei valori:

$$\frac{\sum_0^{199} COUNTER\_PIXEL\_REAL \ * ERROR}{\sum_0^{199} COUNTER\_PIXEL\_REAL}$$

5.2 Risultati iniziali

Inizialmente sono stati eseguiti test considerando tutte le configurazioni degli algoritmi di costo locale, senza considerare la media del box-filtering, per poi utilizzare l'algoritmo semi-globale SGM classico.

Questi algoritmi sono stati eseguiti facendo variare la maschera di SGM, la dimensione della finestra della finestra di aggregazione.

Per quanto riguarda la maschera di SGM, che identifica le direzioni utilizzate, sono stati considerati i seguenti valori:

- Valore 255: dove consideriamo tutti e gli otto percorsi di SGM;
- Valore 149: dove consideriamo solo i quattro percorsi superiori;

Invece la dimensione della finestra del box-filtering è stata fatta variare considerando i valori 1,3 e 5.

Mentre, le due penalità P1 e P2 sono tenute fisse per tutte le esecuzioni:

- P1 = 2;
- P2 = 30;

Considerando queste configurazioni sono stati ottenuti i valori sotto riportati, raggruppati per l'algoritmo dei costi locali:

- ***Trasformata Binaria con griglia completa***

Configurazione Algoritmo	*Errore medio*
Dimensione box-filtering 1 maschera SGM 149	13,19
Dimensione box-filtering 1 maschera SGM 255	10,18
Dimensione box-filtering 3 maschera SGM 149	11,23
Dimensione box-filtering 3 maschera SGM 255	8,75
Dimensione box-filtering 5 maschera SGM 149	10,39
Dimensione box-filtering 5 maschera SGM 255	8,03

- ***Trasformata Binaria con griglia dispari***

Configurazione Algoritmo	*Errore medio*
Dimensione box-filtering 1 maschera SGM 149	13,85
Dimensione box-filtering 1 maschera SGM 255	8,98
Dimensione box-filtering 3 maschera SGM 149	12,23
Dimensione box-filtering 3 maschera SGM 255	7,85
Dimensione box-filtering 5 maschera SGM 149	11,91
Dimensione box-filtering 5 maschera SGM 255	7,39

- ***Trasformata Binaria con griglia pari***

Configurazione Algoritmo	*Errore medio*
Dimensione box-filtering 1 maschera SGM 149	13,69
Dimensione box-filtering 1 maschera SGM 255	8,96
Dimensione box-filtering 3 maschera SGM 149	11,97
Dimensione box-filtering 3 maschera SGM 255	7,79
Dimensione box-filtering 5 maschera SGM 149	11,66
Dimensione box-filtering 5 maschera SGM 255	7,34

- ***Trasformata Ternaria con griglia completa***

Configurazione Algoritmo	*Errore medio*
Dimensione box-filtering 1 maschera SGM 149	12,61
Dimensione box-filtering 1 maschera SGM 255	10,16
Dimensione box-filtering 3 maschera SGM 149	10,59
Dimensione box-filtering 3 maschera SGM 255	8,61
Dimensione box-filtering 5 maschera SGM 149	9,69
Dimensione box-filtering 5 maschera SGM 255	7,77

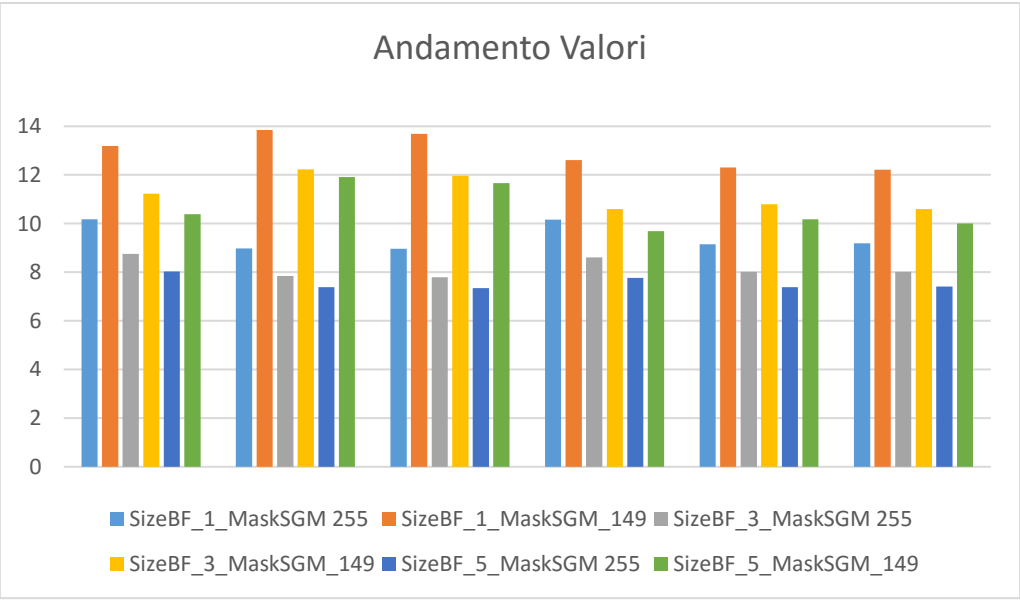
- ***Trasformata Ternaria con griglia dispari***

Configurazione Algoritmo	*Errore medio*
Dimensione box-filtering 1 maschera SGM 149	12,31
Dimensione box-filtering 1 maschera SGM 255	9,15
Dimensione box-filtering 3 maschera SGM 149	10,80
Dimensione box-filtering 3 maschera SGM 255	8,02
Dimensione box-filtering 5 maschera SGM 149	10,18
Dimensione box-filtering 5 maschera SGM 255	7,39

- ***Trasformata Ternaria con griglia pari***

Configurazione Algoritmo	*Errore medio*
Dimensione box-filtering 1 maschera SGM 149	12,22
Dimensione box-filtering 1 maschera SGM 255	9,19
Dimensione box-filtering 3 maschera SGM 149	10,60
Dimensione box-filtering 3 maschera SGM 255	8,01
Dimensione box-filtering 5 maschera SGM 149	10,01
Dimensione box-filtering 5 maschera SGM 255	7,41

Dai risultati ottenuti si può ricavare il seguente grafico il quale mostra l'andamento dei valori:



Si può notare dal grafico ottenuto che per tutte le trasformate applicate abbiamo che i valori più bassi si hanno considerando due tipi di configurazioni:

1. Dimensione del box-filtering 5 griglia e maschera di SGM 255.
2. Dimensione del box-filtering 3 griglia pari e maschera di SGM 255.

Per entrambe, il valore migliore è ottenuto applicando la trasformata Census con griglia pari e rispettivamente per la prima è pari a ***7,34*** e per la seconda è ***7,79***.

5.3 Risultati con media del Box-filtering

Dopo aver trovato le due configurazioni dell'algoritmo ottimale, cioè quelli che permettono di avere un errore medio basso ed accettabile, si è deciso di applicare a questi l'algoritmo di media puntuale all'aggregazione dei costi locale (box-filtering).

L'applicazione di questo algoritmo ha permesso di ricavare queste variazioni dei valori precedenti:

	Valore senza media	*Valore con media*
Trasformata Binaria pari con dimensione box-filtering 5 e maschera SGM 255	7,34	7,40
Trasformata Binaria pari con dimensione box-filtering 3 e maschera SGM 255	7,79	7,86

Dalla tabella sovrastante mostra dei risultati ottimali poiché applicando l'algoritmo di media puntuale il valore viene incrementato solo del 6/7% e questo è un importante beneficio in quanto, come detto nella parte riservata al box-filtering, ci permette di risparmiare dei bit e in hardware è molto favorevole.

5.4 Risultati con varianti di SGM

Un'ulteriore test effettuato al set di immagini, è stato applicare non più l'algoritmo semiglobale classico ma le sue varianti. Vale a dire:

- eSGM
- eSGM_ten_winners
- eSGM_ten_winners_2_scans
- eSGM_sums

I suddetti algoritmi sono stati applicati alla configurazione migliore trovata in precedenza cioè utilizzando la trasformata Census con griglia pari applicando un'aggregazione dei costi locale (box-filtering) con dimensione 5 e tutti le otto direzioni (mask = 255).

I risultati prodotti dagli algoritmi, considerando la configurazione migliore, sono rappresentati dalla seguente tabella:

Algoritmo utilizzato	*Errore medio*
eSGM	7,41
eSGM_ten_winners	7.42
eSGM_ten_winners_2_scans	14,91
eSGM_sums	9,88

Considerando la tabella ottenuta, si può notare che le ultime due varianti, eSGM a doppia passata e eSGM_sums non affidabili allo stato attuale poiché aumentano il valore dell'errore. Mentre sono accettabili i valori di eSGM classico e eSGM_ten_winners.

Dopo aver riscontrato i seguenti risultati, come in precedenza, applichiamo (prima dei seguenti algoritmi) la media locale all'aggregazione dei costi locali, ottenendo:

Algoritmo utilizzato	*Errore medio*
eSGM	7,48
eSGM_ten_winners	7.49
eSGM_ten_winners_2_scans	14,69
eSGM_sums	9,98

Dall'analisi dei risultati ottenuti si evince che tentare di perfezionare un algoritmo stereo già estremamente efficace sia un compito alquanto arduo, e nel farlo si rischia di peggiorare le prestazioni dell'algoritmo di partenza.

Tuttavia, ciò non significa che si tratti di un compito impossibile e lo abbiamo dimostrato, in particolare con due dei quattro approcci proposti, ottenendo risultati decisamente positivi.

Capitolo 6 - Conclusioni

Il lavoro di tesi ha permesso di costruire una pipeline completa di visione stereo software finalizzata all'implementazione su hardware FPGA. Per la realizzazione ho dovuto studiare e analizzare il mondo della visione stereo con tutte le sue sfaccettature, che mi ha permesso di apprezzare questi concetti a me sconosciuti.

Per quanto concerne i risultati ottenuti, come già sottolineato in precedenza, hanno permesso di ricavare benefici per l'implementazione successiva in hardware, come l'applicazione della media puntuale all'aggregazione dei costi locali, la quale permette di diminuire il numero di bit e di conseguenza ottimizzare le risorse, e l'applicazione delle varianti di SGM che permettono a loro volta di essere più "leggeri" in termini di memoria.

Gli sviluppi immediatamente futuri sulla pipeline software saranno:

- Continuare a testare i vari algoritmi con altre differenti tipologie di configurazioni dei parametri fino ad' ora non considerate, per ottenere risultati migliori e più interessanti;
- Utilizzare la pipeline realizzata in immagini non perfettamente rettificate valutando quanto possono variare i risultati sopra analizzati. Quest' ultimo punto è stato affrontato dalla tesi di Luca Buratti *"Valutazione sperimentale di metodologie di rettificazione e impatto su algoritmo di stereo visione"*.

Bibliografia

[1] Tesi di Albertazzi Riccardo "Sistema di visione Stereo su architettura ZYNQ".

[2] S. Mattoccia "Stereo Vision: Algorithms and Applications", 2011.

[3] H. Hirschmueller, "Stereo Processy by Semi-Global Matching and Mutual Information", IEEPAMI 2008.

[4] Tesi di Nigro Simone "Metodologie di Machine Learning applicate alla stereo visione".

[5] H. Hirschmueller and M. Burder and I. Ernst "Memory Efficient Semi-Global matching".

[6] Tesi di Pietrosanti Stefano "Riconoscimento e localizzazione di oggetti mediante visione stereoscopica".

[7] *"Stereo and 3D vision"* *https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect16.pdf*

[8] L. Di Stefano "Computer Vision Laboratory (CVLAB) DEIS/ARCES - University of Bologna"

Appendice: Risultati Esaustivi

In seguito allo sviluppo della tesi e alla realizzazione della pipeline sono stati effettuati dal Dottor Matteo Poggi ulteriori test facendo variare le dimensioni della finestra dell'aggregazione dei costi locale (*box-filtering*).

Nella tesi i valori della finestra da me considerati erano 3 e 5, mentre in questi test sono stati assunti come valori:

- 7
- 9
- 11
- 13

Otteniamo quindi i seguenti risultati:

Algoritmo applicato	*Dimensione*	*Errore senza media*	*Errore con media*
Binaria completa SGM classico	7 x 7	7.58	7.60
Ternaria completa SGM classico	7 x 7	7.28	7.32
Binaria completa SGM classico	9 x 9	7.37	7.39
Ternaria completa SGM classico	9 x 9	7.01	7.04
Binaria completa SGM classico	11 x 11	7.33	7.33
Ternaria completa SGM classico	11 x 11	6.90	6.92
Binaria completa SGM classico	13 x 13	7.33	7.33
Ternaria completa SGM classico	13 x 13	6.90	6.92

Da cui si può notare che la configurazione migliore si ottiene utilizzando la dimensione della finestra 13 x 13 applicando la trasformata ternaria a griglia completa. Eseguire la media dei costi introduce un errore quasi trascurabile.

Ricavata la configurazione migliore, con dimensione della finestra di aggregazione 13 x 13, sono stati ricavati i seguenti valori utilizzando:

- Trasformate a griglia pari e dispari

Algoritmo applicato	*Errore senza media*	*Errore con media*
Binaria completa SGM classico	7.33	7.33
Binaria griglia dispari SGM classico	7.45	7.56
Binaria griglia pari SGM classico	7.36	7.47
Ternaria completa SGM classico	6.90	6.92
Ternaria griglia dispari SGM classico	6.73	6.80
Ternaria griglia pari SGM classico	**6.72**	**6.76**

- Algoritmo semi-globale eSGM:

Algoritmo applicato	*Errore senza media*	*Errore con media*
Binaria completa eSGM	7.39	7.93
Binaria griglia dispari eSGM	7.37	7.51
Binaria griglia pari eSGM	7.26	7.40
Ternaria completa eSGM	6.94	6.97
Ternaria griglia dispari eSGM	6.77	6.84
Ternaria griglia pari eSGM classico	**6.76**	**6.81**

Le due tabelle sopra mostrano come i valori migliori sono stati ottenuti con la trasformata ternaria a griglia pari. Il valore ottimo si ha considerando SGM classico ma, come discusso nella tesi, applicando eSGM abbiamo benefici risparmiando in termini di memoria e risorse.

- Algoritmo SGM con variazione di maschere:

Algoritmo applicato	*Errore senza media*	*Errore con media*
Binaria completa SGM 106	12.88	12.55
Binaria griglia dispari SGM 106	18.82	18.52
Binaria griglia pari SGM 106	18.31	18.10
Ternaria completa SGM 106	12.27	11.96
Ternaria griglia dispari SGM 106	12.97	12.69
Ternaria griglia pari SGM 106	12.53	12.26
Binaria completa SGM 149	10.37	10.57
Binaria griglia dispari SGM 149	14.53	15.32
Binaria griglia pari SGM 149	13.96	14.78
Ternaria completa SGM 149	**9.37**	**9.57**
Ternaria griglia dispari SGM 149	10.62	10.97
Ternaria griglia pari SGM 149	10.28	10.60

Quest'ultima tabella mostra come i risultati peggiorino se all'algoritmo SGM vengano applicate maschere differenti.