UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXVII Ciclo

SVILUPPO DI UN SISTEMA EMBEDDED INTELLIGENTE PER LA VISIONE STEREOSCOPICA

Coordinatore:

Chiar.mo Prof. Marco Locatelli

Tutor:

Chiar.mo Prof. Alberto Broggi

Dottorando: *Gabriele Camellini*

Gennaio 2015

A Eleonora,
ai miei genitori
(e al futuro "pupo")
che mi hanno sempre sostenuto
sia nei momenti felici,
che in quelli difficili.

Sommario

Elenco delle figure

Elenco delle tabelle

Introduzione

Oggigiorno, in molti campi della vita moderna, dall'agricoltura, all'industria, al terziario i sistemi di visione artificiale si stanno largamente diffondendo per compiti di sorveglianza, automazione e sicurezza. Sistemi di telecamere per videosorveglianza e controllo qualità esistono ormai da qualche decina di anni, ma la tendenza dell'ultimo decennio è quella di rendere questi sistemi sempre più intelligenti e autonomi, in modo da svolgere funzioni di alto livello senza la necessità di una supervisione da parte di un operatore umano.
I sistemi di visione artificiale intelligenti, vengono utilizzati in varie situazioni quali:

- autoveicoli autonomi o sistemi di supporto alla guida;
- controllo qualità nelle linee di produzione;
- veicoli autonomi in ambiente strutturato, come per esempio nella movimentazioni di pezzi e merci in campo industriale o nella grandi catene di distribuzione logistica;
- robotica: i robot autonomi sono da sempre al centro della ricerca scientifica che li studia per supportare o sostituire l'uomo in compiti pericolosi o comunque faticosi. Non bisogna poi dimenticare come la robotica, negli ultimi anni sia diventata alla portata degli hobbisti, facendo aumentare la richiesta di sistemi di percezione dell'ambiente ad elevate prestazioni e bassi costi

- UAV (Unmanned Aerial Vehicle): aereo veicoli comandati da remoto, con anche capacità di volo in autonomia, usati principalmente per operazioni di sorveglianza e monitoraggio.
- videosorveglianza e controllo di accessi: in questo campo i nuovi sistemi di visione permettono di andare a sostituire vecchie tipologie di sensori, offrendo migliori prestazioni e costi più bassi.

La visione stereoscopica e tridimensionale

Una delle necessità basilari nei campi robotico, automotive e industriale è la ricostruzione tridimensionale dell'ambiente, perché questo permette una navigazione autonoma in differenti scenari, il riconoscimento e la classificazione di oggetti oltre ad una visualizzazione dell'ambiente attraverso la realtà aumentata.
Tra gli obbiettivi, che si pone la ricerca nel campo della visione artificiale, negli ultimi anni, vi è quindi lo sviluppo di nuovi algoritmi che permettano una migliore percezione dell'ambiente circostante, ripreso attraverso una o più videocamere.
L'elaborazione intelligente di dati provenienti da due telecamere alloggiate su uno stesso supporto (visione stereoscopica) permette di ricostruire la tridimensionalità dell'ambiente inquadrato, attraverso una mappa di disparità densa che descrive per i punti dell'immagine la loro posizione all'interno dello spazio tridimensionale, rispetto al punto di osservazione.
I più moderni algoritmi per la visione stereo (come il Semi Global Matching) generano una mappa di disparità densa per ogni punto della scena osservata, a differenza delle precedenti tecniche che restituivano informazione solo in punti sparsi della scena. In modo analogo anche il flusso ottico denso permette di ricavare informazioni sugli spostamenti dei singoli elementi all'interno della scena ripresa.

Stato dell'arte

I sistemi di stereo-visione, basati su mappa di disparità, sono stati ampiamente oggetto di ricerca [1, 2, 3, 4, 5] e di analisi di prestazioni [6, 17, 15]. I vari algoritmi

offrono diversi compromessi in termini di complessità computazionale, qualità della ricostruzione tridimensionale e robustezza contro immagini sorgenti rumorose.

In particolare, il cosiddetto algoritmo Semi-Global Matching, presentato per la prima volta in [7], in accopiamento ad una trasformata di Census come metrica[Hirschmuller:2009], ha dimostrato di essere una valida soluzione adottata in molte applicazioni reali[VIAC], in quanto insensibile alle variazioni di illuminazione e robusto in presenza di aree con scarsa testurizzazione pur essendo in grado di gestire correttamente grandi discontinuità di profondità, come quelle prodotte da piccoli oggetti e pali. Queste capacità derivano dal fatto che l'approccio SGM ottimizza i costi di corrispondenza sull'intera area dell'immagine aggiungendo un termine regolarizzazione dei valori calcolati prima della minimizzazione.

La fase di ottimizzazione è computazionalmente molto onerosa in quanto richiede un elevato numero di accessi alla memoria per leggere i costi, rendendo molto impegnative le implementazioni in tempo reale; tuttavia, la sua natura massivamente parallela ben si adatta ad una vasta gamma di moderni dispositivi hardware. In particolare, le soluzioni basate su PC possono raggiungere 20 fps con risoluzione VGA su una moderna CPU desktop standard [25] sfruttando sia l'architettura multicore sia le capacità di elaborazione SIMD [1]. Esistono anche implementazioni su GPU che raggiungono più di 60 fps a risoluzione VGA [27]; tuttavia, queste soluzioni richiededono hardware professinale e costoso, con consumi energetici più che raddoppiati.

Il difetto fondamentale di tutte queste soluzioni è il non essere adatte a sistemi hardware embedded; una valida alternativa è l'uso di unità FPGA, che ben si adattano all'elevata esigenza di risorse computazionali massivamente parlallele dell'algoritmo SGM [28, 29]. Le FPGA si caratterizzano per basso costo e basso consumo energetico, elevata affidabilità e adatte anche ad ambienti difficili (ad esempio automobilistico), ma richiedono significativi sforzi di progettazione con tempi di sviluppo notevolmente più lunghi.

[1] Single Instruction Multiple Data

Definizione del problema

Gli algoritmi per la visione stereoscopica densa, dato il notevole carico computazionale, normalmente richiedono un elaboratore potente (generalmente un PC) sul quale venire eseguiti. Sui moderni microprocessori multi-core, un'elaborazione in tempo reale richiede l'utilizzo della maggior parte della capacità computazionale, a scapito del tempo per le elaborazioni di più alto livello volte alla comprensione del mondo circostante.

Il problema che si è affrontato, consiste nell'individuare un algoritmo affidabile che consenta di effettuare una ricostruzione stereoscopica di buona qualità e, successivamente, progettare un'opportuna architettura embedded in grado di integrare al suo interno tutto il processo di visione stereoscopica, in modo che possa essere utilizzata a supporto di sistemi più complessi, per sollevarli dai compiti di basso livello e computazionalmente pesanti; la stessa architettura può essere utilizzata in autonomia per semplici funzioni di controllo basate sulla visione stereoscopica.

Soluzioni e contributi

In questo progetto di ricerca, inizialmente, si è affrontato lo studio di vari algoritmi per valutarne la qualità dell'informazione tridimensionale ricostruita e per proporre nuovi miglioramenti che aumentino il grado di affidabilità nelle varie situazioni operative. Il passo successivo, che rappresenta il tema principale della ricerca, è stato lo studio e l'ingegnerizzazione di un sistema di visione stereoscopica su architetture a basso costo, che consentono al contempo di raggiungere elevate prestazioni, affidabilità e bassi consumi. Tra le varie di architetture disponibili si è scelto un sistema ibrido (SoC: system on chip) che integra una architettura FPGA e un microprocessore. Il risultato ottenuto è un sistema embedded a basso costo e bassi consumi che permette una ricostruzione densa della scena inquadrata ad una velocità di 27.5 fps a risoluzione VGA (640 x 480 pixel).
Il sistema sviluppato in questo progetto risulta di duplice utilità: in primo luogo fornisce ad un elaboratore tradizionale l'informazione di disparità già elaborata, sollevan-

dolo dalle onerose elaborazioni di basso livello e permettendo di concentrarsi sulla comprensione dell'ambiente, (per esempio attraverso tecniche di intelligenza artificiale); il secondo ambito applicativo prevede l'elaborazione, direttamente a bordo del sistema stesso, dell'informazione di disparità per compiti di sicurezza, sorveglianza e automazione.

Struttura della tesi

Il capitolo 1 offre una panoramica sulle teoria alla base della visione stereoscopica, e descivere gli alogortimi di ricostruzione trimensionale analizzati di questo dottorato di ricerca. Nel capitolo 2 viene presentato il lavoro di valutazione dell'accuratezza ottenibile con diversi algoritmi allo stato dell'arte. Il successivo capitolo 3 presenta una panoramica e quale dettaglio sull'hardware scelto e progettato per questa architettura. L'architettura del sistema di elaborazione e la sua implementazione vengono descritti nel capitolo 4, dove vengono dettagliatamente analizzati i diversi blocchi funzionali. Infine nel capitolo e conclusioni vengono analizzate le prestazioni del sistema studiato e sviluppato oltre alle considerazioni su possibili sviluppi futuri.

Capitolo 1

Teoria e Algoritmi di visione stereoscopica

Inizialmente in questo capitolo viene descritta la teoria che sta alla base della visione stereoscopica partendo dal modello della telecamera pinhole. Successivamente si introduce la geometria epipolare, che permette di semplificare il problema della visione stereoscopica.

1.1 Teoria della visione stereoscopica

In questa sezione vengono descritti gli aspetti teorici relativi alla visione stereoscopica. Si inizia descrivendo il processo di *formazione dell'immagine* parlando del modello della *telecamera pinhole* e della *trasformazione prospettica*, per passare successivamente al problema della ricostruzione dell'informazione tridimensionale attraverso la triangolazione delle immagini riprese da due telecamere. In particolare si introducono i concetti fondamentali di geometria epipolare.

1.1.1 Formazione dell'immagine e telecamera pinhole

La *formazione dell'immagine* è il processo che mette in relazione una scena del mondo tridimensionale con l'immagine prodotta sul sensore della telecamera che inqua-

dra tale scena. Questo processo è regolato sia dalla relazione geometrica fra un punto tridimensionale della scena ed un punto bidimensionale dell'immagine, sia da cosa è determinata l'intensità di tale punto. Concentrando l'attenzione sul primo punto, l'osservazione di uno spazio 3D attraverso una telecamera ha come diretta conseguenza la riduzione dell'informazione ad uno spazio 2D (si ha una perdita di informazione). Questo processo è definito *proiezione prospettica* e il modello più semplice che lo descrive è conosciuto come modello *pinhole* della telecamera (ovvero foro di spillo) (Fig. 1.1). Questa telecamera è costituita da una scatola con un foro (teoricamente infinitamente piccolo) su un lato, e il piano sensibile su quello opposto. La luce della scena attraversando il foro forma sul piano sensibile un'immagine capovolta della scena inquadrata.

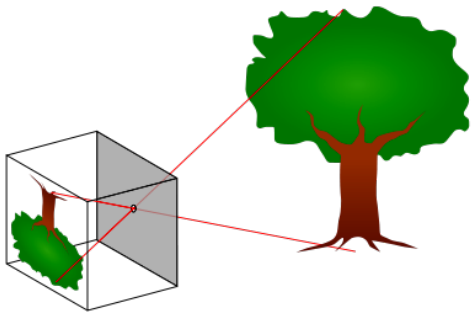


Figura 1.1: Modello pinhole della telecamera.

Nel modello prospettico, raffigurato in Fig. 1.2, si definisce:

- P_W : *punto della scena*; P_I : *punto immagine*;
- $\mathscr{R}$: *piano immagine*; $\mathscr{F}$: *piano focale*;
- O_C : il *centro ottico* (ovvero il foro della telecamera pinhole);
- f : la *lunghezza focale* ovvero la distanza tra I e il punto O_C;
- *l'asse ottico* : quell'asse normale al piano immagine e passante per O_C;

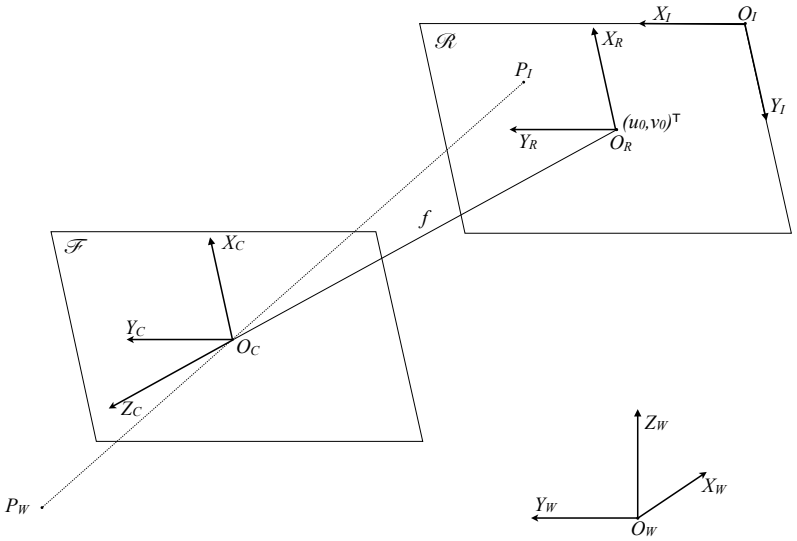


Figura 1.2: Modello Pinhole. $\{O_W\}$ = sistema di riferimento del mondo, $\{O_C\}$ = sistema di riferimento della telecamera, $\{O_R\}$ = sistema di riferimento del piano immagine, $\{O_I\}$ = sistema di riferimento dell'immagine (assi $(X_I, Y_I) \equiv (u,v)$). $\mathscr{F}$ è il piano focale e $\mathscr{R}$ il piano immagine.

- *principal point* : il punto di intersezione dell'asse ottico e con il piano immagine.

Per descrivere analiticamente il modello prospettico della telecamera, si definiscono due sistemi di riferimento. Un primo sistema di riferimento della telecamera $\{X_C, Y_C, Z_C\}$, centrato nel centro ottico O_C con l'asse Z coincidente con l'asse ottico, per rappresentare i punti 3D del monto osservati dalla telecamera. Il secondo sistema di riferimento $\{u,v\}$ è posizionato sul piano immagine con gli assi u e v orientati come X_I e Y_I rispettivamente.

Preso un punto della scena di coordinate 3D $P = P_W = [X,Y,Z]^\top$ e il suo punto sul piano immagine di coordinate $p = P_I = [u,v]^\top$, le equazioni (non lineari) che legano tali coordinate attraverso la proiezione prospettica, sono date da:

$$u = \frac{-f}{Z}X \qquad v = \frac{-f}{Z}Y \tag{1.1}$$

Il segno meno rappresenta la riflessione dell'immagine attraverso il centro ottico O_C e il termine $1/Z$ introduce la componente non lineare del modello. Senza perdere di correttezza, il segno meno può venir omesso considerando un piano immagine posto dall'altro lato, parallelo al piano $\mathscr{R}$ esistente e ad egual distanza f da O_C.
Come si può intuire, la formazione dell'immagine non è una corrispondenza biunivoca: tutti gli infiniti punti mondo su una semiretta passante per il centro ottico O_C, vengono mappati nel medesimo punto immagine. L'informazione che si perde, è la distanza e quindi con una solo immagine non è possibile ricostruire la tridimensionalità della scena.

Il sistema non lineare 1.1 può essere rappresentato con coordinate omogenee per trasformarlo in un sistemare lineare come mostrato in 1.2 . Quindi dati i punti $\tilde{P} = [X,Y,Z,1]^\top$ e $\tilde{p} = [u,v,1]^\top$ il modello di proiezione prospettica diventa:

$$Z\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} -fX \\ -fY \\ Z \end{bmatrix} = \begin{bmatrix} -f & 0 & 0 & 0 \\ 0 & -f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ X \\ Z \\ 1 \end{bmatrix} \tag{1.2}$$

rappresentato in forma matriciale come

$$\tilde{p} \approx M\tilde{P} \tag{1.3}$$

L'equazione 1.3 è definita a meno di un fattore di scala Z e rappresenta la matrice di proiezione prospettica della telecamera.

1.1.2 Triangolazione con due telecamere

Per ovviare alla perdita di informazione tridimensionale della trasformazione prospettica, è possibile effettuare una triangolazione del punto P osservato da due differenti telecamere per determinare la sua distanza. Possiamo quindi ricostruire una scena tridimensionale, ma dobbiamo risolvere il *problema della corrispondenza* o *stereo matching* (*problema dell'accoppiamento*).

Dato un punto p_L sull'immagine della telecamera sinistra, proiezione del punto che vogliamo localizzare nella scena 3D (che può essere posizionato su qualunque

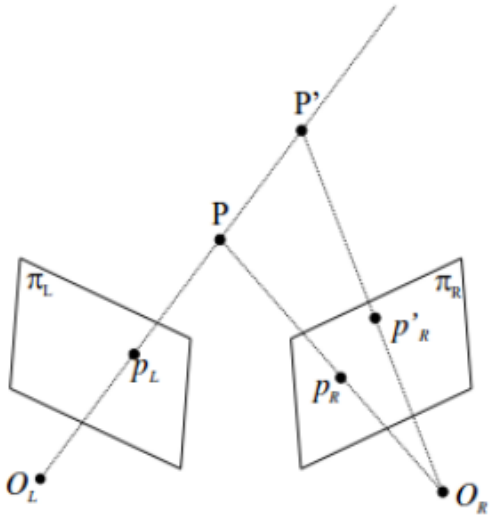


Figura 1.3: Problema della corrispondenza o stereo matching.

punto dell'asse che congiunge O_L a p_L). Se si riesce a individuare anche nell'immagine di destra il punto da localizzare, si può tracciare una linea da O_R a p_R e, incrociandola con la precedente, si trova la reale posizione nello spazio 3D. Trovare il punto omologo di p_L è ciò che chiamiamo *stereo matching*.
Nella visione stereoscopia, solitamente si fa riferimento a due modelli: il *sistema stereo laterale* e quello a *geometria epipolare*.
Il *sistema stereo laterale* o modello *standard* prevede due telecamere con stessa focale, con gli assi (x,y,z) paralleli e piani immagine complanari, come mostrato in figura 1.4. I due sistemi di riferimento risultano quindi solidali a meno di una traslazione orizzontale, che viene definita *baseline b* del sistema stereoscopico 1.5.

$$x_L - x_R = b \qquad y_L = y_R \qquad z_L = z_R \tag{1.4}$$

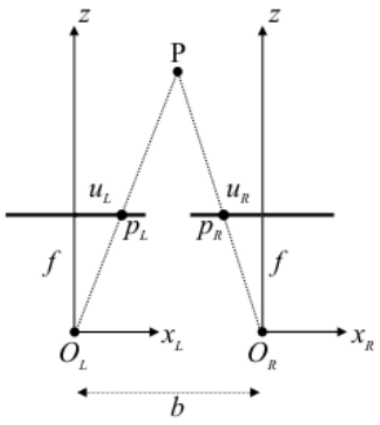


Figura 1.4: Triangolazione di un punto mondo con due telecamere.

I riferimenti sui piani immagine, sono quindi legati dalle seguenti relazioni

$$v_L = v_R = y \cdot \frac{f}{z} \qquad u_L = x_L \cdot \frac{f}{z} \qquad u_R = x_R \cdot \frac{f}{z} \qquad u_L - u_R = b \cdot \frac{f}{z} \tag{1.5}$$

Si definisce la *disparità d* come $d = b * \frac{f}{z}$, che rappresenta lo scostamento orizzontale dei punti immagine relativi ad un punto della scena. Questa quantità dipende dalla profondità o distanza del punto mondo, e quindi permette di ricostruire l'informazione di trimensionalità.

In una configurazione reale di due telecamere non sussistono le ipotesi del modello standard, quindi bisogna considerare il modello della *geometria epipolare* (Fig. 1.5).

Lo spazio di ricerca del problema è sempre 1D, come nel precedente caso (infatti cerchiamo sempre su una retta il punto omologo). Il punto p_L del piano immagine della telecamera di sinistra corrisponde ad una linea sul piano immagine dell'altra. Questa linea viene detta *linea epipolare*. Tutte le linee epipolari passano per un punto che chiamiamo *epipolo* e che è dato dal punto della congiungente i due centri ottici che interseca il piano immagine (idealmente è la proiezione del centro ottico dell'altra telecamera).

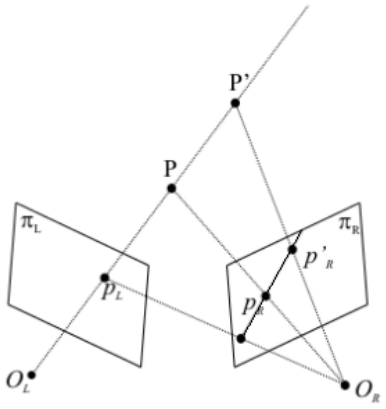


Figura 1.5: Geometria epipolare.

La ricerca è più complessa rispetto allo spazio standard, in quanto la linea epipolare giace su uno spazio bidimensionale. Volendo ricondursi al modello *standard*, a ciascuna delle immagini ottenute va applicata una trasformazione (omografia) detta *rettificazione*. Questa consiste nell'ottenere rette epipolari coniugate orizzontali e collineari. Per ottenere le omografie, che danno la rettificazione, è necessaria la calibrazione delle telecamere (stima dei parametri estrinseci ed intrinseci). La calibrazione fornisce inoltre i parametri b, f e $pixelsize$ necessari alla ricostruzione della scena 3D.

In sintesi, una volta determinate le corrispondenze tra i punti delle due immagini che rappresentano le due proiezioni del medesimo punto della scena reale **P** (elementi corrispondenti), si può determinare la disparità, da cui la profondità di **P**. Come realizzare la corrispondenza?
La geometria epipolare ci semplifica il problema limitando la ricerca di un punto giacente su una determinata linea epipolare ai punti appartenenti alla linea epipolare ad essa coniugata. La rettificazione ci aiuta ulteriormente: dato un punto di coordinate (x,y) sull'immagine di riferimento, il punto ad esso omologo si trova alla medesima coordinata y.

La relazione fra distanza degli oggetti dalle telecamere e disparità degli stessi

sui piani immagine è inversamente proporzionale. Tanto più gli oggetti sono distanti, tanto minore sarà la loro disparità.

Il problema della corrispondenza deve tener conto di un aspetto importante: le *occlusioni*. In ciascuna delle due immagini sono presenti regioni non visibili nell'altra perché occluse da oggetti più vicini. Un algoritmo di matching dovrebbe anche individuare i punti dell'immagine per i quali, causa occlusione, non è possibile risolvere la corrispondenza.

Prendendo come riferimento una delle due immagini, ad esempio quella di sinistra, è possibile costruire la cosiddetta *mappa delle disparità* nella quale ogni pixel rappresenta il valore della sua *disparità* (che come si è visto dipende dalla distanza); in Fig. 1.6 tali valori sono codificati in scala di grigio.

(a) (b)

Figura 1.6: Esempio di una mappa di disparità (b) relativa alla scena (a).

1.2 Algoritmi

La soluzione del problema della visione stereoscopica consiste nel trovare punti corrispondenti di una stessa scena inquadrati da due telecamere differenti. La differenza di coordinate immagine tra i punti corrispondenti, viene detta *disparità* ed è strettamente legata alla posizione dei punti nello spazio tridimensionale della scena, in

particolare è inversamente proporzionale alla distanza. Prendendo a prestito il termine inglese, gli algoritmi che risolvono tale problema vengono detti *algoritmi di stereo matching*; lo sviluppo di tali algoritmi è stato ed è tuttora uno dei filoni di ricerca più attivi nel campo della visione artificiale (o computer vision).
Questi algoritmi possono essere suddivisi in due macro-categorie:

- ***feature-based***: il *matching* viene cercato tra determinate *feature* estratte dall'immagine Fig. 1.7. La mappa di disparità risultante da tali algoritmi è sparsa in quanto vengono analizzati solo certi punti dell'immagine. Il loro punto di forza, consiste nell'essere robusti e computazionalmente veloci. La prima fase del processo di elaborazione consiste nell'estrazione delle *feature*; questa esigenza è comune a vari campi della visione artificiale, quindi il suo risultato può essere condiviso con altri algoritmi usati nel processo di visione, come per esempio i classificatori.

- ***area-based***: il *matching* è effettuato per tutti i pixel dell'immagine di riferimento. Le mappe di disparità risultano quindi dense restituendo un valore per ogni pixel (a meno di aree occluse o ambigue per la mancanza di texture). Il risultato che si ottiene deve essere opportunamente valutato in quanto non è detto sia altrettanto robusto quando quello degli algoritmi *feature-based*. Sicuramente questi algoritmi risultano computazionalmente molto onerosi in quando durante la fase di *matching* vengono presi in considerazione tutti i pixel dell'immagine di riferimento e confrontati con tutti i possibili punti omologhi dell'immagine di confronto.

Focalizzandosi sulla seconda categoria, negli ultimi due decenni, grazie anche alla disponibilità di piattaforme hardware sempre più potenti, molti algoritmi di stereo matching denso sono stati al centro della ricerca nel campo della visione artificiale [1, 2, 3, 4, 5]. Un riassunto e una valutazione di tali algoritmi è stata presentata da Scharstein e Szeliski [6]. In questo interessante lavoro, gli algoritmi di stereo matching densi sono suddivisi in due categorie principali: *algoritmi locali* (local area based methods) e *algoritmi globali* (global optimization based methods). A queste

Figura 1.7: Algoritmo di matching feature-based.

due macro-categorie, se ne aggiunge anche una "ibrida" che raccoglie i cosiddetti *algoritmi semi-globali*. Le caratteristiche principali di queste categorie sono:

- ***algoritmi locali***: la disparità assegnata ad un pixel dipende solo da informazioni dedotte da pixel spazialmente vicini a quello considerato. Generalmente il criterio di matching è costituito dalla similiarità fra finestre (di dimensione prefissata o adattive) centrate nei pixel considerati. Veloci e meno accurati.

- ***algoritmi globali***: la disparità assegnata ad un pixel dipende da informazioni dedotte da tutta l'immagine. Generalmente il problema viene impostato come un problema di minimizzazione di una funzione energia. Computazionalmente onerosi, ma molto accurati.

- ***algoritmi semi-globali***: hanno la stessa impostazione dei globali, ma utilizzando un sotto-insieme dell'intera immagine (es. analizzando la funzione energia solo lungo certi percorsi) e rappresentano un'interessante compromesso fra velocità e accuratezza.

Nel lavoro di classificazione di Scharstein e Szeliski [6], inoltre vengono identificati quattro blocchi funzionali ricorrenti nella maggior parte degli algoritmi stereo area-based densi:

- calcolo delle metriche di accoppiamento delle corrispondenze, dette costi delle corrispondenze,

- strategia di aggregazione dei costi,
- calcolo della disparità,
- raffinamento della disparità.

I suddetti blocchi funzionali verranno analizzati dettagliatamente per l'algoritmo prescelto in questo progetto di ricerca.

In questa sezione sono descritti due differenti algoritmi di ricostruzione stereoscopica. Il primo denominato Semi-Global Matching (abbreviato SGM) è stato pubblicato per la prima volta nel lavoro di [7], e viene descritto molto dettagliatamente in quanto è stato poi effettivamente scelto nell'implementazione del sistema embedded, sebbene utilizzando metriche differenti per la definizione dei costi di inizializzazione. Il secondo algoritmo denominato ELAS [4], utilizza un approccio ibrido: viene eseguita una prima ricerca di corrispondenze tra feature sparse nelle immagini sinistra e destra, successivamente questi valori di disparità sono usati per restringere il range di ricerca di un algoritmo area-base locale basato su finestre di corrispondenza.

1.2.1 SGM - Semi-Global Matching

Un sistema di visione stereoscopica basato sull'algoritmo SGM può essere scomposto nei quattro macroblocchi presentati precedentemente. Il primo, ovvero il calcolo dei costi delle corrispondenze non fa parte dell'algoritmo vero e proprio, che invece rappresenta un metodo usato nella strategia di aggregazione dei costi. Il calcolo dei costi delle corrispondenze verrà descritto successivamente in quanto, essendo indipendente dal resto, può essere scelto secondo differenti strategie. Nelle sezioni successive sono descritte le caratteristiche principali dell'algoritmo suddivise nei diversi blocchi funzionali.

Sarà considerata una coppia di immagini stereo rettificate; con immagine *match* si indicherà l'immagine in cui si cercano corrispondenze per i punti dell'immagine *base*. Con p si indicherà un punto dell'immagine base, mentre con q un punto dell'immagine match.

Lo scopo dell'algoritmo *SGM* consiste nell'identificazione della mappa di disparità D che minimizza una funzione energia $E(D)$ definita su tale mappa:

$$E(D) = E_{data}(D) + E_{smooth}(D) \tag{1.6}$$

dove $E_{data}(D)$ rappresenta il costo di corrispondenza di tutti i pixel, mentre il termine $E_{smooth}(D)$ è un vincolo di regolarizzazione.

Aggregazione dei costi

La procedura di aggregazione dei costi è la caratteristica principale dell'algoritmo SGM. La sua formulazione si basa su una particolare definizione della funzione energia $E(D)$, definita sulla mappa di disparità D, descritta di seguito.

Il termine $E_{data}(D)$ è la somma dei costi C di corrispondenza, o accoppiamento, di tutti i pixel per le disparità D:

$$E_{data}(D) = \sum_{\mathbf{p} \in \mathbf{D}} C(\mathbf{p}, D_{\mathbf{p}}) \tag{1.7}$$

Il secondo termine E_{smooth} penalizza diversamente le variazioni di disparità tra i pixel dell'immagine, aggiungendo una piccola penalità P_1 per tutti i pixels $\mathbf{q}$ con lievi cambiamenti di disparità (variazioni di una unità) all'interno della regione locale (o vicinato) $N\mathbf{p}$ del punto $\mathbf{p}$; oppure penalizzando con penalità P_2 in caso di cambiamenti maggiori di disparità all'interno della stessa regione $N\mathbf{p}$.

$$\begin{aligned} E_{smooth} = & \sum_{\mathbf{q} \in N\mathbf{p}} P_1 \mathbf{T}[|D_{\mathbf{p}} - D_{\mathbf{q}}| = 1] + \\ & \sum_{\mathbf{q} \in N\mathbf{p}} P_2 \mathbf{T}[|D_{\mathbf{p}} - D_{\mathbf{q}}| > 1] \end{aligned} \tag{1.8}$$

con

$$\mathbf{T}[x] = \begin{cases} 1 \text{ se } x \text{ vero} \\ 0 \text{ altrimenti} \end{cases} \tag{1.9}$$

Questa funzione di regolarizzazione che penalizza i cambiamenti di disparità è congrua con l'ipotesi di superfici lisce a tratti (piecewise smooth): la penalità ridotta

P_1 è presente per tenere conto di superfici curve o non perfettamente piane, mentre la penalità elevata P_2 serve per preservare le discontinuità. Per far in modo che la funzione E_{smooth} sia consistente, bisogna sempre imporre la che $P_2 <= P_1$, altrimenti la definizione di energia perde di validità.

Il calcolo della disparità viene quindi definito come la determinazione della mappa di disparità D che minimizza l'espressione 1.6, ovvero la soluzione di un problema di minimizzazione globale. Per molte funzioni che preservano le discontinuità, come quella in esame, la minimizzazione su tutta l'immagine (in 2D) è un problema NP-completo [8]. Viceversa, la minimizzazione lungo una dimensione dell'immagine (solitamente le righe) basata sulla programmazione dinamica viene risolta efficientemente in tempo polinomiale: tale approccio esegue una strategia di aggregazione dei costi provenienti da due orientamenti (gli unici due possibili all'interno di una riga dell'immagine), risultando molto vincolato all'unica dimensione considerata. Infatti, questa soluzione causa spesso evidenti striature, con conseguenti errori nella disparità.

Per sfruttare la velocità della programmazione dinamica e raggiungere una precisione simile ad una minimizzazione globale, l'idea alla base dell'algoritmo SGM è quella di estendere il numero di orientamenti coinvolti nella strategia di aggregazione a tutti gli orientamenti possibili, eseguendo poi una minimizzazione locale, in una sola dimensione, del costo $S(p,d)$ (della disparità d per il punto p) definito come la somma dei costi $L_r(p,d)$ dei cammini di costo minimo lungo una direzione r che terminano in p. Nell'implementazione di questo sistema sono stati utilizzati 8 cammini: quelli orizzontali, verticali e diagonali.

Il comportamento della funzione $E(D)$ viene modellato all'interno del costo aggregato dei cammini in una dimensione, definendolo come la somma del costo della disparità d per il punto p e del costo minimo lungo tale cammino che termina in $p-r$, includendo anche le penalità in maniera opportuna. Il costo $L'_{\mathbf{r}}$ per ogni percorso $\mathbf{r}$ è

aggregato come descritto in Eq. 4.3, definita per ogni pixel e $\mathbf{p}$ e disparità d:

$$\begin{aligned} L'_{\mathbf{r}}(\mathbf{p},d) = C(\mathbf{p},d) + \min(&L'_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d),\\ &L'_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d-1)+P_1, L'_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d+1)+P_1,\\ &\min_i L'_{\mathbf{r}}(\mathbf{p}-\mathbf{r},i)+P_2) \end{aligned} \tag{1.10}$$

In analogia con l'espressione 1.6 e successive, all'interno della funzione di minimo notiamo i due termini con penalità P_1 associati ai cammini con un lieve cambiamento di disparità (in aumento o in diminuzione rispettivamente) e il termine con penalità P_2 associato a tutti i cammini con cambiamenti significativi di disparità.

Trattandosi di una somma di termini sempre positivi, il valore del costo di un cammino può crescere senza limite; tuttavia, sottraendo un termine costante (il più elevato possibile), l'aumento del valore viene limitato senza cambiare la posizione del minimo ricercato. Il costo minimo tra i cammini che terminano in $p-r$ (al variare quindi della disparità) possiede le caratteristiche desiderate: è costante per tutti i valori disparità d e, nel peggiore dei casi, il costo aumenta di P_2 limitando in modo superiore il costo del cammino minimo $L_r \leq C_{max} + P_2$, dove C_{max} è il massimo costo calcolato.

L'espressione 4.3 diventa quindi:

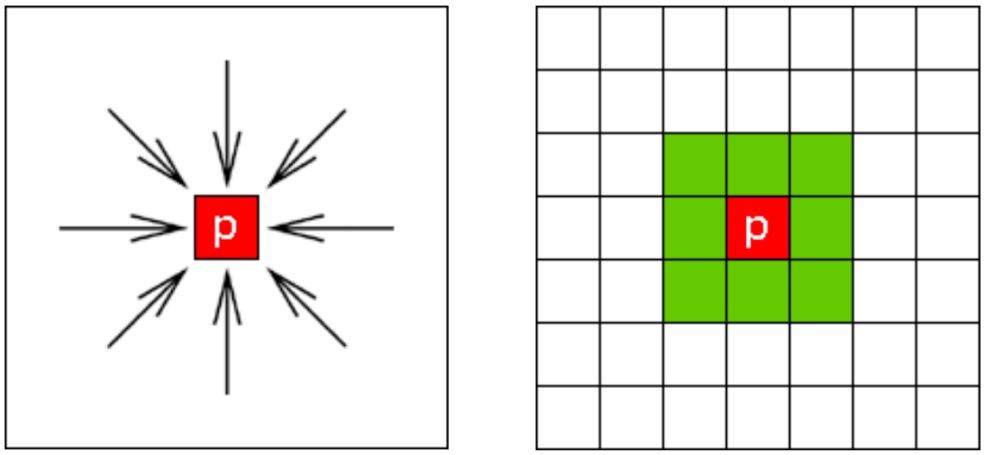
$$\begin{aligned} L_{\mathbf{r}}(\mathbf{p},d) = C(\mathbf{p},d) + \min(&L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d),\\ &L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d-1)+P_1, L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d+1)+P_1,\\ &\min_i L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},i)+P_2) - \min_k L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},k) \end{aligned} \tag{1.11}$$

Nella figura 1.8 è mostrata la disposizione delle direzioni dei cammini monodimensionali nello spazio dell'immagine dei costi.

Il costo aggregato finale $S(\mathbf{p},d)$ della disparità d per il punto $\mathbf{p}$ viene calcolato come:

$$S(\mathbf{p},d) = \sum_{\mathbf{r}} L_{\mathbf{r}}(\mathbf{p},d) \tag{1.12}$$

Dalle considerazioni sul limite superiore dell'equazione 4.3, il limite superiore per $S(\mathbf{p},d)$ è dato dal numero di direzioni n_r dei percorsi di aggregazione, moltiplicato per il massimo costo di un cammino di costo minimo: $S(\mathbf{p},d) \leq n_r \cdot (C_{max} + P_2)$.



(a) (b)

Figura 1.8: Esempio delle direzioni dei cammini (a) che terminano in p con relativa rappresentazione nell'immagine raster (b).

L'insieme di tutti questi valori per ogni pixel dell'immagine, per ogni valore di disparità viene definito *cubo dei costi aggregati*, rappresentato in figura 1.9 con un esempio dei percorsi di aggregazione per un pixel **p**, che vengono ripetuti per ciascun valore di disparità.

Calcolo della disparità e minimizzazione

Il calcolo finale della mappa di disparità, avviene utilizzando una strategia *winner-takes-all* (*WTA*, ovvero il vincitore è quello con valore minimo). Considerando l'immagine base, la disparità del punto **p** è l'ipotesi con costo aggregato $S(\mathbf{p},d))$ minimo:

$$\bar{D}(\mathbf{p}) = \arg\min_{d}(S(\mathbf{p},d)) \tag{1.13}$$

L'insieme di tutti questi valori costituisce la mappa di disparità D.

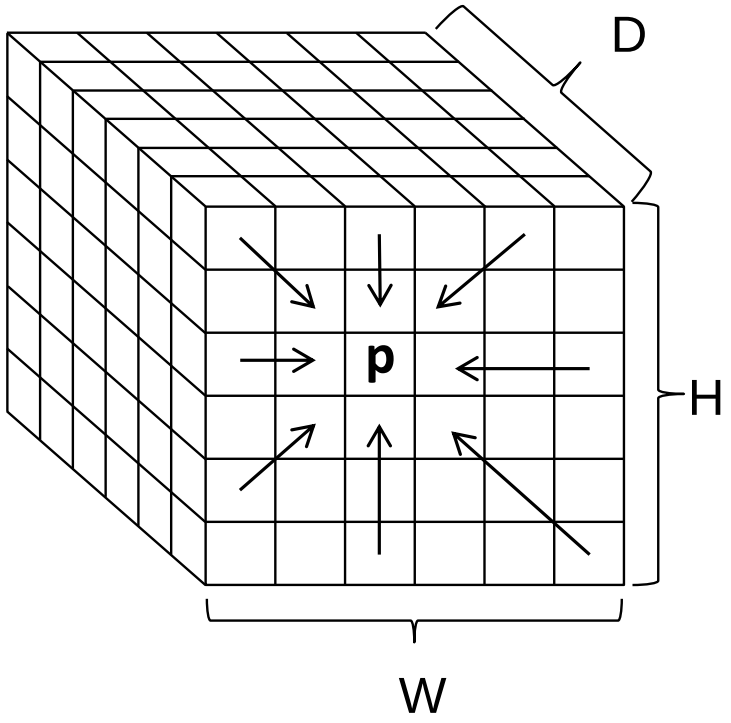


Figura 1.9: Cubo dei costi aggregati.

Visualizzare questa operazione sul cubo dei costi aggregati significa, per ogni punto **p** di ogni riga, analizzare tutti i costi $S(\mathbf{p},d)$ lungo lo spazio della disparità, come mostrato in Fig. 1.10, individuando l'indice del costo minimo.

1.2.2 Semi Global Matching - Filtri

Al fine di migliorare e rifinire i risultati restituiti dall'algoritmo SGM, sono stati studiati un insieme di filtri. Questi filtri operano a più livelli, suddividendosi tra quelli di rifinitura delle disparità, quelli di pre-eleborazione e quelli di post-elaborazione.

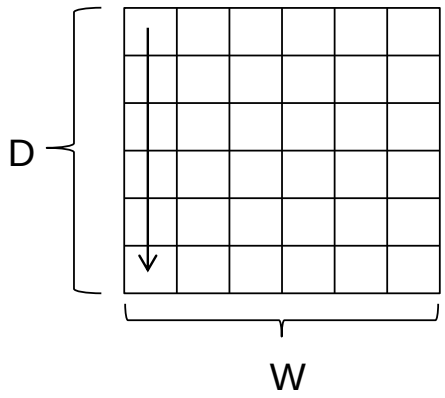


Figura 1.10: Selezione *WTA* della disparità del pixel **p**.

Filtri di rifinitura

Questo insieme di filtri vengono eseguiti durante la fase di generazione della mappa di disparità, permettendo di ottenere una stima più precisa del valore calcolato, oppure invalidando pixel errati, come per esempio nel caso delle occlusioni.

Un primo filtro denominato *del secondo minimo* o di *uniqueness constraint*, cerca di ridurre il numero di ricostruzioni spurie, considerano un pixel **p** valido solo nel caso in cui il rapporto tra il costo minimo, relativo a tale pixel, e il secondo costo minimo è al di sotto di una certa soglia (ovvero il minino trovato è sufficientemente forte).

Il *controllo di consistenza* sinistra/destra (left-right check) prevede di calcolare due mappe di disparità: quella rispetto all'immagine base e quella rispetto all'immagine match. In entrambi i casi, il calcolo della disparità viene eseguito secondo una strategia winner-takes-all. Considerando l'immagine base, la disparità del punto **p** è l'indice del costo aggregato $S(\mathbf{p},d)$ minimo:

$$D_(\mathbf{p}) = \arg\min_{d}(S(\mathbf{p},d)) \tag{1.14}$$

Nel caso dell'immagine match, la strategia è identica: la disparità del punto **q** è l'indice del costo aggregato minimo. In questo caso, i costi delle ipotesi di disparità possono essere o calcolati da zero oppure assunti uguali ai costi delle ipotesi calcolati per i punti corrispondenti nell'immagine base. In questo secondo caso, considerando le due linee epipolari corrispondenti nelle due immagini, al punto **q** viene assegnata la disparità d di costo minimo per il punto $(q_x - d, q_y) = e_{mb}(\mathbf{q}, d)$ nell'immagine base:

$$D_(\mathbf{q}) = \arg\min_{d}(S(e_{mb}(\mathbf{q}, d), d)) \tag{1.15}$$

dove $e_{mb}(\mathbf{q}, d)$ rappresenta la linea epipolare nell'immagine base corrispondente al punto **q**. Dopo aver calcolato le due mappe di disparità, si esegue un controllo sinistra/destra verificando che si siano ottenuti gli stessi valori di disparità (a meno di uno scostamento unitario), garantendo l'univocità delle corrispondenze tra le due immagini; dove ciò non si verifica si è in presenza di errori di assegnazione causati da occlusioni. Le disparità corrispondenti all'immagine di match vengono derivate dallo stesso volume dei costi aggregati dell'immagine base $S(\mathbf{p}, d)$ effettuando una ricerca del minimo lungo i valori diagonali come mostrato in Fig. 1.11.

Un ultimo filtro denominato *precisione sub-pixel* (sub-pixel accuracy), permette di stimare la parte decimale dei valori di disparità attraverso un'interpolazione pesata sui valori dei costi aggregati vicini a quello di costo minimo. Esistono varie strategie di interpolazione, ma come mostrato in [9], quella *equiangolare* è quella che meglio si adatta per stimare la parte decimale del valore vincente di disparità $\bar{d}$ per ogni pixel **p** :

$$\bar{d}_{frac} = \frac{S(\mathbf{p}, \bar{d} - 1) - S(\mathbf{p}, \bar{d} + 1)}{\max(S(\mathbf{p}, \bar{d} - 1), S(\mathbf{p}, \bar{d} + 1)) - S(\mathbf{p}, \bar{d})} \tag{1.16}$$

dove $S(\mathbf{p}, \bar{d} - 1)$ e $S(\mathbf{p}, \bar{d} + 1)$ sono i costi aggregati vicini a quello della disparità vincente (Fig. 1.12

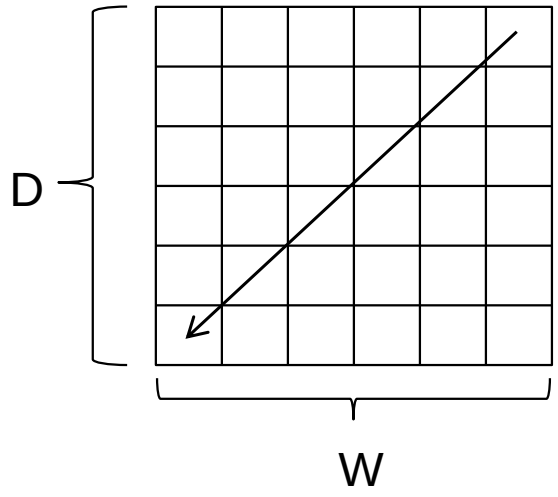


Figura 1.11: Controllo di consistenza sinistra/destra.

Filtri di pre-elaborazione

Questi filtri vengono applicati a monte dell'algoritmo SGM, agendo sulle immagini originali o andando a modificare il processo di calcolo dei costi delle corrispondenze.

- *Filtro gaussiano* – la seguente maschera 3×3 gaussiana di "smoothing" viene applicata ad entrambe le immagini originali in toni di grigio, per attenuare eventuale rumore presente nel processo di acquisizione:

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{1.17}$$

- *Maschera di Census sparsa* – come illustrato in [10], si utilizza una maschera sparsa che segue il pattern visualizzato in Fig. 1.13 per il calcolo dei valori della trasformata di Census, invece che utilizzare una finestra piena centrata nel pixel:

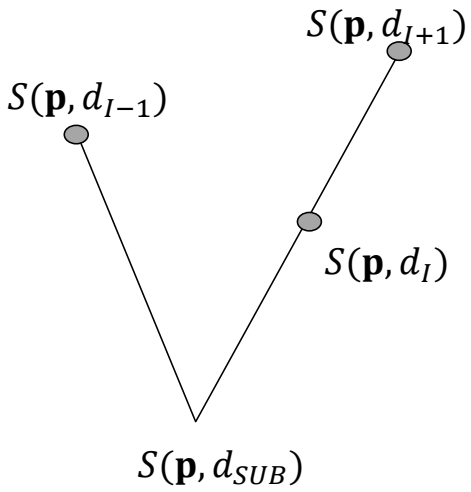


Figura 1.12: Esempio di interpolazione equiangolare per stimare la parte decimale della disparità.

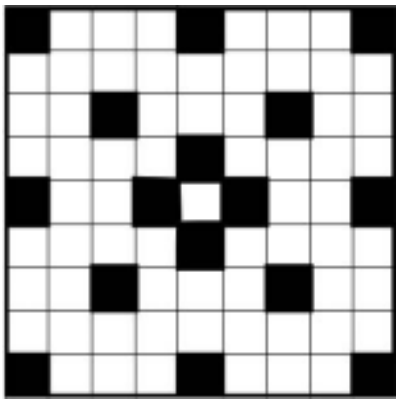


Figura 1.13: Maschera di Census sparsa.

- *Census ternarizzato* – per far in modo di aumentare la quantità di informazione che viene codificata durante la trasformata di Census delle immagini originali in quella risultante, la funzione della trasformata di Census $\xi(I_p, I_{\overline{p}})$ (normal-

mente con risultato binario per ogni pixel) è stata modificata per restituire le seguenti stringhe:

$$\xi(I_p, I_{\overline{p}}) = \begin{cases} 00 \text{ se } I_p - I_{\overline{p}} < -1 \\ 11 \text{ se } I_p - I_{\overline{p}} > 1 \\ 01 \text{ altrimenti} \end{cases} \tag{1.18}$$

- *Aggregazione punteggi di Hamming* – come suggerito in [10], una finestra W di dimensioni 5×5 centrata attorno ad ogni pixel, viene utilizzata per preprocessare ogni costo $C(p,d)$ nel cubo dei costi iniziali:

$$C(p,d) = \frac{1}{25} \sum_{\overline{p} \in W} C(\overline{p}, d) \tag{1.19}$$

Filtri di post-elaborazione

Filtri applicati dopo aver ottenuto la mappa di disparità completa per andare a filtrare eventuali valori spurii (outlier) o per integrare i valori mancanti in piccole zone della mappa.

- *Media adattiva* – un filtro di smoothing adattivo con finestra 8×8 [4] è applicato alla mappa di disparità ottenuta D:

$$D(p) = \frac{\sum_{\overline{p} \in W} D(\overline{p}) \max\{4 - |D(p) - D(\overline{p})|, 0\}}{\sum_{\overline{p} \in W} \max\{4 - |D(p) - D(\overline{p})|, 0\}} \tag{1.20}$$

- *Filtro despeckle* – piccole zone della mappa di disparità con valori molto differenti da quelli dei vicini, sono solitamente dovuti ad errori di associazione tra i pixel delle due immagini, la strategia che viene proposta in [2] è usata per individuare ed eliminare queste zone.

- *Filtro gap* – Un'interpolazione costante lungo i percorsi 1D orizzontali e verticali sulla mappa di disparità è utilizzata per riempire le piccole aree (≤ 3 px) con valori mancanti di disparità [4]. Siano p_L e p_R i primi due pixel con disparità valida prima e dopo il corrente gap di valori mancanti; i valori usati per

riempirli sono calcolati come:

$$d = \begin{cases} \frac{D(p_L)+D(p_R)}{2} \text{ if } |D(p_L)-D(p_R)| < 3 \\ \min\{D(p_L), D(p_R)\} \text{ otherwise} \end{cases} \tag{1.21}$$

1.2.3 Metrica Birchfield-Tomasi

In questa sezione viene descritta brevemente una differente metrica di inizializzazione dei costi delle corrispondenze, che verrà utilizzata nel capitolo 2 tra le varie configurazioni testate. L'implementazione in OpenCV di SGM [11] (BT-SGM nel prosieguo) liberamente utilizzabile, utilizza come metrica la *dissimilarità Birchfield-Tomasi* [12] per inizializzare il volume dei costi.

Siano I_L e I_R le funzioni 1D che rappresentano l'intensità lungo una linea di scansione dell'immagine sinistra e destra rispettivamente, e $\hat{I}_L(x_L)$, $\hat{I}_R(x_R)$ le funzioni di interpolazione lineare tra i punti intorno x_L e x_R. È quindi possibile determinare un valore di confidenza del confronto tra l'intesità in x_L e la regione linearmente interpolata intorno a x_R:

$$\overline{d}(x_L, x_R, I_L, I_R) = \min_{x_R - \frac{1}{2} \leq x \leq x_R + \frac{1}{2}} \left| I_L(x_L) - \hat{I}_R(x) \right| \tag{1.22}$$

e per simmetria:

$$\overline{d}(x_R, x_L, I_R, I_L) = \min_{x_L - \frac{1}{2} \leq x \leq x_L + \frac{1}{2}} \left| I_R(x_R) - \hat{I}_L(x) \right| \tag{1.23}$$

La dissimilarità tra i pixel in x_L e x_R quindi diventa:

$$d(x_L, x_R) = \min\{\overline{d}(x_L, x_R, I_L, I_R), \overline{d}(x_R, x_L, I_R, I_L)\} \tag{1.24}$$

e questo valore viene utilizzato per inizializzare il cubo dei costi per ogni pixel e per ogni valore di disparità.

1.2.4 ELAS - Efficient Large-Scale Stereo Matching

Efficient Large-Scale Stereo Matching (abbreviato *ELAS*), proposto in [4], è un metodo per gestire in modo particolarmente efficiente scenari con valori di disparità molto

elevati che possono derivare dall'utilizzo di *baseline* larghe fra le telecamere, oppure nel caso di immagini ad elevata risoluzione.

Sfrutta robusti abbinamenti di punti di controllo sparsi per per generare una mesh 2D attraverso la *triangolazione di Delaunay*. I valori di disparità di questi punti associati vengono utilizzati, attraverso una funzione lineare a tratti definita sui punti di controllo della mesh, per inizializzare una mappa di disparità, utilizzata successivamente per ridurre le dimensioni dell'intervallo di ricerca delle disparità per i pixel rimanenti.

1.3 Architettura di un sistema stereoscopico

In questa sezione viene presentata la struttura a pipeline scelta per il sistema stereo che è stato progettato durante questa ricerca.

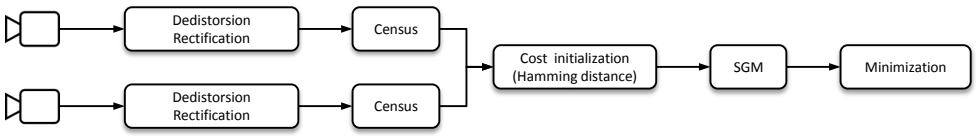


Figura 1.14: Pipeline di un sistema di ricostruzione stereo.

Le fasi di elaborazione coinvolte in un sistema di ricostruzione stereoscopica sono illustrate in Fig. 1.14.

Inizialmente gli effetti combinati della distorsione delle lenti e del disallineamento delle telecamere, viene rimosso dalle immagini di input originali I_L e I_R usando delle *look-up table* (LUT): questo permette di ottenere un paio di immagini rettificate R_L and R_R, che riduce la fase di calcolo delle corrispondenze ad una ricerca 1D lungo linee epipolari orizzontali.

Ogni coppia di coordinate intere $\mathbf{p} = (x, y)$ dell'immagine rettificata, viene associata alla posizione discreta (con coordinate decimali) $\mathbf{i} = (x_i + \alpha, y_i + \beta) = LUT(x, y)$ sull'immagine di ingresso.

L'interpolazione bilineare viene utilizzata per determinare il valore del pixel rettificato $R(x,y)$:

$$R(\mathbf{p}) = (1-\beta)((1-\alpha)I(x_i,y_i) + \alpha I(x_i+1,y_i)) + \\ \beta((1-\alpha)I(x_i,y_i+1) + \alpha I(x_i+1,y_i+1)) \tag{1.25}$$

Le trasformate di Census $n \times n$ [13], C_L e C_R, sono calcolare sulle immagini rettificate, e successivamente utilizzate per inizializzare il cubo dei costi $C(\mathbf{p},d)$ per ogni pixel $\mathbf{p}$ e per ogni valore di disparità d prendendo la *distanza di Hamming H* della coppia di pixel risultante:

$$C(\mathbf{p},d) = H(C_L(x+d,y), C_R(x,y)) \tag{1.26}$$

I passi successivi applicano le varie fasi dell'algoritmo SGM (presentato nella Sec. 1.2.1) fino ad ottenere la mappa di disparità D dal blocco di minimizzazione che si occupa anche di applicare i filtri del secondo minimo, del controllo sinistra/destra e di interpolazione equiangolare.

Capitolo 2

Analisi prestazioni algoritmi di ricostruzione stereo

Negli ultimi anni, l'incremento della potenza di elaborazione dell'hardware COTS [1] (intendendo sia elaboratori tradizionali in versioni industriali e automotive, che veri e propri sistemi di embedded basati su piattaforme hardware ad alte prestazioni e bassi consumi) è in continua crescita e ha permesso ad un numero sempre maggiore di algoritmi, per la ricostruzione tridimensionale a partire dalla visione stereoscopica, di ottenere prestazioni ragionevoli tali da consentirne l'utilizzo in applicazioni reali (per esempio la guida autonoma di autoveicoli).
Un confronto quantitativo e qualitativo dei loro livelli di performance tuttavia, non è semplice, principalmente per la difficoltà di generare informazioni di *ground truth*. Gli storici *data-set* disponibili in questo campo della visione artificiale, erano piccoli e sintetici (ovvero prodotti da immagini sintetizzate virtualmente) oppure ripresi in ambienti controllati [6]; queste caratteristiche limitano la loro utilità come indicatori della bontà attuale degli algoritmi in scenari all'aperto o comunque in situazioni applicative reali. Più recentemente, l'esigenza di opportune metriche di confronto ha portato alla definizione di nuove misure di qualità che sono descritte e utilizzate nel prosieguo.

[1] Commercial Off-The-Shelf

In questo capitolo vengono confrontati i livelli di performance di alcuni algoritmi allo stato dell'arte per la mappatura tridimensionale basata su stereovisione in scenari automotive. Saranno utilizzati sia *data-set* disponibili in letteratura, che *data-set* specificatamente raccolti attraverso una piattaforma di registrazione dedicata, sviluppata presso il nostro laboratorio.
Tra gli algoritmi valutati, quello scelto per la successiva implementazione sulla piattaforma embedded, ovvero SGM, viene analizzato in maggior dettaglio, per testarne differenti varianti in modo da determinarne la configurazione migliore.

2.1 Setup sperimentale

Fornire misure quantitative affidabili sulle prestazioni di algoritmi di *stereo-mapping* all'aperto o in scenari non controllati è un compito difficile, che può essere affrontato con diversi approcci.

Una soluzione [14, 15] è l'utilizzo di un scanner LIDAR [2] high-end [16] per mappare direttamente l'area intorno al veicolo: le misure di distanza di solito hanno un livello di precisione dell'ordine dei centimetri nel range 5-100 m e producono mappe ragionevolmente dense, utilizzando fino a 64 piani di scansione orizzontale.
Un'altra opzione è quella di sfruttare un conoscenza a priori sul *data-set*, come la presenza di spazio libero davanti ad un veicolo guidato manualmente [17] (si presuppone che il conducente mantenga sempre una distanza minima di sicurezza) per individuare i punti tridimensionali erroneamente ricostruiti dall'algoritmo, su un periodo esteso di tempo.
Un ultima soluzione consiste nel sintetizzare una vista virtuale, partendo dalla geometria tridimensionale dell'ambiente ricostruita dall'algoritmo, e confrontarla con i dati registrati da una terza telecamera reale opportunamente posizionata [18, 19].

In questa analisi gli algoritmi sono stati testati usando tutti gli approcci menzionati, in modo da comprendere meglio il loro comportamento effettivo in scenari di applicazioni reali.

[2] Light Detection And Ranging, è una tecnica di telerilevamento che permette di determinare la distanza, o profondità, di un oggetto o di una superficie utilizzando impulsi laser

2.1.1 Dense LIDAR-based ground truth

Come rifermino è stato utilizzato il *data-set* disponibile in [20], che consiste in un insieme di dati di *ground truth* filtrati manualmente, di circa 200 scene riprese in ambiti urbani e rurali.

Il *ground truth* per una data immagine, è ottenuto dalla registrazione di 5 scansioni laser consecutive prima e dopo l'istante di acquisizione della stessa, successivamente accumulate (compensando adeguatamente il moto del veicolo) per ottenere una nuvola di punti tridimensionale; le regioni ambigue, come per esempio i vetri e le recinzioni, sono state rimosse manualmente, e infine la mappa di disparità corrispondente è calcolata usando le informazioni di calibrazione tra laser e telecamera (Fig. 2.1).

Le metriche di performance utilizzate per valutare la bontà delle mappe di disparità (calcolate in pixel), come illustrato dal test [20], sono:

- la percentuale di pixel errati (cioè quelli che hanno un errore di misura superiore a 3 px),
- l'errore medio di corrispondenza in pixel,
- la densità dell'immagine ottenuta.

Tuttavia, l'esatta procedura di calcolo di questi valori, è stata leggermente modificata, in quanto la metrica originale non sembrava sufficientemente equa. In particolare:

- in questa analisi, solo i pixel calcolati non occlusi sono stati considerati. Il test originale fornisce anche statistiche dopo un'interpolazione lineare dei valori mancanti, con l'intento di rendere comparabili tra loro algoritmi sparsi, semidensi e densi; tuttavia tale approssimazione non è ottima e questo si riflette su una penalizzazione negativa degli algoritmi non densi.
- l'errore medio è stato calcolato considerando solo i valori sotto l'errore di endpoint, e non tutti i valori, in modo da avere una stima migliore della bontà dei pixel rilevanti e non errati.

- vengono considerate le statistiche per ogni immagine e non solamente la loro media sull'intera sequenza di test. Per meglio comprendere il dato raccolto, esso è rappresentato in un grafico con la variabile indipendente (asse-x) che indica il valore misurato e quella dipendente (asse-y) la percentuale di frame che rimango sotto tale valore. Gli algoritmi migliori sono quelli con un minore valore x per una data percentuale (es. $y = 90\%$).

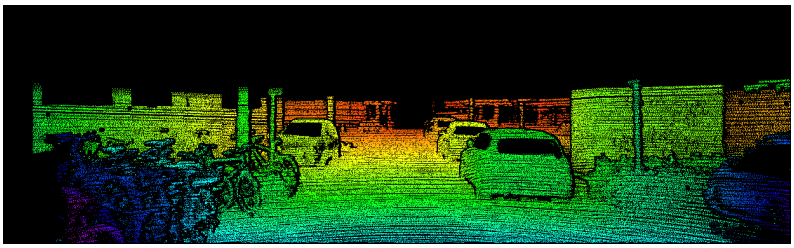
2.1.2 Stima delle false corrispondenze

Questo benchmark è l'adattamento di una delle tecniche descritte in [17]: normalmente, il conducente di un veicolo mantiene una distanza di sicurezza di circa 1 s dal veicolo che lo precede; questo significa che un volume (dipendente dalla velocità) di spazio libero è presente di fronte al veicolo in ogni momento, quindi ogni punto mondo, ricostruito dall'algoritmo di visione stereo-tridimensionale, che cade in questo volume "di sicurezza" deve essere considerato come una stima errata, come mostrato in Fig. 2.2. La percentuale di false corrispondenze $m_{fc} = 100 \times N_{fc}/N$ è il rapporto di punti all'interno del volume libero rispetto al numero totale di punti 3D ricostruiti.

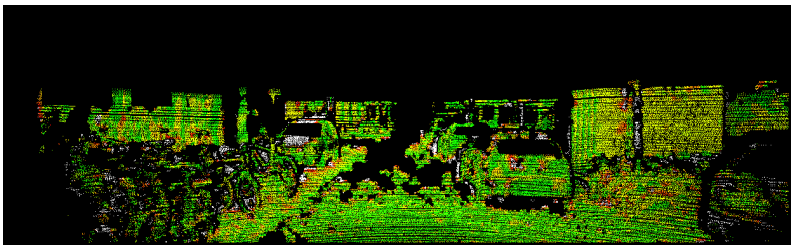
2.1.3 Cross-correlazione normalizzata

Gli approcci descritti precedentemente, hanno qualche limitazione: il LIDAR-based ground truth richiede molto tempo per essere generato e quindi non può essere utilizzato per ottenere un data-set molto grande, mentre le misure sul veicolo che precede possono essere facilmente eseguite anche su lunghe sequenze, ma questa è una misura di performance indiretta, anche se significativa. Come alternativa, l'utilizzo di una terza telecamera [19], come illustrato in Fig. 2.3, permette un confronto diretto tra una vista virtuale ricostruita, a partire dalle visione stereo sulle prime due telecamere, con quella attualmente inquadrata dalla terza telecamera, senza la necessità di un intervento manuale. La mappa di disparità è usata per trasformare i pixel ripresi dalla telecamera di rifermento, nelle coordinate della telecamera di controllo, in modo da creare una immagine virtuale (Fig. 2.4-a) che può essere confrontata con

(a)



(b)



(c)

Figura 2.1: Esempio di LIDAR-based ground truth. a) L'immagine originale, b) La mappa di disparità generata dal LIDAR che rappresenta il ground-truth, e c) la mappa dell'errore, con colori che variano dal verde (errore = 0 px) al rosso (errore = 3 px). I pixel bianchi rappresentano le aree con un errore di ricostruzione maggiore di 3 px.

quella registrata dalla telecamera di controllo (Fig. 2.4-b) per produrre una mappa di cross-correlazione (Fig. 2.4-c). Tuttavia, occorre prestare attenzione a garantire che i risultati siano significativi: la calibrazione della telecamera è una fonte di errore che deve essere minimizzato e anche la presenza di ostacoli occasionali all'interno del

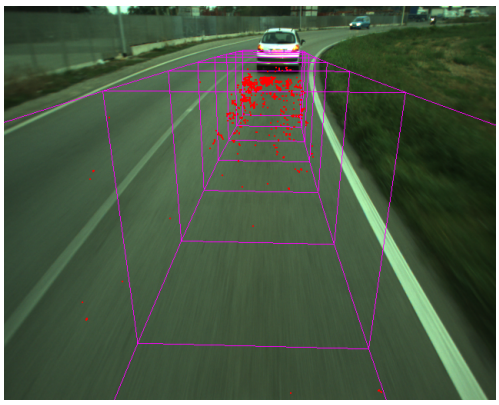


Figura 2.2: Esempio di stima delle false corrispondenze. In viola il volume libero da ostatoli che è presente di fronte al veicolo, e in rosso i punti che ricadono all'interno di esso, generati da false corrispondenze.

volume libero deve essere trattata opportunamente.

La metrica scelta è la Cross-Correlazione Normalizzata (NCC [3]), calcolata nel seguente modo:

$$NCC(I_v, I_c) = \frac{1}{|\Omega|} \sum_{(x,y)\in\Omega} \frac{[I_c(x,y) - \mu_c][I_v(x,y) - \mu_v]}{\sigma_c \sigma_v} \tag{2.1}$$

dove Ω è il sottoinsieme di tutti i pixel che hanno una disparità valida; $\mu_c, \mu_v, \sigma_c, \sigma_v$ sono le medie e le deviazioni standard rispettivamente dell'immagine di controllo e di quella virtuale.

Vale la pena notare come in [18] si consiglia una configurazione con telecamera di riferimento e quella di confronto posizionate alla distanza di 30 cm, mentre la telecamera di controllo a 50 cm da quella di riferimento; tuttavia, la piattaforma di registrazione utilizzata in questo lavoro utilizza distanza minori (rispettivamente 24 e 12 cm), come illustrato in Sez. 2.1.4, in quanto il veicolo è stato equipaggiato una una telecamera trinoculare pre-calibrata. In questo studio, come telecamera di controllo si considera quella centrale, tra la telecamera di riferimento e quella di confronto.

[3] Normalized Cross Correlation

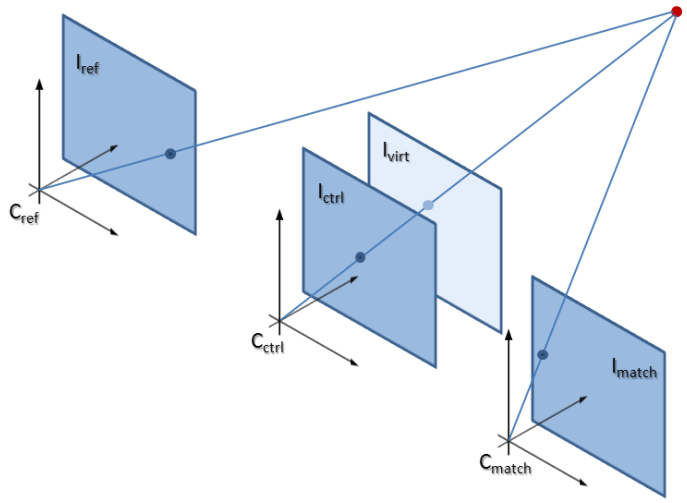


Figura 2.3: Setup della telecamere trinoculare per la valutazione dello stero. La coppia stereo di telecamere di riferimento e di confronto (C_{ref} e C_{match} rispettivamente) sono utilizzate per produrre una immagine virtuale I_{virt} nel sistema di riferimento della telecamera di controllo, che può essere confrontata con l'immagine attualmente registrata da quest'ultima I_{ctrl}.

(a) (b) (c)

Figura 2.4: Esempio metrica NCC. L'immagine di riferimento è usata per produrre l'immagine virtuale a), che è confrontata con l'immagine ripresa dalla telecamere di controllo b), per produrre la mappa di cross-correlazione c).

Questa configurazione rende l'immagine virtuale e quella di controllo più simili e potrebbe potenzialmente migliorare il punteggio NCC a causa dei valori di disparità in gioco più bassi. Tuttavia, la calibrazione di fabbrica della telecamera trinoculare è significativamente migliore di quella che può essere attualmente ottenuta con una calibrazione in laboratorio di singole telecamere; il conseguente miglioramento di precisione è stato ritenuto più rilevante che la quantizzazione introdotta dalla riduzione del range di disparità.

2.1.4 Setup della piattaforma di registrazione

I test descritti in Sez. 2.1.2 and Sez. 2.1.3 sono stati effettuati su dati acquisiti usando il veicolo raffigurato in Fig. 2.5, che è stato equipaggiato con una telecamera frontale trinoculare Point Grey Bumblebee® XB3-13S2C a colori con ottiche da 3.8 mm, e con risoluzione di 1280×960 pixel, montata sulla carrozzeria del veicolo al di sopra del parabrezza. Il sistema di visione è sincronizzato con un laser scanner LIDAR SICK LD-MRS-400001 a 4-piani funzionante a 12.5 Hz, con una risoluzione angolare pari a $0.125\,^\circ$ e un campo visivo di $85\,^\circ$, installato nel paraurti anteriore. Le informazioni GPS e INS (inerziali) sono fornite da un'unità Topcon AGI-3 unit, e sono usate per stimare la traiettoria del veicolo.

(a) (b) (c)

Figura 2.5: La piattaforma di registrazione. I dati sono stati raccolti utilizzando uno dei veicoli elettrici (a) che sono stati allestiti nel 2010 per prendere parte al VisLab Intercontinental Autonomous Challenge (VIAC) [21]. Il sistema di visone (b) è una Point Grey Bumblebee®s XB3-13S2C, sincronizzata con un SICK LD-MRS-400001 LIDAR (c) funzionante alla frequenza di 12.5 Hz

Calibrazione LIDAR con telecamera

Per ottenere risultati significativi dal test descritto in Sez. 2.1.2, l'unità LIDAR è stata utilizzata per individuare la presenza occasionale di veicoli che precedono all'interno dell'area libera definita dal volume di test. Per eseguire questa operazione è necessario misurare il posizionamento relativo della telecamera trinoculare e del laser scanner, e questo è abbastanza impegnativo dato la piccola quantità di dati prodotta dal laser scanner LIDAR a 4 piani disponibile.

La procedura di calibrazione inizia con un primo allineamento grossolano, quindi vengono manualmente associati punti LIDAR facilmente riconoscibili con i corrispondenti pixel dell'immagine registrata. La precisione di ogni associazione è vincolata da diversi fattori, come la risoluzione angolare del LIDAR e le ambiguità derivanti dal limitato numero di piani di scansione che colpiscono ogni superficie; tuttavia, un gran numero di campioni possono essere velocemente raccolti su differenti frame e utilizzati tutti assieme in un framework di minimizzazione non lineare basato sul

Maximum-Likelihood, utilizzando come funzione di costo:

$$\arg\min_{[\mathrm{R}|\mathrm{t}]} \sum_i \|p_i - \mathrm{K}[\mathrm{R}|\mathrm{t}]w_i\|^2 \tag{2.2}$$

dove $[\mathrm{R}|\mathrm{t}]$ è la matrice di rototraslazione incognita da stimare, p un pixel 2D dell'immagine (dedistorta) e w il corrispondente punto mondo. Assumendo come ipotesi il modello pin-hole della telecamera, la matrice di proiezione della telecamera K è definita come:

$$\mathrm{K} = \begin{bmatrix} k & 0 & u_0 \\ 0 & k & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

con k la lunghezza focale in pixel e u_0, v_0 il centro ottico della telecamera. Come risolutore non lineare, è stato scelto il metodo Levenberg-Marquardt [22], per la sua robustezza e la relativa velocità di convergenza.

Calibrazione tra le telecamere

Per poter eseguire il test descritto nella Sez. 2.1.3 con il setup hardware disponibile, le posizioni relative tra tutte le telecamere devono essere note. Purtroppo la telecamera trinoculare Point Grey Bumblebee® XB3-13S2C fornisce unicamente le look-up tables (LUT) di rettificazione e dedistorsione tra le telecamere destra e sinistra e tra quelle destra e centrale. Occorre quindi calcolare la calibrazione mancante tra quella centrale e quella sinistra.

Siano P_{mw} e P_{ms} due punti di disparità omogenea sulla telecamera destra (telecamera di confronto, see Fig. 2.3) rispettivamente nei sistemi di riferimento largo e stretto:

$$P_{mw} = (u_{rw} - d_{rw}, v_{rw}, -d_{rw}, 1) \qquad P_{ms} = (u_{rs} - d_{rs}, v_{rs}, -d_{rs}, 1) \tag{2.4}$$

esiste pertanto una matrice di omografia 3D H affinché $P_{ms} = \mathrm{H}P_{mw}$, nella forma:

$$\mathrm{H} = \mathrm{Q_s^{-1}} \begin{bmatrix} \mathrm{R} & \mathrm{t} \\ 0 & 1 \end{bmatrix} \mathrm{Q_w} \tag{2.5}$$

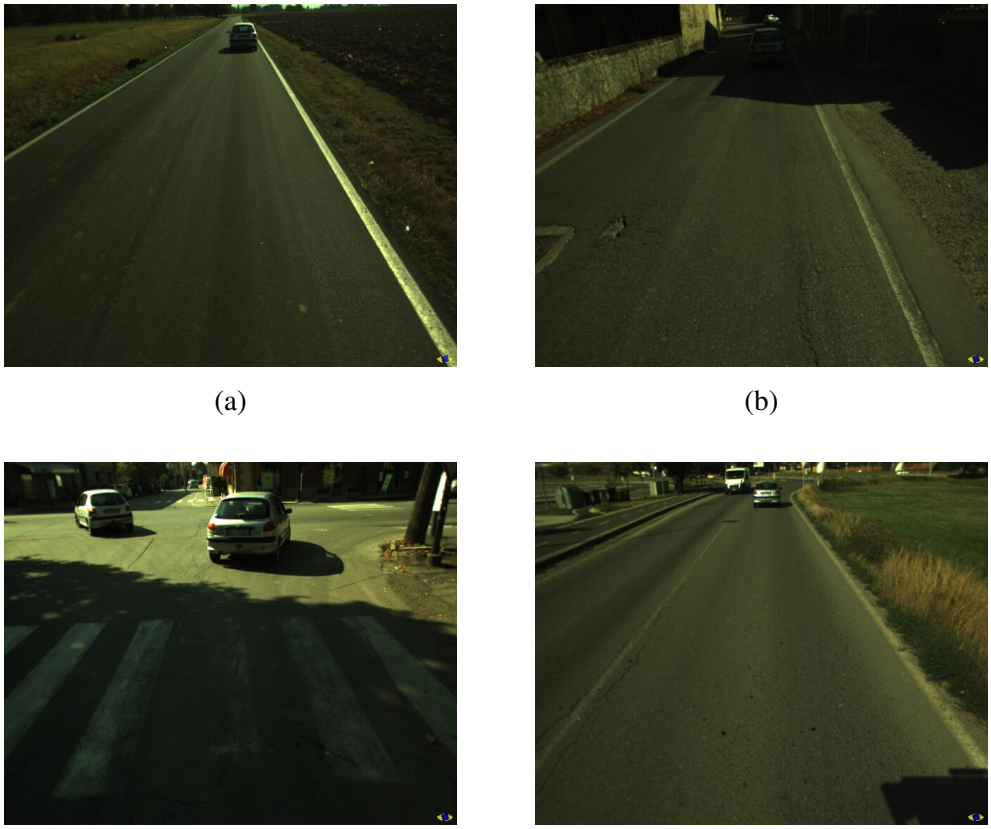
con

$$\mathrm{Q_s^{-1}} = \begin{bmatrix} k_s & 0 & u_{0s} & 0 \\ 0 & k_s & v_{0s} & 0 \\ 0 & 0 & 0 & k_s b_s \\ 0 & 0 & 1 & 0 \end{bmatrix} \qquad \mathrm{Q_w} = \begin{bmatrix} \frac{1}{k_w} & 0 & 0 & -\frac{u_{0w}}{k_w} \\ 0 & \frac{1}{k_w} & 0 & -\frac{v_{0w}}{k_w} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{1}{k_w b_w} & 0 \end{bmatrix} \tag{2.6}$$

I termini $k_{\{w,s\}}$ nell'Eq. 2.6 rappresentano le lunghezze focali delle immagini rettificate rispettivamente delle baseline larga e corta, $u_{0\{w,s\}}$, $v_{0\{w,s\}}$ sono i centri ottici corrispondenti e $b_{\{w,s\}}$ le lunghezze di baseline. I termini R e t nell'Eq. 2.5, invece, rappresentano le componenti di rotazione e traslazione che allineano le due baseline. La rotazione incognita R è molto prossima all'identità siccome le tre telecamere sono praticamente allineate fisicamente e un risolutore lineare si è dimostrato sufficiente per stimare direttamente i termini della matrice H, utilizzando una corrispondenza basata su feature per generare le necessarie coppie di pixel associati. Quando H è nota, il valore di luminanza $I(P_{rw})$ di ogni punto con disparità nota è utilizzato per costruire l'immagine virtuale, attraverso la proiezione nelle coordinate (u_{rs}, v_{rs}), sfruttando Eq. 2.4.

2.1.5 Test data-set

Una sequenza di test è stata registrata in uno scenario misto su strade urbane e rurali, come mostrato in Fig. 2.6. Il set di dati è stato acquisito lungo un giro circolare di 15 Km intorno alle zone del campus dell'università di Parma; la sessione di registrazione è avvenuta intorno alle ore 13:00 di una soleggiata giornata di settembre, e gli scenari includono strette strade di campagna (Fig. 2.6-a), piccoli centri urbani (Fig. 2.6-b), intersezioni (Fig. 2.6-c) e autostrade (Fig. 2.6-d).



(a) (b)

(c) (d)

Figura 2.6: Esempi dei dati registrati.

2.2 Benchmark - Analisi delle prestazioni

In questa sezione vengono inizialmente presentati i risultati dei test eseguiti sui vari filtri presentati nella Sez. 1.2.2. Ogni filtro è stato testato singolarmente con una configurazione standard *Census-SGM* e sono state selezionate tre configurazioni promettenti. Ogni configurazione è stata poi confrontata con altri algoritmi stereo, che condividono alcune delle stesse strategie di filtraggio, come dettagliato nella Tab. 2.1.

Gli approcci utilizzati nel confronto, oltre a quello standard *Census-SGM*, sono una versione di SGM che utilizza una differente metrica di inizializzazione dei costi descritta in 1.2.3 (di seguito *BT-SGM*) e un secondo basato sull'algoritmo *ELAS* 1.2.4.

In questa sezione verranno illustrati i grafici che misurano le performance delle varie configurazioni testate; per brevità, verranno usate le seguenti notazioni:

- LGT – Valutazione del LIDAR-based ground (Sez. 2.1.1);
- NFC – Numero di false corrispondenze (Sez. 2.1.2);
- NCC – Cross-correlazione normalizzata (Sez. 2.1.3).

Tabella 2.1: Configurazioni degli algoritmi. Dopo i testi individuali, i setup di filtri più promettenti sono stati valutati per l'algoritmo *Census-SGM*, mentre per gli algoritmi *BT-SGM* e *ELAS* sono stati seguiti i setup suggeriti in [15].

	Census-SGM			BT-SGM	ELAS
	Config 1	Config 2	Config 3		
Filtro gaussiano	✓	✓	✓	-	-
Mappa di Census sparsa	-	-	-	-	-
Census ternarizzato	-	-	-	-	-
Aggregazione costi di Hamming	-	-	-	-	-
Uniqueness constraint	10	20	20	10	15
Filtro adattivo	✓	✓	✓	-	✓
Filtro despeckle	✓	✓	✓	✓	✓
Filtro gap	✓	✓	-	-	✓
Altri paramentri	P1=10, P2=50, L/R check			vedere [23]	vedere [23]

2.2.1 Filtri isolati

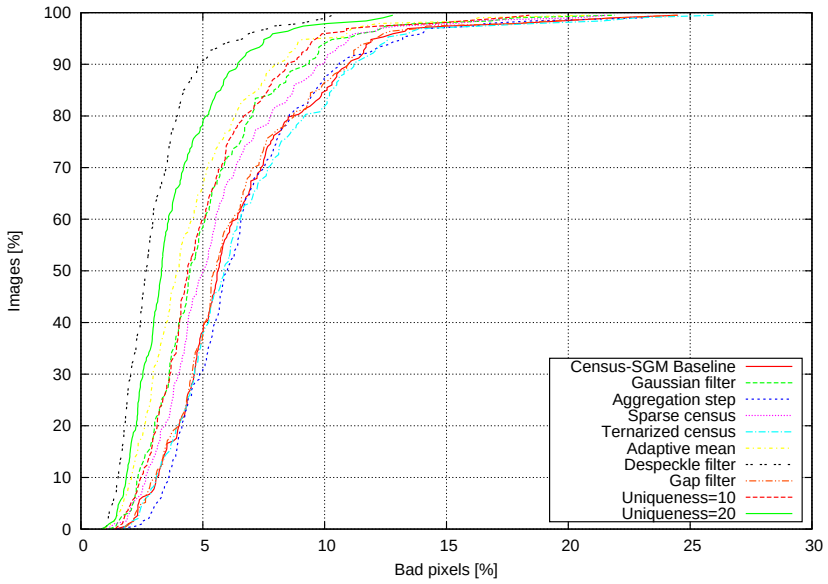
I risultati del test LGT per i vari filtri, presi singolarmente, presentati nella Sez. 1.2.2 sono mostrati in Fig. 2.7. I maggiori miglioramenti si ottengono utilizzando il *filtro*

despeckle; anche l'*uniqueness constraint* con una soglia bassa è efficace nel rimuovere valori spurii, penalizzando però la densità dei risultati. La *media adattiva* riduce notevolmente l'errore di ricostruzione, anche se su scala relativamente piccola (intorno 0.1 px). A livello di errore sub-pixel il *filtro gap* produce peggiori risultati, che possono spiegarsi con il fatto che l'interpolazione a valore costante non è abbastanza accurata per descrivere le variazioni di disparità tra pixel e pixel.

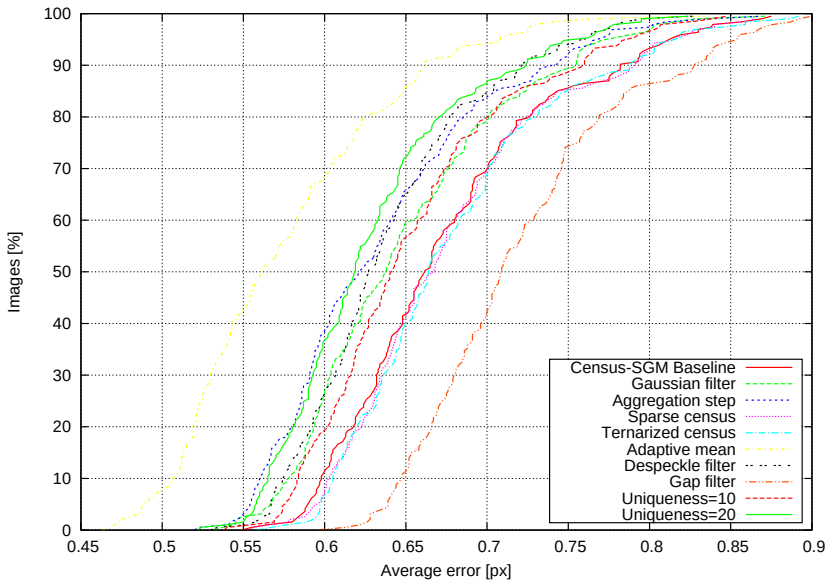
La Fig. 2.8 mostra i risultati per il test NFC: i filtri *despeckle* e *uniqueness* continuano a mostrare miglioramenti, come anche la *media adattiva*, avvalorando l'idea che una loro combinazione probabilmente permette di migliorare le performance di ricostruzione stereoscopica.

I risultati per il test NCC sono rappresentati in Fig. 2.9: sfortunatamente, i punteggi ottenuti dai differenti filtri sono pressoché sovrapposti. Questo comportamento non è facilmente spiegabile, sebbene certi fattori possano contribuirvi:

- i punteggi NCC dipendono dalla luminanza dei punti immagine a confronto, quindi ricostruzioni sbagliate non sono equamente pesate;
- la piccola baseline utilizzata, riduce gli effetti misurabili degli errori di ricostruzione;
- la qualità di ricostruzione è sempre abbastanza buona utilizzando l'algoritmo SGM, indipendentemente dai filtri applicati, e i punteggi dei test potrebbero essere dovuti principalmente ad altre fonti di errore, come ad esempio la calibrazione.



(a)



(b)

Figura 2.7: Performance LGT dei filtri isolati. a) Percentuale pixel errati e b) errore medio.

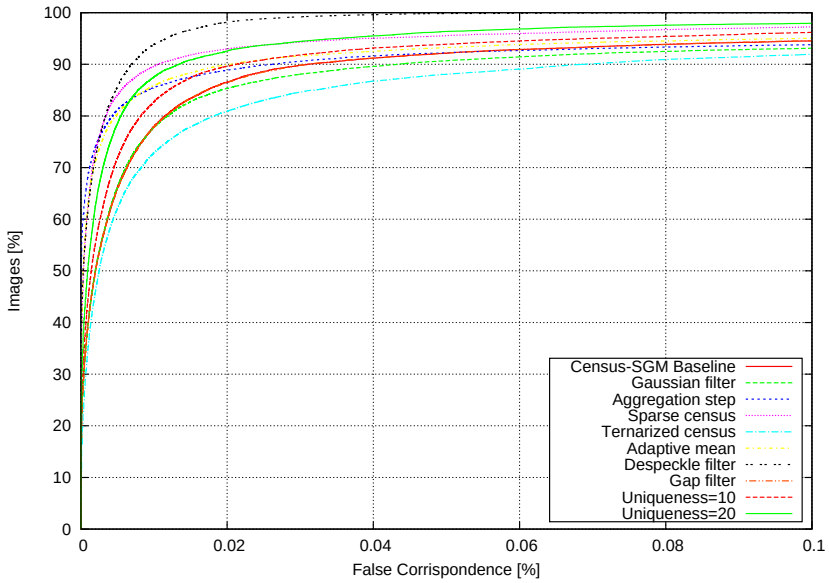


Figura 2.8: Performance NFC dei filtri isolati..

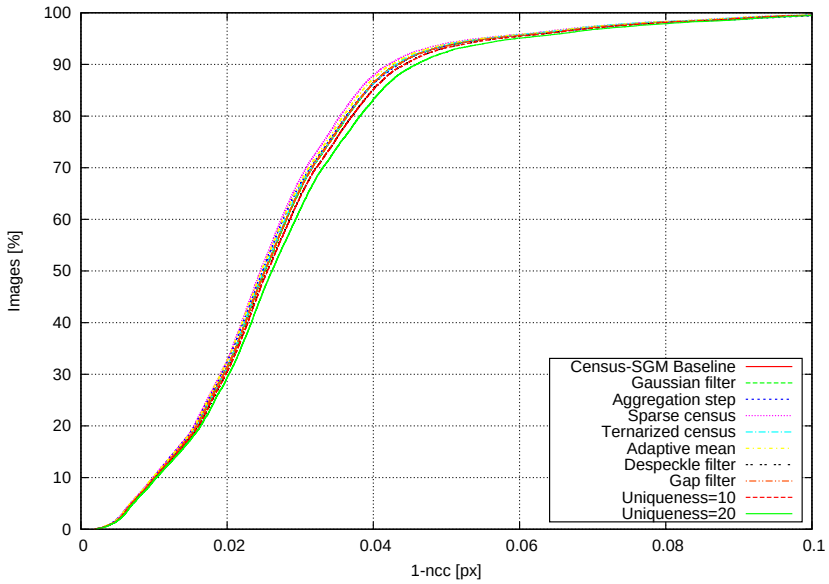
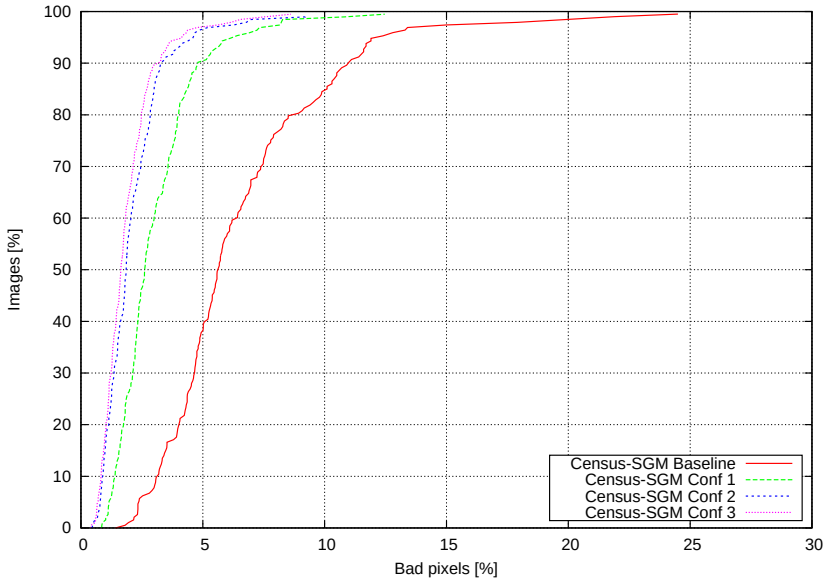


Figura 2.9: Performance NCC dei filtri isolati.

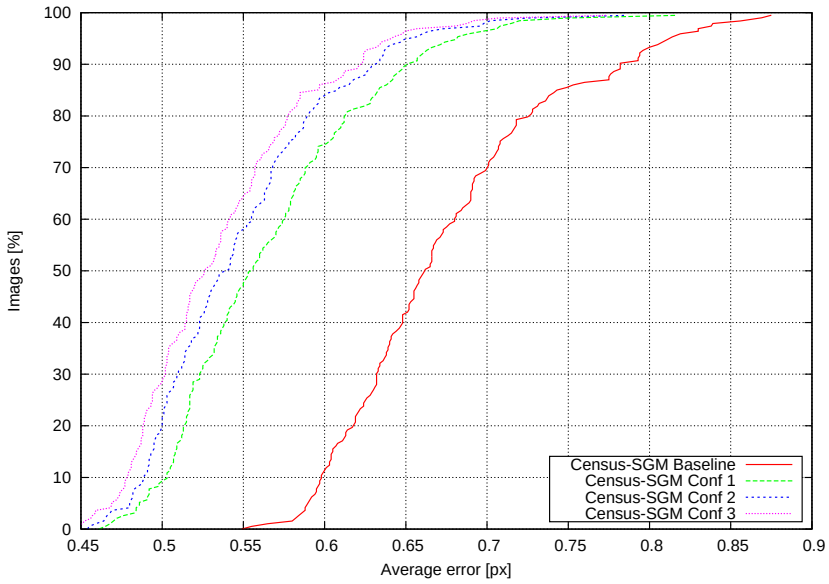
2.2.2 Filtri compositi

Combinando differenti filtri è possibile ottenere risultati migliori rispetto ad usarli singolarmente. La Fig. 2.10 mostra i risultati per le tre configurazioni di *Census-SGM*, descritte in Tab. 2.1, ottenuti con il test LGT. Osservando al 90-esimo percentile nella Fig. 2.10-a si ha un miglioramento intorno al 6% nel numero di pixel errati per le configurazioni 2 e 3; per queste anche l'errore medio dei pixel cala di circa 0.175 px (Fig. 2.10-b). Questo miglioramento, tuttavia, si paga nel calo della densità, come da Fig. 2.10-c: la configurazioni 3, al 90-esimo percentile, ha un densità di circa 58%, mentre la 2 è migliore a circa 65%; invece il metodo base, ha una densità vicina al 78%, con un 12% di pixel errati. Le configurazioni 2 e 3 ottengono migliori risultati anche nel test NFC, riducendo gli errori che cadono all'interno della traiettoria del veicolo, ad un numero trascurabile. Il test NCC, sembra essere in controtendenza, ma come spiegato nella Sez. 2.2.1, è influenzato da altri fattori esterni.

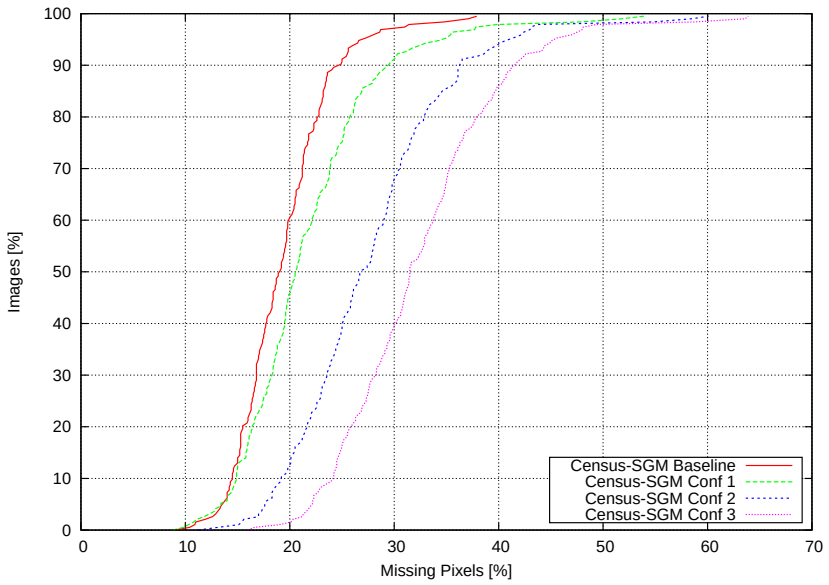


(a)

Figura 2.10: (cont.) Performance LGT dei filtri compositi: a) percentuale pixel errati



(b)



(c)

Figura 2.9: b) errore medio, c) densità della mappa di disparità.

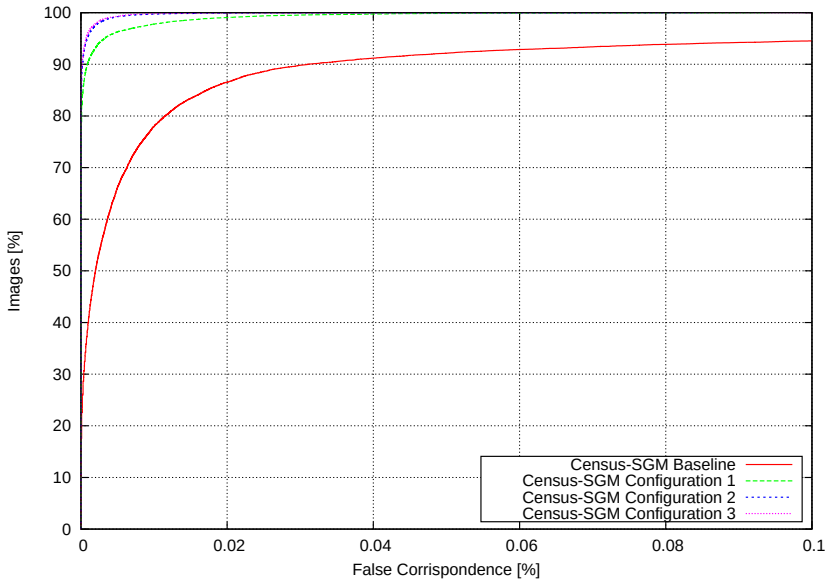


Figura 2.10: Performance NFC dei filtri compositi.

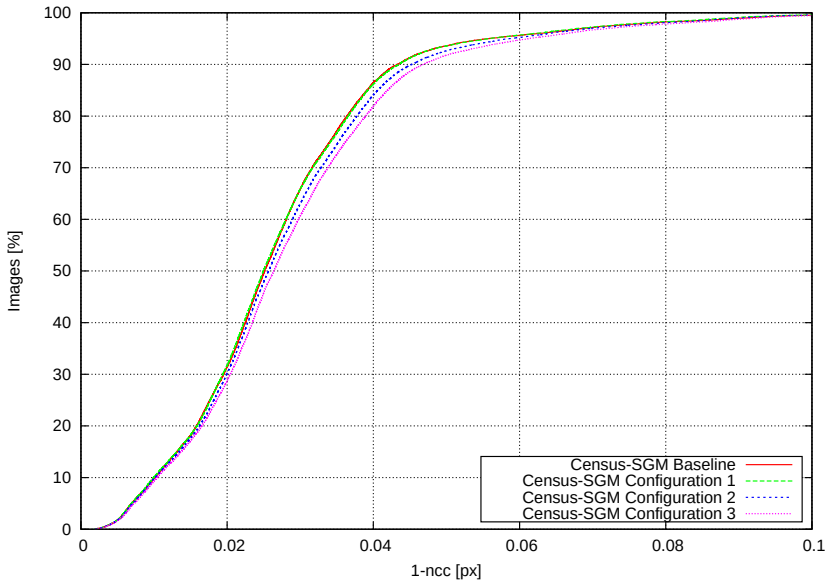
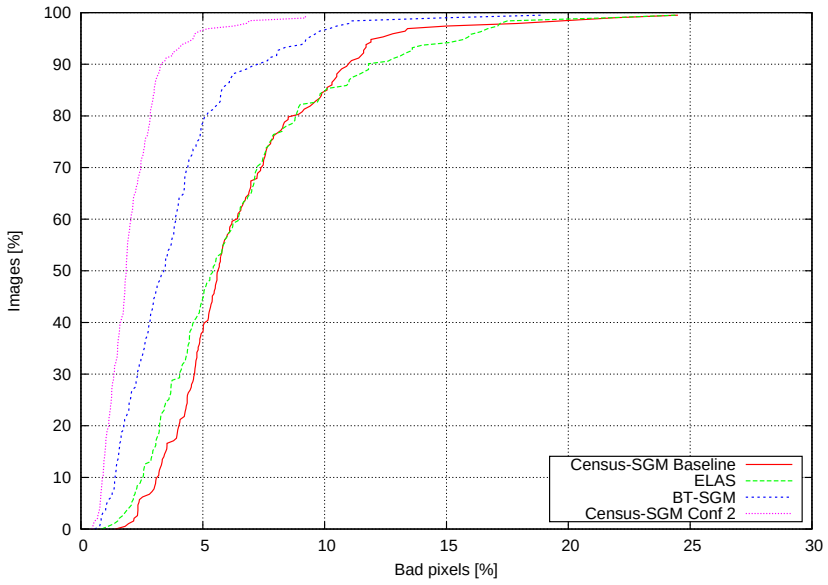


Figura 2.11: Performance NCC dei filtri compositi.

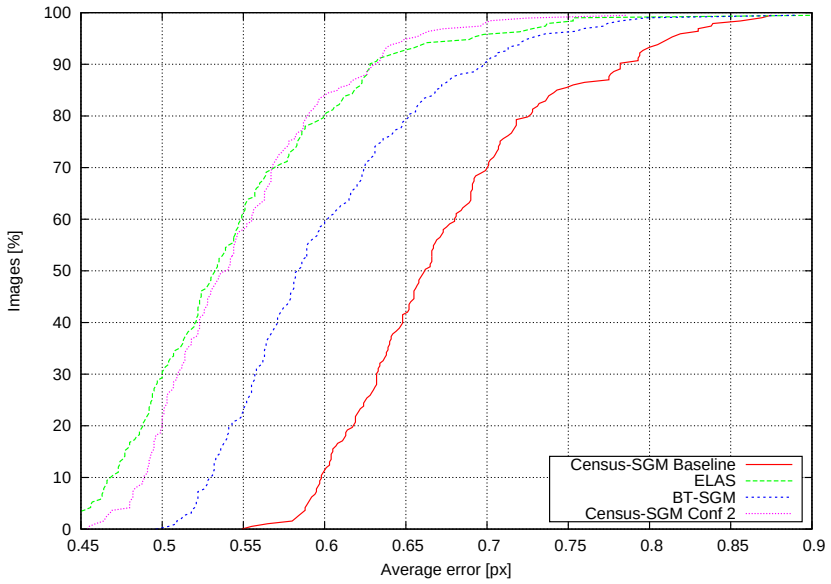
2.2.3 Confronto tra algoritmi

Census-SGM configurazione 2 è stato scelto come miglior compromesso tra qualità di ricostruzione e densità della mappa; i seguenti grafici mostrano le sue performance confrontate con quelle degli altri due approcci descritti nella Sez. 1.2. Il test LGT (Fig. 2.12) mostra un calo dei pixel errati di circa 7.5% al 90-esimo percentile rispetto alla configurazione base, e circa il 4.5% comparandolo al BT-SGM. L'errore medio è anche ridotto di 0.15 px, con Census-SGM conf. 2, in linea con i valori ottenuti da ELAS. La percentuale di pixel mancanti raggiunge circa il 35%, ovvero 12% in più del setup base; tuttavia, la maggior parte di punti non ricostruiti è dovuta ad una migliore rimozione degli errori ed è un comportamento aspettato.

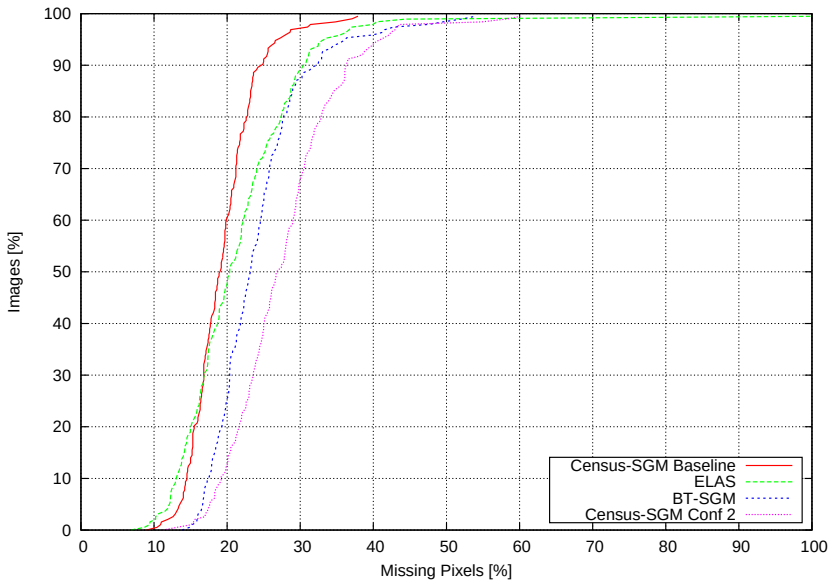


(a)

Figura 2.12: (cont.) Performance LGT degli algoritmi: a) percentuale pixel errati



(b)



(c)

Figura 2.11: b) errore medio, c) densità della mappa di disparità.

Il test NFC produce risultati in linea con quelli ottenuti dal test LGT, significa che Census-SGM configurazione 2 è considerevolmente e misurabilmente migliore rispetto agli altri approcci, è quindi l'algoritmo vincitore in questo confronto.

I risultati NCC per ELAS e BT-SGM sono abbastanza simili e migliori del Census-SGM base (come aspettato), ma il piazzamento del Census-SGM configurazione 2 sembra sospetto (ovvero peggiore del Census-SGM configurazione base). Per questa ragione, in dati ottenuti da questo test dovranno essere studiati più attentamente prima di poter essere utilizzati come affidabili indicatori delle performance degli algoritmi.

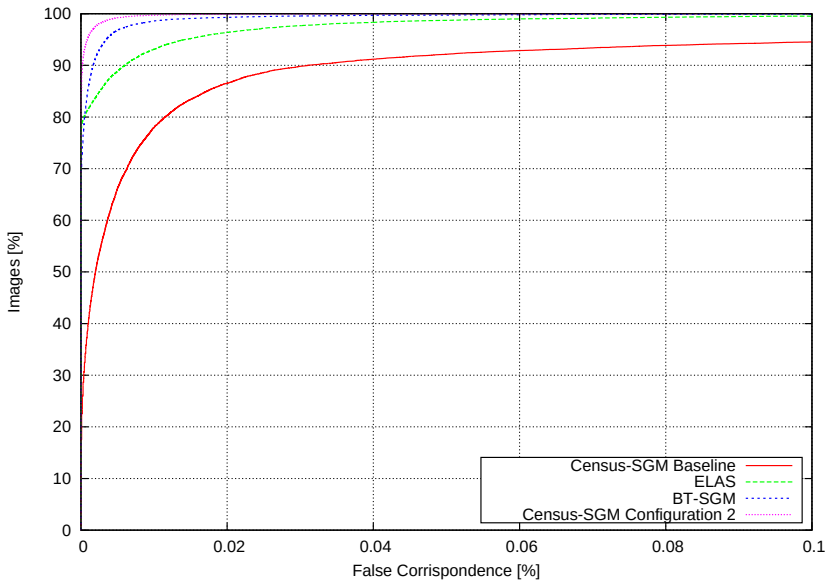


Figura 2.12: Performance NFC algoritmi.

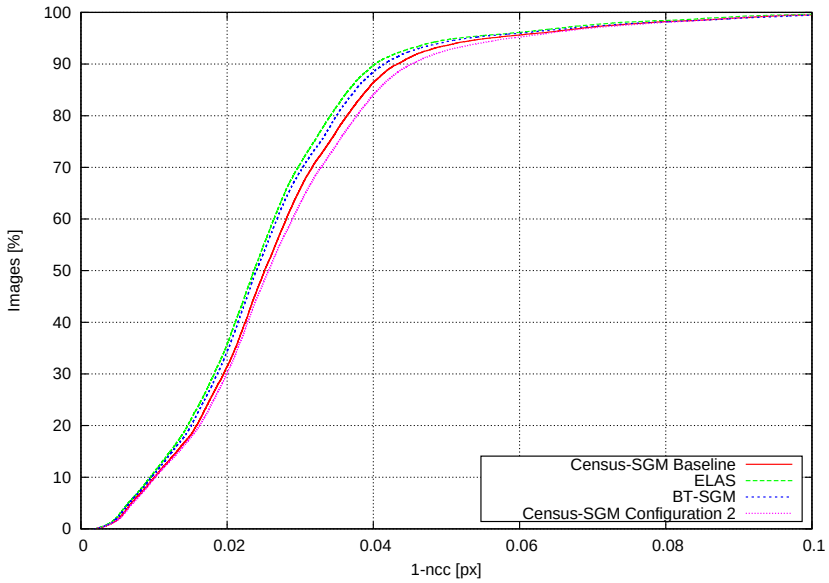


Figura 2.13: Performance NCC algoritmi.

2.3 Conclusioni

I test condotti finora hanno confermato quantitativamente come le strategie di filtraggio mirate possono ridurre notevolmente la quantità di pixel errati, calcolati durante la ricostruzione stereo, ma anche migliorare la precisione dei risultati.

In particolare, Census-SGM configurazione 2 descritto nel paragrafo 2.2 riduce il numero di pixel difettosi del 7,5% al 90-esimo percentile, migliorando nel contempo l'errore medio di 0,15 px, rendendo tale approccio quello migliore tra quelli testati.

Questa valutazione è servita anche a convalidare le strategie impiegate. Mentre il LIDAR-based ground truth (vedi Sez. 2.1.1) è molto efficace nel produrre statistiche attendibili, rimane piuttosto costoso da svolgere, sia in termini di attrezzature necessarie che per il trattamento manuale a posteriori che deve essere eseguito per produrre ogni singolo frame.

In alternativa, la conoscenza del movimento del veicolo (Sez. 2.1.2) può essere sfruttata per identificare una porzione dei punti indebitamente ricostruiti. Il vantaggio di questo approccio è che può essere efficacemente utilizzato per valutare il comportamento di un algoritmo su grandi insiemi di dati, coprendo così una vasta gamma di condizioni ambientali. D'altra parte, la porzione di spazio che può essere controllato è limitata alla zona di fronte al veicolo in movimento, che spesso è la più critica, e può introdurre una caratterizzazione particolare nelle statistiche risultanti.

L'utilizzo di una terza camera per la valutazione (Sez. 2.1.3) è concettualmente interessante, ma in pratica ha dimostrato di produrre scarsi risultati. Saranno necessari ulteriori test per valutare la sua reale efficacia in scenari reali.

Capitolo 3

Architettura hardware del Sistema

Selezionare una opportuna architettura di elaborazione per un sistema di visione stereoscopica è di importanza critica perché influenza pesantemente le caratteristiche del sistema stesso quali performance, costi di produzione e consumo di potenza.

Ci sono molti tipi diversi di sistemi di elaborazione:

- **CPU based**: elaboratori tradizionali caratterizzati da CPU molto potenti grazie ad un numero di core sempre maggiore e frequenze di funzionamento elevate. Integrano grandi quantità di memoria RAM per l'elaborazione con una elevata banda passante. Le moderne CPU offrono un set di istruzioni SIMD [1] molto adatte agli algoritmi stereo, permettendo di eseguire parallelamente una stessa operazioni su più dati. Sfortunatamente non sono affatto adatti ai sistemi embedded perchè onerosi di potenza, ingombranti, e generalmente poco affidabili se utilizzati in contesti molto dinamici.

- **GPU based**: questi sistemi generalmente si affiancano ai precedenti, dei quali ereditano pregi e difetti, aggiungendo un'architettura caratterizzata da un elevato parallelismo, che sfrutta la disponibilità di un elevato numero di processori (centinaia) sui quali suddividere parallelamente il lavoro dell'algoritmo. A pa-

[1] Single Instruction Multiple Data

rità di potenza, rispetto alle CPU permettono di eseguire un'elevata quantità di calcoli parallelamente.

- **Embedded RISC CPU**: caratterizzati da bassi consumi e minori prestazioni rispetto le CPU tradizionali, storicamente utilizzati principalmente come sistemi elaborazione per sistemi di controllo e dispositivi portatili; negli ultimi anni hanno incontrato un notevole aumento delle prestazioni, principalmente dovuto al diffondersi dell'elettronica di consumo (es. gli smartphone). Sfortunatamente
- **DSP**:
- **FPGA**:
- **SoC**:

Come sistema di elaborazione embedded, per sviluppare il sistema e stata scelta una moderna architettura SoC (System-on-Chip) che integra in un unico componente una FPGA e un microprocessore ARM dual-core, soluzione che semplifica l'architettura hardware garantendo alte prestazioni sui canali di comunicazione tra l'elaborazione di basso livello su FPGA e il processore.

3.1 ZYNQ

I dispositivi della famiglia Xilinx Zynq™, rappresentano una moderna architettura SoC (System-on-Chip) che integra una logica programmabile FPGA (PL) e un sistema a microprocessore (PS) basato su un processore dual-core ARM® Cortex™-A9, nello stesso chip fisico (Fig. 3.1). La comunicazione tra PL e PS avviene attraverso una serie di interfacce basate sullo standard AXI; in particolare, quattro canali ad elevate prestazioni (High Performance AXI channels: HP0, HP1, HP2 e HP3) sono stati utilizzati per gestire il trasferimento dei dati temporanei in fase di elaborazione tra la PL e la memoria DDR in entrambe le direzioni. Ogni canale è full-duplex ed ha una banda teorica massima di 1.2 GB/s in ogni direzione, con il supporto per trasferimenti a pacchetti (burst) in multipli di parole da 4 o 8 byte. La memorizzazione dei risultati

finali delle elaborazioni (immagini rettificate e mappa di disparità), avviene utilizzato un canale dedicato di comunicazione detto Accelerated Coherence Port (ACP) per garantire la coerenza della memoria cache dal processore PS (utilizzato per i compiti di più alto livello); sullo stesso canale vengono anche trasferite le immagini di census C_L and C_R che vengono scambiate tra i moduli che implementano il sistema.

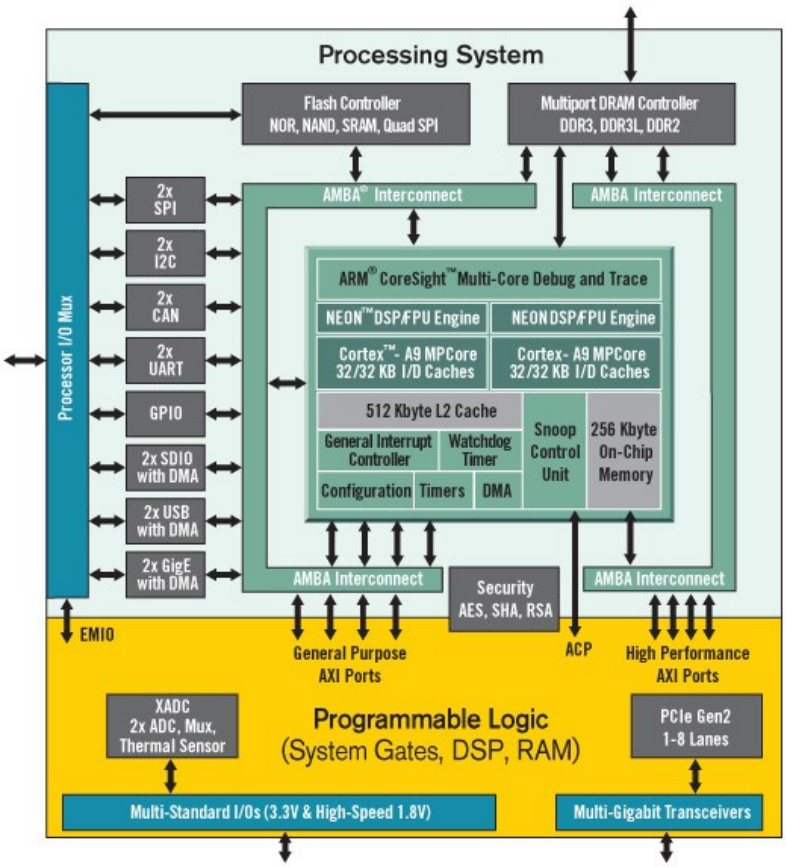


Figura 3.1: Architettura dei dispositivi ZYNQ.

Lo Zynq™ SoC integra inoltre un elevato numero di periferiche di input/output per potersi adattare ai differenti campi di utilizzo di un possibile sistema embedded. In particolare quelle utilizzate in questa architettura di visione stereoscopica sono:

- GigE – interfaccia di rete Gigabit Ethernet usata per spedire il flusso dei risultati dell'elaborazione ovvero le immagini rettificate e la mappa di disparità;
- I^2C – utilizzato per controllare i due sensori video e per interfacciarsi con l'unità IMU;
- QSPI Flash – utilizzata per memorizzare il bitstream del PL, i file delle LUT di rettificazione, il file di configurazione, l'applicazione di streaming o elaborazione e l'immagine del sistema operativo Linux;
- I/O pin – utilizzati per fornire funzionalità aggiuntive quali la possibilità di sincronizzare il sistema con eventi esterni, oppure per fornire all'esterno informazioni di alto livello (es. presenza di ostacoli);
- CAN – in futuro si prevede di utilizzare questa linea di comunicazione, molto usata in campo industriale e automotive, ai fini di configurazione oppure di per output di alto livello (es. posizione degli ostacoli);
- UART – interfaccia seriale standard utilizzata ai fini di debug.

Il sistema di visione stereoscopica, sviluppato in questo progetto di dottorato, è basato sul modello di ZYNQ Z-7020, scelto anche per la disponibilità in versione automotive, perché offre il miglior compromesso tra performance, risorse FPGA e costo.

3.2 Sensori video

Due sensori Aptina MT9V034 CMOS 752×480 in bianco e nero (Fig. 3.2), sono stati selezionati per l'acquisizione delle immagini, perchè rappresentano una soluzione economica, di facile integrazione (interfacciamento tramite bus parallelo) e di classe automotive. Le caratteristiche principali di tali sensori sono descritte in Fig. 3.3.

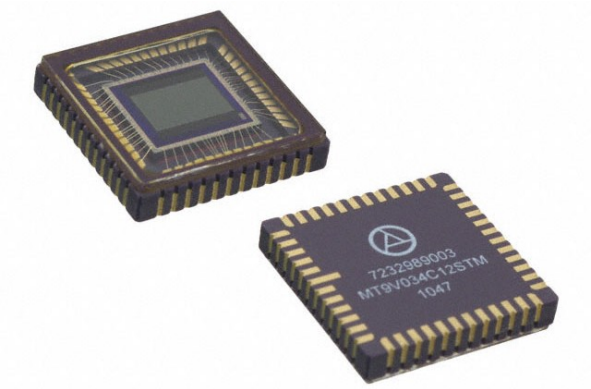


Figura 3.2: Sensore video Aptina MT9V034.

Parameter	Value
Optical format	1/3-inch
Active imager size	4.51mm(H) x 2.88mm(V) 5.35mm diagonal
Active pixels	752H x 480V
Pixel size	6.0 x 6.0µm
Color filter array	Monochrome or color RGB Bayer pattern
Shutter type	TrueSNAP™ Global shutter
Maximum data rate master clock	27 Mp/s 27 MHz
Full resolution	752 x 480
Frame rate	60 fps (at full resolution)
ADC resolution	10-bit column-parallel
Responsivity	4.8 V/lux-sec (550nm)
Dynamic range	>55dB linear; >110dB in HDR mode
Supply voltage	3.3V ±0.3V (all supplies)
Power consumption	<160mW at maximum data rate (LVDS disabled); 120µW standby power at 3.3V
Operating temperature	–30°C to +70°C ambient
Packaging	48-pin CLCC

Figura 3.3: Caratteristiche del sensore video Aptina MT9V034.

3.3 Circuito elettronico

Durante questo progetto di ricerca si è anche affrontato lo studio e progettazione del circuito elettronico del sistema di visione artificiale, in modo da testare l'architettura sviluppata su un prototipo funzionante. Le parti principali che compongono lo schema del circuito sono allegate in appendice A. Le caratteristiche principali che hanno guidato la progettazione del prototipo realizzato sono le seguenti:

- baseline delle telecamere 15 cm
- tensione di alimentazione compresa tra 10 e 36 V
- circuito di alimentazione analogica separato per i sensori video, per ridurre i disturbi di acquisizione
- interfaccia Gigabit Ethernet per l'invio via rete dei dati elaborati
- interfaccia CAN per configurare e monitorare lo stato del sistema
- sistema inerziale a 9 assi.

In Fig. 3.4 è mostrato il circuito stampato progettato.

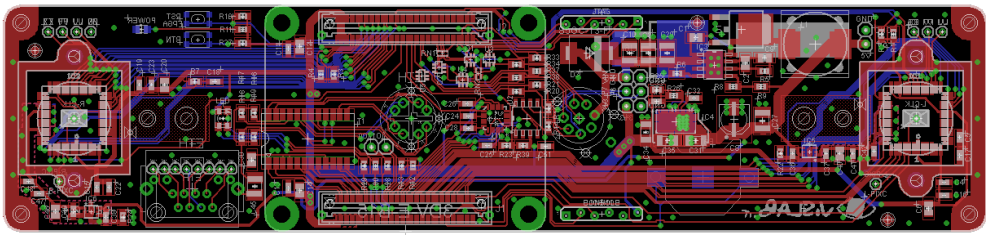


Figura 3.4: Schema del circuito stampato progettato per il sistema di visione stereoscopico.

Capitolo 4

Architettura e progettazione del sistema di elaborazione

4.1 Architettura

Di seguito viene illustrato come la generica architettura di un sistema stereoscopico, presentata nella sezione 1.3 e illustrata in Fig. 1.14, sia stata mappata sull'architettura hardware scelta per questo progetto, presentata nel capitolo 3.

Lo schema del sistema risultante è illustrato in Fig. 4.1, dove si possono individuare due macro blocchi, il cui funzionamento interno è descritto più dettagliatamente nelle sezioni successive del capitolo, quello che si occupa principalmente della rettificazione (Sec. 4.1.1) e quello che implementa l'algoritmo di minimizzazione SGM (Sec. 4.1.3).

Il principio alla base dell'algoritmo SGM, ovvero la fase di aggregazione dei costi, come è stato descritto nella sezione 1.2, viene eseguita su 8 percorsi, 4 ortogonali e 4 diagonali sul piano X,Y del cubo dei costi. Il cubo dei costi aggregati ha una dimensione considerevole, pari alla risoluzione dell'immagine moltiplicata per il valore massimo di disparità supportato ($636 \times 476 \times 128$ Byte ≈ 39 MByte). Tale quantità di memoria non è possibile trovarla in nessun dispositivo FPGA, nemmeno quelli di fascia più alta, quindi bisogna per forza appoggiarsi ad una memoria esterna DDR.

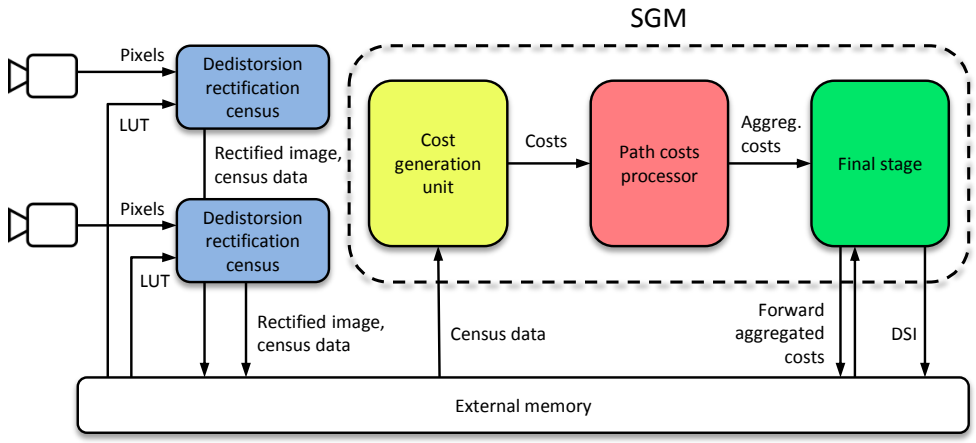


Figura 4.1: Architettura hardware. Per semplicità, sono state omesse le FIFO intermedie, tra i singoli blocchi, e le BlockRAM usate come buffer temporanei.

Questo comporta che i dati non siano direttamente accessibili dalla FPGA, ma vanno caricati di volta in volta dalla DDR nelle BlockRAM. Sfortunatamente l'aggregazione lungo gli 8 percorsi richiede di iniziare ad aggregare da posizioni opposte nel cubo dei costi, quindi bisognerebbe accedere contemporaneamente a dati che non possono risiedere fisicamente nelle BlocKRAM della FPGA.

Il processo di aggregazione dei costi è stato quindi separato in due distinti passi, a quali si farà riferimento come *fase di andata* e *fase di ritorno*; durante la fase di andata vengono elaborati i percorsi corrispondenti a 0°, 45°, 90° and 135°, mentre nella fase di ritorno si coprono i percorsi a 180°, 225°, 270° and 315°, come mostrato in Fig. 4.2. Inoltre, i 4 passi della *fase di andata* sono sommati insieme prima di essere trasferiti alla memoria esterna DDR in modo da ridurre la banda passante richiesta. Tutti e 4 i canali AXI HP sono utilizzati per scrivere e successivamente rileggere i costi generati nella *fase di andata*, in quanto i dati prodotti ad ogni ciclo di clock dallo stadio di generazione dei costi (32 byte) sono esattamente 4 volte la larghezza di parola dei canali (64 bit). Il canale ACP, invece, gestisce tutti i rimanenti trasferimenti

(caratterizzati da una minore larghezza di banda): le lookup table di rettificazione, le trasformate di Census, le immagini rettificate sinistra e destra e infine la mappa delle disparità.

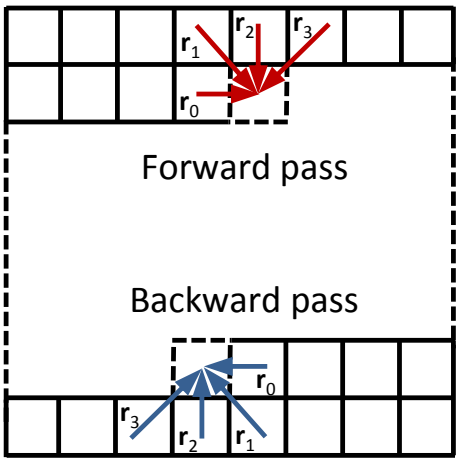


Figura 4.2: Gli 8 percorsi dell'aggregazione di SGM suddivisi in fase di andata e di ritorno.

Infine in Fig. 4.3 è mostrato quali unità sono attive in ogni preciso momento durante l'elaborazione di un frame. La lettura dei sensori video, la rettificazione e la trasformata di Census sono sincrone ed eseguite progressivamente (ogni unità fornisce i dati alla successiva). I dati prodotti, attraverso uno scambio in memoria esterna, permettono, parallelamente a queste elaborazioni, la generazione dei costi e l'elaborazione della *fase di andata* di SGM e vengono trasferite su rete le immagini rettificate sinistra e destra. La *fase di ritorno* e la generazione della mappa delle disparità inizia dopo il completamento della *fase di andata* ed eseguita in parallelo con lo streaming della mappa stessa (DSI) e la fase di esposizione del sensore per il frame successivo.

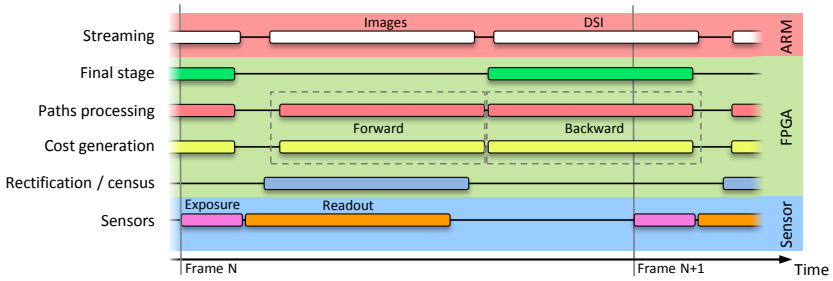


Figura 4.3: Diagramma temporale del sistema. I colori dei vari passi, corrispondono a quelli delle corrispondenti unità in Fig. 4.1.

4.1.1 Rettificazione

La rettificazione e la trasformata di Census sono eseguite indipendentemente sulle immagini catturate da ciascuna telecamera.

L'operazione di rettificazione consiste nell'inversione della deformazione introdotta dalla distorsione delle lenti e dagli errori di allineamento dei sensori video (questo fattore può essere limitato con un'accurata progettazione meccanica del sistema, ma rimane sempre presente entro certi limiti), i quali portano ad uno sfasamento dei pixel delle immagini originali registrate dalle telecamere. Per correggere questo effetto, viene utilizzata una look-up table (LUT), che fornisce, per ogni pixel dell'immagine rettificata, le coordinate del corrispondente pixel dell'immagine originale dalla telecamera. Per aumentare la precisione, evitando di introdurre artefatti nell'immagine rettificata, le coordinate sono frazionali e quindi richiedono di applicare un'interpolazione tra i quattro pixel dell'immagine originale intorno a tali coordinate. Come accennato nel capitolo iniziale, viene utilizzata un'interpolazione bilineare descritta dall'equazione (4.1).

$$\begin{aligned} R(\mathbf{p}) = (1-\beta)((1-\alpha)I(x_i,y_i) + \alpha I(x_i+1,y_i)) + \\ \beta((1-\alpha)I(x_i,y_i+1) + \alpha I(x_i+1,y_i+1)) \end{aligned} \tag{4.1}$$

Per eseguire questa operazione, il sistema legge i pixel provenienti dalla telecamera e li memorizza in un buffer circolare. La dimensione di questo buffer, definisce la capacità massima di dedistorsione del sistema. Date le limitate risorse di BlockRAM della FPGA, e contando quelle già allocate per le altre unità di elaborazione del sistema, tale buffer ha un'estensione di 128 linee; quindi un pixel rettificato può essere localizzato non più di 64 linee sopra o sotto il corrispondente pixel originale. Questa quantità di righe è sufficientemente ampia per gestire ottiche con una lunghezza focale maggiore o uguale a 2.8mm. L'immagine originale viene quindi memorizzata in un buffer di 128 linee centrate attorno alla posizione corrente; una serie di indici permette di mantenere traccia della locazione di memoria attualmente utilizzata, permettendo di usare il buffer in modalità circolare andando progressivamente a sovrascrivere le linee più vecchie con quelle nuove acquisite.

I dati della LUT sono memorizzati nella memoria DDR esterna utilizzando un formato compresso e incrementale. Dopo uno studio delle LUT necessarie per rettificare differenti tipi di ottiche (da focali di 2.4mm in su) si è calcolato che il massimo scostamento orizzontale e verticale tra i pixel originali di due pixel rettificati, non supera mai il valore di 1.75 pixel. Questo significa che i dati necessari per elaborare il pixel rettificato successivo in orizzontale, non si discostano mai più di 1.75 pixel interi sull'immagine originale sia in X che in Y. Essendo sia X che Y funzioni continue, che da un pixel all'altro cambiano nella parte intera di +/-1 pixel, 2 bit sono sufficienti. Per ogni pixel viene quindi memorizzata nella LUT la sola differenza rispetto al pixel precedente (in orizzontale) sia per la parte X che Y. Inoltre si è ritenuto sufficiente una precisione di 2 bit per la parte di interpolazione decimale.

Il risultato occupa un solo byte per ogni pixel, così suddiviso:

$$X_i \quad X_i \quad X_d \quad X_d \qquad Y_i \quad Y_i \quad Y_d \quad Y_d \tag{4.2}$$

con

- X_i X_i = 2 bit parte intera sulla X
- X_d X_d = 2 bit parte decimale sulla X

- Y_i Y_i = 2 bit parte intera sulla Y
- Y_d Y_d = 2 bit parte decimale sulla Y.

L'unico caso da gestire in modo particolare, è rappresentato dal primo pixel di ogni riga (non ha senso fare la differenza rispetto l'ultimo pixel della riga precedente) che viene memorizzato in coordinate assolute X, Y rispetto all'immagine originale.

Dopo che i dati LUT di ogni pixel rettificato sono caricati, il sistema calcola l'indirizzo corrispondente sull'immagine originale e legge i corrispondenti 4 pixel attorno ad esso, dal buffer circolare. L'interpolazione di questi 4 valori viene eseguita con un'aritmetica *fixed point* utilizzando solo due cifre decimali. Date queste esigenze particolari e limitate, il blocco di logica combinatoria che si occupa dell'interpolazione è stato progettato per calcolare il risultato senza la necessità di costosi moltiplicatori hardware.

Per poter eseguire l'operazione di interpolazione in un solo ciclo di clock, bisogna accedere parallelamente ai 4 pixel dell'immagine originale. Il buffer circolare da 128 linee, se venisse implementato come un unico blocco di BlockRAM, permetterebbe di accedere ad unico valore ad ogni ciclo di clock (più esattamente a 2 valori se si sfruttasse la doppia porta di accesso). Nell'implementazione del sistema si è quindi pensato di suddividere i pixel dell'immagine originale su 4 buffer circolari differenti, così suddivisi:

- RAW_00 : memorizza i pixel pari delle righe pari (es. (0,0) (0,2) (0,4) ecc.)
- RAW_01 : memorizza i pixel dispari delle righe pari (es. (0,1) (0,3) (0,5) ecc.)
- RAW_10 : memorizza i pixel pari delle righe dispari (es. (1,0) (1,2) (1,4) ecc.)
- RAW_11 : memorizza i pixel dispari delle righe dispari (es. (1,1) (1,3) (1,5) ecc.)

In questo modo ad ogni ciclo di clock vengono caricati i dati dei 4 pixel necessari per calcolare l'equazione di interpolazione (4.1) e alla stessa frequenza viene prodotto un nuovo valore di pixel rettificato.

I valori dei pixel rettificati vengono inviati al blocco successivo e anche scritti nella memoria esterna DDR, per rendere disponibile l'immagine rettificata per lo streaming o utilizzi successivi.

4.1.2 Trasformata di Census

I pixel rettificati sono memorizzati in un buffer temporaneo per calcolare la trasformata di *Census*. Siccome la finestra utilizzata per la trasformata ha dimensioni 5×5, richiede di accedere solamente alle 2 linee sopra e sotto a quella corrente. Quindi è necessario memorizzare temporaneamente in un buffer le ultimi 5 righe calcolate dell'immagine rettificata, utilizzando un approccio a finestra mobile in modo analogo a quanto fatto per l'immagine originale prima della rettificazione. Per eseguire la trasformata di Census 5×5, è necessario accedere contemporaneamente a 25 pixel dell'immagine rettificata, ma le BlockRAM offrono solamente un accesso in parallelo a 2 soli valori. Per ovviare a questo problema è stata usata la seguente strategia, che richiede di accedere a soli 4 nuovi valori ogni volta (quindi utilizzando due sole BlockRAM a doppia porta).

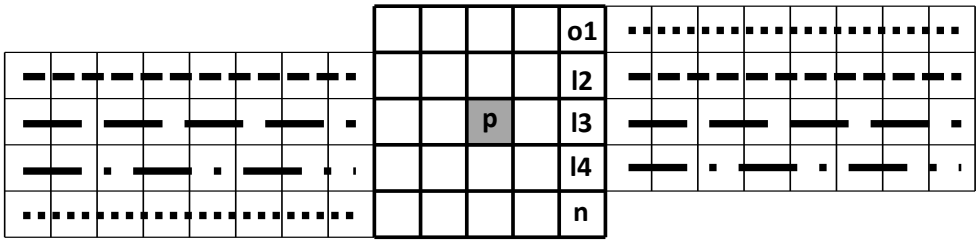


Figura 4.4: Trasformata di Census 5×5 a finestra mobile.

Come rappresentato in Fig. 4.4 viene utilizzato uno shift register 5×5 per memorizzare i pixel nell'intorno di **p** di cui si sta calcolando la trasformata di Census. Le colonne si questo shift register vengono traslate al passaggio al pixel successivo e quella nuova è caricata con 4 pixel letti dal buffer delle righe (**o1**, **l2**, **l3**, **l4**) più il nuovo pixel rettificato **n**. Inoltre questo valore viene memorizzato nel buffer delle

righe andando a sostituire il pixel in posizione **o1**. Sfruttando questo stratagemma è possibile ridurre a 4 righe la dimensione del buffer temporaneo, in quanto la prima e la quinta riga vengono gestite come una unica, sostituendo i vecchi pixel, non più necessari perché già caricati nello shift register, con quelli nuovi rettificati.

Un semplice circuito combinatorio calcola la trasformata di Census, restituendo una stringa di 24bit che viene successivamente memorizzata nella memoria esterna DDR e resa disponibile per la successiva fase di calcolo dei costi nel blocco SGM.

4.1.3 Semi-Global Matching

Il blocco Semi-Global Matching (SGM) inizia caricando i costi di Census memorizzati nella memoria esterna DDR e genera la mappa di disparità, utilizzando una *fase di andata* seguita da una *fase di ritorno*. Il cubo dei costi intermedio calcolato durante la fase di andata, è progressivamente salvato nella memoria esterna DDR, mentre il calcolo dei costi (dal confronto delle trasformate di Census) è ricalcolato in entrambi le fasi, poiché meno oneroso che salvarlo temporaneamente. L'algoritmo implementato analizza l'immagine riga per riga, dal pixel sinistro a quello destro all'interno della riga stessa.

L'implementazione di SGM è suddivisa funzionalmente in tre moduli separati, internamente collegati attraverso delle FIFO (strutture dati Firs-In-First-Out) e comunica con gli altri moduli solo attraverso la memoria DDR esterna. I moduli, che sono descritti più dettagliatamente in seguito, calcolano rispettivamente:

- I costi iniziali $C(\mathbf{p},d)$, ovvero la distanza di Hamming fra le trasformate di Census;
- Il costo aggregato dei percorsi $L_{\mathbf{r}}(\mathbf{p},d)$ per i 4 percorsi della fase di andata e quella di ritorno;
- Il costo dei cubi aggregati per tutti i percorsi e da questo la mappa di disparità a costo minimo.

Strategia di parallelizzazione dell'algoritmo SGM

Le architetture FPGA, sono utilizzate solitamente per implementare degli algoritmi massivamente paralleli. Nel caso dell'algoritmo SGM la fase più computazionalmente onerosa che ci interessa parallelizzare è quella di aggregazione dei costi secondo l'Eq. 4.3. Tale operazione deve essere eseguita riga per riga, per ogni pixel della riga, per ogni valore di disparità e per ogni singolo percorso.

Esistono quindi tre possibili strategie di parallelizzazione:

- calcolare parallelamente i costi aggregati per diversi pixel per un dato percorso e una data disparità;
- calcolare parallelamente i costi aggregati per i diversi valori di dispartità, per un singolo pixel.
- calcolare parallelamente i costi aggregati lungo i diversi percorsi, per un singolo pixel;

La prima opzione non permetterebbe di raggiungere un parallelismo effettivo, in quanto il costo aggregato di un determinato pixel dipende anche dal costo aggregato del pixel precedente lungo il cammino orizzontale; si potrebbe quindi ipotizzare una struttura a pipeline tra più righe, ma non sarebbe facilmente estendibile ad un elevato numero di pixel a causa delle limitate risorse disponibili.

La seconda opzione richiede di calcolare parallelamente tutti i costi aggregati per un dato pixel (128 valori di dispartià). Tali costi andrebbero memorizzati temporaneamente per tutti i pixel di una riga per tutti i percorsi di aggregazione, in quanto necessari successivamente nell'elaborazione dei percorsi dei pixel della riga successiva, bisogna quindi attentamente valutare la quantità di memoria disponibile a bordo della FPGA per memorizzare tali dati.

La terza opzione invece, non presenta nessuna controindicazione, in quanto ciascun percorso di aggregazione non dipende dagli altri, ma solo dai dati precedenti di tale percorso (in particolare i costi del pixel precedente e il minimo trovato lungo tutto il percorso).

La strategia di parallelizzazione vincente può risultare dalla combinazione delle possibili parallelizzazioni. Intuitivamente verrebbe da pensare di eseguire in parallelo il calcolo dei 128 costi per i 4 percorsi di ogni singolo pixel, ma vedremo che non sarà quella vincente ne in termini di velocità ne di risorse utilizzate. Per valutare quale sia la migliore strategia di parallelizzazione, è stato necessario svolgere un attento studio della quantità di memoria richiesta per memorizzare i dati temporanei e anche analizzare la complessità delle reti combinatorie risultanti, che influisce direttamente sulla frequenza di funzionamento del sistema.

La risoluzione video, che si è scelta per quest'architettura, è 640×480 pixel; escludendo i 2 pixel sui bordi esterni necessari per la trasformata di Census, la risoluzione utile della mappa di dispartià risultante sarebbe 636×476. Siccome la strategia di aggregazione dei costi di SGM opera per righe sulla immagine di Census, il dato che interessa in particolare è 636 pixel per riga.

I dispositivi Xilinx ZYNQ 3.1, scelti per l'architettura del sistema, mettono a disposizione delle BlockRAM da 36 Kbit con doppia porta di accesso sia in lettura che scrittura, configurabili con parole di diversa ampiezza (32k x 1, 16k x 2, 8k x 4, 4k x 8, 2k x 16, 1k x 32).

La prima strategia di parallelizzazione che si è ipotizzata, consiste nel calcolare contemporaneamente i costi delle 128 disparità per tutti e 4 i cammini di aggregazione. Per memorizzare i dati temporanei necessari al processo di aggregazione, si necessita dei seguenti vettori:

- *dcost[128][636]*: i costi di matching per ogni pixel per ogni valore di disparità;
- *path[4][128][636]*: i costi di ogni percorso (dei 4 possibili), per ogni pixel per ogni valore di disparità;
- *aggr[128][636]*: i costi aggregati per ogni pixel per ogni valore di disparità;
- *min[4][636]*: i costi minimi per ogni percorso (dei 4 possibili), per ogni pixel fra tutte le possibili disparità.

Senza entrare nel dettaglio dell'allocazione delle singole BlockRAM, in base alla larghezza dei dati e alle diverse modalità di accesso in lettura e scrittura, il totale delle

BlockRAM, solo per questa fase, sarebbe 225, valore decisamente maggiore delle 140 disponibili nel modello ZYNQ Z7020 scelto per questa architettura. Suddividendo i dati in modo opportuno sarebbe possibile ridurre le BlockRAM richiese a sole 161, che comunque non rientrerebbe nelle specifiche. Inoltre non bisogna dimenticare che anche tutti gli altri blocchi funzionali dell'architettura avranno una loro esigenza di BlockRAM.

Il calcolo dei costi di 128 disparità in parallelo introduce un ulteriore problema nel momento in cui se ne volesse calcolarne il valore minimo (valore utilizzato per il pixel successivo lungo il percorso): ne risulterebbe una rete combinatoria molto estesa, quindi con tempi di propagazione elevati e bassa frequenza di elaborazione. Per ovviare a questo problema bisognerebbe suddividere tale rete in più cicli di clock (almeno 3), ma non si potrebbe trarre vantaggio dalla struttura a pipeline che si otterrebbe, in quanto ogni elaborazione successiva richiede il minimo calcolato in quella precedente, e quindi dovrebbe aspettare.

La soluzione individuata consiste nel suddividere il calcolo delle 128 disparità in 4 momenti diversi. Significa quindi calcolare in parallelo 32 disparità per i tutti i 4 cammini. In questo modo vengono comunque richiesti 4 cicli di clock per elaborare un intero pixel, ma le reti combinatorie si riducono di un fattore 4 (quindi meno risorse) per il calcolo dei percorsi; inoltre l'albero di minimizzazione è più semplice dovendo calcolare il minimo tra soli 32 valori. Per trovare il minimo di tutte le 128 disparità viene confrontato il valore calcolato al ciclo precedente con il nuovo minimo dei 32 valori correnti. Solo nel caso in cui il minimo fosse trovato nell'ultimo gruppo di 32 valori, si avrebbe una situazione di stallo che richiederebbe l'introduzione di 1 ciclo di clock di ritardo, per permettere il propagarsi di tale valore verso gli ingressi della rete combinatoria per il calcolo del pixel successivo. Per ovviare a questo problema si potrebbe passare ad una rete combinatoria a più livelli (esattamente 7) che permetta di spezzare il problema di calcolo del minimo, in modo che non si crei mai la situazione di stallo, ma la frequenza di funzionamento dovrebbe quasi raddoppiare per mantenere gli stessi tempi di elaborazione per pixel. Frequenze alte impongono vincoli di implementazione che possono essere difficilmente rispettati.

Modulo generazione dei costi - Cost generation unit

Il modulo di generazione dei costi esegue una scansione delle trasformate di Census delle immagini sinistra e destra e calcola la distanza di Hamming per 128 valori di disparità (ovvero confronta un pixel dell'immagine sinistra con i 128 pixel dell'immagine destra, partendo dal suo corrispondente e spostandosi a sinistra). I valori dei pixel dell'immagine sinistra sono memorizzati in un registro a scorrimento da 128 elementi (in quanto ogni pixel successivo dell'immagine destra, viene confrontato anche con i pixel sinistri usati in precedenza) e il costo è calcolato contando il numero dei 24 bit che risultano a 1 dopo l'operazione di XOR logico tra ogni coppia di trasformate di Census. Il conteggio è implementato accumulando il risultato di 4 blocchi LUT a 6 ingressi e 1 uscita, ciascuno dei quali conta un segmento di 6 bit della XOR sulle trasformate da 24 bit.

Ad ogni ciclo di clock, vengono calcolati i costi per 32 valori diversi di disparità e memorizzai in una FIFO di uscita. Dopo 4 cicli di clock, tutti i valori delle 128 disparità per un pixel sono completi e si passa ad elaborare il pixel successivo. Siccome la fase di andata e quella di ritorno operano sui costi in modo molto simile, possono condividere la maggior parte dell'implementazione hardware a meno di quella parte che si occupa di inizializzare il registro a scorrimento (dovendo operare in due direzioni differenti).

Modulo di calcolo dei percorsi - Path costs processor

Questo modulo implementa il nucleo dell'algoritmo SGM, ovvero il calcolo dei costi lungo i percorsi $L_r(\mathbf{p}, d)$. Come descritto in 4.1.3, contemporaneamente vengono calcolati i costi dei percorsi per 32 valori di disparità sui 4 percorsi differenti in ogni fase, usando una struttura a pipeline a 4 livelli. Successivamente i 4 percorsi sono aggregati sommandoli per essere memorizzati nella FIFO di uscita. Come il calcolo delle distanze di Hamming, anche questo modulo ha un throughput (tasso di produttività) di un pixel ogni 4 cicli di clock. (Uno schema parzialmente completo del moduolo di calcolo dei percorsi, si trova in appenidce B.)

L'implementazione di un blocco elementare del modulo di calcolo dei percorsi è illustrato in Fig. 4.5, che segue la formulazione di calcolo dei costi già presentata nell'equazione 4.3.

$$\begin{aligned} L_{\mathbf{r}}(\mathbf{p},d) = C(\mathbf{p},d) + \min(&L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d), \\ &L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d-1)+P_1, L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},d+1)+P_1, \\ &\min_i L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},i)+P_2) - \min_k L_{\mathbf{r}}(\mathbf{p}-\mathbf{r},k) \end{aligned} \tag{4.3}$$

Questo blocco viene replicato 32 volte per ognuno dei 4 percorsi, per un totale di 128 blocchi che processano parallelamente 32 livelli di disparità. Mentre quasi tutti gli input sono dipendenti dal valore della disparità d, il minimo, su tutte le disparità del pixel precedente lungo il percorso, non lo è e quindi non richiede registri temporanei nella pipeline perché viene aggiornata solo ogni 4 cicli di clock al termine dell'elaborazione di un intero pixel. Il multiplexer inferiore seleziona l'uscita tra il costo iniziale e quello calcolato lungo il percorso, per gestire correttamente i casi di bordo che si hanno all'inizio dell'immagine e quindi all'inizio dei percorsi.

I costi calcolati lungo i percorsi $\mathbf{r}_1$, $\mathbf{r}_2$ e $\mathbf{r}_3$, come in Fig. 4.2, sono temporaneamente strutturati e memorizzati in opportune BRAM della FPGA, per essere utilizzati nel calcolo dei pixel della riga successiva. Viceversa, i costi calcolati lungo il percorso $\mathbf{r}_0$ sono direttamente inviati all'opportuno stadio della pipeline per i pixel adiacenti lungo la direzione orizzontale.

Oltre ad essere sommati per generare il costo dei cubi parziali, memorizzato nella memoria esterna DDR attraverso la FIFO di uscita, tutti i costi sono anche caricati in un albero di funzioni di minimizzazione per estrarre il minimo tra tutti i valori di disparità (usato nei calcoli successivi del pixel per ogni percorso). Tale albero è scomposto in pipeline in modo da incrementarne le performance. Il valore del minimo è continuamente aggiornato in un registro per essere disponibile nell'elaborazione del pixel successivo. Tuttavia nel caso in cui il minimo per il percorso orizzontale si venga a trovare nell'ultimo blocco di 32 disparità, bisogna forzare un ciclo di latenza a tutto il sistema di elaborazione, per dar modo a tale valore di propagarsi attraverso i registri.

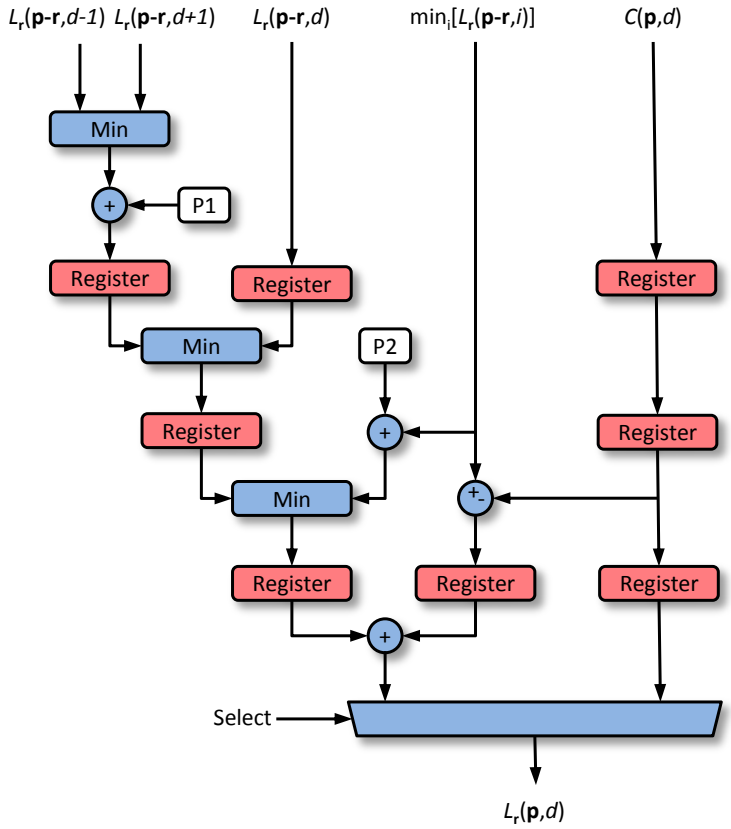


Figura 4.5: Blocco elementare del modulo di calcolo dei percorsi. In blu la logica combinatoria, mentre in rosso quella sequenziale ovvero i registri di ritardo.

La *fase di andata* e la *fase di ritorno* sono computazionalmente identiche, l'unica differenza consiste nell'ordine con cui vengono caricati i pixel. Mentre la *fase di andata* inizia dall'angolo superiore sinistro dell'immagine, quella *di ritorno* presuppone che il primo pixel processato sia quello nell'angolo inferiore destro. Tuttavia, non è necessario conoscere quale fase sia correntemente in esecuzione, in quanto l'ordine dei pixel viene gestito esternamente dal blocco che si occupa dell'interfacciamento verso la memoria DDR per il trasferimento dei dati.

Stadio finale - Final stage

Lo stadio finale, o stadio di minimizzazione, ha il compito di calcolare la mappa di disparità con precisione subpixel (i valori di disparità sono rappresentati in fixed-point con 5 bit di decimali) applicando anche i filtri del secondo minimo e del controllo sinistra/destra per invalidare i pixel non corretti. La Fig. 4.6 descrive in modo semplificato questo modulo (in particolare omettendo il check L/R che complicherebbe assai la trattazione).

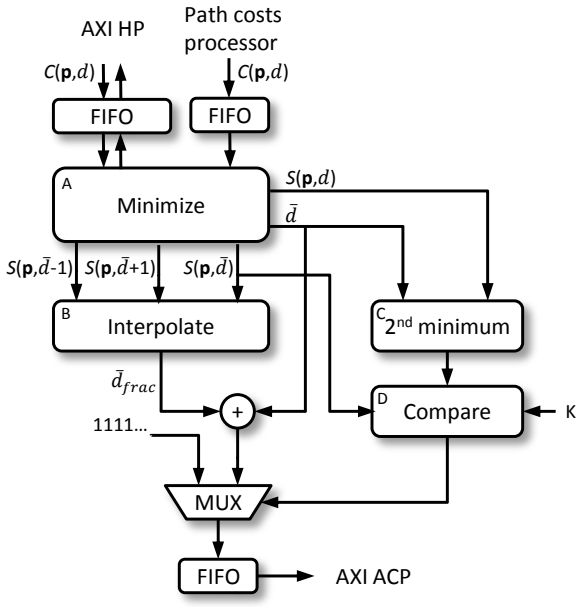


Figura 4.6: Schema a blocchi dello stadio finale

Durante la *fase di andata* lo stadio finale riceve i dati dal modulo di calcolo dei percorsi e trasferisce tali dati alla memoria DDR esterna per memorizzarli temporaneamente. Viceversa, durante la *fase di ritorno* riceve i dati dal modulo di calcolo dei percorsi, (32 costi per ogni ciclo di clock) e li aggrega con quelli riletti dalla memoria esterna, in un unico vettore dei costi aggregati $S(\mathbf{p},d)$ per ogni pixel $\mathbf{p}$. La disparità $\bar{d}$ è calcolata come l'indice del valore del vettore che ha valore minimo

(vedere Eq. 1.13). Questi due compiti sono svolti dal blocco A, che inoltre genera come uscita anche il valore del costo minimo e quello dei costi vicini per le disparità $\bar{d}-1$ e $\bar{d}+1$. Questi dati sono successivamente elaborati dal blocco B, che introduce la precisione decimale del valore di disparità del pixel, attraverso un'interpolazione equiangolare tra i valori dei costi forniti secondo l'Eq. 4.4.

$$\bar{d}_{frac} = \frac{S(\mathbf{p},\bar{d}-1) - S(\mathbf{p},\bar{d}+1)}{\max(S(\mathbf{p},\bar{d}-1), S(\mathbf{p},\bar{d}+1)) - S(\mathbf{p},\bar{d})} \tag{4.4}$$

Parallelamente, il blocco C si occupa di rimuovere dal vettore $S(\mathbf{p},d)$ il costo minimo e quello per le disparità $\bar{d}-1$ e $\bar{d}+1$, dopodiché calcola il nuovo minimo sui costi rimanenti. Successivamente viene eseguito un confronto tra il primo minimo moltiplicato per 128 e il secondo minimo moltiplicato per una costante K e sul risultato di questo confronto viene controllato un multiplexer di uscita selezionando come valore finale la disparità in virgola fissa oppure un valore di disparità invalido corrispondente a bit tutti 1.

Il risultato finale della disparità viene memorizzato nella FIFO di uscita, per essere poi trasferita nella memoria DDR.

Capitolo 5

Analisi dei risultai

In questo capitolo, vengono presentate le caratteristiche finali dell'architettura oggetto del progetto di ricerca ed alcuni esempi dei dati forniti in output. Successivamente si analizzeranno le risorse dell'FPGA utilizzate. Infine si presenta un'analisi comparativa delle performance ottenute rispetto ad altre tipologie di architetture presentate in letteratura.

5.1 Caratteristiche

La Fig. 5.1 mostra il prototipo, realizzato al termine del progetto di ricerca, per dimostrare le qualità dell'architettura studiata e progettata.

Le caratteristiche principali dell'architettura realizzata sono:

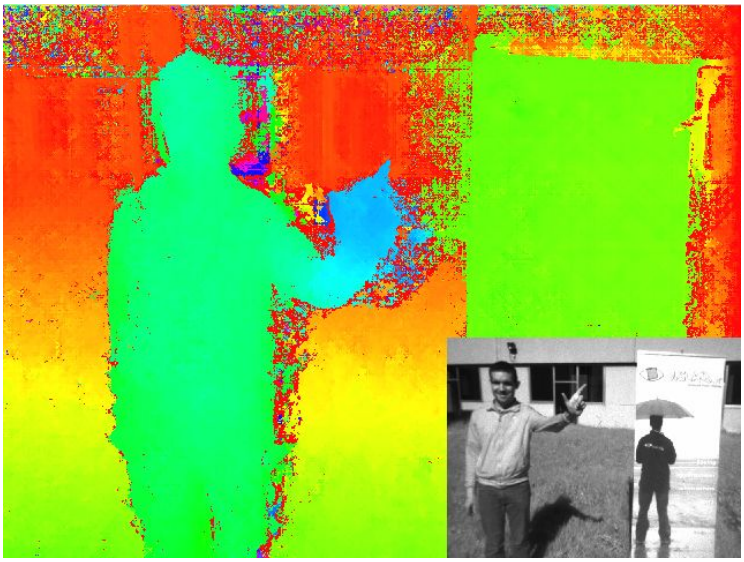
- 640×480 pixel @ 27fps (bassa latenza)
- Rettificazione delle immagini per ottiche 2.8mm e superiori
- Algoritmo computazionale : *trasformata di Census + Semi Global Matching (8 percorsi) + Minimizzazione e filtri*
- 128 livelli di disparità con precisione sub-pixel

Figura 5.1: Prototipo del sistema di visione stereoscopica.

- Capacità di rettificazione pari a 128 linee, corrispondenti a ottiche maggiori di 2.8 mm
- Architettura basata su Xilinx ZYNQ-7020 SoC (FPGA + dual core ARM Cortex-A9)
- Dati in output : immagini rettificate sinistra e destra + mappa di disparità
- Interfaccia GigabitEthernet per l'output dei dati
- Bassa potenza (circa 5 W)
- Trigger esterno
- Capacità di elaborazioni di alto livello a bordo del sistema.

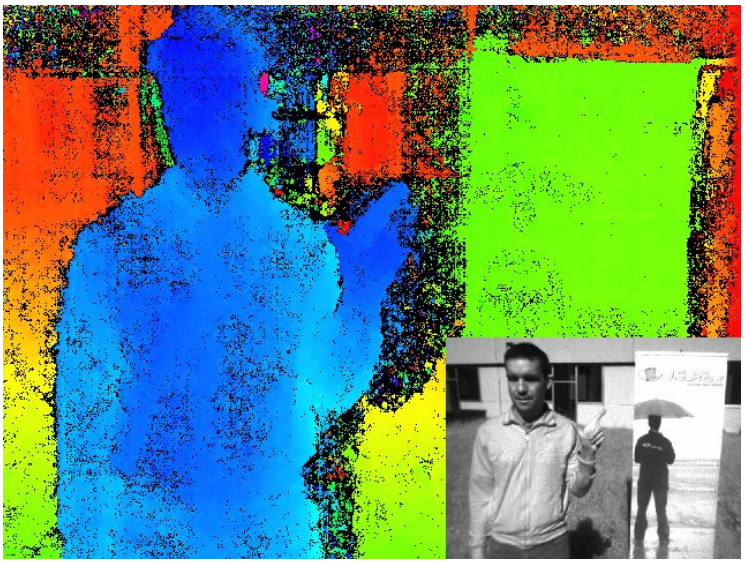
L'utilizzo dell'algoritmo SGM ad 8 percorsi (rispetto alla versione a 4 percorsi più facilmente implementabile) permette di migliorare la qualità della ricostruzione tridimensionale in condizioni avverse (bassa luminosità o scarsa texture delle superfici inquadrate).

In Fig. B.1, vengono mostrate varie esempi di mappe di disparità calcolate dal prototipo del sistema realizzato. L'informazione di disparità, è codificata attraverso differenti colori, che spaziano dal blu, al verde, al giallo e al rosso al diminuire del valore. Ricordando che la distanza è inversamente proporzionale alla disparità, si può riconoscere come la tridimensionalità della scena inquadrata (mostrata nell'angolo delle singole immagini) venga correttamente ricostruita. Gli esempi mostrano sia scenari esterni che interni, con o senza filtri sulla disparità.

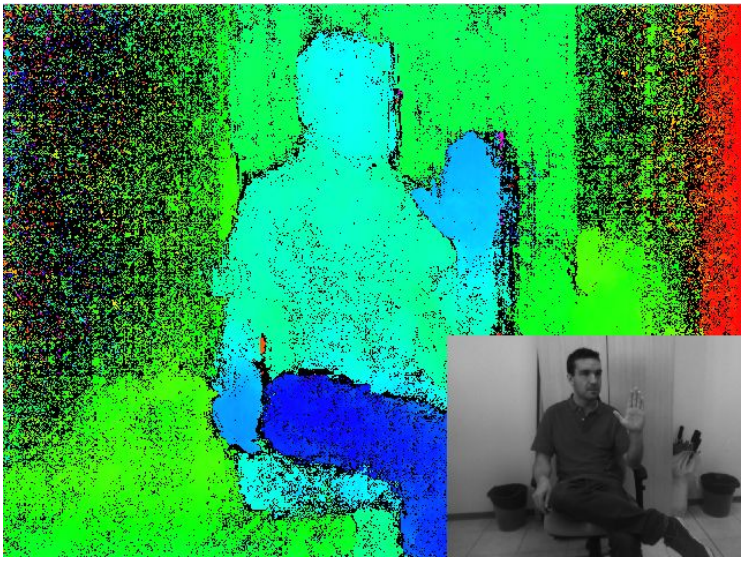


(a)

Figura 5.2: (cont.) Esempio di mappa di disparità: a) scenario esterno senza filtri,



(b)



(c)

Figura 5.1: b) scenario esterno con filtri controllo sinistra/destra e secondo minimo, (c) scenario interno.

5.2 Risorse

Una delle caratteristiche che giustifica la bontà della scelta del modello di FPGA, da utilizzare per un'architettura, sono le percentuali di risorse effettivamente utilizzate, mostrate in Tab. 5.1.

Tabella 5.1: Utilizzo risorse FPGA.

Risorsa	Utilizzate	Totali	Percentuale
Registri	25223	106400	23 %
LUT	26630	53200	50 %
BRAM (36 kbit)	116	140	83 %
DSP	48	220	21 %

L'utilizzo di registri e LUT si attesta intorno al 50 % permettendo di ottenere una buona implementazione che garantisce frequenze di funzionamento elevate. L'utilizzo delle BlockRAM è molto elevato, maggiore del 80 %, ed è il risultato di un attento studio per trovare il giusto compromesso tra prestazioni e risorse utilizzate. In Tab. 5.2 viene dettagliatamente descritta l'allocazione delle BlockRAM utilizzate da ciascun blocco funzionale dell'architettura. Si può notare come il 43 % delle BlockRAM disponibili è utilizzato dal blocco di generazione dei costi di SGM, il 29 % è usato dalla stadio di rettificazione, il 9 % per la comunicazione con la memoria esterna e solo 1 % per la trasformata di Census. La maggior parte delle BlockRAM del blocco SGM sono usate per memorizzare i costi precedenti lungo ogni cammino e la loro quantità dipende dalla larghezza dell'immagine e dal numero di disparità supportate. Si potrebbe pensare di allocare ogni ulteriore BlockRAM da 36 Kbit per la rettificazione, aumentando il corrispondente buffer di circa 6 linee ciascuna, ma così facendo si raggiungerebbe una percentuale di utilizzo talmente alta, da penalizzare le prestazioni di implementazione, diminuendo la frequenza massima di funzionamento.

La fase di implementazione produce le seguenti frequenze di funzionamento per i 3 dominii di clock utilizzati:

Tabella 5.2: Allocazione delle BlockRAM.

			BlockRAM 18Kbit		BlockRAM 36Kbit	
		Layout	N	%	N	%
Camera L&R	Canali AXI	64bit x 512	0	0%	6	4%
	Rettificazione	8bit x 20480	0	0%	40	29%
	Census	8bit x 2048	2	1%	0	0%
SGM	Canali AXI	64bit x 512	0	0%	7	5%
	Costi iniziali	160bit x 512	1	1%	2	2%
	Costi aggregati	256bit x 512	0	0%	4	3%
	Temporanee	32bit x 512	2	1%	0	0%
	Costi Minimi	8bit x 512	2	1%	0	0%
	Percorsi	234bit x 512	8	3%	48	35%
TOT			17	6%	107	77%
	2 x 18Kbit = 1 x 36Kbit				116	83%

Allocazione delle BlockRAM tra i vari blocchi funzionali. I canali AXI rappresentano le risorse allocate per la comunicazione con la memoria esterna DDR. Ogni BlockRAM da 36Kbit, puó essere utilizzata come due BlockRAM da 18Kbit; l'ultima riga della tabella rappresenta l'utilizzo totale delle BlockRAM disponibili.

- 26.7 MHz – Acquisizione, dedistorsione, rettificazione e Census, sincroni con il clock di acquisizione dalla telecamera;
- 100 MHz – Pipeline di elaborazione SGM;
- 150 MHz – Comunicazione con la memoria esterna DDR.

questi valori di frequenza sono compatibili con l'hardware disponibile e permettono di ottenere tranquillamente una frequenza di funzionamento dell'architettura pari a 27 fps.

Un'altra risorsa che ha richiesto un attento studio in fase di progettazione, è stata l'allocazione della banda di trasferimento disponibile con la memoria esterna DDR. Il traffico con la memoria è dettagliato in Tab. 5.3. La maggior parte della banda è

consumata per memorizzare e rileggere i costi aggregati parziali durante le fasi di andata e ritorno: l'architettura sviluppata può essere configurata per eseguire la sola fase di andata (SGM a 4 percorsi) eliminando quindi la necessità di memorizzare i dati intermedi (in modo analogo a quanto descritto in [29]) e funzionare a frame-rate più elevati, ma ciò a discapito della qualità della mappa di disparità in quando l'approccio a 2 passi è essenziale per produrre risultati consistenti anche in scenari di utilizzo sfavorevoli.

5.3 Prestazioni rispetto allo stato dell'arte

Un confronto delle prestazioni tra l'architettura sviluppata e altre implementazioni SGM disponibili in letteratura è presentata in Tab. 5.4. Siccome i vari lavori presentati, operano su immagini di dimensioni differenti e anche su numero di livelli dispartià diversi, come metrica di confronto si è utilizzato il *disparity rate*, ovvero il numero di disparità che il sistema è in grado di valutare ogni secondo; in questo modo vengono equamente pesate: la dimensione dell'immagine, il frame rate di funzionamento e il numero di livelli dispartà.

Si sono prese in considerazione anche architetture di elaborazione basate su CPU e GPU per dimostrare come questo sistema embedded sia in grado di raggiungere prestazioni in linea con tipologie hardware diverse. Le prestazioni raggiunte da que-

Tabella 5.3: Traffico con la memoria esterna DDR.

	Forward pass			Backward pass		
	Data	Read	Write	Data	Read	Write
HP0	$S_{fwd}(\mathbf{p},[0-31])$	-	0.58 GB/s (41%)	$S_{fwd}(\mathbf{p},[0-31])$	0.58 GB/s (47%)	-
HP1	$S_{fwd}(\mathbf{p},[32-63])$	-	0.58 GB/s (41%)	$S_{fwd}(\mathbf{p},[32-63])$	0.58 GB/s (47%)	-
HP2	$S_{fwd}(\mathbf{p},[64-95])$	-	0.58 GB/s (41%)	$S_{fwd}(\mathbf{p},[64-95])$	0.58 GB/s (47%)	-
HP3	$S_{fwd}(\mathbf{p},[96-127])$	-	0.58 GB/s (41%)	$S_{fwd}(\mathbf{p},[96-127])$	0.58 GB/s (47%)	-
ACP	LUTs, R_L, R_R, C_L, C_R	0.27 GB/s (22%)	0.27 GB/s (22%)	$C_L, C_R, \bar{D}$	0.21 GB/s (18%)	0.053 GB/s (4%)
DDR	All	0.27 GB/s (60%)	2.6 GB/s (60%)	All	2.5 GB/s (59%)	0.053 GB/s (1%)

La massima banda utilizzata su ciascun canale AXI, connesso verso il controllore della memoria DDR, è 0.58 GB/s per la lettura e 2.6 GB/s per la scrittura.

sta architettura si collocano ai livelli più elevati; sebbene due soluzioni offrono un *disparity rate* più alto [27, 29] e una essenzialmente identico [25], tutte richiedono dispositivi hardware considerevolmente più costosi con consumi di potenza di uno o due ordini di grandezza maggiori. In particolare la soluzione che raggiunge il maggiore *disparity rate* [29], sfrutta un'implementazione di SGM basata su soli 4 percorsi di aggregazione, restituendo quindi mappe di disparità meno precise in molti scenari di utilizzo reale.

Tabella 5.4: Confronto delle implementazione di algoritmi di ricostruzione stereoscopica

Implementazione	Piattaforma hardware	Algoritmo	Dim. Immagine $[px]$	Tempo $[ms]$	Disparity rate $[10^6/s]$
Gehrig ECVW10 [24]	Intel® Core™ i7 975 EX @ 3.3 GHz	CT + SGM (8) + MF + L/R	640 × 320 @ 128	224	117
Broggi IROS11 [25]	Intel® Core™ i7 920 @ 3.20 GHz	CT + SGM (8) + L/R	640 × 320 @ 128	27	970
Hirschmüller ISVC10 [26]	NVIDIA® GeForce™ 8800 Ultra	HMI +SGM (8) +MF +L/R	640 × 480 @ 128	238	165
Nedevschi IV10 [9]	NVIDIA® GeForce™ GTX 280	CT + SGM (8) + L/R + MF	512 × 383 @ 56	19	578
Banz ICCV11 [27]	NVIDIA® Tesla C2050	RT + SGM (8) +MF	640 × 480 @ 128	16	2457
Gehrig ICVS09 [28]	Xilinx® Virtex-4 FX140	ZSAD + SGM (8) + L/R	2 × 340 × 200 @ 64	40	218
Banz SAMOS10 [29]	Xilinx® Virtex-5 LX 220T-1	RT + SGM (4) + L/R + MF	640 × 480 @ 128	9.7	4053
this paper	Xilinx® Zynq™ 7020	CT + SGM (8) + L/R + 2^{nd}min	640 × 480 @ 128	33	1192

Panoramica delle implementazioni SGM valutate; tra parentesi il numero di percorsi di aggregazione. Sono utilizzate diverse funzioni di costo , descritte come: trasformata di Census (CT), trasformata del rank (RT), hierarchical mutual information (HMI), e zero-mean sum of absolute differences(ZSAD). L/R indica il controllo di consistenza sinistra-destra, MF il filtraggio mediano e 2^{nd}min il filtro sul rapporto fra minimo e 2^{o} minimo.

Conclusioni

La ricostruzione tridimensionale è un campo di ricerca molto studiato e seguito sia in ambito accademico che industriale/commerciale. Questo progetto di ricerca ha affrontato lo studio, lo sviluppo e l'ingegnerizzazione di un sistema embedded di visione stereoscopica in grado di fornire una ricostruzione tridimensionale densa, senza l'ausilio di un elaboratore tradizionale, per compiti di sorveglianza, automazione e sicurezza.

L'attività del primo anno di ricerca, volta alla definizione delle specifiche del progetto, ha affrontato le seguenti tematiche:

- studio dell'architettura del sistema embedded: si è scelta una moderna tecnologia SoC (System On Chip) che integra in un unico chip una FPGA (per le elaborazioni più onerose quali la disparità stereo) e un microprocessore ARM dual-core (per le attività di elaborazione di più alto livello);

- valutazione algoritmi di visione stereo (basandosi sullo stato dell'arte), proponendo una versione ottimizzata dell'algoritmo SGM unita allo sviluppo di nuovi filtri di post-elaborazione per aumentare l'accuratezza e la qualità della disparità stereo.

Durante il secondo anno, sono stati affrontati sia la progettazione elettronica del sistema embedded, sia lo studio e l'analisi dell'algoritmo di elaborazione stereo, per svilupparne una implementazione su architettura FPGA (massivamente parallela).

L'ultimo anno è stato dedicato all'ingegnerizzazione del sistema, per arrivare ad un prototipo funzionante, e parallelamente si sono studiati e sviluppati i filtri post-

elaborazione. Inoltre sono state analizzate applicazioni reali per dimostrare l'utilità del sistema ai fini di sorveglianza, automazione e sicurezza.

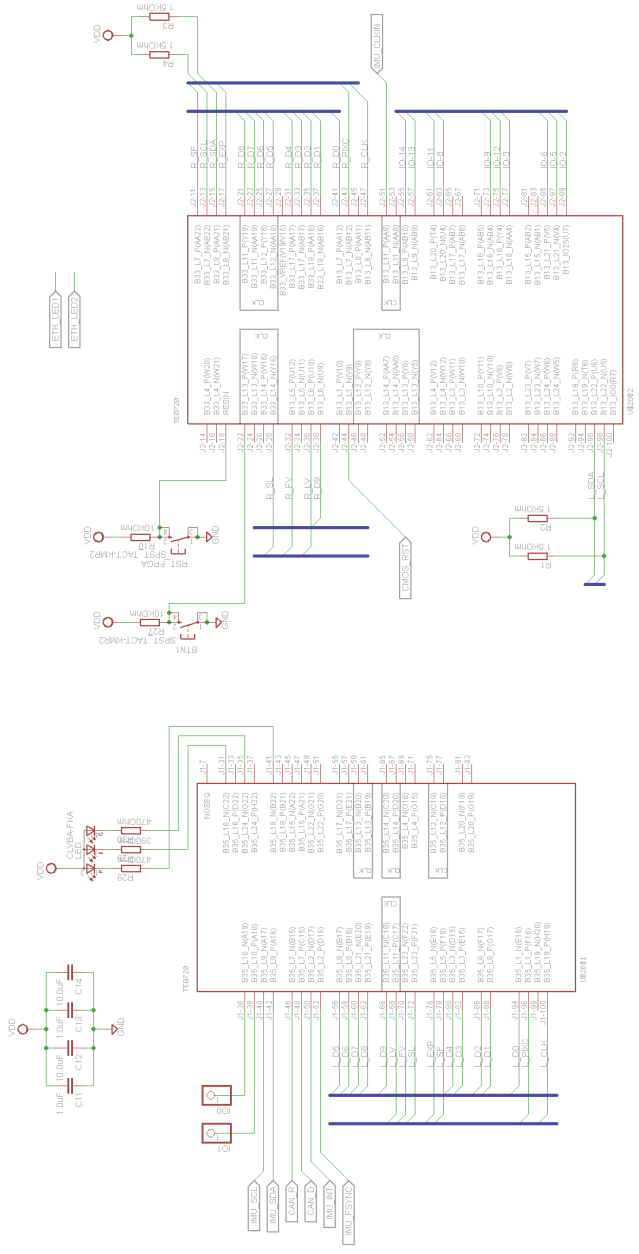
I vantaggi offerti da questo sistema si misurano in termini di prestazioni, dimensioni, potenza richiesta, costi rispetto a sistemi basati su personal computer o GPU attualmente utilizzati.

Possibili sviluppi futuri, riguardano lo studio e sviluppo di applicazioni intelligenti per automazione e sorveglianza, che possano essere elaborate direttamente a bordo dell'architettura studiata, senza l'ausilio di elaboratori tradizionali esterni.

Appendice A

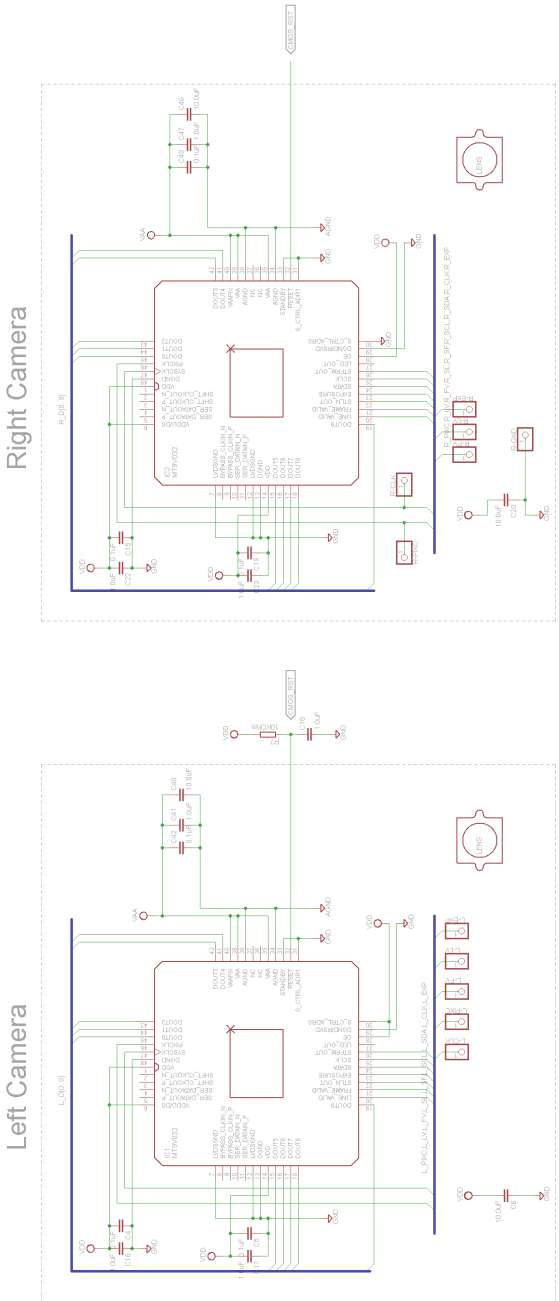
Circuito elettronico

Di seguito sono riportati gli schematici principali che descrivono il circuito elettronico sviluppato per realizzare il prototipo del sistema di visione stereoscopico.



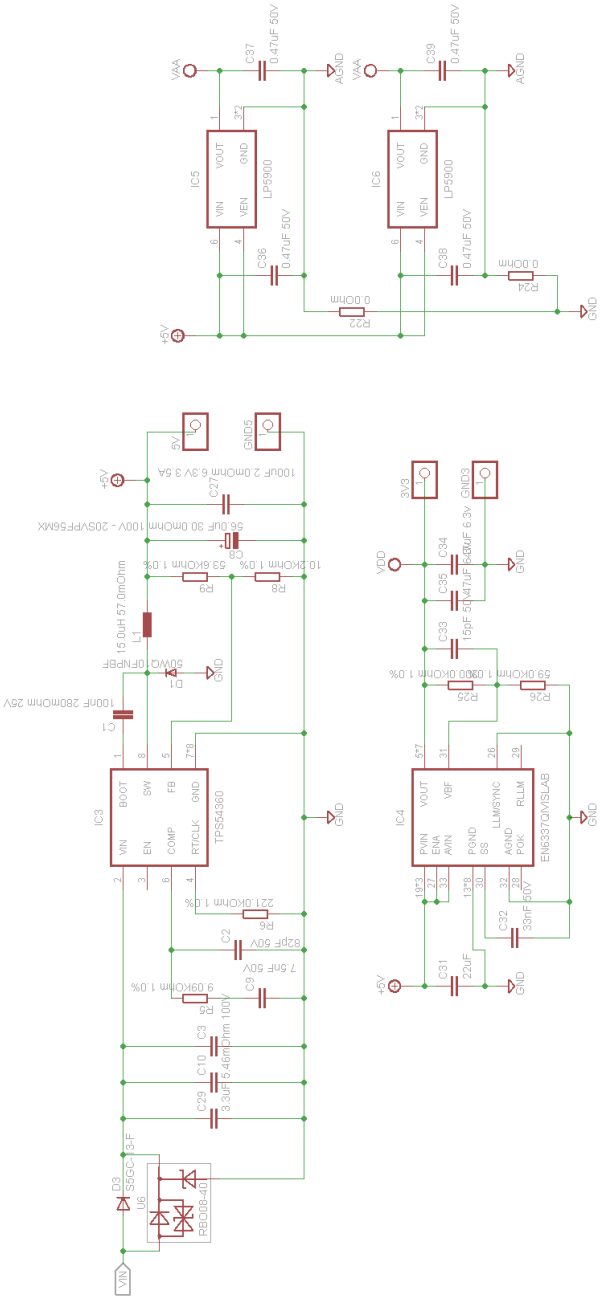
(a)

Figura A.1: (cont.) Schematico del circuito elettronico del sistema: (a) stadio di elaborazione,



(b)

Figura A.0: (b) sensori video,

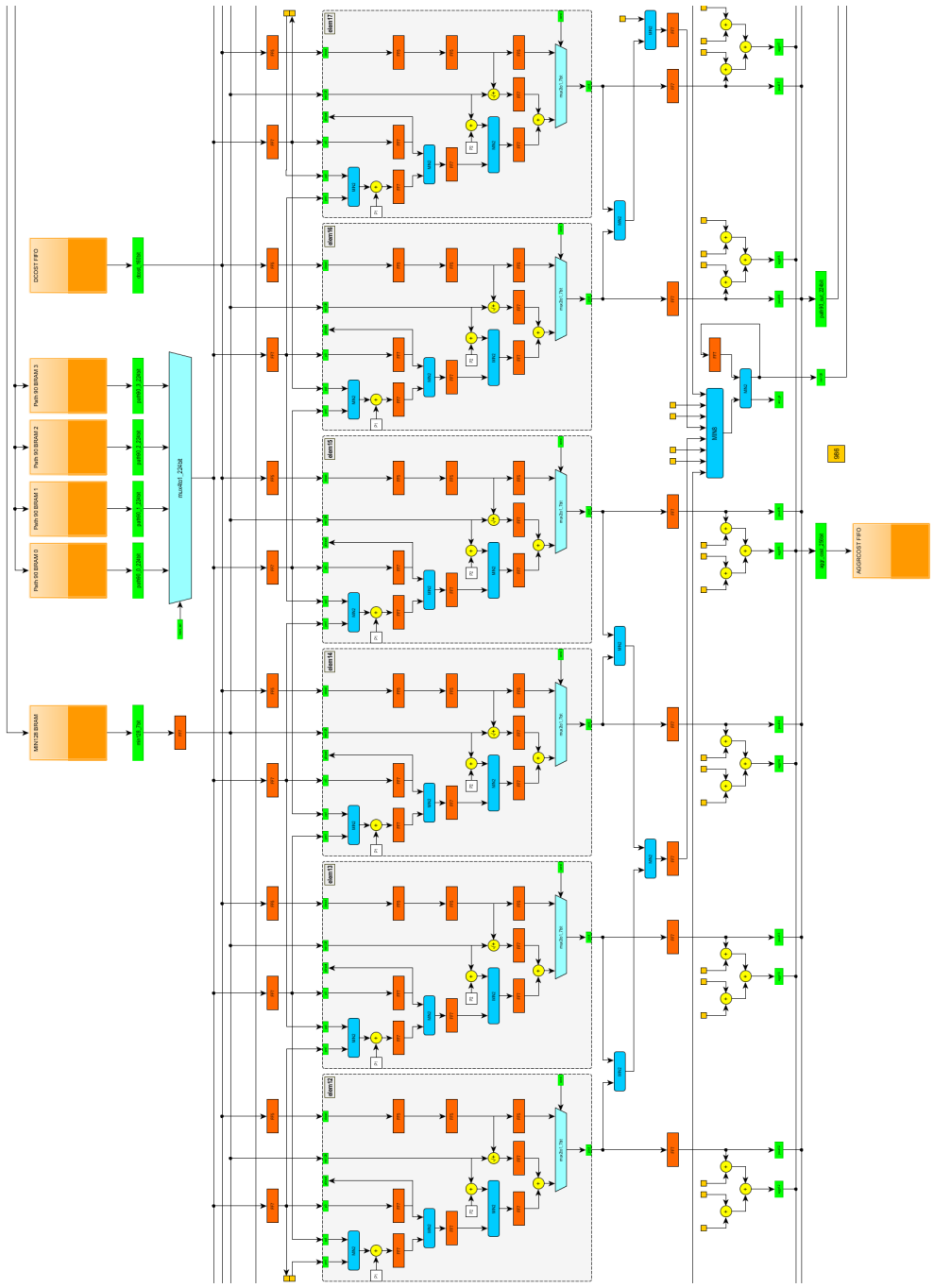


(c)

Figura A.-1: (c) stadio di alimentazione.

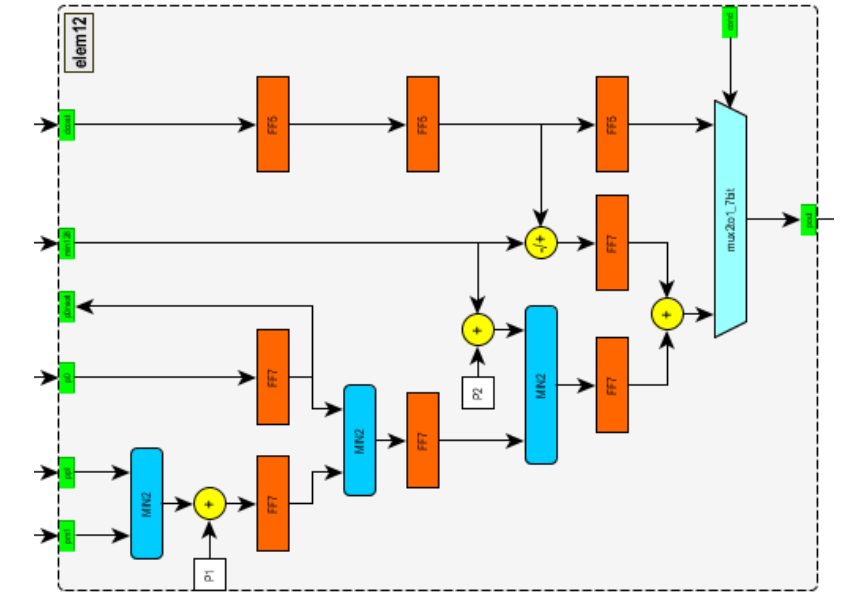
Appendice B

Schema a blocchi del modulo di aggregazione dei costi

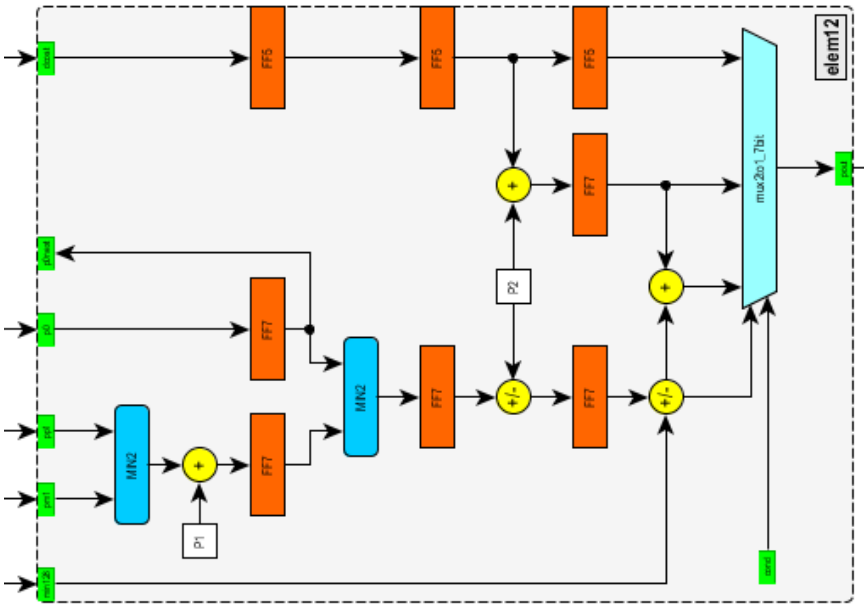


(a)

Figura B.1: (cont.) Schema a blocchi del modulo di aggregazione dei costi: (a) una porzione delle modulo di aggregazione dei costi (si possono individuare le FIFO di ingresso e uscita ed una parte dei 32 blocchi che implementano la funzione di calcolo del costo minimo lungo un cammino),



(b)



(c)

Figura B.0: (b) funzione di calcolo del costo minimo per un generico cammino, (c) funzione di calcolo del costo minimo specializzata per il cammino orizzontale, dove la propagazione del minimo precedente rappresenta il percorso critico.

Bibliografia

[1] Olivier Faugeras, Bernard Hotz, Hervé Mathieu, Thierry Viéville, Zhengyou Zhang, Pascal Fua, Eric Théron, and Projet Robotvis. Real time correlation-based stereo: Algorithm, implementations and applications, 1996.

[2] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(2):328–341, February 2008. URL: `http://dx.doi.org/10.1109/TPAMI.2007.1166`, `doi:10.1109/TPAMI.2007.1166`.

[3] Mirko Felisa and Paolo Zani. Incremental Disparity Space Image computation for automotive applications. In *Procs. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, St.Louis, Missouri, USA, October 2009. `arXiv:doi:10.1109/IROS.2009.5353897`.

[4] Andreas Geiger, Martin Roser, and Raquel Urtasun. Efficient large-scale stereo matching. In *Proceedings of the 10th Asian conference on Computer vision - Volume Part I*, ACCV'10, pages 25–38, Berlin, Heidelberg, 2011. Springer-Verlag. URL: `http://dl.acm.org/citation.cfm?id=1964320.1964325`.

[5] Rene Ranftl, Stefan Gehrig, Thomas Pock, and Horst Bischof. Pushing the Limits of Stereo Using Variational Stereo Estimation. In *IV*, 2012. to appear.

[6] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47:7–42, 2001.

[7] Heiko Hirschmuller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, CVPR '05, pages 807–814, Washington, DC, USA, 2005. IEEE Computer Society. doi:10.1109/CVPR.2005.56.

[8] Olga Veksler. *Efficient Graph-based Energy Minimization Methods in Computer Vision*. PhD thesis, Ithaca, NY, USA, 1999. AAI9939932.

[9] I. Haller, C. Pantilie, F. Oniga, and S. Nedevschi. Real-time semi-global dense stereo solution with improved sub-pixel accuracy. In *Intelligent Vehicles Symposium (IV), 2010 IEEE*, pages 369–376, 2010. doi:10.1109/IVS.2010.5548104.

[10] C.D. Pantilie and S. Nedevschi. Sort-sgm: Subpixel optimized real-time semiglobal matching for intelligent vehicles. *Vehicular Technology, IEEE Transactions on*, 61(3):1032 –1042, march 2012. doi:10.1109/TVT.2012.2186836.

[11] http://opencv.willowgarage.com.

[12] Stan Birchfield and Carlo Tomasi. A pixel dissimilarity measure that is insensitive to image sampling. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(4):401–406, April 1998. URL: http://dx.doi.org/10.1109/34.677269, doi:10.1109/34.677269.

[13] Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. In Jan-Olof Eklundh, editor, *Computer Vision âĂŤ ECCV '94*, volume 801 of *Lecture Notes in Computer Science*, pages 151–158. Springer Berlin Heidelberg, 1994. URL: http://dx.doi.org/10.1007/BFb0028345, doi:10.1007/BFb0028345.

[14] Sandino Morales and Reinhard Klette. Ground truth evaluation of stereo algorithms for real world applications. In *ACCV Workshops (2)*, pages 152–162, 2010.

[15] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR)*, Providence, USA, June 2012.

[16] `http://velodynelidar.com/lidar/hdlproducts/hdl64e.aspx`.

[17] Pascal Steingrube, Stefan K. Gehrig, and Uwe Franke. Performance evaluation of stereo algorithms for automotive applications. In *Proceedings of the 7th International Conference on Computer Vision Systems: Computer Vision Systems*, ICVS '09, pages 285–294, Berlin, Heidelberg, 2009. Springer-Verlag. `doi:10.1007/978-3-642-04667-4_29`.

[18] Sandino Morales, Simon Hermann, and Rehinard Klette. Real-world stereo-analysis evaluation. Technical Report MItech-TR-77, The University of Auckland, New Zealand, 2011.

[19] Sandino Morales and Reinhard Klette. A third eye for performance evaluation in stereo sequence analysis. In *Proceedings of the 13th International Conference on Computer Analysis of Images and Patterns*, CAIP '09, pages 1078–1086, Berlin, Heidelberg, 2009. Springer-Verlag. URL: `http://dx.doi.org/10.1007/978-3-642-03767-2_131`, `doi:10.1007/978-3-642-03767-2_131`.

[20] `http://www.cvlibs.net/datasets/kitti`.

[21] Massimo Bertozzi, Luca Bombini, Alberto Broggi, Michele Buzzoni, Elena Cardarelli, Stefano Cattani, Pietro Cerri, Stefano Debattisti, Rean Isabella Fedriga, Mirko Felisa, Luca Gatti, Alessandro Giacomazzo, Paolo Grisleri, Maria Chiara Laghi, Luca Mazzei, Paolo Medici, Matteo Panciroli, Pier Paolo Porta, and Paolo Zani. The VisLab Intercontinental Autonomous Challenge: 13,000

km, 3 months, no driver. In *Procs. 17th World Congress on ITS*, Busan, South Korea, October 2010.

[22] K. Levenberg. A method for the solution of certain nonlinear problems in least squares. *Q. Appl. Math*, 2:164–168, 1944.

[23] `http://www.cvlibs.net/datasets/kitti/eval_stereo_flow.php?benchmark=stereo`.

[24] S.K. Gehrig and C. Rabe. Real-time semi-global matching on the cpu. In *ECVW10*, pages 85–92, 2010.

[25] Alberto Broggi, Michele Buzzoni, Mirko Felisa, and Paolo Zani. Stereo obstacle detection in challenging environments: the VIAC experience. In *Procs. IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems*, pages 1599–1604, San Francisco, California, USA, September 2011. `arXiv:doi:10.1109/IROS.2011.6048211`.

[26] Heiko Hirschmüller and Ines Ernst. Mutual information based semi-global stereo matching on the gpu. In *ISVC (1) 08*, pages 228–239, 2008.

[27] C. Banz, H. Blume, and P. Pirsch. Real-time semi-global matching disparity estimation on the gpu. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 514–521, 2011. `doi:10.1109/ICCVW.2011.6130286`.

[28] Stefan K. Gehrig, Felix Eberli, and Thomas Meyer. A real-time low-power stereo vision engine using semi-global matching. In *Proceedings of the 7th International Conference on Computer Vision Systems: Computer Vision Systems*, ICVimages/Capitolo4S '09, pages 134–143, Berlin, Heidelberg, 2009. Springer-Verlag.

[29] C. Banz, S. Hesselbarth, H. Flatt, H. Blume, and P. Pirsch. Real-time stereo vision system using semi-global matching disparity estimation: Architecture and

fpga-implementation. In *Embedded Computer Systems (SAMOS), 2010 International Conference on*, pages 93–101, 2010. doi:10.1109/ICSAMOS.2010.5642077.

Ringraziamenti

Giunti all'ultima pagina, è il momento di ricordare e ringraziare tutte le persone che hanno contribuito a questa importante esperienza durata 3 anni. Come ben sapete, sono una persona di poche parole, e ognuno di voi sa quanto gli devo in amicizia e gratitudine.

Il primo grazie va ad Alberto, che ha reso possibile tutto questo, e in secondo luogo a tutti i colleghi, o meglio amici, che mi hanno affiancato in questi anni, in particolare Mirko Felisa e Paolo Medici che hanno combattuto tenacemente insieme a me per raggiungere questo risultato. Un ringraziamento particolare anche a Francesco Gregoretti, Roberto Passerone e Claudio Passerone.

Infine ringrazio, l'Eleonora, la mia famiglia e gli amici che in tutti questi anni non mi hanno mai fatto mancare il loro affetto e compagnia.

Grazie a tutti.