D3 - Rapporto tecnico contenente la descrizione del prototipo e la personalizzazione della soluzione prodotta

DRAGO – Digitalizzazione reperti archeologici garganici by open-source

Prima emissione: 30/04/2015

Distribuzione: PUBBLICA

INDICE GENERALE

Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)

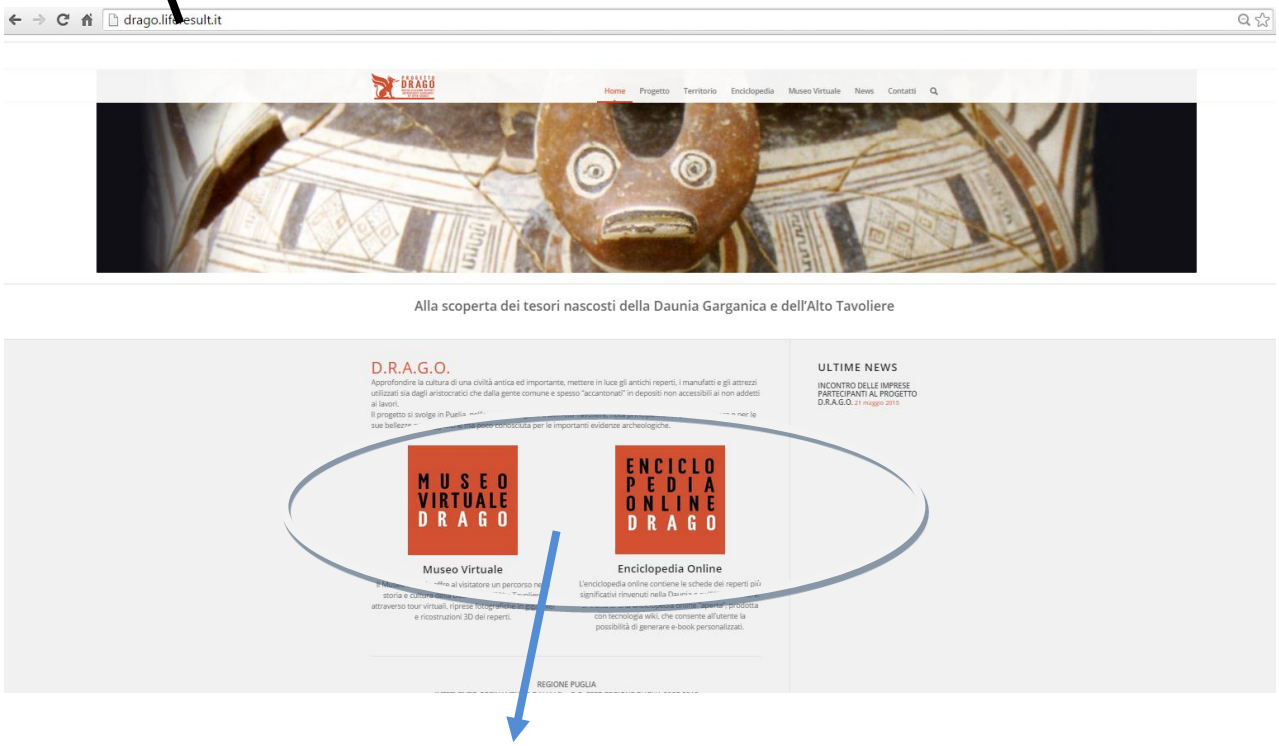
STORIA DELLE MODIFICHE EFFETTUATE

Non applicabile in quanto è la prima versione del documento

1 REQUISITI DI SISTEMA

L'utente ha la possibilità di accedere al sito del progetto D.R.A.G.O attraverso l'indirizzo http:\\drago.liferesult.it
All'interno del sito si trova il tasto "OMNIACULTURE" che trasporta l'utente sulla piattaforma OMNIACULTURE su un'altra pagina internet priva di credenziali di accesso, quindi pubblica.

http:\\drago.liferesult.it



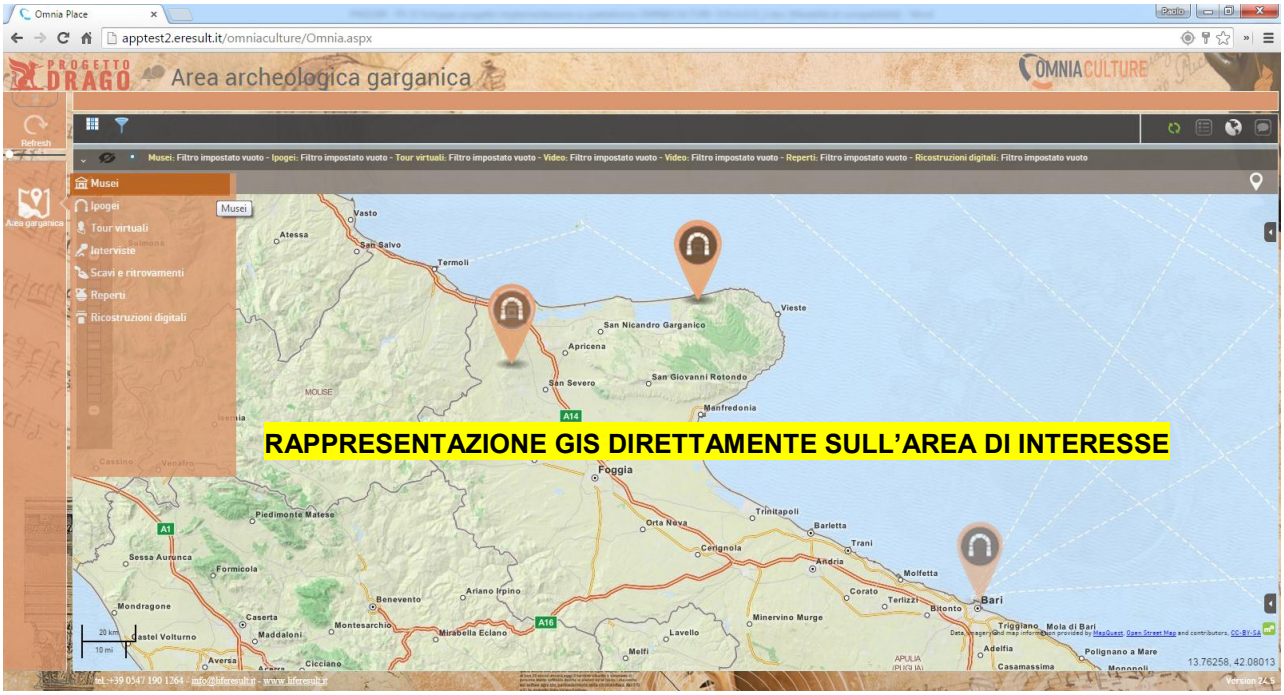
TASTI DI COLLEGAMENTO AL SITO WWW.OMNIACULTURE.IT ALL'INTERNO DEL SITO DRAGO

La nuova pagina internet OMNIACULTURE presenta la struttura indicata di seguito, in cui all'avvio viene mostrata la seguente schermata introduttiva:

Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)

Dal pulsante Accedi si arriva alla applicazione web, che mostra un ribbon sulla parte destra ed una mappa GIS, sulla quale sono collocate più icone, diverse tra loro, a seconda della categoria che rappresentano.
Questo tipo di visualizzazione (GIS) fa riferimento alla categoria “Area archeologica garganica”. Selezionandola sul ribbon verrà visualizzata la mappa e verrà nascosto l’albero di navigazione, gli elementi da selezionare si trovano sotto forma di icone nell’area di visualizzazione.

Con un click del mouse su un'icona, presente sulla pianta, si apre una breve didascalia che ne spiega il contenuto.

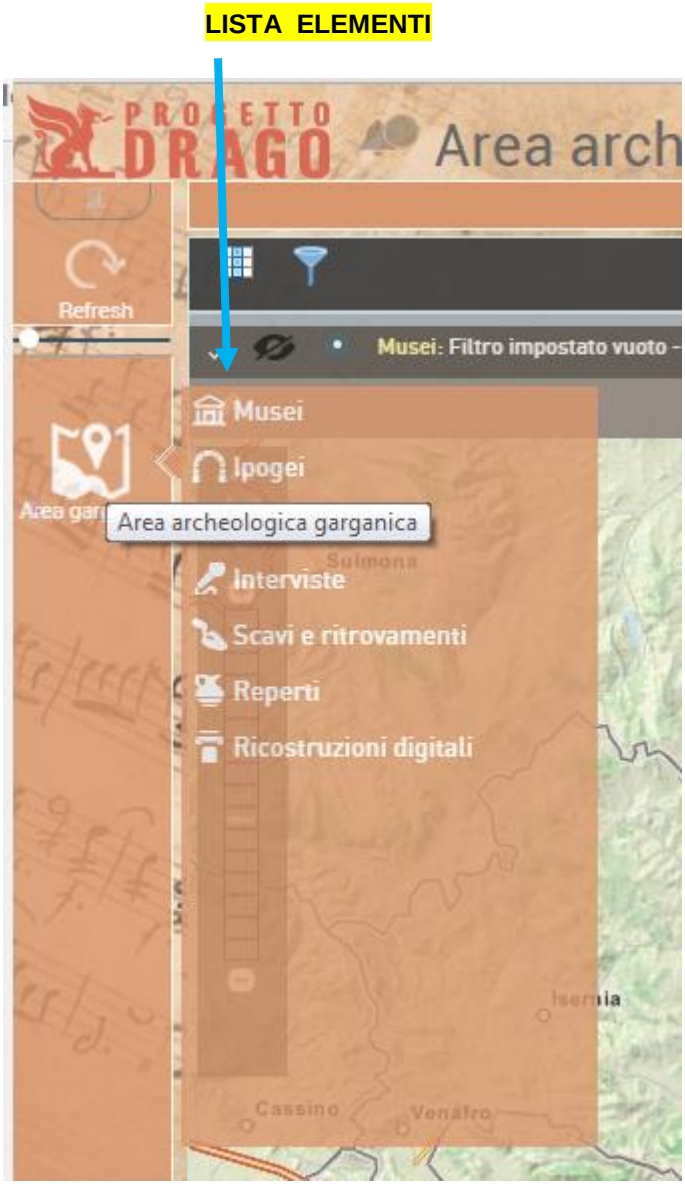


Le "categorie" presenti nel ribbon sono:

- Musei;
- Ipogei;
- Tour virtuali;
- Interviste;
- Scavi e ritrovamenti
- Reperti;
- Ricostruzioni digitali;

Ognuna di queste categorie da origine ad una lista di elementi visibili nell'albero di navigazione, ora visibile perché contenente dati. Oltre al nome della categoria c'è anche l'icona corrispettiva.

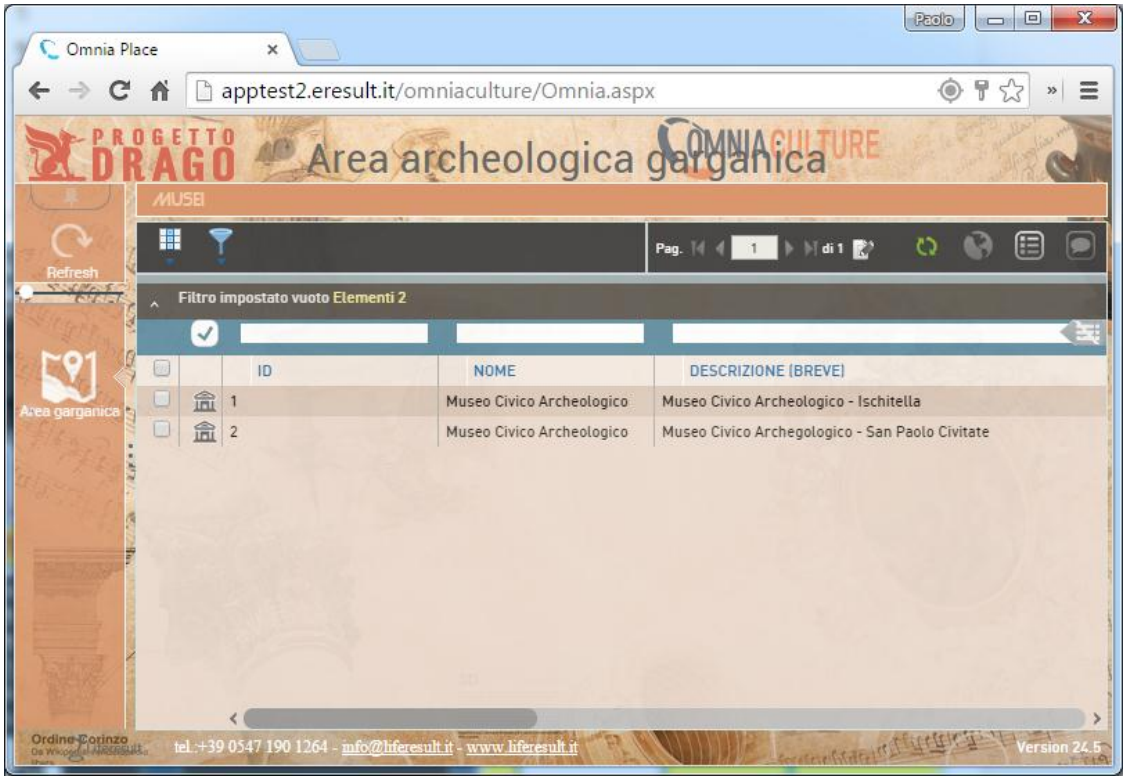
Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)



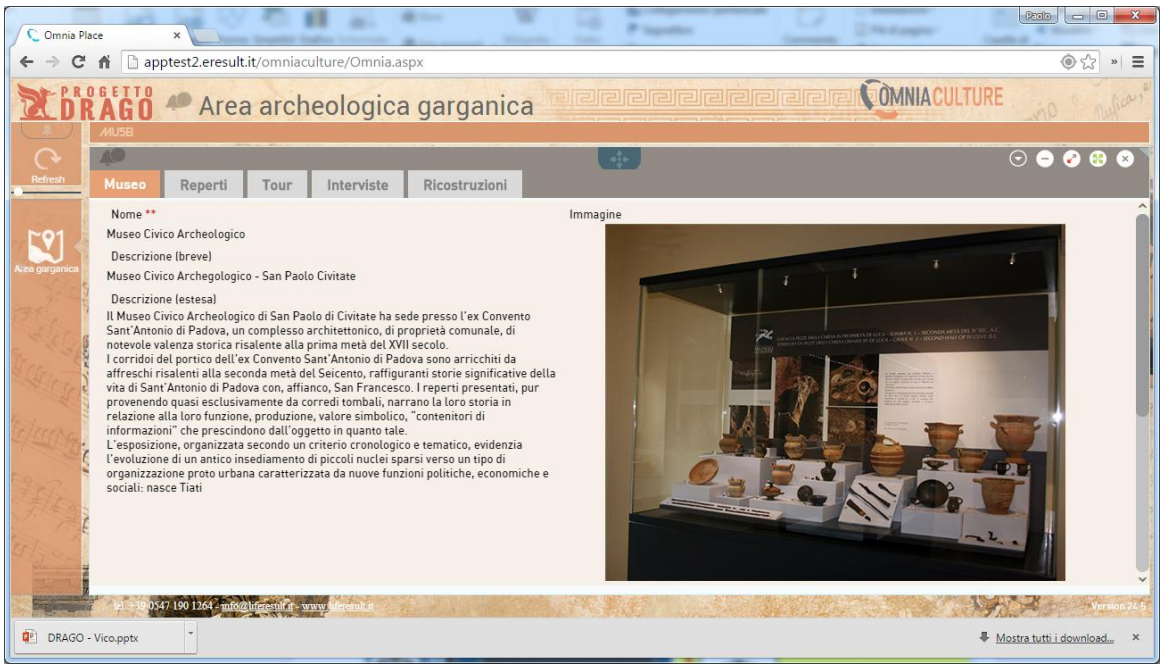
Selezionando quindi l'icona MUSEI del ribbon, l'albero viene popolato con l'elenco di tutti i musei.
Selezionando l'icona IPOGEI del ribbon, l'albero viene popolato con l'elenco di tutti gli Ipogei.
Analogamente per le altre icone.

Nello specifico le categorie si trovano nella ribbon. Nell' albero troviamo la lista relativa alla categoria selezionata. All'interno dell'area di visualizzazione si apre invece la visualizzazione a schede, che permette una fruizione intuitiva della sottocategoria selezionata con in più la facilità di switchare sulle altre categorie presenti all'interno della sottocategoria selezionata.
All'interno delle schede oltre all'inserimenti di valori alfanumerici ci sono anche collegamenti multimediali di elementi presenti nel database.
Analizziamo le SCHEDE nello specifico.

1.1 MUSEO

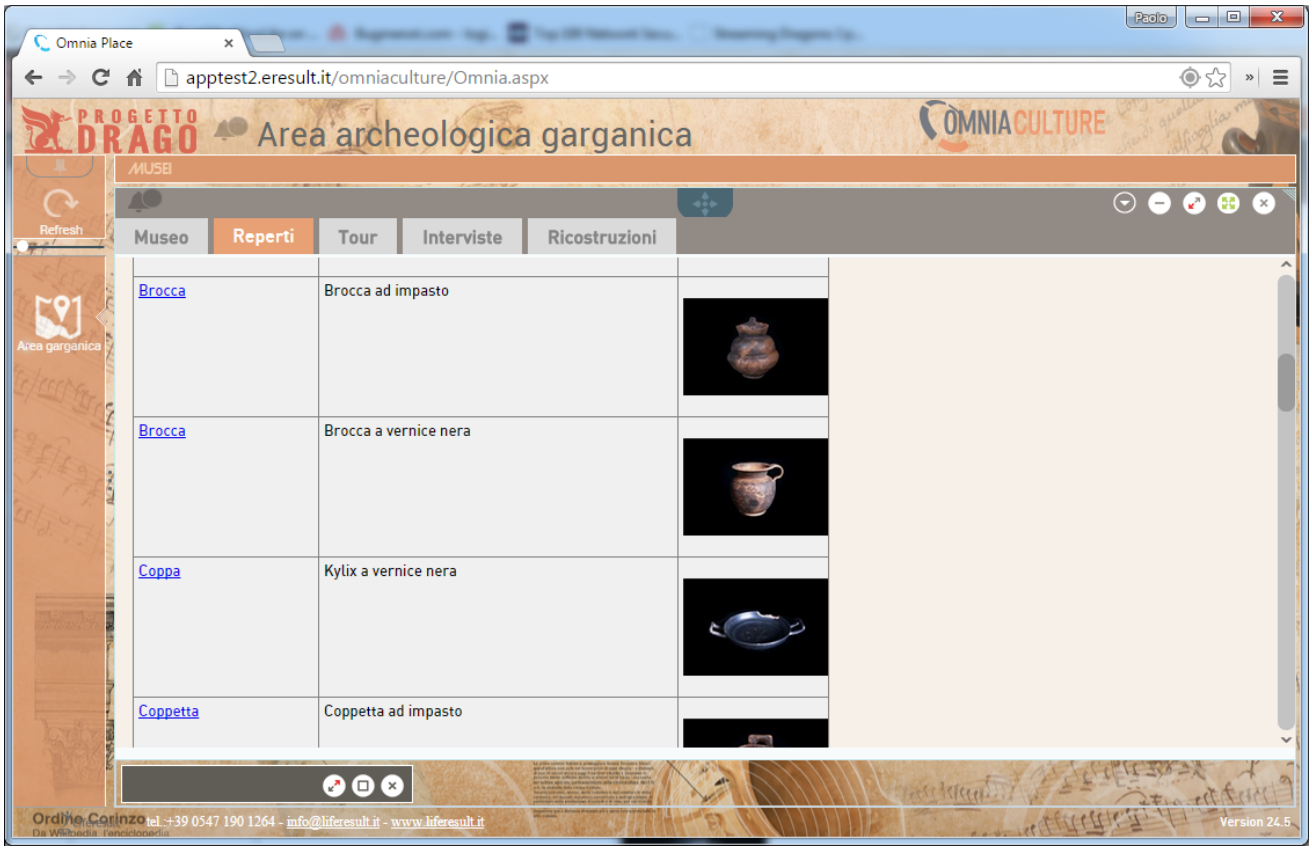


La scheda museo si presenta con la foto del museo stesso e con la relativa scheda descrittiva. Oltre a questi valori è presente una finestra all'interno della quale, con collegamento embedde, viene collocato il virtual tour del relativo museo fruibile dall'utente sulla scheda.



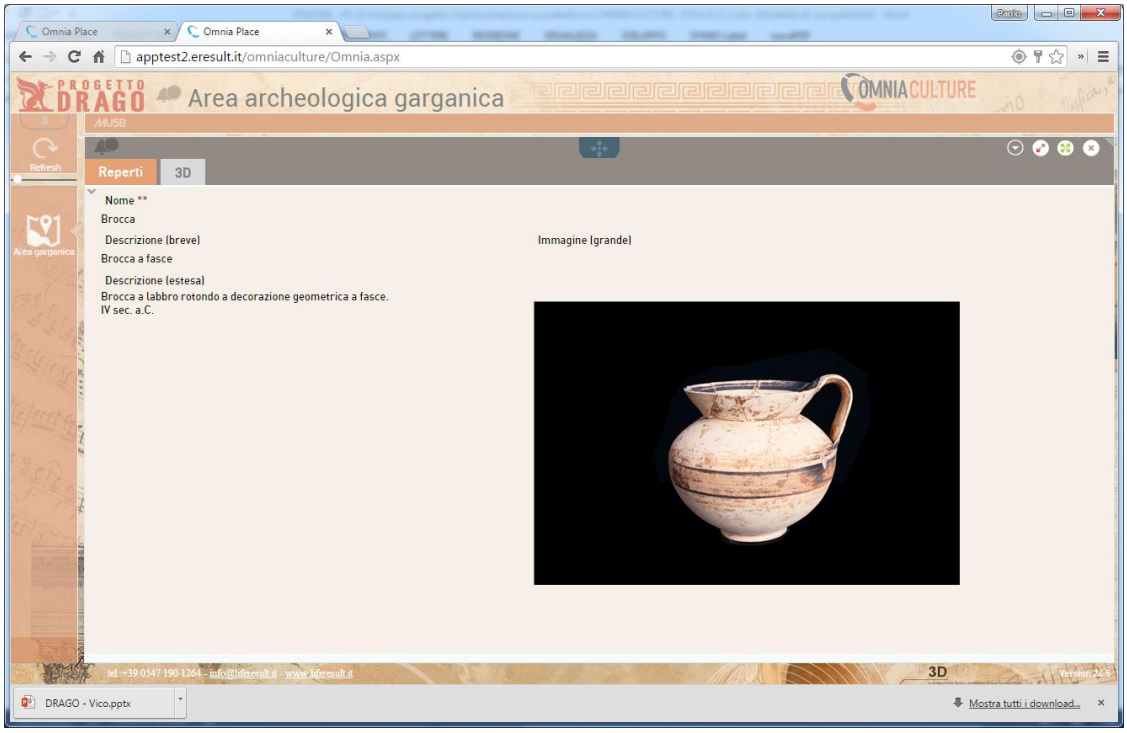
1.2 REPERTI

La scheda “reperti” si presenta con la lista di tutte le immagni acquisite presso il relativo sito archeologico.
Sono organizzate in questo modo: “Immagine + Nome+ descrizione breve”
Selezionando la foto desiderata, si apre all’interno di un frame embedded.

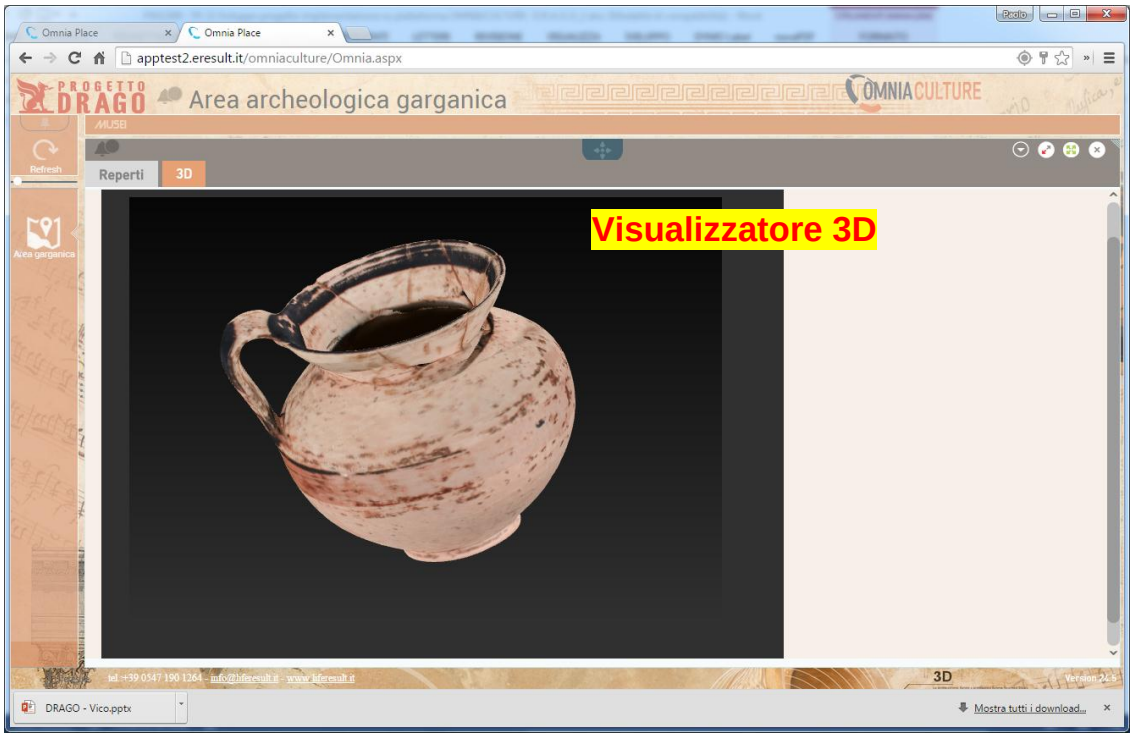


Qui l’utente può visualizzare la foto all’interno di un visualizzatore più ampio. Inoltre una scheda descrittiva contenente una descrizione lunga approfondisce quanto presente in fotografia.
Una volta terminata la visualizzazione si può tornare all’elenco delle scansioni semplicemente facendo click sul comune tasto “X” presente nella parte superiore destra del frame.

Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)



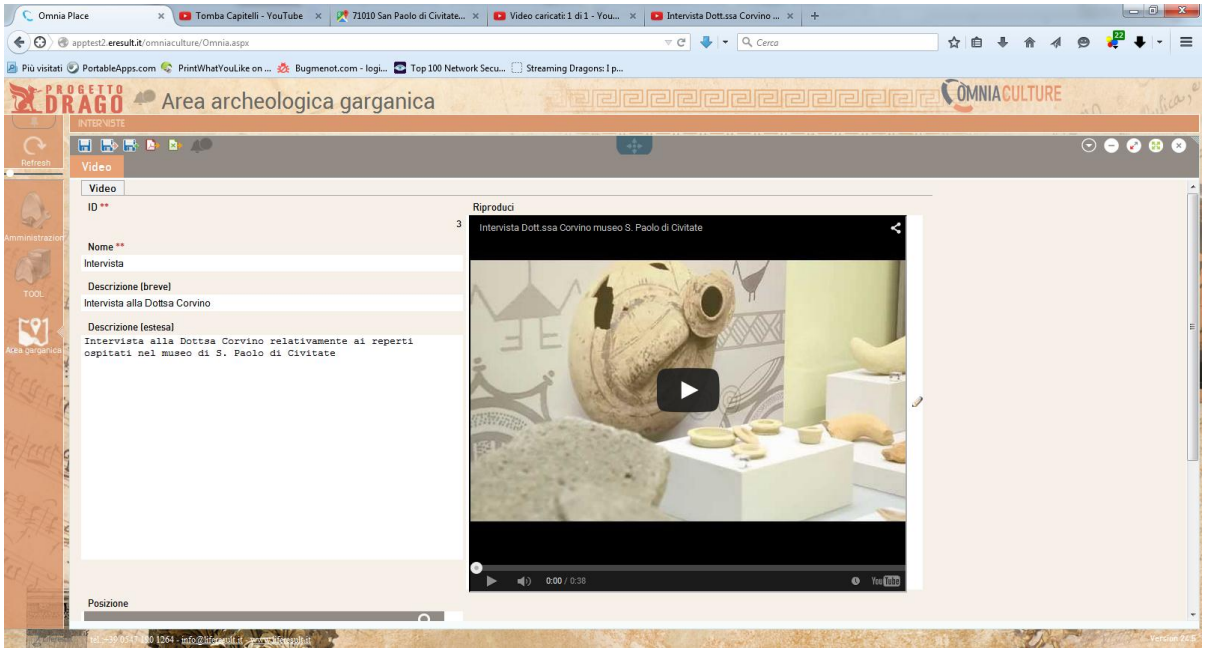
La scheda scansioni si presenta con la lista di tutte le scansioni acquisite presso il relativo museo, queste si presentano con il nome del reperto, una descrizione breve ed una relativa foto. Qui l'utente può fruire della visualizzazione 3D e di una descrizione più approfondita. Una volta terminata la visualizzazione si può tornare all'elenco delle scansioni semplicemente facendo click sul comune tasto "X" presente nella parte superiore destra della scheda parallela



Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)

1.3 INTERVISTA

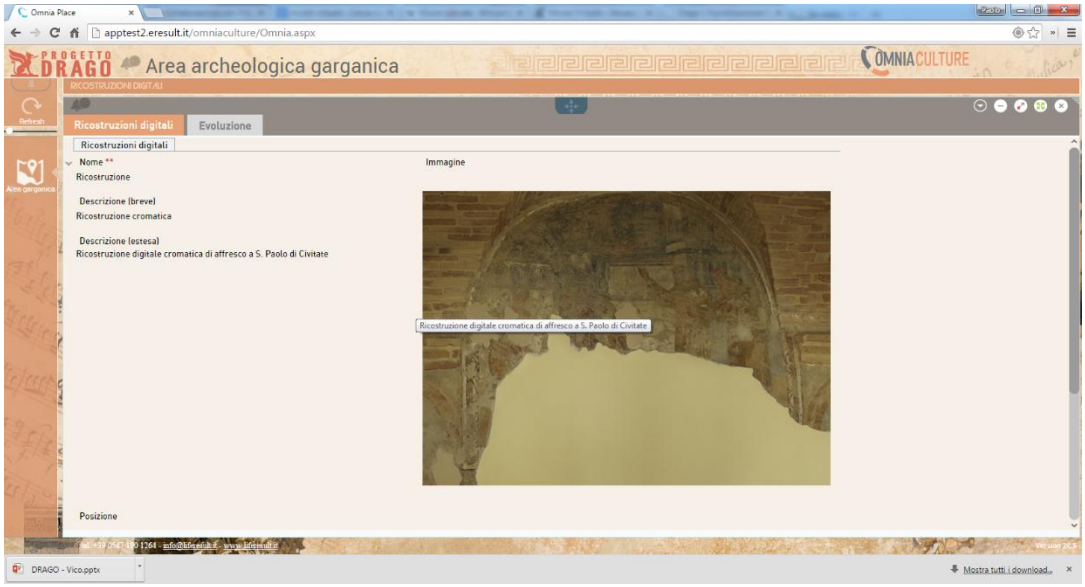
La scheda intervista si presenta con la lista di tutte le interviste acquisite presso il relativo sito archeologico. Queste si presentano con il nome “intervista + nome intervistato + sito archeologico”.
Selezionando l’intervista desiderata, si apre in un frame embedded.
Qui l’utente può fruire della videointervista. Una volta terminata la visualizzazione si può tornare all’elenco delle interviste semplicemente facendo click sul comune tasto “X” presente nella parte superiore destra del frame

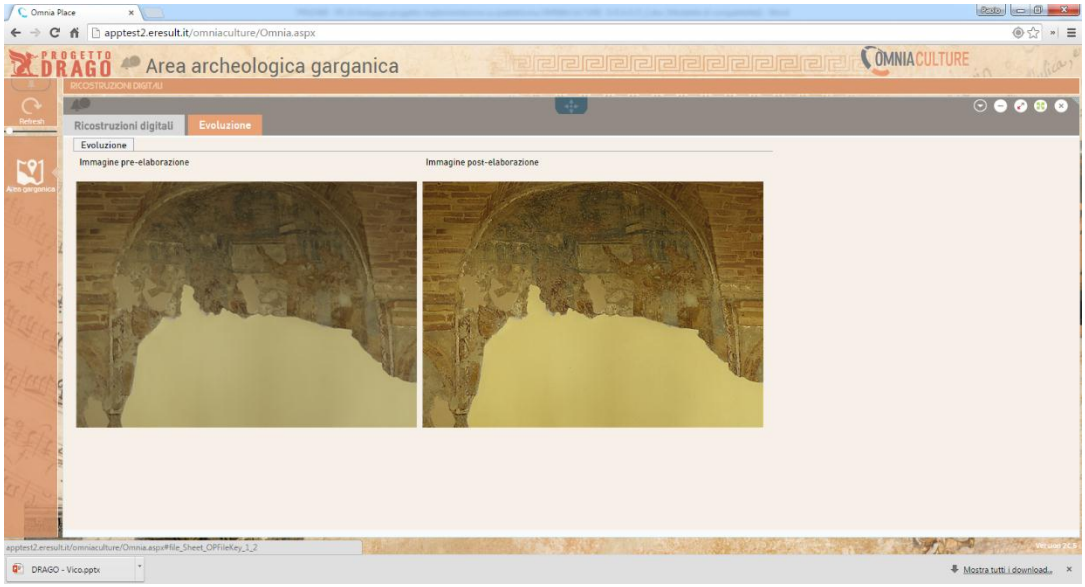


Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)

1.4 RICOSTRUZIONI

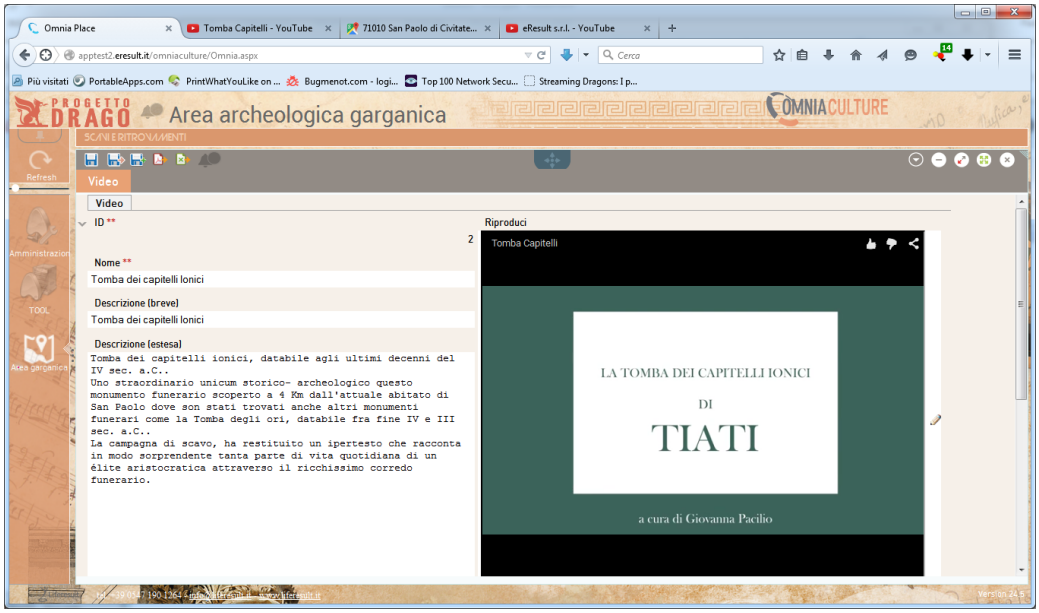
La scheda “RICOSTRUZIONI” si presenta con la lista di tutti gli elementi ricostruiti digitalmente acquisiti sempre presso il sito archeologico relativo alla sottocategoria selezionata.
Sono organizzate in questo modo: “Immagine + Nome+ descrizione breve”
Selezionando ‘elemento desiderato, questo si apre all’interno di un frame embedded.
Qui l’utente può visualizzare e confrontare l’elemento prima e dopo il restauro digitale.
Entrambe le immagini sono ulteriormente selezionabili e dunque visualizzabili in una pagina dedicata che restituisce una maggiore proporzione di visualizzazione dei file che rende quindi possibile una più corretta e approfondita osservazione dello stesso.
Come per le altre schede parallele, una volta terminata la visualizzazione si può tornare all’elenco delle ricostruzioni semplicemente facendo click sul comune tasto “X” presente nella parte superiore destra del frame.





1.5 SCAVI E RITROVAMENTI

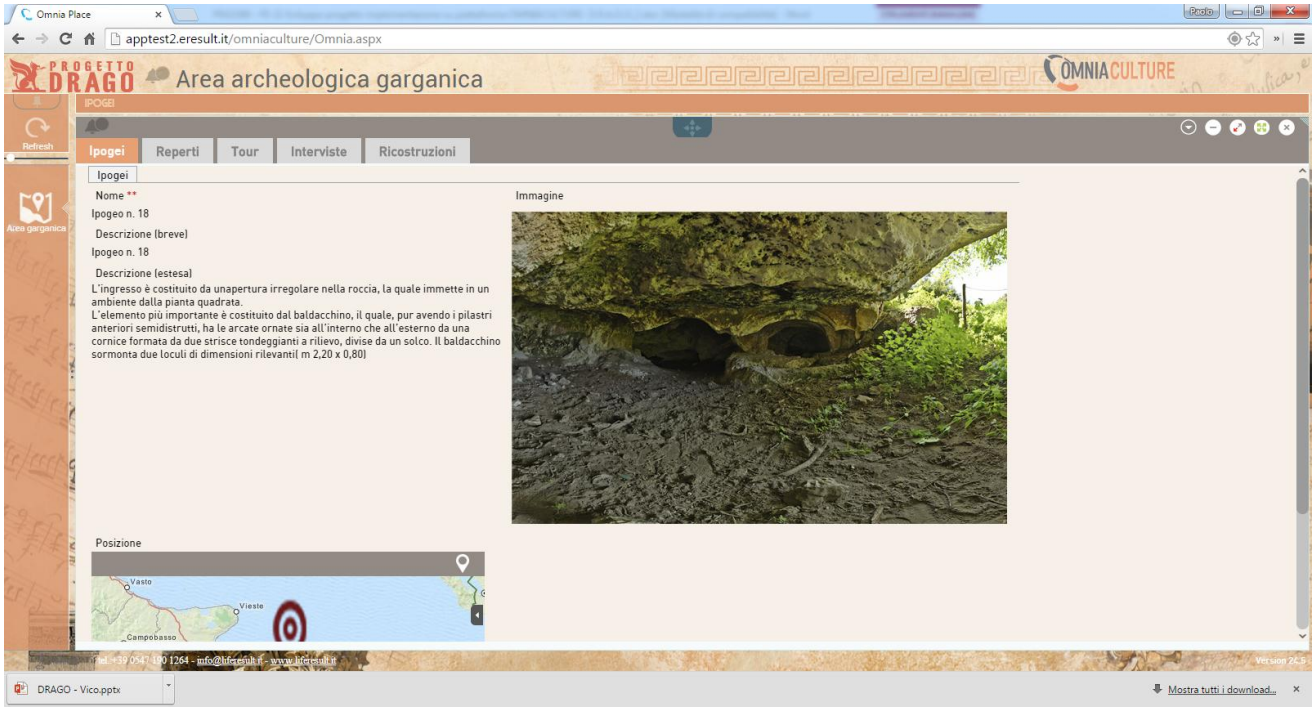
La scheda “SCAVI E RITROVAMENTI” si presenta con la lista di tutti i video inerenti al sito archeologico relativo alla sottocategoria selezionata.
Sono organizzate in questo modo: “Nome+ sito archeologico”
Selezionando ‘elemento desiderato, questo si apre all’interno di un frame.
Qui l’utente può visualizzare l’elemento all’interno di un player collocato con un <embed> all’interno della scheda stessa. I video possono essere collegati da un link Youtube.
Il video può essere visualizzato anche in una pagina dedicata che ne permette la visualizzazione con proporzioni maggiori.
Come per le altre schede parallele, una volta terminata la visualizzazione si può tornare all’elenco delle scansioni semplicemente facendo click sul comune tasto “X” presente nella parte superiore destra del frame.



ULTERIORE INGRANDIMENTO DEL VIDEO IN UN’ALTRA PAGINA (PER ESEMPIO YOUTUBE)

1.6 IPOGEI

La scheda “Ipogei” si presenta con la lista di tuti gli ipogei e relative foto.
Sono organizzate in questo modo: “Immagine + Nome+ sito archeologico”
Selezionando l’elemento desiderato, questo si apre all’interno di un livello parallelo.
Qui l’utente può visualizzare la foto incasellata all’interno di una finestra più ampia. Inoltre una scheda descrittiva contenente una descrizione lunga approfondisce quanto presente in fotografia.
La sezione 360° da all’utente la possibilità di esplorare l’elemento tramite un tour virtuale collegato tramite un <Embed> all’interno della stessa scheda.
Il pulsante “reperti”, situato nella sezione inferiore della scheda, trasporta l’utente su un’altra scheda parallela all’interno della quale c’è la lista degli elementi ritrovati all’interno del luogo in cui si stava navigando.
A seconda dell’elemento scelto e selezionato dall’utente , questo si apre in un’altra scheda parallela.
Una volta terminata la visualizzazione si può tornare all’elenco delle scansioni semplicemente facendo click sul comune tasto “X” presente nella parte superiore destra di tutte le schede parallele.



Per ogni categoria, la visualizzazione della sottocategoria sarà la stessa finora illustrata. L’elenco presente nella colonna delle sottocategorie varierà in base alla categoria selezionata. L’unica categoria a non avere la sottocategoria è “Area archeologica garganica” che viene presentato all’interno dell’area di visualizzazione a mezzo GIS.

2 ALGORITMI IMPLEMENTATI

Per le attività di acquisizione delle immagini e la realizzazione dei modelli 3D e tour virtuale, si sono sviluppati software in grado di eseguire funzionalità avanzate di elaborazione in post-processing per reperti e fotografie.
In particolare, gli strumenti implementati sono in grado di eseguire operazioni di camera calibration applicando algoritmi di lens undistortion e demosaicking. Queste funzioni sono necessarie a migliorare la resa qualitativa delle immagini acquisite e a migliorare le performance degli algoritmi di elaborazione e ricostruzione 3D delle stesse. Sono state considerate le tecniche avanzate basate sulla geometria prospettiva e sulla teoria di elaborazione digitale dei segnali al fine di raggiungere il miglior compromesso tra qualità dei risultati e costo computazionale. Quest'ultimo aspetto risulta di particolare criticità nel progetto vista la quantità e le dimensioni delle immagini da elaborare. Le performance qualitative e quantitative del software sono state verificate sulle immagini acquisite durante il corso del progetto.

2.1 Camera calibration: lens undistorsion

Nella gran parte dei casi i sistemi ottici forniscono un'immagine abbastanza fedele della scena che si osserva. A volte, però, le immagini osservate presentano delle differenze rispetto all'immagine desiderata. Tali variazioni sono per lo più riscontrabili nelle zone più lontane dal centro dell'immagine.
L'obbiettivo di una qualsiasi camera fotografica, per quanta cura possa essere stata posta nella sua progettazione e costruzione, non è mai totalmente privo di difetti.

2.1.1 Il problema della distorsione

La distorsione è un'aberrazione ottica extra-assiale per cui l'ingrandimento lineare di una lente varia con la distanza r dall'asse ottico.
Si possono distinguere sostanzialmente due tipi di distorsione radiale detti di tipo 'barrel' e di tipo 'pin cushion'. La distorsione di tipo barrel trasforma un oggetto che per esempio è perfettamente quadrato o rettangolare in un altro oggetto che appare sulla fotografia con i contorni non più squadrati bensì leggermente curvi con una curvatura convessa verso l'interno dell'oggetto. Viceversa, la distorsione di tipo 'pin cushion' spinge verso i quattro angoli dell'immagine gli oggetti vicini a tali angoli:

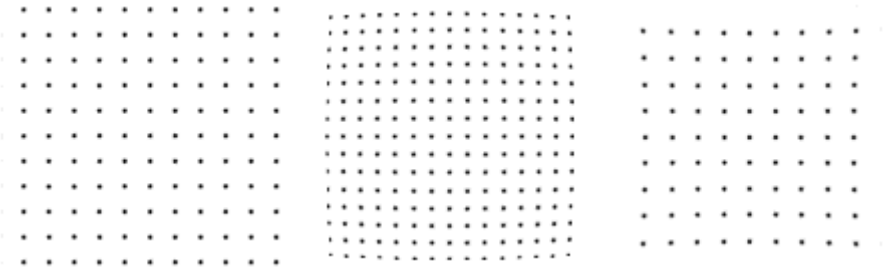


Figura 1 – reticolo non distorto, distorsione barrel e distorsione pin cushion

Il risultato definitivo che comporta la distorsione è quello di spostare ogni pixel dell' immagine reale dalla posizione corretta in una posizione differente determinata appunto dall' entità dell' effetto distorcente e quindi dalla qualità del sistema ottico interno all' apparecchio fotografico.
Un modello matematico che può essere usato per la compensazione digitale della distorsione radiale prevede l' utilizzo di un polinomio con soli termini di grado dispari del tipo:

$$r_{reale} = k_1 * r_{oss} + k_3 * r_{oss}^{\ 3} + k_5 * r_{oss}^{\ 5} \quad (1)$$

dove k_1, k_3 e k_5 sono opportuni coefficienti reali mentre r_{reale} è la distanza radiale del pixel in esame dal centro dell' immagine che avremmo se non ci fosse distorsione radiale mentre a causa di questa il pixel appare nell'immagine ad una distanza radiale r_{oss} dal centro dell'immagine.

L'entità dell'aberrazione si esprime convenzionalmente come deviazione percentuale riferita all'angolo del campo inquadrato: positiva per l'aberrazione a cuscinetto, negativa per quella a barilotto.

2.1.2 Algoritmo correttivo

L' algoritmo di compensazione implementato prevede i seguenti passi:

1. Individuare le coordinate del centro dell' immagine.
2. Calcolare la distanza radiale r_{oss} di ciascun pixel dell' immagine distorta dal centro.
3. Applicare la trasformazione (1) per ciascun pixel.
4. Spostare il pixel dalla posizione ross alla posizione rreale appena calcolata.

Osservare che lo spostamento di ogni pixel avviene esclusivamente in modo radiale cioè l'angolazione del punto in esame dal centro dell'immagine rimane invariata.
Notare inoltre che il procedimento appena descritto comporterebbe una grossa introduzione di alias nell'immagine compensata perchè bisognerebbe ricorrere ad una discretizzazione delle distanze risultanti dalla funzione di trasformazione (1) affinchè il pixel spostato capiti in una posizione consentita dell'immagine compensata. Infatti è ben noto che tutti i pixel di un'immagine sono caratterizzati da un numero di riga e di colonna e tale numero è ovviamente un numero intero. Tale condizione non sarebbe rispettata se non si provvedesse alla discretizzazione delle due nuove coordinate del pixel ricavate dall'applicazione della funzione di trasformazione. Per questo motivo si è scelto di implementare un diverso algoritmo che utilizzi l'interpolazione tra pixel in modo da ridurre al minimo l'effetto dell'alias aggiunto con la compensazione della distorsione. Si deve pertanto usare una diversa funzione di trasformazione:

$$r_{oss} = t * r_{reale} + a * r_{reale}^{\ 3} + b * r_{reale}^{\ 5} \quad (2)$$

dove i coefficienti sono diversi da quelli della funzione precedente. Il nuovo algoritmo di compensazione è il seguente:

1. Si costruisce l'immagine compensata calcolando per ogni pixel (che è ancora da identificare) la distanza radiale dal centro.
2. Si applica la trasformazione (2) ponendo come r_{reale} il valore calcolato al punto 1.
3. Dato il valore r_{oss} calcolato con la (2) si determinano le coordinate del pixel relativo e si effettua l'interpolazione tra pixel per ottenere il pixel interpolato.

4. Si associa al pixel dell'immagine compensata del punto 1. il valore interpolato del pixel del punto 3.
5. Ripetendo i punti sopra per tutti i pixel dell'immagine definita al punto 1. si riesce a costruire l'immagine compensata definitiva priva di alias.

I coefficienti a e b nella (2) devono essere scelti in modo da eliminare l' effetto della distorsione radiale nell'immagine compensata. Il coefficiente t può essere scelto in modo che l'immagine compensata risulti normalizzata rispetto all'immagine di partenza.
In altri termini possiamo fare in modo che la massima distanza radiale dell'immagine distorta rimanga invariata nell'immagine compensata e quindi imponiamo che

$$r_{reale} = r_{oss} = r_{MAX}$$

e siccome

$$r_{oss} = t*r_{reale} + a*r_{reale}^{3} + b*r_{reale}^{5}$$

si ottiene che

$$r_{MAX} = t*r_{MAX} + a*r_{MAX}^{3} + b*r_{MAX}^{5}$$

da cui

$$t = 1 - a*r_{MAX}^{2} - b*r_{MAX}^{4}$$

e

$$r_{oss} = (1 - a*r_{MAX}^{2} - b*r_{MAX}^{4})*r_{reale} + a*r_{reale}^{3} + b*r_{reale}^{5}$$

Questa relazione ha solo due gradi di libertà perchè si ha la possibilità di agire solo sui parametri a e b mentre il terzo parametro è vincolato dalla scelta dei parametri precedenti e dalla dimensione radiale massima dell'immagine.

2.1.3 Esempi applicativi

Di seguito alcuni esempi di applicazione dell'algoritmo implementato:

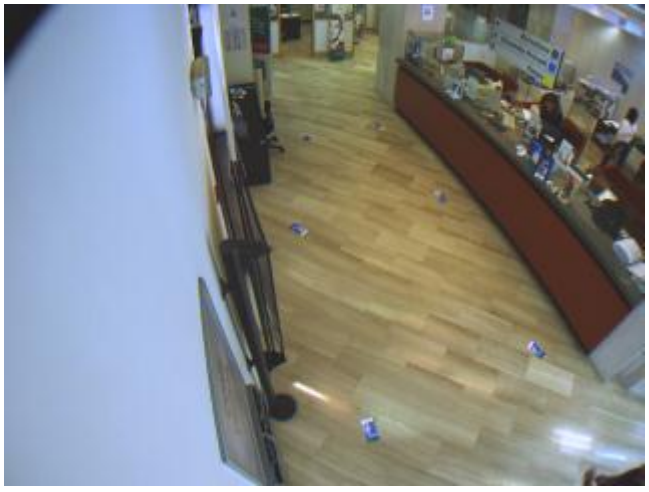


Figura 2 – Esempio 1: frame distorto acquisito dalla telecamera

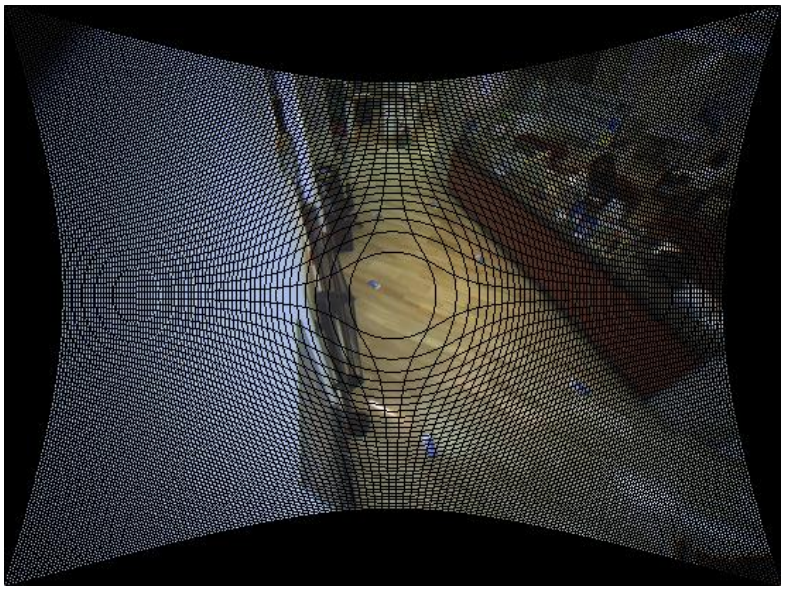


Figura 3 – Esempio 1: frame anti-distorto. I pixel neri corrispondono ai punti dell'immagine di cui non si ha informazione

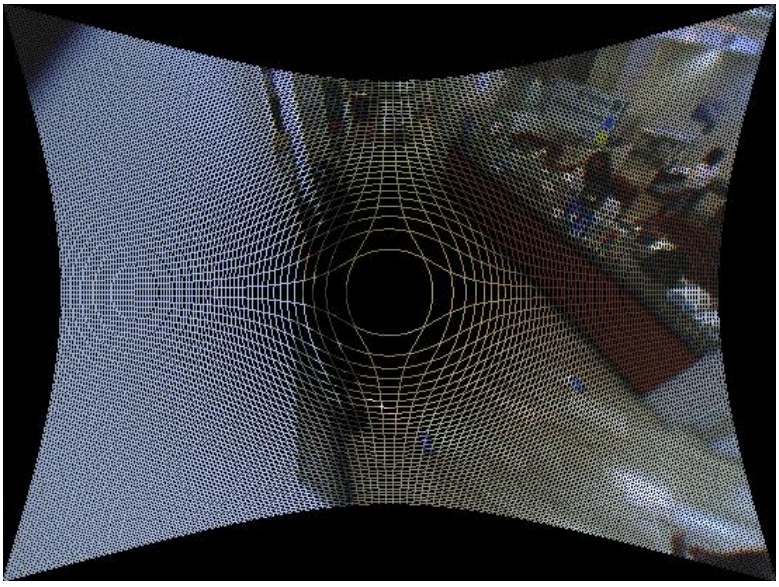


Figura 4 – Esempio 1: i pixel di cui non si hanno informazione sono ricavati per interpolazione

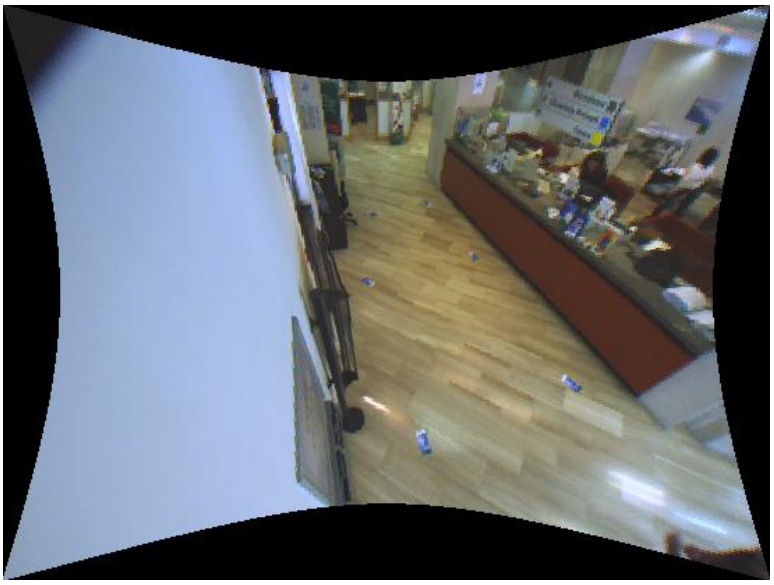


Figura 5 – Esempio 1: immagine risultante

Figura 6 - Esempio 2: frame distorto acquisito dalla telecamera

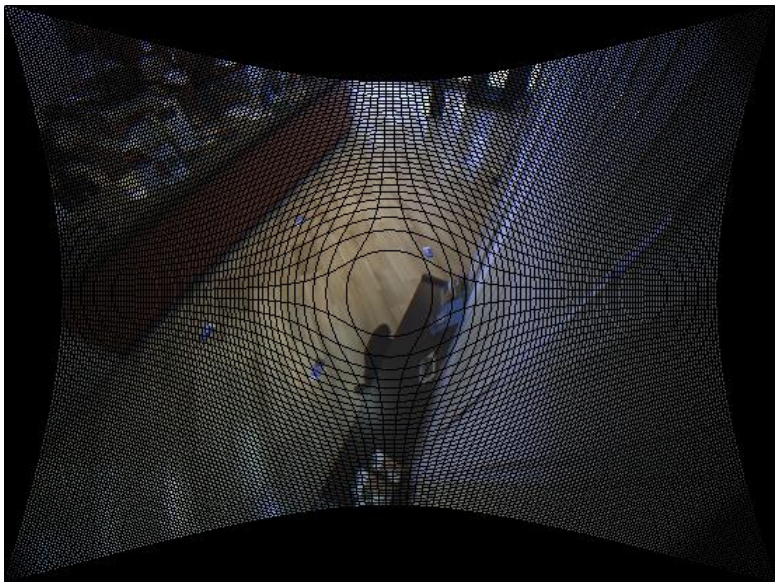


Figura 7 - Esempio 2: frame anti-distorto. I pixel neri corrispondono ai punti dell'immagine di cui non si ha informazione

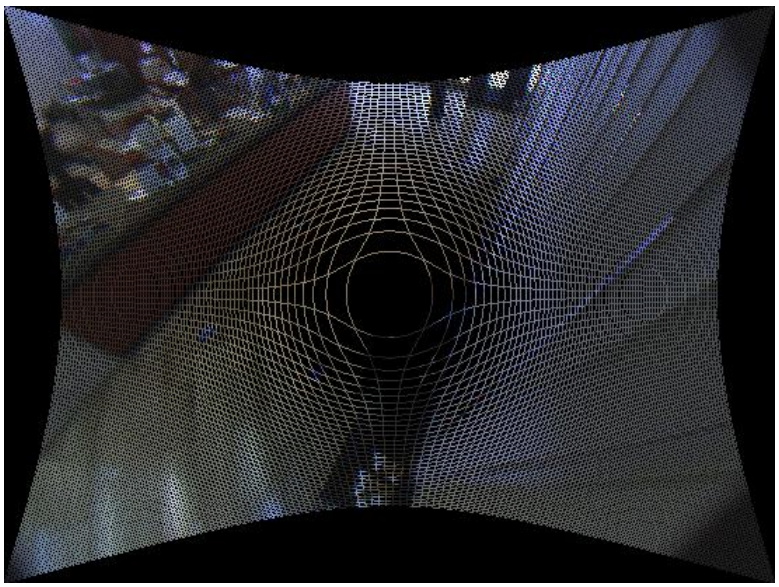


Figura 8 - Esempio 2: i pixel di cui non si hanno informazione sono ricavati per interpolazione

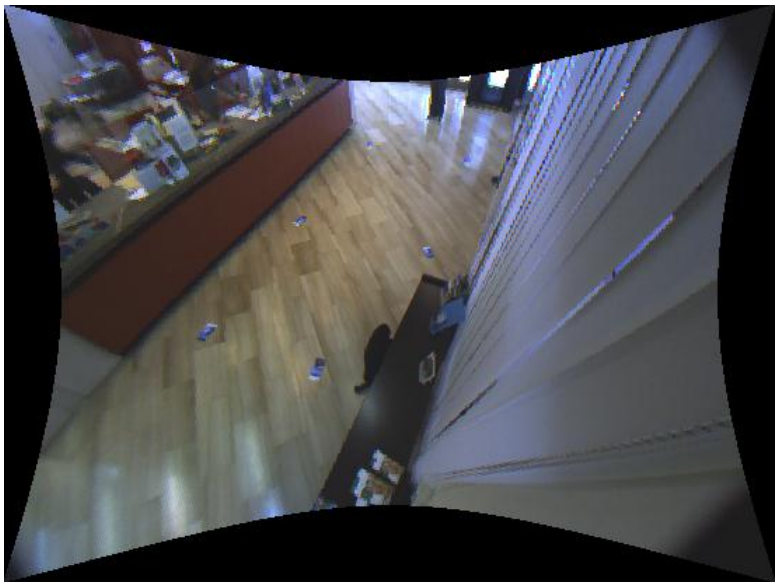


Figura 9 - immagine risultante

2.2 Camera calibration: demosaicking

La rappresentazione di un'immagine digitale a colori richiede la conoscenza di almeno tre campioni di colore per ogni pixel. Solitamente vengono utilizzati il rosso, il verde ed il blu, basi dello spazio di colore RGB. Pertanto l'acquisizione di un'immagine a colori da parte di una fotocamera digitale necessita di tre sensori per pixel, ognuno sensibile ad una specifica lunghezza d'onda. Il posizionamento dei tre sensori non è semplice e comporta alcuni inconvenienti di sfasamento se questi sensori vengono disposti sullo stesso piano con l'utilizzo di un prisma per indirizzare la luce, o di allungamento dei tempi di esposizione nel caso di sensori disposti in serie, come nel sensore Foveon. Quindi la maggior parte delle fotocamere e telecamere digitali non acquisisce l'immagine con tre componenti, bensì utilizza un solo sensore per pixel, ottenendo una griglia di valori corrispondenti ai diversi colori. Tale griglia, detta anche Color Filter Array (CFA) oppure mosaico, alterna campioni di rosso, verde e blu secondo alcune disposizioni specifiche, la più nota dalle quali viene chiamata Bayer pattern, dal nome del suo inventore Bryce Bayer.

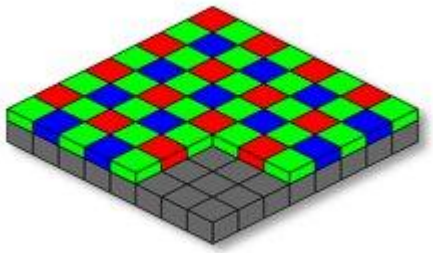


Figura 10 - Matrice di Bayer

L'immagine a colori viene ricostruita in un secondo momento attraverso un'interpolazione dei dati acquisiti dal

sensore, comunemente chiamata demosaicking. Tale ricostruzione, se eseguita in maniera adeguata, è in grado di produrre un'immagine di buona qualità visiva. Essa però necessita di tecniche specifiche dal momento che i normali algoritmi di interpolazione di immagini non sono in grado di sfruttare a fondo l'informazione contenuta nella griglia di valori acquisiti dal sensore. Pertanto sono stati sviluppati molti algoritmi in grado di ottenere ricostruzioni efficienti, con diversi trade-off tra complessità computazionale e prestazioni.

Il demosaicking è il processo che permette di ottenere un'immagine full RGB a partire da un'immagine acquisita con un CFA, in questo capitolo verranno descritte le principali tecniche di demosaicking suddivise in quattro aree di appartenenza: metodi euristici, metodi edge-directed, metodi nel dominio della frequenza, metodi basati sulla trasformata wavelet ed infine metodi di ricostruzione.

2.2.1 Interpolazione bilineare

Le prime tecniche di interpolazione per la ricostruzione di un'immagine completa a colori sono molto semplici, intuitive e veloci dal punto di vista computazionale. Una delle prime utilizzate è la semplice trasposizione o spostamento di un pixel vicino, si pensi al solo canale verde nel pattern Bayer, esistono metà pixel già identificati e metà da ricostruire, ciascun pixel verde viene ricopiato dal suo vicino a sinistra, un analogo procedimento viene poi applicato agli altri canali; ovviamente questa è una tecnica di interpolazione molto rude che porta moltissimi artefatti cromatici e ad una bassa resa visiva.
Una tecnica più interessante ed ancora molto diffusa seppur molto basilare è l'interpolazione bilineare, ciascun pixel viene stimato come una media aritmetica dei pixel vicini; si consideri il canale verde nel pattern Bayer come in Figura 11, il pixel in posizione centrale viene calcolato in questo modo:

$$G(2,3) = \frac{1}{4}G(1,3) + \frac{1}{4}G(2,2) + \frac{1}{4}G(2,4) + \frac{1}{4}G(3,3)$$

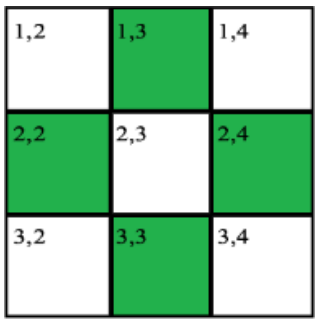


Figura 11 - canale verde

Per gli altri canali ovviamente si possiedono meno pixel già assegnati, si ricorre quindi ad un procedimento analogo in due step, prima si calcolano i punti della cornice del blocchetto in Figura 12, successivamente il punto centrale esattamente come nel canale verde:

$$R(2,2) = \frac{1}{2}[R(1,2) + R(3,2)]$$

$$R(2,4) = \frac{1}{2}[R(1,4) + R(3,4)]$$

$$R(1,3) = \frac{1}{2}[R(1,2) + R(1,4)]$$

$$R(3,3) = \frac{1}{2}[R(3,2) + R(3,4)]$$

$$R(2,3) = \frac{1}{4}[R(1,3) + R(2,2) + R(2,4) + R(3,3)]$$

Il procedimento è simile per il canale blu con gli opportuni indici, per quanto riguarda il bordo dell'immagine bisogna applicare un approccio leggermente diverso (come capita in ogni algoritmo di demosaicking) sempre attraverso delle opportune medie o traslazioni di pixel.

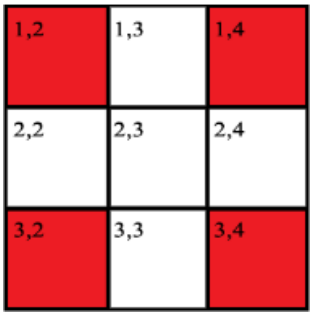


Figura 12 - canale rosso

L'interpolazione bilineare è un metodo di demosaicking ancora molto utilizzato tuttavia è risaputo che il suo funzionamento è accettabile solo nelle zone di colore uniforme ed introduce molti errori lungo i contorni e nelle zone ad alta frequenza. L'errore che affligge maggiormente questo metodo è lo Zipper Effect il quale si presenta nei punti dove il colore subisce un cambio brusco, questo tipo di errore viene risaltato nelle componenti ad alta frequenza ad esempio nella palizzata in Figura 13.

Figura 13 - Esempio di interpolazione bilineare

2.2.2 Metodi euristici

Gli approcci di ricostruzioni delle immagini euristici si fondano su assunzioni e considerazioni sui colori e sulle immagini piuttosto che su strutture matematiche.
Precedentemente si era detto che l'interpolazione bilineare fallisce laddove si presenta un cambio repentino di colore nell'immagine, questa problematica è stata affrontata da Cok [1].
Il metodo di Cok cerca di mantenere costante il colore tra pixel vicini, l'immagine ricostruita del canale verde viene ottenuta tramite interpolazione bilineare, invece per quanto concerne i canali rosso e blu, viene effettuata una media dei valori di tonalità.
L'algoritmo si basa sul concetto di tonalità come un rapporto tra i valori del colore in questione sul valore del verde ricostruito, ad esempio si consideri la Figura 14, volendo ricostruire il valore B(2,2), dopo che si era già ricostruito totalmente il canale verde, bisogna calcolare:

$$B(2,2) = \frac{1}{2} G(2,2) \left[\frac{B(1,2)}{G(1,2)} + \frac{B(3,2)}{G(3,2)} \right]$$

$$B(3,3) = \frac{1}{2} G(3,3) \left[\frac{B(3,2)}{G(3,2)} + \frac{B(3,4)}{G(3,4)} \right]$$

$$B(2,3) = \frac{1}{4} G(2,3) \left[\frac{B(3,2)}{G(3,2)} + \frac{B(3,4)}{G(3,4)} + \frac{B(1,2)}{G(1,2)} + \frac{B(1,4)}{G(1,4)} \right]$$

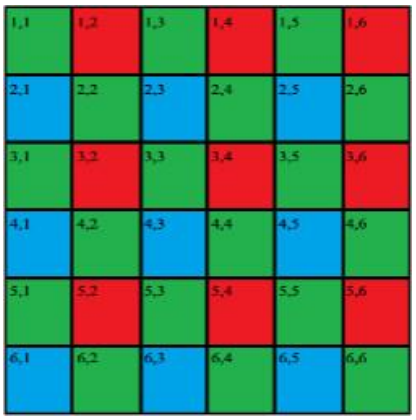


Figura 14 - Bayer pattern

L'approccio al problema della ricostruzione viene invece affrontato da Freeman [2] con l'introduzione dell'operatore mediano: dopo aver applicato l'interpolazione bilineare ai tre canali, per ogni singolo pixel non presente nativamente nel Bayer pattern, viene applicata una media delle differenze dei valori circostanti, ad esempio si consideri la Figura 12:

$$R(2,3) = G(2,3) + \frac{1}{8} \sum_{\substack{i,j=1,2,3 \\ (i,j)\neq(2,3)}} (R(i,j) - G(i,j))$$

lo stesso si applica al canale blu con ovvie sostituzioni.

Tuttavia entrambi i metodi, sia quello di Cok che di Freeman, anche se portano a migliorie rispetto

all'interpolazione bilineare e restando ad un livello computazionale basso, presentano forti limitazioni, soprattutto lungo i contorni delle figure e nelle componenti ad alta frequenza dell'immagine. Si può comunque affermare che tra i due metodi Freeman ha un comportamento migliore nelle zone di transizione di colore.

Figura 15 - Esempio di metodo di Cok

2.2.3 Metodi nel dominio della frequenza

Esistono tutta una serie di tecniche di ricostruzione delle immagini che sfruttano la correlazione esistente tra i canali e in particolare in questa sezione vengono affrontati i metodi nel dominio della frequenza.

Seguendo l'approccio descritto in *[3]*, partiamo dall'immagine originale X(n) dove sono presenti per intero i tre canali RGB, quando questa viene catturata con un CFA, otteniamo una versione campionata dell'immagine data da f_{CFA} $[n_1, n_2]$ con $\{n_1, n_2\}$ appartenenti alla matrice rettangolare utilizzata Λ, che rappresenta l'output del sensore.

La matrice Λ è ripartita in tre sottoinsiemi disgiunti, Λ_R ,Λ_G e Λ_B , le componenti che stiamo cercando di stimare completamente vengono chiamate f_R, f_G, f_B. Il sottocampionamento può essere rappresentato come una moltiplicazione delle funzioni sottocampionate m_i $[n_1, n_2]$ che hanno valore 1 se appartiene alla matrice Λ, e zero altrimenti.

$$m_G[n_1, n_2] = \frac{1}{2}(1 + (-1)^{n_1 + n_2})$$

$$m_R[n_1, n_2] = \frac{1}{4}(1 + (-1)^{n_1})(1 + (-1)^{n_2})$$

$$m_B[n_1, n_2] = \frac{1}{4}(1 + (-1)^{n_1})(1 - (-1)^{n_2})$$

Il segnale ottenuto dal CFA può venire espresso con:

$$f_{CFA}[n_1, n_2] = \sum_{i \in \{R,G,B\}} f_i[n_1, n_2] m_i[n_1, n_2]$$

Sviluppando i termini e passando alla luminanza e crominanza si ottiene:

$$f_{CFA}[n_1, n_2] = f_L[n_1, n_2] + f_{C1}[n_1, n_2]\, exp\left(\frac{j2\pi(n_1 + n_2)}{2}\right) + f_{C2}[n_1, n_2]\, exp\left(\frac{j2\pi(n_1 - n_2)}{2}\right)$$

La formula appena scritta può essere interpretata come una componente in banda base di luminanza f_L, una componente f_{C1} e una seconda componente f_{C2} entrambe contenenti informazioni di crominanza modulate in banda passante.

Ora utilizzando la trasformata di Fourier si ottiene:

$$F_{CFA}(u, v) = F_L(u, v) + F_{C1}(u - 0.5, v - 0.5) + F_{C2}(u - 0.5, v) - F_{C2}(u, v - 0.5)$$

dove con {u, v} sono state espresse in cicli per pixel (c/px).

Ogni segnale in cui i valori dei tre canali sono tra loro identici $f_R = f_G = f_B$ le componenti di crominanza sono zero (siamo in presenza di un valore in scala di grigi).
In generale le componenti di crominanza avranno minore energia e larghezza di banda rispetto alla componente di luminanza.

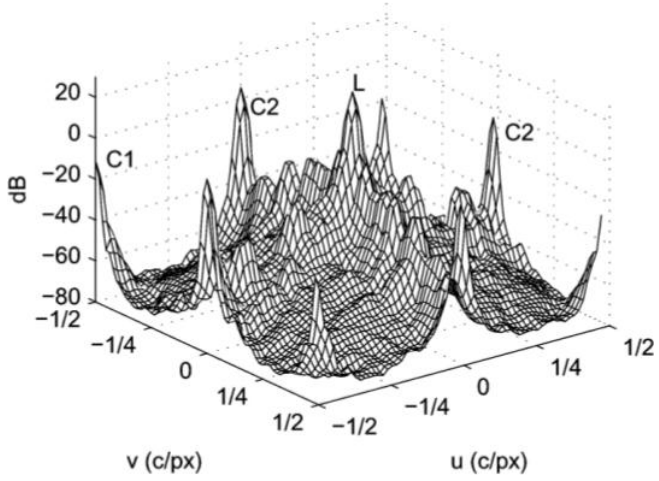


Figura 16 - Esempio di densità spettrale di potenza

Nella Figura 16, si osserva che esiste interferenza o cross-talk molto accentuata tra le componenti di luminosità f_L e di crominanza f_{C2}; da questo spettro è possibile recuperare la componente di luminosità con un filtro passa basso e le componenti di crominanza con dei filtraggi appropriati in banda passante. Successivamente utilizzando le formule inverse introdotte in precedenza si possono trasformare i valori nel dominio RGB. Questo approccio non elimina la sovrapposizione delle componenti f_L e f_{C2}, tuttavia esistendo due copie separate della componente di crominanza C2, ognuna delle quali ha una diversa sovrapposizione con la luminanza è possibile ridurre in modo significativo il cross talk.

Nell'articolo [4] gli autori propongono un metodo capace di ridurre l'aliasing tra le componenti. Si consideri inizialmente il canale verde: in un color filter array di tipo Bayer questo canale presenta il doppio dei campioni rispetto agli altri, nel dominio della frequenza significa che tale canale presenta meno aliasing rispetto al rosso ed al blu ed inoltre che le componenti rosse e blu possono perdere le informazioni delle alte frequenze che sono presenti invece nella componente verde.

Si considera la trasformata zeta del segnale in ingresso:

$$R_S(z_1, z_2) = \frac{1}{4}R(z_1, z_2) - \frac{1}{4}R(-z_1, z_2) + \frac{1}{4}R(z_1, -z_2) - \frac{1}{4}R(-z_1, -z_2)$$

$$B_S(z_1, z_2) = \frac{1}{4}B(z_1, z_2) - \frac{1}{4}B(-z_1, z_2) + \frac{1}{4}B(z_1, -z_2) - \frac{1}{4}B(-z_1, -z_2)$$

$$G_S(z_1, z_2) = \frac{1}{2}G_S(z_1, z_2) + \frac{1}{4}G_S(-z_1, -z_2)$$

dove i termini del tipo $X(z_1, z_2)$ rappresentano i termini utili in bassa frequenza, ed i termini del tipo $X(-z_1, z_2)$, $X(z_1, -z_2)$, $X(-z_1, -z_2)$ rappresentano i termini di aliasing, che possono distorcere l'immagine in ingresso dove questa contiene componenti in alta frequenza.

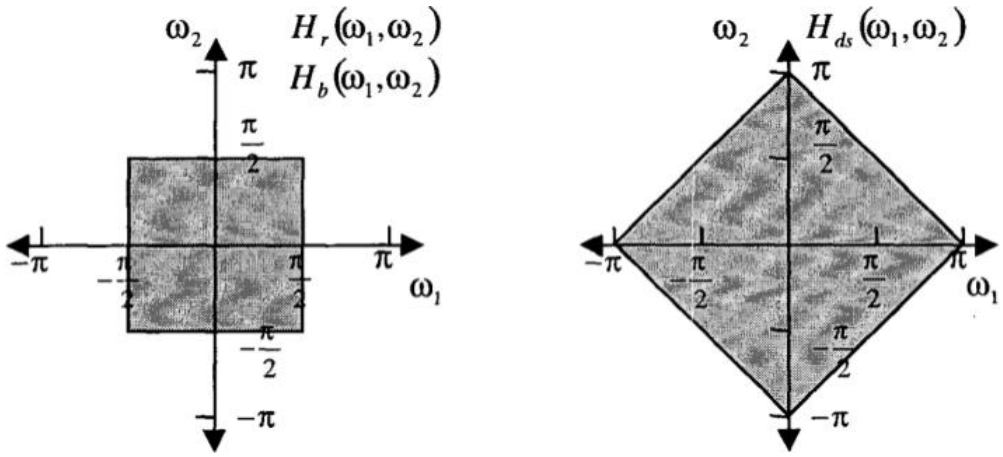


Figura 17 - Filtraggi ideali per l'interpolazione del CFA

In Figura 17 si nota che un CFA introduce una interpolazione dove per le componenti rosse e blu viene utilizzato un filtro di tipo passa-basso rettangolare, mentre per la componente verde viene utilizzato un filtro passa-basso a forma di diamante.

Considerando la forte correlazione esistente tra i canali e che il verde possiede più informazioni rispetto agli altri due, un metodo per ricostruire i canali rosso e blu è utilizzando l'informazione in bassa frequenza dei canali rosso e blu e l'informazione in alta frequenza del canale verde, con questa idea sono stati creati i filtri per le alte frequenze verticale ed orizzontale H_v, H_h in Figura 18, l'uscita di questi filtri viene quindi sommata all'uscita dei filtraggi passa-basso aumentando in questo modo la nitidezza dei canali rosso e blu. Per rimuovere le componenti di aliasing dal canale rosso e blu, è necessario sottrarre ed aggiungere termini del canale verde in questo modo:

$$\hat{R}(z_1, z_2) = R(z_1, z_2)H_r(z_1, z_2) + G_h(z_1, z_2) + G_h(-z_1, z_2) + G_v(z_1, z_2) - G_v(z_1, -z_2)$$

$$\hat{B}(z_1, z_2) = B(z_1, z_2)H_b(z_1, z_2) + G_h(z_1, z_2) - G_h(-z_1, z_2) + G_v(z_1, z_2) - G_v(z_1, -z_2)$$

Dove:

$$G_h(z_1, z_2) = G_s(z_1, z_2)H_h(z_1, z_2)$$

$$G_v(z_1, z_2) = G_s(z_1, z_2)H_v(z_1, z_2)$$

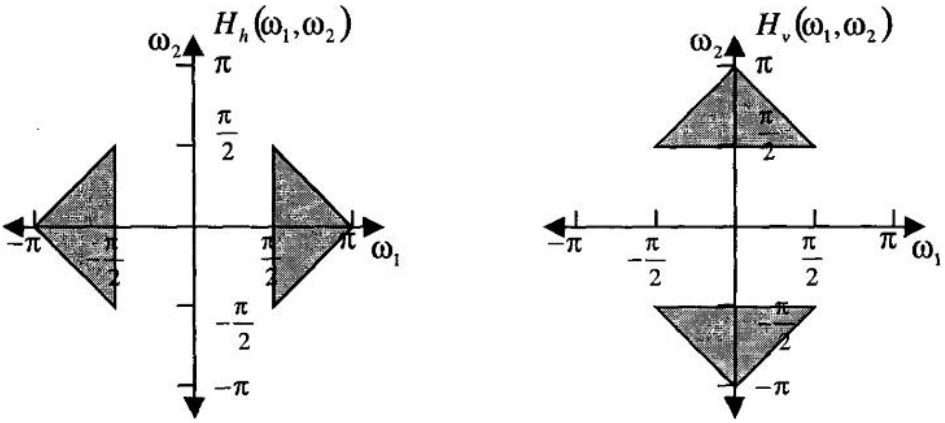


Figura 18 - Filtraggi ideali passa-alto per il canale verde

Nell'articolo [5] viene riportata un'analisi ancora nel dominio della frequenza, i principi alla base di questo metodo sono due: le alte frequenze sono simili tra le tre componenti di colore, e le alte frequenze lungo gli assi orizzontali e verticali sono essenziali per una ricostruzione di qualità; nelle posizioni del rosso e del blu le diverse componenti in frequenza vengono modulate come in Figura 19, mentre nei pixel verdi, che sono disposti a quinconce, i termini di differenza di colore modulati alle frequenze (0, ±π) e (±π, 0) svaniscono.

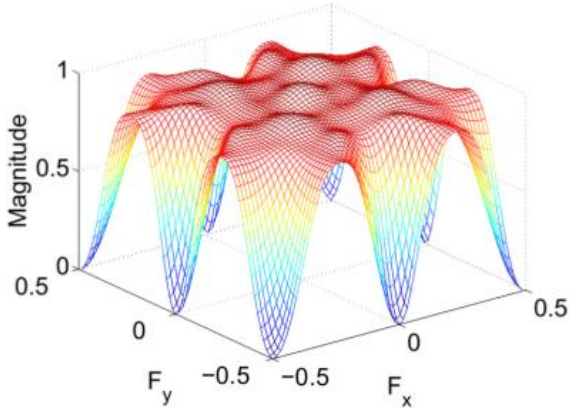


Figura 19 - Trasformata di Fourier del filtro passa-basso proposto in [6]

Nelle posizioni appena descritte ci sono quindi meno sovrapposizioni spettrali tra crominanza e luminanza e quindi la stima della luminanza per i pixel verdi viene calcolata con il filtro passa-basso proposto in Figura 20, mentre per i pixel rossi e blu viene utilizzato un approccio del tipo adattativo. Successivamente i colori vengono ricostruiti con una interpolazione bilineare di $R - \hat{L}, G - \hat{L}, B - \hat{L}$ utilizzando i valori noti dal CFA.

Poiché esiste una forte correlazione tra le componenti di colore nelle regioni ad alta frequenza, anche le componenti ad alta frequenza della luminosità del verde sono fortemente correlate a quelle dei canali rosso e blu.

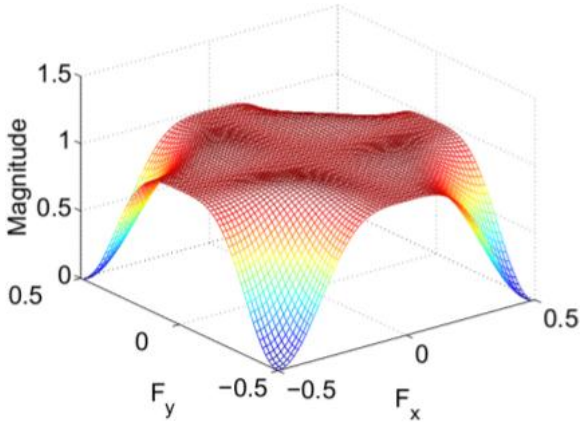


Figura 20 - Spettro del filtro passa-basso per la ricostruzione della luminanza del verde

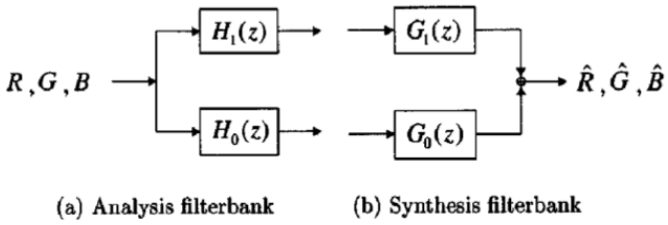


Figura 21 - Banco di filtri utilizzato in [7]

L'algoritmo prevede che inizialmente venga interpolato il canale verde, con un metodo semplice di demosaicking quale il bilineare o edge-directed, poi utilizzando i campioni del blu nel pattern di Bayer si ottiene una versione sottocampionata del canale blu, quindi i valori del canale verde interpolati corrispondenti alle posizioni di blu nel pattern di Bayer vengono utilizzati per formare un'immagine sottocampionata del canale verde. Ora vengono decomposti i due canali sotto-campionati (verde e blu) nelle loro quattro sottobande, ovvero applicando il filtro passa-basso $h_0 = \frac{1}{4}[1,2,1]$ e passa-alto $h_1 = \frac{1}{4}[1,-2,1]$ alle righe e alle colonne per ciascun canale si ottengono le quattro sotto-bande LL, LH, HL, HH. Le sotto-bande ad alte frequenze (HL, LH, HH) del canale verde vengono sostituite con le sottobande corrispondenti del canale blu, quindi viene ricostruito il canale verde e inseriti i pixel nelle loro posizioni corrispondenti nella stima iniziale del canale verde.

Questa procedura viene ripetuta con le opportune sostituzioni anche per la ricostruzione dei pixel verde che sono situati nei pixel rossi nel pattern.

3 BIBLIOGRAFIA

[1] D. R. Cok, "Signal processing method and apparatus for producing interpolation and chrominance values in a sampled color image signal," 1986.

[2] W. T. Freeman, "Method and apparatus for reconstructing missing color samples," 1988.

[3] E. Dubois, "Frequency-Domain Methods for Demosaicking of Bayer-Sampled," *IEEE signal processing letters,* vol. 12, no. 12, december 2005.

[4] J. W. Glotzbach, R. W. Schafer and K. Illgner, "A method of color filter array interpolation with alias cancellation properties," in *Proceedings IEEE International Conference on Image Processing*, 2001.

[5] N. X. Lian, L. Chang, Y. P. Tan and V. Zagorodnov, "Adaptive filtering for color filter array demosaicking," *IEEE Trans. Image Process. 16 (10),* pp. 2515-2525, 2007.

[6] D. Alleysson, S. Susstrunk and J. Herault, "Linear demosaicking inspired by human visual system," *IEEE Trans. on Image Processing,* vol. 14, no. 4, pp. 439-449, 2005.

[7] B. K. Gunturk, Y. Altunbasak and R. M. Mersereau, "Color plane interpolation using alternating projections," *IEEE Trans. Image Process. 11 (9),* pp. 997-1013, 2002.

APPENDICE A) CODIFICA SOFTWARE

d_out_gen.m

```
function [d2u_row, d2u_col, d2u_row_map, d2u_col_map, u2d_row, u2d_col, ...
    u2d_row_map, u2d_col_map, ccd_d_msize_mt, ccd_d_nsize_mt, ccd_u_msize_mt,...
    ccd_u_nsize_mt, ccd_u_link_map, ccd_lensgain_map, ccd_px_fov_map, ccd_und_msize,
ccd_und_nsize] = ...
    d_out_gen(ccd_msize, ccd_nsize, coeff_px_size, ccd_px_size_hor_mt,
ccd_px_size_ver_mt,...
    lens_u_param, ccd_hdiag2_c2c_mt, cam_f_mt)



% This function generate the arrays (d2u_row, d2u_col and u2d_row, u2d_col)
% to pass from distorted images to undistorted images and vice versa.
% Vengono anche generate le matrici d2u_row_map, d2u_col_map, u2d_row_map e
% u2d_col_map. La funzione di queste 4 matrici è identica a quella dei loro
% corrispondenti vettori. All'utente è lasciata la libertà di scegliere se
% adoperare le look up table per passare da coordinate distorte a antidistorte (e
% viceversa) in forma matriciale o vettoriale.
% The function also produces the gain map, the ccd_px_fov_map and the matching map between
```


Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)

```
distorted
% pixels and undistorded real pixels.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%    NOTE    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Nella generazione di d2u e u2d il lens_u_param (e di ccd_hdiag2_c2c_mt)
% adoperato  varrebbe solo per i punti la cui posizione
% in coordinate assolute è inclusa nell'area in metri dell'immagine
% a partire dalla quale è stato calcolato lens_u_param stesso.
% Per coordinate all'esterno di tale area viene comunque adoperato il valore di
lens_u_param
% identificato su un'area più piccola. Per questo motivo la funzione d_output_generation
% non possiede come ingressi spatial_limit_r e spatial_limit_c e non
% effettua un controllo sulle coordinate assolute massime associabili
% a d2u per verificare che esse siano all'interno dell'area valida.
% Si noti che l'informazione sull'estensione dell'area a partire dalla
% quale è stato identificato lens_u_param è mantenuta da ccd_hdiag2_c2c_mt)
% E' importante, però, prestare attenzione all'uso di d2u e u2d che viene
% fatto in seguito, ricordando i limiti spaziali per i quali sono stati
% calcolati.

% La funzione d_output_generation, partendo da indici di pixel, passa a
% lavorare in coordinate metriche assolute per poi ritornare a indici pixel.
% Questa procedura è necessaria se il pixel del ccd può essere
% rettangolare. Nel caso si limita il pixel ad essere solo quadrato si
% può lavorare solo in indici di pixel.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% This function maps an image to a greater undistorted image, so lens_u_param MUST be
% positive.
%
%
% Syntax example:
% [d2u_row, d2u_col, u2d_row, u2d_col, ccd_d_msize_mt, ccd_d_nsize_mt,
% ccd_u_msize_mt, ccd_u_nsize_mt, ccd_u_link_map, ccd_lensgain_map, ccd_px_fov_map] =
%  = d_output_generation(480, 640, 2*(9.9e-6), 0.7, 3.5e-3)
%
%
% INPUTS:
```


Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)

```
%
% ccd_msize           --> number of distorted image rows.
%
% ccd_nsize           --> number of distorted image columns.
%
% lens_u_param        --> undistorting constant (adimensionale). Maggiore è
%                         il suo valore, più intensa è la distorsione.
%                         Attenzione al fatto che lens_u_param consente di
%                         passare da coordinate distorte a antidistorte
%                         (attraverso la formula del guadagno adoperata in
%                         seguito).
%
% coeff_px_size       --> coefficiente moltiplicativo per le dimensioni
%                         del pixel del ccd. Questo coefficiente tiene
%                         conto dei meccanismi di accorpamento di più
%                         pixel che si verificano quando il ccd funziona
%                         ad una risoluzione inferiore a quella massima.
%
% ccd_px_size_hor_mt  --> dimensione in metri lungo le colonne di un pixel
%                         del ccd distorto;
%
% ccd_px_size_ver_mt  --> dimensione in metri lungo le righe di un pixel
%                         del ccd distorto;
%
% ccd_hdiag2_c2c_mt   --> (half diagonal squared center to center)
%                         distanza al quadrato tra il centro di una delle
%                         immagini da cui è stato identificato lens_u_param
%                         ed il centro di uno dei
%                         quattro pixel ai vertici dell'immagine stessa.
%                         E' in metri. Questo parametro è associato
%                         indissolubilmente al valore di lens_u_param
%                         trovato e fa parte del modello di distorsione
%                         identificato. Pur potendo unire lens_u_param e
%                         ccd_hdiag2_c2c_mt in un unico parametro si è
%                         deciso di tenerli separati per semplicità.
%
% cam_f_mt            --> camera focus in meters.
%
%
% OUTPUTS:
```

```
%
% d2u_row             --> vector to pass from distorted to undistorted
%                         images row component.
%
% d2u_col             --> vector to pass from distorted to undistorted
%                         images col component.
%
% d2u_row_map         --> matrice delle dimensioni di ccd_msize x ccd_nsize.
%                         d2u_row_map(i,j) presenta il valore di coordinata riga
%                         antidistorta per il pixel (i,j) del ccd distorto.
%
% d2u_col_map         --> matrice delle dimensioni di ccd_msize x ccd_nsize.
%                         d2u_col_map(i,j) presenta il valore di coordinata
%                         colonna antidistorta per il pixel (i,j) del ccd distorto
%
% u2d_row             --> vector to pass from undistorted to distorted
%                         images row component.
%
% u2d_col             --> vector to pass from undistorted to distorted
%                         images col component.
%
% u2d_row_map         --> matrice delle dimensioni di ccd_und_msize x ccd_und_nsize.
%                         u2d_row_map(i,j) presenta il valore di coordinata riga
%                         distorta per il pixel (i,j) del ccd antidistorto.
%
% u2d_col_map         --> matrice delle dimensioni di ccd_und_msize x ccd_und_nsize.
%                         u2d_col_map(i,j) presenta il valore di coordinata
%                         colonna distorta per il pixel (i,j) del ccd
%                         antidistorto.
%
% ccd_d_msize_mt      --> metric dimension of the distorted image
%                         (in the rows direction).
%
% ccd_d_nsize_mt      --> metric dimension of the distorted image
%                         (in the columns direction).
%
% ccd_u_msize_mt      --> metric dimension of the undistorted image
%                         (in the rows direction).
%
% ccd_u_nsize_mt      --> metric dimension of the undistorted image
```


Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)

```
%                          (in the columnss direction).
%
% ccd_u_link_map       --> matrix of same size of undistorted image. The
%                          cell (i,j) of this matrix stores 1 if the
%                          correspondent pixel come from an actual pixel in
%                          the distorted image. 0 otherwise.
%
% ccd_lensgain_map     --> CCD_D_MSIZExCCD_D_NSIZE matrix. The (i,j) element stores the
%                          value of the gain of the distortion for the i,j
%                          coordinates.
%
% ccd_px_fov_map       --> CCD_D_MSIZExCCD_D_NSIZE matrix. The (i,j) element stores the
%                          value of the field of wiew for the single i,j
%                          pixel.
%
% ccd_und_msize        --> numero di righe del ccd antidistorto
%
% ccd_und_nsize        --> numero di colonne del ccd antidistorto


%%%%%%%%%%%%%%%%%%%%%%%%%  SECTION 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%  generating arrays and matrices to pass from undistorted  %%%%%%%%
%%%%%%%%%%%%%%%%   to distorted and vice versa      %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% ccd_und_msize   --> number of rows of the undistorted image;
% ccd_und_nsize   --> number of columns of the undistorted image;

% NOTA
% I sistemi di riferimento usati sono due: il sistema di riferimento
% matrice one based ed il sistema di riferimento metrico assoluto.
% Per sistema di riferimento matrice one based si intende
% un sistema di riferimento nel quale il punto di coordinate (ii,jj) corrisponde
% al centro del pixel (ii,jj).
% Ad esempio, il centro del pixel (1,1) è posizionato nelle coordinate (1,1) nel
% sistema di riferimento assunto.
%
% Il sistema di riferimento metrico assoluto è orientato come il sistema di
% riferimento matrice one based, ma è in metri ed ha origine in (ccd_center_row,
ccd_center_col)
% Il termine 'assoluto' serve a sottolineare il contrasto con il sistema di
```


Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)

```
% riferimento in pixel, le cui reali misure cambiano relativamente alla
% dimensione del pixel.
%
% In particolare nel corso della funzione si parte dal sistema di
% riferimento matrice one based, si passa a lavorare nel metrico assoluto e si ritorna
% a quello matrice one based.


% Definizione delle dimensioni metriche assolute  dell'immagine distorta in ingresso.
% I d2u e u2d calcolati fanno riferimento ad un'area di queste dimensioni
% assolute. Se, invece, si considera la risoluzione dell'immagine, occorre
% tenere anche conto delle dimensioni del pixel (qualora non si vogliono
% commettere errori nell'uso di d2u e u2d).
% Per non sbagliare, controllare sempre le dimensioni metriche assolute
% delle immagini da distorcere o antidistorcere.
ccd_d_msize_mt = ccd_msize * (coeff_px_size*ccd_px_size_ver_mt);
ccd_d_nsize_mt = ccd_nsize * (coeff_px_size*ccd_px_size_hor_mt);

% coordinates in meters of the distorted ccd center in the matrix
% one based reference system
ccd_center_r = (ccd_msize/2 + 0.5) * (coeff_px_size*ccd_px_size_ver_mt);    % coordinata
nella direzione delle righe
ccd_center_c = (ccd_nsize/2 + 0.5) * (coeff_px_size*ccd_px_size_hor_mt);    % coordinata
nella direzione delle colonne


% Definizione dei coefficienti del polinomio che descrive l'antidistorsione.
% il polinomio generico a cui si fa riferimento è del tipo y = a*x^3 + b*x^2
% + c*x + d_scalar. Nel caso della antidistorsione il coefficiente a è pari a
% (lens_u_param/ccd_d_hdiag2_c2c_mt); il coefficiente b è uguale a zero; c
% è pari a 1; d_scalar è zero.
% Si noti che la variabile x del polinomio è (nel ccd distorto) la distanza
% tra il centro del pixel(i,j) in coordinate metriche assolute e l'origine
% del sistema di riferimento metrico assoluto. Tale distanza coincide con
% il raggio associato al pixel (i,j) se da coordinate metriche assolute si passa a
coordinate polari.
a = lens_u_param/ccd_d_hdiag2_c2c_mt;

% inizializzazioni
ccd_d_radius_map = zeros(ccd_msize,ccd_nsize);
ccd_d_angle_map = zeros(ccd_msize,ccd_nsize);
```

```
d2u_row_map = zeros(ccd_msize,ccd_nsize);
d2u_col_map = zeros(ccd_msize,ccd_nsize);


%%%%%%%%% CICLO DI GENERAZIONE DI d2u_row_map E DI d2u_col_map  %%%%%%%%%%
for rr = 1: ccd_msize
    for cc = 1: ccd_nsize

        % passaggio da coordinate indice one based a coordinate
        % metriche assolute
        rr_d_mt = rr * (coeff_px_size*ccd_px_size_ver_mt) - ccd_center_r;
        cc_d_mt = cc * (coeff_px_size*ccd_px_size_hor_mt) - ccd_center_c;

        % passaggio da coordiante metriche assolute a coordinate polari
        ccd_d_radius_map(rr, cc) = sqrt(rr_d_mt^2 + cc_d_mt^2);
        ccd_d_angle_map(rr, cc) = atan2(rr_d_mt, cc_d_mt);

        % calcolo della coordinata raggio antidistorta attraverso il
        % polinomio di terzo grado.
        % Per il distorto pixel(rr, cc) il valore del raggio antidistorto
        % corrispondente è radius_u, mentre l'angolo associato al pixel
        % antidistorto rimane uguale al valore associato al pixel distorto.
        radius_u = a*ccd_d_radius_map(rr, cc)^3 + ccd_d_radius_map;

        % passaggio da coordinate polari a coordinate metriche assolute
        d2u_row_map(rr, cc) = radius_u(rr, cc) * sin(ccd_d_angle_map(rr, cc));
        d2u_col_map(rr, cc) = radius_u(rr, cc) * cos(ccd_d_angle_map(rr, cc));

        % Passaggio da coordinate metriche assolute a coordinate indice con
        % decimali.
        % E' ancora richiesto il rounding e una traslazione a sistema one
        % based. I valori delle matrici d2u presentano l'origine come centro
        % di simmetria e questa proprietà viene sfruttata nel rounding.
        d2u_row_map(rr, cc) = d2u_row_map(rr, cc)./(coeff_px_size*ccd_px_size_ver_mt);
        d2u_col_map(rr, cc) = d2u_col_map(rr, cc)./(coeff_px_size*ccd_px_size_hor_mt);


        %%%%%%%%%%%%%%%% rounding di d2u_row_map e d2u_col_map %%%%%%%%%%%%%%%%%%%%%%

        % NOTE SULLA SIMMETRIA
```


Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)

```
% Il passaggio da indici decimali a indici interi può essere effettuato
% tramite la funzione round() senza problemi di asimmetria nel caso gli
% assi di simmetria sono coincidenti con l'origine del sistema di
% riferimento. Ciò non è vero se vi è un asse di simmetria con coordinate
% diverse da zero. Ad esempio, le coordinare r1 = 0.5 e r2 = 1.5, simmetriche
% rispetto all'asse r = 1, vengono convertite in r1'=round(0.5) = 1 e
% r2'=round(1.5)=2, non più simmetriche rispetto all'asse r. Si noti che
% round() mantiene la simmetria delle coordinate r1 ed r2 rispetto l'asse r
% diverso da zero, se la parte decimale di r1 ed r2 è diversa da 0.5. Solo
% se la parte decimale è pari a 0.5 occorre effettuare il rounding tramite
% ceil() e floor() per mantenere la simmetria.
% Nel caso l'asse sia r = 0, come esempio di conferma di quanto detto, si
% noti che r1 = -2.5 ed r2 = 2.5, simmetrici, sono convertiti in
% r1'=round(-2.5)=-3 e r2'=round(2.5)=3, ancora simmetrici.
% Concludendo il rounding viene di seguito implementato con la semplice
% funzione round(), poichè i valori in d2u_row_map e d2u_col_map sono
% simmetrici rispetto all'origine.


% Operazioni propedeutiche al rounding vero e proprio.
% Il passaggio da valori valori decimali a interi con assi di simmetria
% sull'origine presenta il notevole vantaggio di conservare la simmetria
% anche se si adopera la sola funzione round(). Occorre, però, prestare
% attenzione al fatto che, poichè non è stata effettuata ancora nessuna
% traslazione, il pixel(i,j) ha, in questo punto del codice, coordinate
% (i-0.5, j-0.5), nell'ipotesi che la distorsione non abbia alterato la
% sua posizione. Effettuare un rounding su questi valori può creare
% problemi. Il seguente esempio illustra il problema.
% Si consideri un asse di simmetria
% nel sistema di riferimento one based pari a  r=1, e due pixel
% da antidistorcere subito a destra e sinistra dell'asse con coordinate
% dei centri pari a r1=0.5 e r2=1.5. Nel sistema di riferimento centrato
% sull'asse di simmetria r, i pixel hanno coordinate r1_rif=-0.5 e
% r2_rif=0.5. Dopo l'antidistorsione, rimanendo in tale sistema di
% riferimento, i pixel hanno coordinate r1_u=-0.500001 e r2_u=0.500001 (poichè
l'antidistorsione
% sta allargando l'immagine iniziale). Il round su questi valori ci darà
% r1_u'=-1 ed r2_u'=1. La distanza tra i due centri che prima era pari a 1,
% adesso è 2. E' comparso un pixel tra i due reali che non ha una
% corrispondenza con pixel iniziali. Il problema è che vorremmo che il
% round fissasse le posizioni dei pixel antidistori a r1_u'=1 e r2_u'=1 solo
```

```
        % se l'antidistorsione ha dato valori di r1_u e r2_u che si discostano da
        % r1_rif e r2_rif per una variazione maggiore di 0.5. Nel caso in cui la
variazione
        % di posizione introdotta dall'antidistorsione è inferiore a 0.5, le posizioni
        % relative tra i due pixel non devono variare.
        % Nell'esempio appena visto invece è bastato un delta di 0.00001 a far variare le
        % posizioni dei pixel. Ciò non va bene.
        % La soluzione è traslare di 0.5 e verso infinito (+infinito o -infinito a seconda
del quadrante)
        % tutte le coordinate in d2u_row_map e d2u_col_map.
        % La traslazione di 0.5 consente di far funzionare round() in base alle
        % variazioni di 0.5 nelle posizioni dei pixel.
        % Ora sorge un altro problema. La traslazione, infatti,
        % in sè cambia la posizione relativa tra tutti i pixel di un quadrante e
        % quelli di un altro (aggiungendo sempre un pixel senza corrispondenza a
        % cavallo dell'asse di simmetria). E' facile correggere ciò traslando, dopo
        % il round(), tutte le coordinate di 0.5 verso l'origine.

        if d2u_row_map(rr, cc) < 0
            d2u_row_map(rr, cc) = d2u_row_map(rr, cc) - 0.5;
        else
            d2u_row_map(rr, cc) = d2u_row_map(rr, cc) + 0.5;
        end

        if d2u_col_map(rr, cc) < 0
            d2u_col_map(rr, cc) = d2u_col_map(rr, cc) - 0.5;
        else
            d2u_col_map(rr, cc) = d2u_col_map(rr, cc) + 0.5;
        end

        % rounding
        d2u_row_map(rr, cc) = round(d2u_row_map(rr, cc));
        d2u_col_map(rr, cc) = round(d2u_col_map(rr, cc));

        % La traslazione propedeutica al rounding deve essere rieffettuata, in modo
        % inverso, per poter ripristinare la distaza relativa iniziale tra i
        % quadranti.
        if d2u_row_map(rr, cc) < 0
            d2u_row_map(rr, cc) = d2u_row_map(rr, cc) + 0.5;
        else
            d2u_row_map(rr, cc) = d2u_row_map(rr, cc) - 0.5;
```


Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)

```
        end

        if d2u_col_map(rr, cc) < 0
            d2u_col_map(rr, cc) = d2u_col_map(rr, cc) + 0.5;
        else
            d2u_col_map(rr, cc) = d2u_col_map(rr, cc) - 0.5;
        end
        %%%%%%%%%%  fine rounding di d2u_row_map e d2u_col_map   %%%%%%%%%
    end  % fine del ciclo "cc = 1: ccd_nsize"
end      % fine del ciclo "rr = 1: ccd_msize"


% Traslazioni necessarie per ricondurre d2u_row_map e d2u_col_map al sistema
% matrice one based.
% Alla fine delle traslazioni il minimo elemento in d2u_row_map e
% d2u_col_map ha valore pari a 1.
min_d2u_row_map = min(min(d2u_row_map));
min_d2u_col_map = min(min(d2u_col_map));
d2u_row_map = d2u_row_map - min_d2u_row_map +1;
d2u_col_map = d2u_col_map - min_d2u_col_map +1;


% calcolo del numero di righe e colonne dell'immagine antidistorta
max_d2u_row_map = max(max(d2u_row_map));
max_d2u_col_map = max(max(d2u_col_map));
ccd_und_msize = max_d2u_row_map;
ccd_und_nsize = max_d2u_col_map;


% calcolo delle dimensioni in metri dell'immagine antidistorta lungo le righe
% e lungo le colonne
ccd_u_msize_mt = ccd_und_msize * (coeff_px_size*ccd_px_size_ver_mt);
ccd_u_nsize_mt = ccd_und_nsize * (coeff_px_size*ccd_px_size_hor_mt);

% Creazione di ccd_u_link_map, d2u_row e d2u_col.
% Dato il pixel distorto di coordinate (ii,jj) è possibile leggere le coordinate
% riga e colonna del suo corrispondente pixel antidistorto all'elemento ii + ccd_msize*(jj
-1)
% nei vettori d2u_row e d2u_col.
ccd_u_link_map = uint8(zeros(ccd_und_msize, ccd_und_nsize));
d2u_row = zeros(ccd_msize*ccd_nsize,1);
```

```
d2u_col = zeros(ccd_msize*ccd_nsize,1);

for cc = 1:ccd_nsize
    for rr = 1:ccd_msize
        ccd_u_link_map(d2u_row_map(rr,cc), d2u_col_map(rr,cc)) = 1;
        d2u_row((cc-1)*ccd_msize + rr) = d2u_row_map(rr,cc);
        d2u_col((cc-1)*ccd_msize + rr) = d2u_col_map(rr,cc);
    end
end

figure(1)
imshow(ccd_u_link_map,[])
disp('Press space to continue')
pause

d2u_row = int32(d2u_row);
d2u_col = int32(d2u_col);
d2u_row_map = int32(d2u_row_map);
d2u_col_map = int32(d2u_col_map);



% coordinate in metri del centro del ccd antidistorto nel sistema di
% riferimento matrice one based
ccd_u_center_r = (ccd_und_msize/2 + 0.5) * (coeff_px_size*ccd_px_size_ver_mt);    %
coordinata nella direzione delle righe
ccd_u_center_c = (ccd_und_nsize/2 + 0.5) * (coeff_px_size*ccd_px_size_hor_mt);    %
coordinata nella direzione delle colonne


% inizializzazioni
ccd_u_radius_map = zeros(ccd_und_msize,ccd_und_nsize);
ccd_u_angle_map = zeros(ccd_und_msize,ccd_und_nsize);
u2d_row_map = zeros(ccd_und_msize,ccd_und_nsize);
u2d_col_map = zeros(ccd_und_msize,ccd_und_nsize);

%%%%%%%%%%%%%%% CICLO DI GENERAZIONE DI u2d_row E u2d_col %%%%%%%%%%%%%%%%
for cc = 1: ccd_und_nsize
    for rr = 1: ccd_und_msize

        % passaggio da coordinate indice one based a coordinate
        % metriche assolute
        rr_u_mt = rr * (coeff_px_size*ccd_px_size_ver_mt) - ccd_u_center_r;
```

```
cc_u_mt = cc * (coeff_px_size*ccd_px_size_hor_mt) - ccd_u_center_c;

% passaggio da coordiante metriche assolute a coordinate polari
ccd_u_radius_map(rr, cc) = sqrt(rr_u_mt^2 + cc_u_mt^2);
ccd_u_angle_map(rr, cc) = atan2(rr_u_mt, cc_u_mt);

% inversione del polinomio di terzo grado associato
% Nota: le espressioni di x1, x2, x3 sono le soluzioni analitiche
% di un generico polinomio di terzo grado, semplificate al caso
% particolare del polinomio dell'antidistorsione.
d = - ccd_u_radius_map(rr, cc);

x1 = - (2^(1/3)*(3*a))./(3*a*( - 27*a^2*d + sqrt(4*(3*a).^3 + (-
27*a^2*d).^2)).^(1/3)) ...
    + (- 27*a^2*d + sqrt(4*(3*a)^3 + (- 27*a^2*d).^2)).^(1/3)./(3*2^(1/3)*a);
x2 = ((1 + i*sqrt(3))*(3*a))./(3*2^(2/3)*a*(- 27*a^2*d + sqrt(4*(3*a)^3 +  ...
    (-27*a^2*d).^2)).^(1/3)) - (1 - i*sqrt(3))*(- 27*a^2*d + sqrt(4*(3*a).^3 + ...
    (-27*a^2*d).^2)).^(1/3)/(6*2^(1/3)*a);
x3 = ((1 - i*sqrt(3))*(3*a))./(3*2^(2/3)*a*(- 27*a^2*d + sqrt(4*(3*a).^3  ...
    + (-27*a^2*d).^2)).^(1/3)) - (1 + i*sqrt(3))*(- 27*a^2*d + ...
    sqrt(4*(3*a)^3 + (- 27*a^2*d).^2)).^(1/3)./(6*2^(1/3)*a);

% selezione della soluzione reale tra x1, x2 e x3
% Nota: il calcolo effettuato presuppone che non vi siano
% contemporanea x1, x2, x3 di valore reale. Ciò è vero sinchè il
% coefficiente a è positivo, c = 1 e il coefficiente b pari a zero.
x_aux_mod = [x1, x2, x3];
[min_val  min_idx] = min([abs(imag(x1)) abs(imag(x2)) abs(imag(x3))]);
radius_d = real(x_aux_mod(1, min_idx));

u2d_row_map(rr, cc) = radius_d * sin(ccd_u_angle_map(rr, cc));
u2d_col_map(rr, cc) = radius_d * cos(ccd_u_angle_map(rr, cc));

% Passaggio da coordinate metriche assolute a coordinate indice con
% decimali.
% E' ancora richiesto il rounding e una traslazione a sistema one based
% ma in questo caso il centro di simmetria è l'origine, informazione
% sfruttata nel rounding
u2d_row_map(rr, cc) = u2d_row_map(rr, cc)./(coeff_px_size*ccd_px_size_ver_mt);
u2d_col_map(rr, cc) = u2d_col_map(rr, cc)./(coeff_px_size*ccd_px_size_hor_mt);
```

```
        %%%%%%%%%%  rounding per u2d_row_map e u2d_col_map   %%%%%%%%%%%%%%
        %%%% (del tutto analogo a quello per d2u_row_map e d2u_col_map %%%%

        % traslazione propedeutica al rounding
        if u2d_row_map(rr, cc) < 0
            u2d_row_map(rr, cc) = u2d_row_map(rr, cc) - 0.5;
        else
            u2d_row_map(rr, cc) = u2d_row_map(rr, cc) + 0.5;
        end

        if u2d_col_map(rr, cc) < 0
            u2d_col_map(rr, cc) = u2d_col_map(rr, cc) - 0.5;
        else
            u2d_col_map(rr, cc) = u2d_col_map(rr, cc) + 0.5;
        end


        % rounding
        u2d_row_map(rr, cc) = round(u2d_row_map(rr, cc));
        u2d_col_map(rr, cc) = round(u2d_col_map(rr, cc));

        % La traslazione propedeutica al rounding deve essere rieffettuata, in modo
        % inverso, per poter ripristinare la distaza relativa iniziale tra i
        % quadranti.
        if u2d_row_map(rr, cc) < 0
            u2d_row_map(rr, cc) = u2d_row_map(rr, cc) + 0.5;
        else
            u2d_row_map(rr, cc) = u2d_row_map(rr, cc) - 0.5;
        end

        if u2d_col_map(rr, cc) < 0
            u2d_col_map(rr, cc) = u2d_col_map(rr, cc) + 0.5;
        else
            u2d_col_map(rr, cc) = u2d_col_map(rr, cc) - 0.5;
        end
    end
end

% traslazione dopo la quale u2d_row_map(1,1) = 1,
```

```
% u2d_row_map(CCD_u_MSIZE,ccd_und_nsize) = ccd_msize,
% u2d_col_map(1,1) = 1 e u2d_col_map(CCD_u_MSIZE,ccd_und_nsize) = ccd_nsize
u2d_row_map = u2d_row_map + (ccd_msize-1)/2 +1;
u2d_col_map = u2d_col_map + (ccd_nsize-1)/2 +1;


% Generazione di u2d_row e u2d_col.
% Dato il pixel antidistorto di coordinate (ii,jj) è possibile leggere le coordinate
% riga e colonna del suo corrispondente pixel distorto all'elemento ii + ccd_und_msize*(jj
-1)
% nei vettori u2d_row e u2d_col.

u2d_row = zeros(ccd_und_nsize*ccd_und_msize,1);
u2d_col = zeros(ccd_und_nsize*ccd_und_msize,1);
for cc = 1:ccd_und_nsize
    for rr = 1:ccd_und_msize
        u2d_row((cc-1)*ccd_und_msize + rr) = u2d_row_map(rr,cc);
        u2d_col((cc-1)*ccd_und_msize + rr) = u2d_col_map(rr,cc);
    end
end

u2d_row = int32(u2d_row);
u2d_col = int32(u2d_col);
u2d_row_map = int32(u2d_row_map);
u2d_col_map = int32(u2d_col_map);


%%%%%%%%%%%%%%%%%%%%%%%% END OF SECTION 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  SECTION 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%  generating ccd_lensgain_map   %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% NOTA: pur potendo includere la generazione di ccd_lensgain_map nel ciclo
% di generazione di XY_u, si è preferito, creare una sezione a parte per
% chiarezza di lettura del codice.

% Nel sistema di riferimento one based le coordinate del centro della
% matrice di pixel e quindi del centro della distorsione sono poste in
```

```
% (ccd_msize/2+0.5, ccd_nsize/2+0.5). Le coordinate del centro di ogni pixel,
% invece, coincidono rispettivamente con gli indici che ne
% determinano la posizione nella matrice.

ccd_lensgain_map = zeros(ccd_msize, ccd_nsize);

% computing centre of distortion in meters
distortion_center_row_mt = (ccd_msize/2 + 0.5) * (coeff_px_size*ccd_px_size_ver_mt);
distortion_center_col_mt = (ccd_nsize/2 + 0.5) * (coeff_px_size*ccd_px_size_hor_mt);



for ii = 1:ccd_msize
    for jj = 1:ccd_nsize
        % getting distance between point to undistort (the center
        % of the (ii, jj) pixel) and distortion centre.
        radius2_c2c = (ii * (coeff_px_size*ccd_px_size_ver_mt)  -
distortion_center_row_mt).^2 + (jj * (coeff_px_size*ccd_px_size_hor_mt) -
distortion_center_col_mt).^2;
        % calcolo del guadagno della distorsione associato ad ogni pixel
        % della matrice
        ccd_lensgain_map(ii,jj) = 1 + radius2_c2c.*(lens_u_param/ccd_hdiag2_c2c_mt);
    end
end
figure(1)
imshow(ccd_lensgain_map,[])

% Normalizzazione di ccd_lensgain_map tra i valori "zero" e "uno"
min_val_gain = min(min(ccd_lensgain_map));
max_val_gain = max(max(ccd_lensgain_map));

for ii = 1:ccd_msize
    for jj = 1:ccd_nsize
        ccd_lensgain_map(ii,jj) = (ccd_lensgain_map(ii,jj) - min_val_gain)/(max_val_gain -
min_val_gain);
    end
end



%%%%%%%%%%%%%%%%%%%%%%%%%% END OF SECTION 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  SECTION 3  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%  generating ccd_px_fov_map  %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% pixel_FOV_row  --> field of view in the row direction for a pixel i,j
% pixel_FOV_col  --> field of view in the columns direction for a pixel i,j


% Notes: you should check ind_row and ind_col sign to calculate
% pixel_FOV_row and pixel_FOV_col. If you find a negative value
% for ind_row or ind_col, you should change the signs of pixel_FOV_row
% and pixel_FOV_col respectively.
% This is not necessary since atan( ) is negative for negative argument.

% Nel sistema di riferimento one based le coordinate del fuoco della
% camera sono coincidenti con il centro della matrice ccd o dell'immagine
% ad essa associata.
focus_row = ccd_msize/2 + 0.5; % row coordinate of the camera focus
focus_col = ccd_nsize/2 + 0.5; % col coordinate of the camera focus

% ccd_px_fov_map inizialization
ccd_px_fov_map = zeros(ccd_msize, ccd_nsize);

% Il FOV di ogni pixel richiede di calcolare la distanza
% di ciascuno dei quattro bordi di ogni pixel dal fuoco della camera.
% Le coordinate riga dei due bordi verticali del pixel (i,j)
% nel sistema di riferimento matrice one based
% sono i-0.5 e i+0.5 se i è la coordinata riga del centro del pixel. Le
% coordinate dei bordi orizzontali del pixel (i,j) sono j+0.5 e j-0.5.
% La tangente dell'angolo tra il fuoco e uno dei bordi del pixel è data
% dal rapporto: 'distanza in metri tra bordo e fuoco' / 'distanza focale in
% metri'. La differenza tra gli angoli associati ai due bordi del pixel
% restituisce l'angolo incluso tra le due rette congiungenti il fuoco e i
% due bordi del pixel. Tale angolo si considera FOV_row o FOV_col del pixel
% a seconda se è stato calcolato rispettivamente per la coppia di bordi
% orizzontali (cioè lungo le righe) o verticali (lungo le colonne).
```

```
for ind_row= 1:ccd_msize
    for ind_col=1:ccd_nsize
        pixel_FOV_row = atan((ind_row + 0.5 - focus_row) *
(coeff_px_size*ccd_px_size_ver_mt)/cam_f_mt)...
            - atan((ind_row - 0.5 - focus_row) *
(coeff_px_size*ccd_px_size_ver_mt)/cam_f_mt);
        pixel_FOV_col = atan((ind_col + 0.5 - focus_col) *
(coeff_px_size*ccd_px_size_hor_mt)/cam_f_mt)...
            - atan((ind_col - 0.5 - focus_col) *
(coeff_px_size*ccd_px_size_hor_mt)/cam_f_mt);


        ccd_px_fov_map(ind_row, ind_col) = pixel_FOV_row * pixel_FOV_col;  % the pixel's
total field of view is the
        % product of horizontal and vertical field of view
    end
end



% Normalizzazione di ccd_px_fov_map tra i valori "zero" e "uno"
min_val_fov = min(min(ccd_px_fov_map));
max_val_fov = max(max(ccd_px_fov_map));


for ind_row= 1:ccd_msize
    for ind_col=1:ccd_nsize
        ccd_px_fov_map(ind_row,ind_col) = - (ccd_px_fov_map(ind_row,ind_col) -
min_val_fov)/(max_val_fov - min_val_fov) + 1;
    end
end



%%%%%%%%%%%%%%%%%%%%%%%%%    END OF SECTION 3    %%%%%%%%%%%%%%%%%%%%%%%%%%%
```

fitline.m

```
function [dist, C] = fitline(XY)

% This function takes a list of point as argument and then it calculates
% the equation of the regression line for that points. After that this
% function evaluates the mean distance input points from the found line.
%
% INPUT:
%
% XY    --> 2xn matrix of point belonging to the same line;
```

```
%
% OUTPUTS:
%
% dist  --> sum of distance of each point from the computated regression
%           line;
% C     --> coefficient of the computated regression line.
% getting number of points belonging to the line

[npts, cols] = size(XY);

% checking to have needed points' coordinates

if cols ~=2

error('Data is not 2D');

end

% checking to have enough point to compute a regression line

if npts < 3

error('Too few points to fit line');

end

% Set up constraint equations of the form  AC = 0,
% where C is a column vector of the line coefficients
% in the form   c(1)*X + c(2)*Y + c(3) = 0.

A = [XY ones(npts,1)];  % Build constraint matrix

[u d v] = svd(A);        % Singular value decomposition of A.

C = v(:,3);              % Solution is last column of v.

dist = abs(C(1) .* XY(:,1) + C(2) .* XY(:,2) + C(3))/sqrt(C(1)^2 + C(2)^2); % distance of
a point from a line

dist = mean(dist); % taking the mean value of the distance of all points
```

obj_distortion.m

```
function fitness = obj_distortion(lens_u_param, linelist, ccd_hdiag2_c2c_mt)

% This function outputs the fitness of a linelist: the more the fitness is

% low, the more the lines in the linelist are straight.
```

```
% INPUTS:
%
% lens_u_param       --> undistortion parameter;
%
% linelist           --> list of lines to undistort;
%
% ccd_hdiag2_c2c_mt  --> distance between the center of the image (equal to the reference
%                        system origin) and the center of a vertex pixel in the image.

%                       The name ccd_hdiag2_c2c_mt means "ccd half diagonal squared center
to
%                       center".
% OUTPUT:
%
% fitness   --> fitness of the undistorted linelist;
%%%%%%%%%%%%%%%%%%%%%%% SECTION 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%% getting undistorted linelist points %%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This section undistort according to parameter lens_u_param all the points in the
% current linelist and generates an undistorted linelist for fitenss
% evaluation.
%
% u_linelist  --> undistorted linelist;
% counting number of line to undistort
regions_numb = length(linelist);
% Iniziallizzazione di u_linelist.
u_linelist = cell(size(linelist,1),1); %u_linelist and linelist have the same dimensions
% undistorting each of the input lines
for ii = 1:regions_numb
    for jj = 1 : length(linelist{ii})

        % for the (ii,jj) pixel, radius_square is the square value of the distance between
```

```
the center
        % of the image (equal to the metric absolute reference system origin) and the
        % (ii,jj) pixel center
        % nota: u_linelist{ii} è la matrice elemento ii del cell array u_linelist.
        % con u_linelist{ii}(a,b) si indica l'elemento (a,b) della matrice
        % identificata da u_linelist{ii}.
        radius2_c2c = linelist{ii}(jj,1)^2 + linelist{ii}(jj,2)^2;
        gain = 1 + radius2_c2c.*(lens_u_param/ccd_hdiag2_c2c_mt);
        u_linelist{ii}(jj,1) = linelist{ii}(jj,1) * gain;
        u_linelist{ii}(jj,2) = linelist{ii}(jj,2) * gain;
    end
end
%%%%%%%%%%%%%%%%%%%%%% END OF SECTION 1 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% SECTION 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%% evaluating fitness of undistorted linelist %%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This section gets a cell array of different correlated points and
% computes the regression line for each group of point. It produces also the
% sum of mean distances of each group of points from its regression line.
%
% fitness   --> sum of distances of each point of each line from the
%               regression line to which the point should belong.
% cell_C    --> regression lines vector in the form ax+by+c.
dist_vect = zeros(regions_numb,1);
for ii = 1:regions_numb

    [dist C] = fitline(u_linelist{ii}); % Getting distance and line's equation for each
group of points
    dist_vect(ii) = dist; % writing distance for each line

end
% summing all lines distances
```

```
fitness  = sum(dist_vect);

%%%%%%%%%%%%%%%%%%%%% END OF SECTION 2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

frame_demosaicking.cpp

```
#include " frame_demosaicking.h"


#define GREENPOSITION 0
#define REDPOSITION 1
#define BLUEPOSITION 2


void demosaicking_adams(float threshold, int redx,int redy,float *ired,float *igreen,float
*iblue,float *ored,float *ogreen,float *oblue,int width,int height)
{
	wxCopy(ired,ored,width*height);
	wxCopy(igreen,ogreen,width*height);
	wxCopy(iblue,oblue,width*height);

	// Initializzazione
	int bluex = 1 - redx;
	int bluey = 1 - redy;

	// CFA Mask of color per pixel
	unsigned char* mask = (unsigned char *) malloc(width*height*sizeof(unsigned char));

	for(int x=0;x<width;x++)
		for(int y=0;y<height;y++){

			if (x%2 == redx && y%2 == redy) mask[y*width+x] = REDPOSITION;
			else  if (x%2 == bluex && y%2 == bluey) mask[y*width+x] = BLUEPOSITION;
			else   mask[y*width+x] = GREENPOSITION;

		}


	// Interpolate the green channel by bilinear on the boundaries
	// make the average of four neighbouring green pixels: Nourth, South, East, West
	for(int x=0;x<width;x++)
		for(int y=0;y<height;y++)
			if ( (mask[y*width+ x] != GREENPOSITION) && (x < 3 || y < 3 || x>= width
- 3 || y>= height - 3 )) {
```

```
                        int gn, gs, ge, gw;

                        if (y > 0)  gn = y-1;      else    gn = 1;
                        if (y < height-1)   gs = y+1;  else  gs = height - 2 ;
                        if (x < width-1)  ge = x+1;  else  ge = width-2;
                        if (x > 0) gw = x-1;       else  gw = 1;

                        ogreen[y*width + x] = (ogreen[gn*width + x] +  ogreen[gs*width +
x] + ogreen[y*width + gw] +  ogreen[y*width + ge])/ 4.0;

                    }

      // Interpolate the green by Adams algorithm inside the image
      // First interpolate green directionally
      for(int x=3;x<width-3;x++)
            for(int y=3;y<height-3;y++)
                  if (mask[y*width+ x] != GREENPOSITION ) {


                        int l = y*width+x;
                        int lp1 = (y+1)*width +x;
                        int lp2 = (y+2)*width +x;
                        int lm1 = (y-1)*width +x;
                        int lm2 = (y-2)*width +x;

                        // Compute vertical and horizontal gradients in the green channel
                        float adv = fabsf(ogreen[lp1] - ogreen[lm1]);
                        float adh = fabsf(ogreen[l-1] - ogreen[l+1]);
                        float dh0, dv0;

                        // If current pixel is blue, we compute the horizontal and
vertical blue second derivatives
                        // else is red, we compute the horizontal and vertical red second
derivatives
                        if (mask[l] == BLUEPOSITION){

                              dh0 = 2.0 * oblue[l] - oblue[l+2] - oblue[l-2];
                              dv0 = 2.0 * oblue[l] - oblue[lp2] - oblue[lm2];

                        } else {
```


Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)

```
                        dh0 = 2.0 * ored[l] - ored[l+2] - ored[l-2];
                        dv0 = 2.0 * ored[l] - ored[lp2] - ored[lm2];

                    }
                    // Add vertical and horizontal differences
                    adh = adh + fabsf(dh0);
                    adv = adv + fabsf(dv0);

                    // If vertical and horizontal differences are similar, compute an isotropic average
                    if (fabsf(adv - adh) < threshold)

                        ogreen[l] = (ogreen[lm1] +  ogreen[lp1] +  ogreen[l-1] + ogreen[l+1]) /4.0 + (dh0 + dv0) / 8.0;

                    // Else If horizontal differences are smaller, compute horizontal average
                    else if (adh < adv )

                        ogreen[l] = (ogreen[l-1] + ogreen[l+1])/2.0 + (dh0)/4.0;

                    // Else If vertical differences are smaller, compute vertical average
                    else if ( adv < adh )

                        ogreen[l] = (ogreen[lp1] + ogreen[lm1])/2.0 + (dv0)/4.0;

                }

    // compute the bilinear on the differences of the red and blue with the already interpolated green
    demosaicking_bilinear_red_blue(redx,redy,ored,ogreen,oblue,width,height);


    free(mask);
}

/**
 * \brief  Classical bilinear interpolation of red and blue differences with the green channel
```

```
 *
 *
 * @param[in]  ored, ogreen, oblue  original cfa image with green interpolated
 * @param[out] ored, ogreen, oblue  demosaicked output
 * @param[in]  (redx, redy)  coordinates of the red pixel: (0,0), (0,1), (1,0), (1,1)
 * @param[in]  width, height size of the image
 *
 */


void  demosaicking_bilinear_red_blue(int  redx,int  redy,float  *ored,float  *ogreen,float
*oblue,int width,int height)
{
	//Initializations
	int bluex = 1 - redx;
	int bluey = 1 - redy;


	// Mask of color per pixel
	unsigned char* mask = (unsigned char *) malloc(width*height*sizeof(unsigned char));

	for(int x=0;x<width;x++)
		for(int y=0;y<height;y++){

			if (x%2 == redx && y%2 == redy) mask[y*width+x] = REDPOSITION;
			else  if (x%2 == bluex && y%2 == bluey) mask[y*width+x] = BLUEPOSITION;
			else   mask[y*width+x] = GREENPOSITION;


		}
	// Compute the differences
	for(int i=0; i < width*height;i++){

		ored[i] -= ogreen[i];
		oblue[i] -= ogreen[i];


	}
	// Interpolate the blue differences making the average of possible values depending
on the CFA structure
	for(int x=0; x < width;x++)
		for(int y=0; y < height;y++)
			if (mask[y*width+x] != BLUEPOSITION){
```


Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)

```
                        int gn, gs, ge, gw;

                        // Compute north, south, west, east positions
                        // taking a mirror symmetry at the boundaries
                        if (y > 0)  gn = y-1;      else    gn = 1;
                        if (y < height-1)   gs = y+1;  else  gs = height - 2 ;
                        if (x < width-1)  ge = x+1;  else  ge = width-2;
                        if (x > 0) gw = x-1;       else  gw = 1;

                        if (mask[y*width+x] == GREENPOSITION && y % 2 == bluey)
                              oblue[y*width+x]      =      (      oblue[y*width+ge]      +
oblue[y*width+gw])/2.0;
                        else if (mask[y*width+x] == GREENPOSITION  && x % 2 == bluex)
                              oblue[y*width+x]      =      (      oblue[gn*width+x]      +
oblue[gs*width+x])/2.0;
                        else {
                              oblue[y*width+x] =   (oblue[gn*width+ge] + oblue[gn*width
+ gw]   +   oblue[gs*width + ge] +   oblue[gs*width +gw])/4.0;
                        }
                  }
      // Interpolate the blue differences making the average of possible values depending
on the CFA structure
      for(int x=0;x<width;x++)
            for(int y=0;y<height;y++)
                  if (mask[y*width+x] != REDPOSITION){
                        int gn, gs, ge, gw;

                        // Compute north, south, west, east positions
                        // taking a mirror symmetry at the boundaries
                        if (y > 0)  gn = y-1;      else    gn = 1;
                        if (y < height-1)   gs = y+1;  else  gs = height - 2 ;
                        if (x < width-1)  ge = x+1;  else  ge = width-2;
                        if (x > 0) gw = x-1;       else  gw = 1;

                        if (mask[y*width+x] == GREENPOSITION && y % 2 == redy)
                              ored[y*width+x]      =      (      ored[y*width+ge]      +
ored[y*width+gw])/2.0;
                        else if (mask[y*width+x] == GREENPOSITION  && x % 2 == redx)
                              ored[y*width+x]      =      (      ored[gn*width+x]      +
ored[gs*width+x])/2.0;
                        else {
                              ored[y*width+x] =   (ored[gn*width+ge] + ored[gn*width +
gw]   +   ored[gs*width + ge] +   ored[gs*width +gw])/4.0;
```

```
                        }
                    }
        // Make back the differences
        for(int i=0;i<width*height;i++){

            ored[i] += ogreen[i];
            oblue[i] += ogreen[i];
        }
        free(mask);
}
```

```
/**
 * \brief  NLmeans based demosaicking
 *
 * For each value to be filled, a weigthed average of original CFA values of the same channel is performed.
 * The weight depends on the difference of a 3x3 color patch
 *
 * @param[in]  ired, igreen, iblue  initial demosaicked image
 * @param[out] ored, ogreen, oblue  demosaicked output
 * @param[in]  (redx, redy)  coordinates of the red pixel: (0,0), (0,1), (1,0), (1,1)
 * @param[in]  bloc  research block of size (2+bloc+1) x (2*bloc+1)
 * @param[in]  h kernel bandwidth
 * @param[in]  width, height size of the image
 *
 */
void demosaicking_nlmeans(int bloc, float h,int redx,int redy,float *ired,float *igreen,float *iblue,float *ored,float *ogreen,float *oblue,int width,int height)
{
        // Initializations
        int bluex = 1 - redx;
        int bluey = 1 - redy;
        // CFA Mask of color per pixel
        unsigned char *cfamask = new unsigned char[width*height];

        for(int x=0;x<width;x++)
```

```
		for(int y=0;y<height;y++){

			if (x%2 == redx && y%2 == redy) cfamask[y*width+x] = REDPOSITION;
			else	if (x%2 == bluex && y%2 == bluey) cfamask[y*width+x] = BLUEPOSITION;
			else	cfamask[y*width+x] = GREENPOSITION;

		}

	wxCopy(ired,ored,width*height);
	wxCopy(igreen,ogreen,width*height);
	wxCopy(iblue,oblue,width*height);

	// Tabulate the function Exp(-x) for x>0.
	int luttaille = (int) (LUTMAX*LUTPRECISION);
	float *lut = new float[luttaille];

	sFillLut(lut, luttaille);

	// for each pixel
	for(int y=2; y <height-2; y++)
		for(int x=2; x<width-2; x++)
		{
			// index of current pixel
			int l=y*width+x;

			// Learning zone depending on the window size
			int imin=MAX(x-bloc,1);
			int jmin=MAX(y-bloc,1);

			int imax=MIN(x+bloc,width-2);
			int jmax=MIN(y+bloc,height-2);


			// auxiliary variables for computing average
			float red=0.0;
			float green=0.0;
			float blue=0.0;

			float rweight=0.0;
			float gweight=0.0;
```

```
float bweight=0.0;


// for each pixel in the neighborhood
for(int j=jmin;j<=jmax;j++)
      for(int i=imin;i<=imax;i++) {

            // index of neighborhood pixel
            int l0=j*width+i;

            // We only interpolate channels differents of the current
pixel channel
            if (cfamask[l]!=cfamask[l0]) {


                  // Distances computed on color
                  float some = 0.0;

                  some = l2_distance_r1(ired,  x, y, i,
                                                  j, width);
                  some += l2_distance_r1(igreen,  x, y, i,
                                                   j, width);
                  some += l2_distance_r1(iblue,  x, y, i,
                                                  j, width);



                  // Compute weight
                  some= some / (27.0 * h);

                  float weight = sLUT(some,lut);

                  // Add pixel to corresponding channel average

                  if (cfamask[l0] == GREENPOSITION)  {

                        green += weight*igreen[l0];
                        gweight+= weight;

                  } else if (cfamask[l0] == REDPOSITION) {
```


Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)

```
                                        red += weight*ired[l0];
                                        rweight+= weight;

                                    } else {

                                        blue += weight*iblue[l0];
                                        bweight+= weight;
                                    }

                                }

                            }


                    // Set value to current pixel
                    if (cfamask[l] != GREENPOSITION && gweight > fTiny)  ogreen[l]  =  green
/ gweight;
                    else  ogreen[l] = igreen[l];

                    if ( cfamask[l] != REDPOSITION && rweight > fTiny)  ored[l]  =  red /
rweight ;
                    else    ored[l] = ired[l];

                    if  (cfamask[l] != BLUEPOSITION && bweight > fTiny)   oblue[l] =  blue
/ bweight;
                    else  oblue[l] = iblue[l];


                }

        delete[] cfamask;
        delete[] lut;

}

/**
 * \brief  Iterate median filter on chromatic components of the image
 *
 *
 * @param[in]  ired, igreen, iblue  initial  image
 * @param[in]  iter  number of iteracions
```

```
 * @param[out] ored, ogreen, oblue  filtered output
 * @param[in]  (redx, redy)  coordinates of the red pixel: (0,0), (0,1), (1,0), (1,1)
 * @param[in]  side  median in a (2*side+1) x (2*side+1) window
 * @param[in]  projflag if not zero, values of the original CFA are kept
 * @param[in]  width, height size of the image
 *
 */

void chromatic_median(int iter,int redx,int redy,int projflag,float side,float *ired,float
*igreen, float *iblue,float *ored,float *ogreen,float *oblue,int width,int height)
{
	int size=height*width;

	// Auxiliary variables for computing chromatic components
	float *y=new float[size];
	float *u=new float[size];
	float *v=new float[size];
	float *u0=new float[size];
	float *v0=new float[size];

	int bluex=1-redx;
	int bluey=1-redy;


	// For each iteration
	for(int i=1;i<=iter;i++){


		// Transform to YUV
		wxRgb2Yuv(ired,igreen,iblue,y,u,v,width,height);

		// Perform a Median on YUV component
		wxMedian(u,u0,side,1 ,width,height);
		wxMedian(v,v0,side,1 ,width,height);


		// Transform back to RGB
		wxYuv2Rgb(ored,ogreen,oblue,y,u0,v0,width,height);

		// If projection flag activated put back original CFA values
		if (projflag)
```


Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)

```
		{

			for(int x=0;x<width;x++)
				for(int y=0;y<height;y++){

					int l=y*width+x;

					if (x%2==redx && y%2==redy) ored[l]=ired[l];

					else if (x%2==bluex && y%2==bluey) oblue[l]=iblue[l];

					else ogreen[l]=igreen[l];

				}

		}

		wxCopy(ored,ired,size);
		wxCopy(ogreen,igreen,size);
		wxCopy(oblue,iblue,size);

	}


	// delete auxiliary memory
	delete[] y;
	delete[] u;
	delete[] u0;
	delete[] v;
	delete[] v0;

}

/**
 * \brief Demosaicking chain
 *
 * for h in {16,4,1} do
 * {
 *
 *		u <- NL_h(u0);			Apply NLmeans demosaicking
```

```
 *
 *            u <- CR(u);                          Apply chromatic regularization
 *
 *            u0 <- u;
 *
 * }
 *
 * Output <- u;
 *
 *
 * @param[in]  ired, igreen, iblue  initial  image
 * @param[out] ored, ogreen, oblue  filtered output
 * @param[in]  (redx, redy)  coordinates of the red pixel: (0,0), (0,1), (1,0), (1,1)
 * @param[in]  width, height size of the image
 *
 */
```

```
void ssd_demosaicking_chain(int redx,int redy,float *ired,float *igreen,float *iblue,float
*ored,float *ogreen,float *oblue,int width,int height)
{
	////////////////////////////////////////////// Process
	float h;
	int dbloc = 7;
	float side = 1.5;
	int iter = 1;
	int projflag = 1;
	float threshold = 2.0;

	demosaicking_adams(threshold,redx,redy,ired, igreen, iblue, ored, ogreen, oblue,
width,height);
	h = 16.0;
	demosaicking_nlmeans(dbloc,h,redx,redy,ored,ogreen,oblue,ired,igreen,iblue,width,he
ight);
	chromatic_median(iter,redx,redy,projflag,side,ired,igreen,iblue,ored,ogreen,oblue,w
idth,height);
	h = 4.0;

	demosaicking_nlmeans(dbloc,h,redx,redy,ored,ogreen,oblue,ired,igreen,iblue,width,he
ight);
	chromatic_median(iter,redx,redy,projflag,side,ired,igreen,iblue,ored,ogreen,oblue,w
idth,height);
```

```
	h = 1.0;
	demosaicking_nlmeans(dbloc,h,redx,redy,ored,ogreen,oblue,ired,igreen,iblue,width,he
ight);

	chromatic_median(iter,redx,redy,projflag,side,ired,igreen,iblue,ored,ogreen,oblue,w
idth,height);



}
```

frame_demosaicking.h

```
#ifndef _FRAME_DEMOSAICKING_H_
#define _FRAME_DEMOSAICKING_H_
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "libAuxiliary.h"

/**
 *
 *  The green channel is interpolated directionally depending on the green first and red
and blue second directional derivatives.
 *  The red and blue differences with the green channel are interpolated bilinearly.
 *
 * @param[in]  ired, igreen, iblue  original cfa image
 * @param[out] ored, ogreen, oblue  demosaicked output
 * @param[in]  (redx, redy)  coordinates of the red pixel: (0,0), (0,1), (1,0), (1,1)
 * @param[in]  threshold value to consider horizontal and vertical variations equivalent
and average both estimates
 * @param[in]  width, height size of the image
 *
 */

void demosaicking_adams(float threshold, int redx,int redy,float *ired,float *igreen,float
*iblue,float *ored,float *ogreen,float *oblue,int width,int height);
/**
 * \brief  Classical bilinear interpolation of red and blue differences with the green
channel
 *
 *
 * @param[in]  ored, ogreen, oblue  original cfa image with green interpolated
```

```
 * @param[out] ored, ogreen, oblue  demosaicked output
 * @param[in]  (redx, redy)  coordinates of the red pixel: (0,0), (0,1), (1,0), (1,1)
 * @param[in]  width, height size of the image
 *
 */

void  demosaicking_bilinear_red_blue(int  redx,int  redy,float  *ored,float  *ogreen,float
*oblue,int width,int height);



/**
 * For each value to be filled, a weigthed average of original CFA values of the same channel
is performed.
 * The weight depends on the difference of a 3x3 color patch
 *
 * @param[in]  ired, igreen, iblue  initial demosaicked image
 * @param[out] ored, ogreen, oblue  demosaicked output
 * @param[in]  (redx, redy)  coordinates of the red pixel: (0,0), (0,1), (1,0), (1,1)
 * @param[in]  bloc  research block of size (2+bloc+1) x (2*bloc+1)
 * @param[in]  h kernel bandwidth
 * @param[in]  width, height size of the image
 *
 */


void  demosaicking_nlmeans(int  bloc,  float  h,int  redx,int  redy,float  *ired,float
*igreen,float *iblue,float *ored,float *ogreen,float *oblue,int width,int height);


/**
 * \brief  Iterate median filter on chromatic components of the image
 *
 *
 * @param[in]  ired, igreen, iblue  initial  image
 * @param[in]  iter  number of iteracions
 * @param[out] ored, ogreen, oblue  filtered output
 * @param[in]  (redx, redy)  coordinates of the red pixel: (0,0), (0,1), (1,0), (1,1)
 * @param[in]  side  median in a (2*side+1) x (2*side+1) window
 * @param[in]  projflag if not zero, values of the original CFA are kept
 * @param[in]  width, height size of the image
 *
 */
```

```
void chromatic_median(int iter,int redx,int redy,int projflag,float side,float *ired,float *igreen, float *iblue,float *ored,float *ogreen,float *oblue,int width,int height);


/**
 * \brief Demosaicking chain
 *
 * for h in {16,4,1} do
 * {
 *
 *            u <- NL_h(u0);                    Apply NLmeans demosaicking
 *
 *            u <- CR(u);                       Apply chromatic regularization
 *
 *            u0 <- u;
 *
 * }
 *
 * Output <- u;
 *
 *
 * @param[in]  ired, igreen, iblue  initial  image
 * @param[out] ored, ogreen, oblue  filtered output
 * @param[in]  (redx, redy)  coordinates of the red pixel: (0,0), (0,1), (1,0), (1,1)
 * @param[in]  width, height size of the image
 *
 */


void ssd_demosaicking_chain(int redx,int redy,float *ired,float *igreen,float *iblue,float *ored,float *ogreen,float *oblue,int width,int height);


#endif
```

frame_diff.cpp

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

#include "io_tiff.h"

int main(int argc, char **argv)
```

```
{

    size_t nx, ny;
    size_t _nx, _ny;
    float *data, *data2;
    float *data_ptr, *data2_ptr;
       float delta;
    float scale, tmp;
       int i, c;
       float sdif[3];

    /* "-v" option : version info */
    if (2 <= argc && 0 == strcmp("-v", argv[1]))
    {
        fprintf(stdout, "%s version " __DATE__ "\n", argv[0]);
        return EXIT_SUCCESS;
    }
    /* wrong number of parameters : simple help info */
    if (5 != argc)
    {
        fprintf(stderr, "usage : %s A.tiff B.tiff D out.tiff\n",
             argv[0]);
        fprintf(stderr, "        D : max difference to visualize\n");
        return EXIT_FAILURE;
    }

    /* rgb sigma parameters */
    delta = atof(argv[3]);
    if (0 >= delta)
    {
        fprintf(stderr, "delta must be > 0\n");
        return EXIT_FAILURE;
    }

    /* read the two images */
    data = (float *) read_tiff_rgba_f32(argv[1], &nx, &ny);
    data2 = (float *) read_tiff_rgba_f32(argv[2], &_nx, &_ny);

    /* check the image sizes */
    if ((nx != _nx) || (ny != _ny))
```

```
{
    fprintf(stderr, "image sizes don't match\n");
    free(data);
    free(data2);
    return EXIT_FAILURE;
}
/* compute the difference */
   {

          data_ptr = data;
          data2_ptr = data2;
          scale = 128. / delta;

          for (c=0; c < 3; c++)
          {

                 i = 0;
                 sdif[c] = 0.0f;

                 while (i < nx * ny)
                 {

                        /* data -= data2 */
                        *data_ptr -= *data2_ptr++;

                        sdif[c]  +=  *data_ptr * (*data_ptr);

                        /* data *= 128 / delta */
                        /* differences > delta are out out the [-128,128] interval */
                        *data_ptr *= scale;
                        /* data += 128 */
                        *data_ptr += 128.;
                        /* round the values to the closest integer,
                         * and truncate to [0..255] */
                        tmp = floor(*data_ptr + .5);
                        *data_ptr++ = (tmp > 255 ? 255. : (tmp < 0 ? 0. : tmp));

                        i++;
                 }
```

```
                    sdif[c] /= (float) (nx*ny);
                    sdif[c] = sqrt(sdif[c]);

                }

                printf("Root Mean Square Error (R,G,B) = (%2.2f, %2.2f, %2.2f)\n", sdif[0],
sdif[1], sdif[2]);

        }

    /* save the result */
    write_tiff_rgba_f32(argv[4], data, nx, ny);

    free(data);
    free(data2);

    return EXIT_SUCCESS;
}
```


Life Result S.r.l. – Via S. De Stefano, n° 23 – 71121 Foggia (FG)