



SAPIENZA
UNIVERSITÀ DI ROMA

Allineamento, Denoising e Stacking di immagini lunari mediante tecniche tradizionali e Unsharp Masking basato su Deep Learning

Facoltà di Ingegneria dell'Informazione, Informatica e Statistica
Laurea Triennale in Ingegneria Informatica

Andrea Spinelli

Matricola 1985877

Relatore

Prof. Thomas Alessandro Ciarfuglia

Anno Accademico 2023/2024

Allineamento, Denoising e Stacking di immagini lunari mediante tecniche tradizionali e Unsharp Masking basato su Deep Learning

Laurea Triennale. Sapienza Università di Roma

© 2024 Andrea Spinelli. Tutti i diritti riservati

Questa tesi è stata composta con \LaTeX e la classe Sapthesis.

Email dell'autore: andreaspinelli2002@gmail.com

*«Tutta colpa della Luna, quando si avvicina troppo alla Terra fa impazzire tutti»
— William Shakespeare*

Sommario

TODO

Indice

1	Introduzione	1
1.1	Evoluzione dell'astrofotografia	1
1.1.1	Breve storia e sviluppo tecnologico	1
1.2	Stato dell'arte: strumentazione e tecniche moderne	2
1.2.1	Hardware	3
1.2.2	Software e algoritmi nell'astrofotografia	3
1.3	Rumore e artefatti nelle immagini astronomiche	4
1.4	La Luna	5
2	Elaborazione di immagini lunari	7
2.1	Calibrazione di immagini	7
2.1.1	Bias Frames	7
2.1.2	Dark Frames	8
2.1.3	Flat Frames	9
2.1.4	Processo completo di calibrazione	10
2.2	Allineamento delle immagini	11
2.2.1	Feature Detection e Matching ORB, SIFT e SURF	11
2.2.2	Trasformazioni omografiche: RANSAC	13
2.2.3	Algoritmo di allineamento completo	15
2.3	Pre-processing delle immagini	15
2.3.1	Denoising tramite reti neurali: DnCNN	15
2.3.2	Unsharp Masking e personalizzazione	18
2.3.3	Rimozione dello sfondo	19
2.4	Stacking delle immagini	20
2.4.1	Algoritmi di stacking	20
2.5	Post-Processing delle immagini	21

2.5.1	Miglioramento di nitidezza e contrasto	22
2.5.2	Bilanciamento del colore	22
3	Implementazione	25
3.1	Architettura del software	25
3.1.1	Calibrazione	26
3.1.2	Allineamento	27
3.1.3	Pre-processing	30
3.1.4	Stacking	33
3.1.5	Post-processing	33
3.2	Sfide affrontate e soluzioni adottate	33
4	Valutazione dei risultati e metriche di qualità	34
4.1	Metriche di valutazione con riferimento	34
4.1.1	SSIM (Structural Similarity Index Measure)	34
4.1.2	SNR (Signal-to-Noise Ratio)	34
4.2	Metriche di valutazione senza riferimento	34
4.2.1	NIQE (Naturalness Image Quality Evaluator)	34
4.2.2	BRISQUE (Blind/Referenceless Image Spatial Quality Evaluator)	34
4.2.3	LIQE (Language-Image Quality Evaluator)	34
4.2.4	Motivazione della scelta di LIQE come metrica di riferimento	34
4.3	Analisi e miglioramenti ottenuti	34
4.3.1	Effetti della calibrazione	34
4.3.2	Impatto del denoising	34
4.3.3	Benefici dello stacking	34
4.3.4	Miglioramenti con sharpening e contrasto	34
	Conclusions	35
	Acknowledgements	36

Capitolo 1

Introduzione

1.1 Evoluzione dell'astrofotografia

L'**astrofotografia** si occupa di fotografare oggetti celesti come stelle, pianeti, galassie e nebulose. Questa pratica ha radici risalenti al XIX secolo: nel 1822 fu scattata la prima foto nella storia da Nicéphore Niépce, e già nel 1840 John William Draper catturò la prima immagine della Luna, segnando l'inizio di una nuova era nell'osservazione astronomica. Nel 1850, William Cranch Bond e John Adams Whipple scattarono la prima fotografia di una stella, *Vega*, con un'esposizione di 1000 secondi.

1.1.1 Breve storia e sviluppo tecnologico

Nei primi anni le immagini erano ottenute con *lastre fotografiche di vetro*, che richiedevano tempi di esposizione estremamente lunghi. Catturare immagini di oggetti deboli come nebulose e galassie era un'impresa ardua e poteva richiedere ore o notti intere di esposizione, rendendo il processo molto laborioso. Le prime immagini si concentravano infatti su oggetti luminosi, come la Luna e i pianeti, mentre stelle più deboli e galassie rimanevano al di là delle capacità tecnologiche dell'epoca [22]. Con la nascita della *fotografia a colori* gli astronomi poterono registrare le diverse tonalità di colore degli oggetti celesti, rendendo possibile ottenere informazioni circa la composizione chimica e la temperatura di stelle e nebulose. La vera rivoluzione arrivò con l'avvento della **fotografia digitale** e l'introduzione di *dispositivi a carica accoppiata* (*CCD*) alla fine degli anni '60. I *CCD* avevano sensibilità alla luce notevolmente superiore rispetto alle lastre fotografiche, permettendo tempi di esposizione più brevi e maggiore qualità dell'immagine. Ciò consentì di rilevare oggetti celesti più deboli e ridurre significativamente il rumore nelle immagini [2].

Con la diffusione dei computer, l'elaborazione delle immagini digitali divenne parte integrante dell'astrofotografia. Tecniche come la *calibrazione*, l'*allineamento*, la *riduzione del rumore* e lo *stacking* delle immagini hanno permesso di ottenere risultati di qualità superiore rispetto alle immagini singole. L'utilizzo di algoritmi avanzati consentì di rivelare dettagli nascosti, migliorare il contrasto e ridurre al minimo il rumore aumentando l'accuratezza delle osservazioni astronomiche [7].

Oggi l'astrofotografia è accessibile non solo ad astronomi professionisti, ma anche ad appassionati dilettanti. L'ampia disponibilità di telescopi, fotocamere e software avanzati ha reso possibile catturare immagini di alta qualità anche con strumenti di costo contenuto. L'astrofotografia è diventata un hobby popolare tra appassionati di astronomia e fotografia, che condividono le proprie immagini e scoperte sui social e forum online come *Cloudy Nights* o *AstroBin*.

1.2 Stato dell'arte: strumentazione e tecniche moderne

L'astrofotografia moderna si avvale di strumenti sofisticati e tecniche avanzate per catturare immagini di alta qualità. I telescopi sono dotati di montature motorizzate che compensano il movimento apparente del cielo, consentendo esposizioni più lunghe senza sfocature. Le fotocamere digitali, spesso equipaggiate con sensori *CCD* o *CMOS*, sono in grado di catturare immagini ad alta risoluzione anche di oggetti celesti deboli quali galassie o nebulose, preservandone i dettagli. I software di elaborazione delle immagini offrono strumenti avanzati per la calibrazione, l'allineamento e la post-produzione delle immagini, consentendo di ottenere risultati di qualità professionale.



Figura 1.1. *Pillars of Creation, Nebulosa dell'Aquila*, ripresi dal *James Webb* combinando dati acquisiti da *NIRCam* e *MIRI*. Immagine originale disponibile su [14]

Esistono diversi telescopi spaziali, come l'*Hubble Space Telescope* [9] della *NASA* e il *James Webb Space Telescope* [6], gestito congiuntamente da *ESA*, *NASA* e *CSA*. Questi strumenti, orbitando al di fuori dell'atmosfera terrestre, sono in grado di catturare immagini ad altissima risoluzione e sensibilità, libere dalle distorsioni e dall'inquinamento luminoso terrestre. Sfruttano a proprio vantaggio fenomeni fisici come le *lenti gravitazionali* e la *diffrazione* per osservare oggetti nello spazio profondo, ben oltre le capacità degli strumenti terrestri. Questi telescopi, inoltre, sono dotati di telecamere come la *NIRCam* (Near Infrared Camera) e la *MIRI* (Mid Infrared Instrument), che permettono di osservare l'universo in bande di luce altrimenti invisibili, rivelando dettagli nascosti e processi fisici altrimenti inaccessibili (Figura 1.1), e si dimostrano fondamentali per la ricerca astronomica.

1.2.1 Hardware

Nell'astrofotografia tradizionale si utilizzano principalmente tre tipi di strumenti:

- **Telescopi:** I telescopi sono fondamentali nell'astrofotografia; ne esistono di diversi tipi, tra cui rifrattori, riflettori e catadiottrici, ciascuno con caratteristiche specifiche. I telescopi moderni variano da piccoli modelli portatili a grandi strutture professionali, come l'*ELT* (Extremely Large Telescope), con uno specchio principale di ben 39m di diametro, che è ancora in costruzione ma si prevede diventerà operativo entro il 2027 [16]. La scelta del telescopio dipende dall'oggetto celeste da osservare e dal livello di dettaglio desiderato, oltre che dal budget disponibile.
- **Fotocamere:** Le fotocamere *CCD* (Charge-Coupled Device) sono ampiamente utilizzate in ambito astronomico per la loro alta sensibilità e basso *rumore elettronico*. Negli ultimi anni, le fotocamere *CMOS* (Complementary Metal-Oxide-Semiconductor) hanno guadagnato popolarità in quanto più performanti e più accessibili economicamente. Tali fotocamere offrono elevate risoluzioni, velocità di lettura più rapide e buona *efficienza quantica*, rendendole adatte sia per l'uso professionale che amatoriale [11].
- **Montature:** Una montatura stabile e precisa è essenziale per compensare la rotazione terrestre durante le lunghe esposizioni.
 - Le *montature equatoriali* sono progettate per seguire il movimento apparente delle stelle nel cielo, consentendo di mantenere gli oggetti celesti centrati nell'inquadratura.
 - Le *montature altazimutali*, più semplici da utilizzare, richiedono sistemi di derotazione o software di correzione più sofisticati per lunghe esposizioni, a causa della rotazione di campo.
 - Le *montature computerizzate*, dotate di sistemi *GoTo*, permettono di puntare automaticamente verso specifici oggetti celesti e di tracciarli con precisione.

1.2.2 Software e algoritmi nell'astrofotografia

Con il tempo i software utilizzati in ambito astrofotografico sono arrivati al punto tale per cui non è necessario essere dotati di un *hardware* professionale per catturare immagini di corpi celesti anche dal proprio cortile (o "*from my backyard*").

I software principalmente utilizzati, quali *PixInsight* o *AutoStakkert* implementano diversi algoritmi in grado di migliorare sensibilmente i risultati finali:

- **Calibrazione delle immagini:** La calibrazione è un passaggio cruciale per rimuovere artefatti e rumori dovuti alla strumentazione dalle immagini astronomiche. Questo processo utilizza diversi insiemi di frame di calibrazione: *bias frames*, *dark frames* e *flat frames* (più nel dettaglio nella *Sezione 2.1*)

dai quali è possibile estrarre informazioni sul rumore dell'immagine, così da poterlo sottrarre alla stessa [7].

- **Allineamento delle immagini:** L'allineamento è necessario per combinare correttamente più immagini dello stesso oggetto. Algoritmi di feature detection come **ORB** (Oriented FAST and Rotated BRIEF) [18], **SIFT** (Scale-Invariant Feature Transform) e **SURF** (Speeded Up Robust Features) identificano punti caratteristici nelle immagini per calcolare *trasformazioni omografiche*, utilizzate per correggere differenze di scala, rotazione e prospettiva tra le immagini (più nel dettaglio nella [Sezione 2.2](#)) [2] [15].
- **Riduzione del rumore:** La riduzione del rumore migliora la qualità finale delle immagini. Tecniche tradizionali come l'unsharp masking [2] accentuano i dettagli sottraendo una versione sfocata dell'immagine originale. Approcci più avanzati utilizzano reti neurali convoluzionali profonde, come **DnCNN** (Denoising Convolutional Neural Network) [26], che apprendono a rimuovere il rumore preservando i dettagli attraverso l'addestramento su grandi dataset (più nel dettaglio nelle sezioni 2.3 e ??).
- **Stacking delle immagini:** Lo *stacking* ("impilamento") combina multiple esposizioni per ottenere un'unica immagine finale ottimizzandone il rapporto segnale-rumore. Questa tecnica riduce il rumore casuale e mette in evidenza dettagli deboli non visibili in singole esposizioni. Metodi come il **Weighted Average Stacking** assegnano un peso a ciascuna immagine in base ad un criterio prefissato, per ottenere in seguito un'immagine data dalla media ponderata dei valori dei singoli frames in input (più nel dettaglio nella [Sezione 2.4](#)).

1.3 Rumore e artefatti nelle immagini astronomiche

La qualità delle immagini astronomiche è influenzata da diversi tipi di rumore e artefatti, che devono essere mitigati per ottenere risultati ottimali (vedremo nella [Sezione 2.3.1](#) come ridurli). I principali tipi di rumore e artefatti includono:

- **Rumore termico:** Il rumore termico (*Dark Current*) è generato dall'agitazione termica degli elettroni all'interno del sensore della fotocamera, producendo un segnale anche in assenza di luce. Questo tipo di rumore aumenta con la temperatura del sensore ed è particolarmente significativo nelle lunghe esposizioni. Per ridurlo, molti sensori astronomici sono raffreddati tramite *sistemi termoelettrici* o *criogenici*. La sottrazione dei *dark frames* durante la calibrazione permette di correggere questo rumore.
- **Rumore del sensore:** Include diversi tipi di rumore intrinseco al sensore della fotocamera:
 - **Rumore di lettura:** deriva dall'elettronica durante il processo di lettura e digitalizzazione del segnale dal sensore. È generalmente costante e può essere minimizzato utilizzando componenti elettronici di alta qualità.

Sebbene il rumore di lettura non possa essere eliminato, l'uso di *bias frames* nella calibrazione aiuta a compensare l'offset introdotto dall'elettronica.

- **Rumore di schema fisso:** Il rumore di schema fisso (*Fixed Pattern Noise*) è causato dalle variazioni di sensibilità tra i pixel, si manifesta come pattern ripetuti nell'immagine. L'utilizzo dei *flat frames* nella calibrazione aiuta a correggere queste imperfezioni.
- **Amp glow:** È una luminescenza causata dal calore generato dai circuiti di amplificazione del sensore, visibile come un bagliore ai bordi dell'immagine. La sottrazione dei *dark frames* e il raffreddamento del sensore contribuiscono a ridurre questo fenomeno.
- **Altri artefatti:** altri artefatti includono la vignettatura, un oscuramento ai bordi dell'immagine dovuto alle caratteristiche ottiche del sistema e mitigabile con i *flat frames*, e i *pixel caldi e morti*, ossia pixel che rimangono sempre accesi o spenti, che possono essere mappati e corretti durante l'elaborazione.

1.4 La Luna

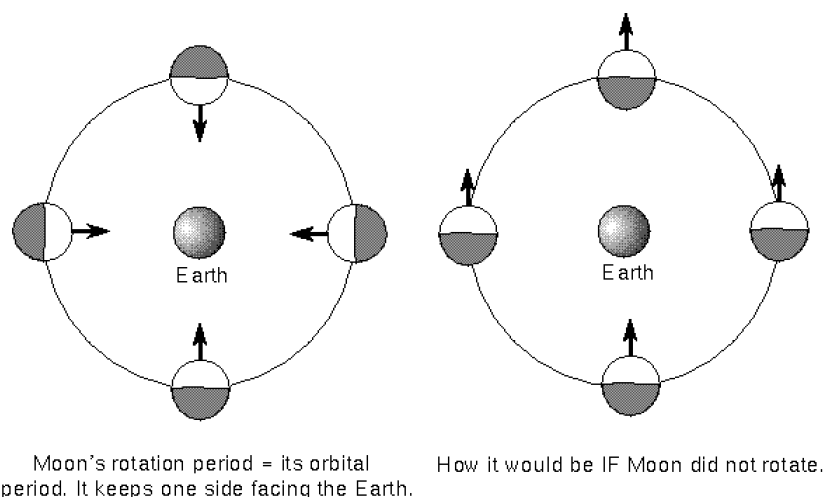


Figura 1.2. Dimostrazione grafica del moto di rotazione e rivoluzione della Luna [21]

Tra i corpi celesti più affascinanti e accessibili nell'astrofotografia spicca sicuramente la **Luna**. Grazie alla sua vicinanza alla Terra, alla sua luminosità e alle sue dimensioni apparenti relativamente grandi, è possibile catturare immagini lunari di alta qualità anche in assenza di strumentazione professionale. Fotografare la Luna rispetto ad altri corpi celesti presenta diversi vantaggi:

- **Rotazione sincrona:** La Luna mostra sempre la stessa faccia verso la Terra a causa della sua *rotazione sincrona*, ovvero il suo periodo di rotazione attorno al proprio asse è uguale al periodo di rivoluzione attorno alla Terra (circa 27,3 giorni) [Figura 1.2]. Questo fenomeno, dovuto alla *risonanza mareale*,

comporta diverse semplificazioni nell'elaborazione delle immagini, eliminando la necessità di processi di derotazione [23].

- **Luminosità elevata:** La Luna è l'oggetto celeste più luminoso nel cielo notturno. Questa luminosità consente di utilizzare tempi di esposizione più brevi rispetto ad altri oggetti astronomici (da 1/250 a 1/100 di secondo, a seconda della percentuale di illuminazione, contro tempi fino a 20-30 secondi per oggetti dello spazio profondo), riducendo gli effetti del **rumore elettronico** e del movimento apparente.
- **Facilità di localizzazione:** Essendo facilmente visibile ad occhio nudo, la Luna non richiede sistemi di puntamento sofisticati o montature equatoriali per essere fotografata. Un semplice treppiede è sufficiente per stabilizzare la fotocamera durante la cattura delle immagini.

Nell'astrofotografia planetaria in generale, quando il soggetto non è la Luna, i pianeti sono notevolmente distanti dalla Terra, e piccole perturbazioni dell'atmosfera possono causare rumore e artefatti fastidiosi. È possibile ovviare a questo problema registrando dei video (piuttosto che singole immagini) e rimuovere tali artefatti tramite una tecnica chiamata *Lucky Imaging*, combinata con tecniche di *derotazione* per mitigare l'effetto della rotazione del pianeta attorno al proprio asse.

L'astrofotografia lunare professionale consiste spesso nello scattare immagini che catturano sezioni diverse della Luna; queste vengono prima processate singolarmente e poi unite mediante tecniche di *stitching*. Un'altra tecnica molto diffusa è quella utilizzata per ottenere le cosiddette *Mineral Moon*: si eseguono diversi scatti applicando filtri a banda stretta sui telescopi, per poi elaborare i canali RGB singolarmente, facendo risaltare la presenza di minerali diversi sulla superficie lunare [13].

Fotografare la Luna presenta anche diverse sfide. Essa è caratterizzata da un elevato contrasto tra le aree illuminate e quelle in ombra, specialmente durante le fasi parziali, rendendo difficile ottenere un'esposizione bilanciata che catturi dettagli in entrambe le zone [19]. Nonostante la sua vicinanza, l'atmosfera terrestre influenza la qualità delle immagini lunari, introducendo turbolenze (*seeing*), dispersione e attenuazione della luce, portando ad una riduzione di nitidezza e contrasto nei risultati [19]. Infine, sebbene la luminosità della Luna consenta di fotografarla anche da aree urbane, l'inquinamento luminoso può ancora influenzare la qualità, specialmente quando si vuole catturare dettagli più fini o nelle fasi meno illuminate.

Nel contesto di questo progetto, sono state scattate personalmente immagini della Luna utilizzando una **Fujifilm FinePix S1**, una fotocamera bridge dotata di uno zoom ottico fino a 50×. Senza l'ausilio di telescopi o montature specializzate, ma semplicemente con l'utilizzo di un treppiede standard, è stato possibile catturare immagini dettagliate della superficie lunare.

Questa scelta evidenzia come, grazie alle moderne tecnologie e alle tecniche di elaborazione delle immagini, sia possibile ottenere risultati di qualità anche con attrezzature relativamente modeste. Le immagini acquisite sono state utilizzate per testare e validare gli algoritmi sviluppati nel corso del progetto, dimostrando l'efficacia delle metodologie proposte nell'ottimizzazione di fotografie lunari.

Capitolo 2

Elaborazione di immagini lunari

Questo capitolo si propone di approfondire le tecniche di elaborazione delle immagini lunari, illustrando le nozioni teoriche alla base degli algoritmi implementati nel progetto. Ogni tecnica verrà descritta in dettaglio, partendo da calibrazione e allineamento, passando per pre-processing e stacking, per concludere con il post-processing. Quando necessario, verranno forniti pseudocodice e descrizioni dei processi matematici applicati alle immagini; l'implementazione sarà invece discussa nel capitolo 3.

2.1 Calibrazione di immagini

La **calibrazione** delle immagini è un passaggio fondamentale nell'astrofotografia, necessario per rimuovere rumore e artefatti introdotti dalla strumentazione. In particolare, nel contesto delle immagini lunari, la calibrazione è utile per rimuovere il rumore termico e i difetti del sensore, oltre a uniformare l'illuminazione dell'immagine. Questo processo è composto da tre fasi principali: la cattura di *bias frames*, *dark frames* e *flat frames*. Tali scatti devono essere acquisiti con la fotocamera nello stesso stato in cui sono state scattate le immagini lunari, in particolare nelle stesse condizioni termiche. Infatti, quando viene eseguita una sessione di molti scatti, o con lunghe esposizioni, la macchinetta tende a scaldarsi causando effetti non sempre trascurabili, e sono proprio quelli che vogliamo mitigare mediante la fase di calibrazione. In questa fase gli scatti della luna vengono denominati *light frames* [7].

2.1.1 Bias Frames

I **bias frames** son scatti acquisiti con il tempo di esposizione più breve possibile (il minimo supportato dalla macchina fotografica, idealmente zero), ISO uguale a quello dei light frames e con l'otturatore della fotocamera chiuso. Questi frame catturano il **rumore di bias**, un segnale di offset introdotto dall'elettronica del sensore in assenza di luce. Il rumore di bias è presente in tutte le immagini acquisite con una macchina fotografica, e varia leggermente da pixel a pixel.

Per correggere questo rumore si calcola il cosiddetto **master bias** combinando i

diversi bias frames, generalmente calcolandone la media. Il master bias viene poi sottratto da tutte le immagini acquisite, inclusi gli altri frame di calibrazione.

Algoritmo 2.1 - Calcolo del master bias:

Data la lista di bias frames B_f , l'algoritmo calcola il master bias M_b

```

1: function CALCULATE_MASTER_BIAS( $B_f$ )
2:    $N \leftarrow$  numero di bias frames
3:    $M_b \leftarrow 0$ 
4:   for  $i \leftarrow 1$  to  $N$  do
5:      $M_b \leftarrow M_b + \frac{B_f[i]}{N}$  ▷ Media dei bias frames
6:   end for
7:   return  $M_b$ 
8: end function

```

Applicazione: Per applicare il master bias a un'immagine, si sottrae semplicemente il master bias dall'immagine originale. In formula, data un'immagine *original* e il master bias *master_bias*, l'immagine calibrata *final_image* sarà data da:

$$final_image = original - master_bias$$

2.1.2 Dark Frames

I **dark frames**, acquisiti con stessi ISO e tempi di cattura dei light frames, ma con l'otturatore chiuso, catturano il **rumore termico** causato dall'agitazione termica degli elettroni nel sensore. Questo rumore aumenta con il tempo di esposizione e con la temperatura del sensore e può variare significativamente tra scatti differenti.

Per correggere il rumore termico, si calcola il **master dark** combinando i diversi dark frames, generalmente calcolandone la media. Il master dark viene poi sottratto ai light frames e ai flat frames. È importante sottrarre il master bias dai dark frames prima di calcolare il master dark, per evitare di sommare due volte il rumore di bias.

Algoritmo 2.2 - Calcolo del Master Dark:

Data la lista di dark frames D_f e il master bias M_b , l'algoritmo calcola il master dark M_d .

```

1: function CALCULATE_MASTER_DARK( $D_f, M_b$ )
2:    $N \leftarrow$  numero di dark frames
3:    $M_d \leftarrow 0$ 
4:   for  $i \leftarrow 1$  to  $N$  do
5:      $D_c \leftarrow D_f[i] - M_b$  ▷ rimozione del bias
6:      $M_d \leftarrow M_d + \frac{D_c}{N}$  ▷ media dei dark frames
7:   end for
8:   return  $M_d$ 
9: end function

```

Applicazione: Per applicare il master dark a un'immagine, si sottrae semplicemente il master dark dall'immagine originale. In formula, data un'immagine *original* e il master dark *master_dark*, l'immagine calibrata *final_image* sarà data da:

$$final_image = original - master_dark$$

2.1.3 Flat Frames

I **flat frames** sono scatti acquisiti fotografando una sorgente di luce uniforme, come un cielo crepuscolare o un pannello luminoso. Vengono acquisiti con lo stesso tempo di esposizione e ISO dei light frames, ma con l'otturatore aperto. Questi frame catturano le variazioni nella risposta dei pixel del sensore e il **rumore di vignettatura** introdotto dal sistema ottico, ovvero la diminuzione dell'illuminazione verso i bordi dell'immagine.

Per correggere queste imperfezioni si calcola un **master flat** combinando i diversi flat frames, solitamente attraverso la media. Prima di calcolare il master flat, è necessario sottrarre sia il master bias che il master dark dai flat frames. Una volta calcolato il master flat, questo viene normalizzato dividendolo per il valore medio dei suoi pixel. Il master flat viene poi utilizzato per normalizzare le immagini scientifiche, dividendo ogni pixel dell'immagine per il corrispondente valore nel master flat.

Algoritmo 2.3 - Calcolo del Master Flat:

Data la lista di flat frames F_f , il master bias M_b e il master dark M_d , l'algoritmo calcola il master flat normalizzato M_f .

```

1: function CALCULATE_MASTER_FLAT( $F_f, M_b, M_d$ )
2:    $N \leftarrow$  numero di flat frames
3:    $M_f \leftarrow 0$ 
4:   for  $i \leftarrow 1$  to  $N$  do
5:      $F_c \leftarrow F_f[i] - M_b - M_d$                                  $\triangleright$  rimozione di bias e dark
6:      $M_f \leftarrow M_f + \frac{F_c}{N}$                                      $\triangleright$  media dei flat frames
7:   end for
8:    $M_f \leftarrow \frac{M_f}{\text{mean}(M_f)}$                                  $\triangleright$  normalizzazione
9:   return  $M_f$ 
10: end function

```

Applicazione: Per applicare il master flat a un'immagine, si divide semplicemente l'immagine originale:

$$final_image = \frac{original}{M_f}$$

Nel contesto di questo progetto, i flat frames non sono stati acquisiti per due principali motivi. In primo luogo, la loro acquisizione risulta complessa e richiede condizioni specifiche che non sono state facilmente riproducibili durante le sessioni fotografiche.

In secondo luogo, l'effetto della vignettatura non ha rappresentato un problema significativo nelle immagini ottenute, poiché la Luna era posizionata verso il centro degli scatti e occupava meno di un terzo dell'altezza dell'immagine. Nonostante ciò, il processo di calibrazione è stato comunque implementato per consentire l'utilizzo di flat frames, garantendo flessibilità e scalabilità del metodo di elaborazione adottato.

2.1.4 Processo completo di calibrazione

Il processo completo di calibrazione applica in sequenza le correzioni con i master frames; si sottraggono master bias e master dark e si divide per il master flat:

Algoritmo 2.4 – Calibrazione di un'immagine:

Data un'immagine Img e i master frames M_b , M_d e M_f , l'algoritmo calcola l'immagine calibrata Out .

```

1: function CALIBRATE_IMAGE( $Img, M_b, M_d, M_f$ )
2:    $Out \leftarrow \frac{Img - M_b - M_d}{M_f}$ 
3:   return  $Out$ 
4: end function

```

Applicazione: Per calibrare un'immagine, si applica la funzione *calibrate_image* con i master frames calcolati. In formula, data un'immagine *original* e i master frames M_b , M_d e M_f , l'immagine calibrata *final_image* sarà data da:

$$final_image = \frac{original - M_b - M_d}{M_f}$$

Questo processo (illustrato in figura 2.1) permette di ottenere immagini corrette da rumori elettronici, termici e da imperfezioni ottiche.

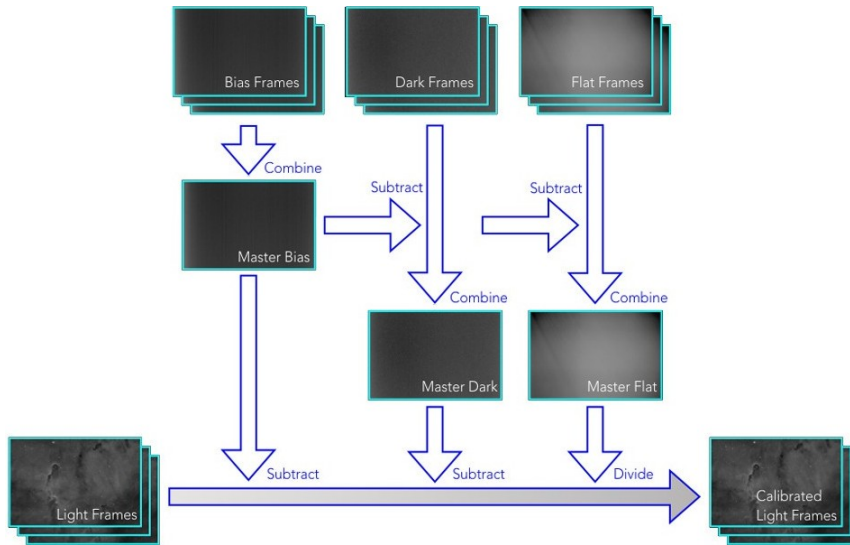


Figura 2.1. Illustrazione schematica dell'uso di immagini di calibrazione [24]

2.2 Allineamento delle immagini

L'allineamento delle immagini è un passaggio cruciale nell'elaborazione delle immagini lunari, necessario per compensare eventuali spostamenti o rotazioni tra gli scatti e per combinare efficacemente più immagini tramite tecniche di *stacking*. Questo processo si basa sull'identificazione di punti caratteristici comuni tra le immagini e sul calcolo delle trasformazioni geometriche necessarie (nel nostro caso **trasformazioni omografiche**) per sovrapporle perfettamente. L'allineamento delle immagini può essere eseguito manualmente, ma è preferibile utilizzare algoritmi automatici per garantire una maggiore precisione e riproducibilità.

2.2.1 Feature Detection e Matching ORB, SIFT e SURF

Gli algoritmi di *feature detection* e *matching* individuano punti di interesse nei distintivi nelle immagini, come bordi, angoli o altre caratteristiche uniche, e calcolando descrittori che rappresentano l'intensità locale attorno a ciascun punto. Successivamente, i descrittori vengono confrontati per trovare corrispondenze tra i punti di interesse delle diverse immagini, consentendo di determinare le trasformazioni geometriche necessarie per allineare le immagini.

Tra i vari algoritmi di feature detection e matching disponibili, tre si distinguono per la loro efficacia e diffusione: ORB (Oriented FAST and Rotated BRIEF), SIFT (Scale-Invariant Feature Transform) e SURF (Speeded-Up Robust Features), i quali sono in grado di identificare punti di interesse invarianti rispetto a rotazioni, traslazioni e scalature, e sono particolarmente adatti per l'allineamento di immagini astronomiche.

- **SIFT** (Scale-Invariant Feature Transform): è noto per la sua elevata accuratezza e robustezza a cambiamenti di scala, rotazione e illuminazione. L'algoritmo identifica i keypoints costruendo una piramide di immagini a diverse scale e cercando i massimi locali. L'orientamento di ogni keypoint viene determinato analizzando gli istogrammi dell'orientamento del gradiente nell'intorno del punto stesso. Un descrittore SIFT ha generalmente 128 dimensioni e viene calcolato campionando gli orientamenti del gradiente in una griglia 16x16 attorno al punto chiave. Questo algoritmo è molto preciso, ma anche computazionalmente costoso, il che può essere problematico per applicazioni in tempo reale o con grandi volumi di dati [10].
- **SURF** (Speeded-Up Robust Features): è stato sviluppato come alternativa più veloce a SIFT. Utilizza un'approssimazione del determinante dell'Hessiana per il rilevamento di keypoints e un descrittore basato sulla somma delle risposte alle wavelet di Haar. Questo approccio rende SURF più efficiente dal punto di vista computazionale rispetto a SIFT, pur mantenendo un buon livello di accuratezza e robustezza. Inoltre, SURF integra le informazioni sul gradiente all'interno di un sotto-patch, migliorando le performance nella rilevazione di caratteristiche in presenza di rumore. Tuttavia, come SIFT, anche SURF è coperto da brevetti che ne limitano l'utilizzo [1].

- **ORB** (Oriented FAST and Rotated BRIEF): è un descrittore binario veloce, progettato per essere efficiente dal punto di vista computazionale e libero da restrizioni di licenza. Combina il rilevatore di keypoints FAST, noto per la sua rapidità, con un descrittore BRIEF ruotato, che è efficiente da calcolare e confrontare. ORB aggiunge un componente di orientamento veloce e accurato a FAST, consentendo di calcolare in modo efficiente le caratteristiche BRIEF orientate. Per migliorare le prestazioni, ORB utilizza un metodo di apprendimento per decorrelare le caratteristiche BRIEF, garantendo invarianza rotazionale. Rispetto ad algoritmi come SIFT e SURF, ORB offre prestazioni comparabili in molte situazioni, pur essendo significativamente più veloce; dimostra una notevole resistenza al rumore gaussiano, anche se, in presenza di forti distorsioni prospettiche, può risultare meno preciso [18].

Nel contesto di questo progetto, è stato scelto di utilizzare l'algoritmo ORB grazie alla sua efficienza computazionale e alla sua robustezza nei confronti di rotazioni e traslazioni, comuni nelle immagini acquisite senza montature motorizzate. Tuttavia l'implementazione prevede la possibilità di utilizzare anche SIFT e SURF, mediante un'interfaccia comune per la selezione dell'algoritmo di feature detection e matching.

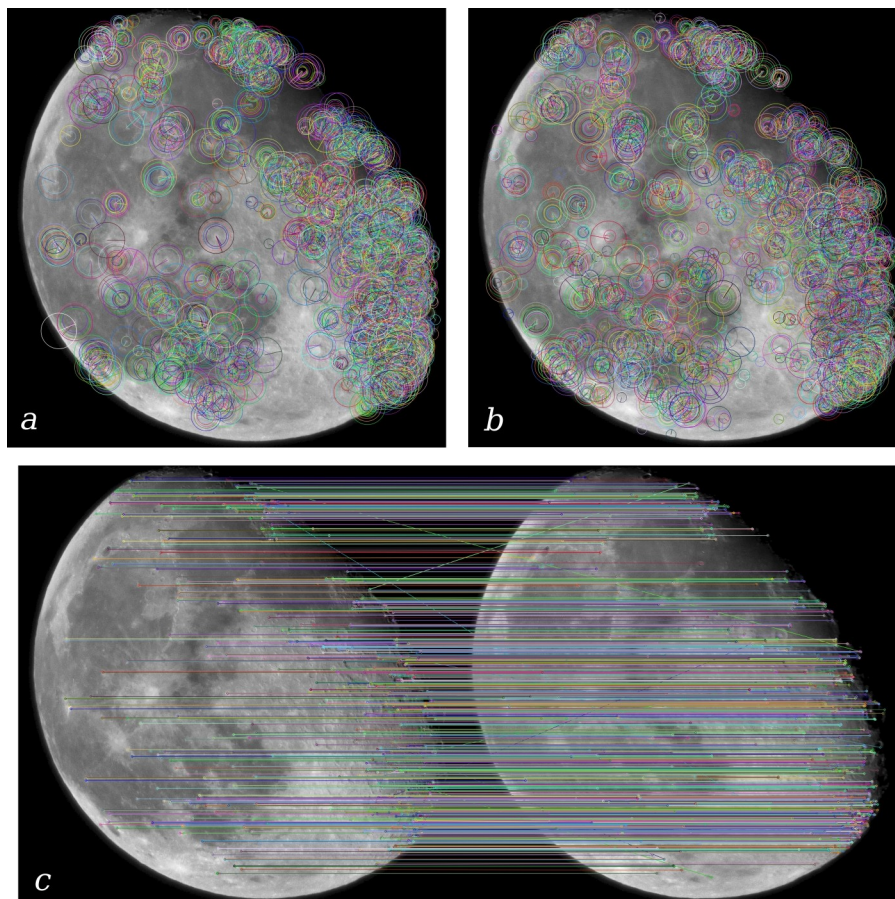


Figura 2.2. Applicazione di ORB (a-b) e RANSAC (c) su due scatti da me acquisiti.
a-b: visualizzazione di 5000 *keypoint* estratti da due immagini
c: visualizzazione dei *matches* tra i keypoints delle due immagini

2.2.2 Trasformazioni omografiche: RANSAC

Una **omografia** è una trasformazione geometrica che mappa punti da un piano a un altro, mantenendo la collinearità e la connettività dei punti. Nel contesto dell'allineamento delle immagini, l'omografia viene utilizzata per correggere le differenze di posizione, scala, rotazione e prospettiva tra le immagini.

L'omografia è rappresentata da una matrice 3×3 denotata con \mathbf{H} che descrive una trasformazione tra due piani proiettivi. La relazione tra un punto nell'immagine di origine (x, y) e il suo corrispondente nell'immagine trasformata (x', y') è data dalla seguente equazione:

$$\begin{bmatrix} x' \\ y' \\ \omega' \end{bmatrix} = H \cdot \begin{bmatrix} x \\ y \\ \omega \end{bmatrix}$$

dove ω e ω' sono i fattori di scala che consentono di rappresentare le trasformazioni proiettive e le coordinate finali sono ottenute dividendo per ω' :

$$\begin{bmatrix} x'' \\ y'' \end{bmatrix} = \begin{bmatrix} x'/\omega' \\ y'/\omega' \end{bmatrix}$$

L'omografia può essere calcolata a partire da un set di corrispondenze tra punti nelle due immagini, utilizzando l'algoritmo **RANSAC** (Random Sample Consensus) per stimare i parametri della matrice H .

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

L'idea alla base di RANSAC è che gli *inlier* (le corrispondenze corrette) concordano tra loro sulla trasformazione da stimare, mentre gli *outlier* (le corrispondenze errate) non concordano e tendono a essere incoerenti [5].

1. Seleziona casualmente un sottoinsieme minimo di corrispondenze.
2. Stima il modello (in questo caso, l'omografia) usando il sottoinsieme selezionato.
3. Calcola un errore di adattamento per tutte le corrispondenze.
4. Determina gli inliers come le corrispondenze con errore inferiore a una soglia.
5. Se il numero di inlier è superiore a una soglia, ricalcola il modello usando tutti gli inlier e termina.
6. Altrimenti, ripete i passaggi precedenti per un numero prefissato di iterazioni.

RANSAC dipende da due parametri: il *numero di iterazioni* e la *soglia di errore*. Un numero maggiore di iterazioni determina quante volte l'algoritmo estrae campioni casuali. Deve essere sufficientemente grande per garantire una probabilità alta di trovare un modello senza outlier. La soglia di errore definisce il massimo errore accettabile per considerare una corrispondenza come inlier. Esistono altri parametri, uno dei quali è il *numero di corrispondenze* da selezionare per stimare l'omografia, solitamente 4. Qui sotto è riportato un esempio di pseudocodice per il calcolo dell'omografia tramite RANSAC.

Algoritmo 2.5 – Algoritmo RANSAC per la stima dell'omografia:

Dati *matches*, la lista di matches, *k*, il numero massimo di iterazioni, *thr*, la soglia per determinare gli inlier e *n*, il numero di corrispondenze da selezionare per l'omografia, l'algoritmo ritorna la matrice omografica stimata.

```

1: function ESTIMATE_HOMOGRAPHY(matches, k, thr)
2:    $H^* \leftarrow \text{null}$                                 ▷ Miglior omografia stimata
3:    $score^* \leftarrow 0$                                 ▷ Miglior punteggio
4:   for  $i \leftarrow 1$  to  $k$  do
5:      $R \subseteq \text{matches}$  con  $|R| = n$                     ▷ Seleziona  $n$  corrispondenze casuali
6:      $H \leftarrow$  Stima l'omografia usando  $R$ 
7:      $score \leftarrow 0$ 
8:     for  $m \in \text{matches}$  do
9:       if errore( $H$ ,  $m$ ) <  $thr$  then                    ▷ Errore come distanza euclidea
10:         $score \leftarrow score + 1$                         ▷ Conta gli inlier
11:      end if
12:    end for
13:    if  $score > score^*$  then                                ▷ Seleziona il modello migliore
14:      Aggiorna  $H^*$  e  $score^*$ 
15:    end if
16:  end for
17:  return  $H^*$ 
18: end function

```

Il numero di iterazioni k necessarie per garantire una certa probabilità di successo P dipende dalla percentuale di inlier attesa p e dal numero minimo di punti n richiesti per stimare il modello [17]:

$$k = \frac{\log(1 - P)}{\log(1 - p^n)}$$

Ad esempio, se si prevede che l'80% delle corrispondenze siano inlier ($p = 0,8$), si desidera una probabilità di successo del 99% ($P = 0,99$) e si utilizzano $n = 4$ punti per stimare l'omografia, il numero di iterazioni necessarie è:

$$k = \frac{\log(1 - 0.99)}{\log(1 - 0.8^4)} \approx 17.6$$

2.2.3 Algoritmo di allineamento completo

Per ottenere un allineamento più preciso delle immagini, spesso si applica una fase di pre-elaborazione volta a evidenziare le caratteristiche significative. In particolare, i fotogrammi destinati all'estrazione delle caratteristiche vengono convertiti in scala di grigi e sottoposti a filtri per migliorarne nitidezza e contrasto. È importante sottolineare che queste operazioni vengono eseguite su una copia dei fotogrammi, utilizzata esclusivamente per l'estrazione delle caratteristiche, preservando così i dati originali. In questa sezione, tali operazioni sono sintetizzate nella funzione `enhance()`, (più nel dettaglio nella [Sezione 3.1.2](#)).

L'algoritmo completo per l'allineamento delle immagini, che combina le fasi di estrazione delle caratteristiche, corrispondenza dei descrittori, stima dell'omografia tramite RANSAC e applicazione della trasformazione alle immagini, è illustrato nell'[Algoritmo 2.6](#).

Algoritmo 2.6 Allineamento delle immagini:

Data un insieme di immagini I , restituisce l'insieme di immagini allineate A

```

1: function ALIGN_IMAGES( $I$ )
2:   Seleziona un riferimento  $r$                                 ▷ Tipicamente la più nitida
3:    $f \leftarrow$  un algoritmo tra ORB, SIFT, SURF
4:    $k_r, d_r \leftarrow$  f.calculate_descriptors( $r$ )              ▷ Calcola keypoints e descrittori di  $r$ 
5:   for ogni  $i$  in  $I$  do
6:      $k, d \leftarrow$  f.calculate_descriptors( $i$ )              ▷ Calcola keypoints e descrittori di  $i$ 
7:      $m \leftarrow$  match_descriptors( $d_r, d$ )                  ▷ Trova i match tra i descrittori
8:      $H \leftarrow$  ESTIMATE_HOMOGRAPHY( $k_r, k, m$ )              ▷ Calcola l'omografia
9:      $a \leftarrow$  apply_transformation( $i, H$ )                  ▷ Applica l'omografia
10:    Aggiungi  $a$  a  $A$ 
11:   end for
12:   return  $A$ 
13: end function

```

2.3 Pre-processing delle immagini

Il pre-processing delle immagini è quella fase di elaborazione dei singoli scatti (ormai calibrati e allineati), necessaria per migliorare la qualità delle immagini e prepararle per il processo di stacking. Questa fase comprende principalmente il denoising, l'incremento della nitidezza e del contrasto, e l'applicazione di filtri per ridurre l'effetto di *banding* e di *moiré*.

2.3.1 Denoising tramite reti neurali: DnCnn

La riduzione del **rumore** (*denoising*) è un passaggio fondamentale per migliorare la qualità delle immagini, specialmente quando si lavora con scatti acquisiti in condizioni non ideali. Nelle immagini lunari, il rumore può nascondere dettagli importanti e compromettere l'efficacia delle tecniche successive, come lo *stacking*.

Tecniche tradizionali di denoising prevedono l'utilizzo di filtri lineari e non, come il filtro gaussiano o il filtro mediano.

- **Filtro mediano:** è un filtro non lineare utilizzato principalmente per ridurre il rumore impulsivo (come il rumore "sale e pepe") in un'immagine. Sostituisce il valore di ogni pixel con la mediana dei valori dei pixel circostanti all'interno di una finestra (o kernel) di dimensione predefinita. Dato un'immagine $I(x, y)$ e una finestra di dimensione $m \times m$ centrata sul pixel (x, y) , il valore filtrato $I'(x, y)$ è dato da:

$$I'(x, y) = \text{mediana}\{I(i, j) | (i, j) \in \text{finestra}\}$$

Dove la *mediana* è il valore centrale dei pixel ordinati all'interno della finestra.

- **Filtro gaussiano:** è un filtro lineare che applica un'operazione di convoluzione tra l'immagine e una funzione Gaussiana. È utilizzato per ridurre il rumore e sfocare l'immagine, preservando le strutture principali. La funzione Gaussiana bidimensionale con varianza σ^2 è data da:

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

Il filtro gaussiano applicato all'immagine $I(x, y)$ è definito come:

$$I'(x, y) = \sum_{i=-k}^k \sum_{j=-k}^k I(x+i, y+j) G(i, j)$$

Dove k è la dimensione del kernel e σ controlla l'ampiezza della distribuzione Gaussiana.

- **Filtro bilaterale:** è un'operazione di filtraggio che combina l'informazione spaziale (distanza tra i pixel) e quella radiometrica (differenza di intensità tra i pixel), rendendolo ideale per ridurre il rumore preservando i bordi dell'immagine. La formula generale per il filtro bilaterale è la seguente:

$$I'(x, y) = \frac{1}{W(x, y)} \sum_{(i, j) \in S} G_{\sigma_s} \left(\sqrt{(x-i)^2 + (y-j)^2} \right) G_{\sigma_r} (I(x, y) - I(i, j)) I(i, j)$$

Dove (x, y) e (i, j) : sono i pixel della finestra locale S in cui viene calcolato il filtro, mentre $I(x, y)$ e $I(i, j)$ sono i valori di intensità rispettivamente del pixel centrale p e del pixel q all'interno della finestra. $W(x, y)$ È il fattore di normalizzazione che assicura che i pesi totali siano normalizzati.

$$W(x, y) = \sum_{(i, j) \in S} G_{\sigma_s} \left(\sqrt{(x-i)^2 + (y-j)^2} \right) G_{\sigma_r} (I(x, y) - I(i, j))$$

Il filtro bilaterale è computazionalmente costoso, poiché richiede il calcolo di due funzioni Gaussiani per ogni pixel dell'immagine.

Questi filtri possono ridurre il rumore ma tendono a sfocare l'immagine e a ridurre la nitidezza dei dettagli. Inoltre, potrebbero non essere efficaci contro tutti i tipi di rumore presenti nelle immagini lunari. Per superare questo problema, negli ultimi anni sono state sviluppate tecniche di denoising basate su reti neurali, che sfruttano la capacità delle reti di apprendere modelli complessi e non lineari direttamente dai dati.

In questo progetto è stato utilizzato **DnCNN** (Denoising Convolutional Neural Network), una rete neurale profonda progettata specificamente per il denoising di immagini. DnCNN è composto da 17 strati convoluzionali, seguiti da funzioni di attivazione ReLU e da un layer di regressione:

- **Strato di input:** accetta l'immagine rumorosa normalizzata.
- **Strato convoluzionale iniziale:** un layer convoluzionale con 64 filtri 3×3 , padding di 1 pixel, seguito da una funzione di attivazione ReLU.
- **Strati convoluzionali intermedi:** 15 strati convoluzionali con 64 filtri 3×3 , padding di 1 pixel, ciascuno seguito da *batch normalization* e funzione di attivazione ReLU.
- **Strato convoluzionale finale:** un layer convoluzionale con un filtro 3×3 , padding di 1 pixel, che fornisce l'output.
- **Strato di output:** fornisce una stima del rumore presente nell'immagine.

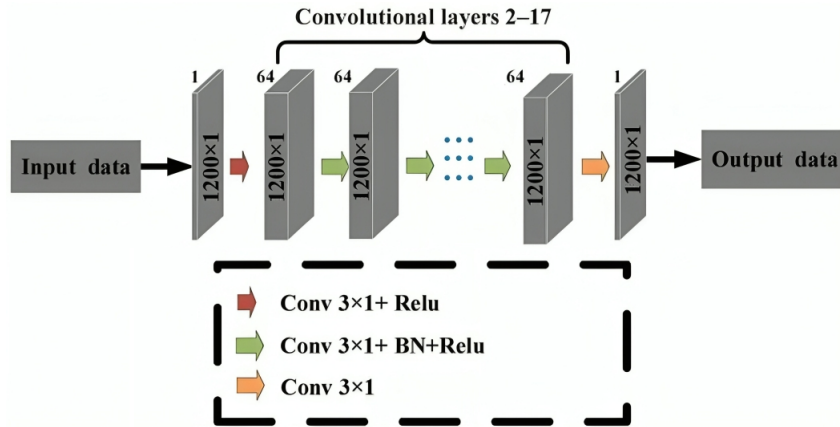


Figura 2.3. Architettura di DnCNN [25]

La rete è addestrata per apprendere la mappatura residua $R(I) = I - I_{\text{denoised}}$, dove I è l'immagine rumorosa e I_{denoised} è l'immagine a cui è stato rimosso il rumore. Durante l'inferenza, la nuova immagine si ottiene sottraendo il rumore stimato dall'immagine di input:

$$I_{\text{denoised}} = I - R(I)$$

Un modello pre-addestrato di DnCNN (DnCNN-S-25: con $\sigma = 25$, e addestrato sul dataset [Train400](#) di Kai Zhang) è stato utilizzato per ridurre il rumore nelle immagini lunari, migliorando la qualità delle immagini e preparandole per il processo di stacking. La procedura utilizzata per la riduzione del rumore con DnCNN è riportata nell'[Algoritmo 2.7](#).

Algoritmo 2.7 – Riduzione del rumore con DnCNN:

Data un'immagine I , l'algoritmo restituisce l'immagine denoised Out .

```

1: function DENOISE_IMAGE( $I$ )
2:    $I_n \leftarrow$  Normalizza  $I$  tra 0 e 1                                ▷ Normalizzazione
3:    $T \leftarrow$  Converti  $I_n$  in tensore
4:    $M \leftarrow$  Carica il modello pre-addestrato DnCNN
5:    $N \leftarrow M(T)$                                                 ▷ Stima del rumore
6:    $Out \leftarrow I_n - N$                                             ▷ Sottrazione del rumore
7:    $Out \leftarrow$  Denormalizza  $Out$  al range originale
8:   return  $Out$ 
9: end function

```

2.3.2 Unsharp Masking e personalizzazione

L'**Unsharp Masking** è una tecnica utilizzata per aumentare la nitidezza dei dettagli in un'immagine. Consiste nel sottrarre una versione sfocata dell'immagine originale dall'immagine stessa, enfatizzando i dettagli e i bordi [20]. La procedura standard è la seguente:

Algoritmo 2.8 Unsharp Mask:

Data un'immagine I , l'algoritmo restituisce l'immagine nitida Out .

```

function UNSHARP_MASK( $I, \alpha$ )
   $I_B \leftarrow$  Gaussian_Blur( $I$ )                                ▷ Sfocatura
   $M \leftarrow I - I_B$                                            ▷ Calcola la maschera
   $Out \leftarrow I + \alpha \cdot M$                                 ▷ Amplifica i dettagli
  return  $Out$ 
end function

```

L'Unsharp Masking è un metodo semplice ed efficace per migliorare la nitidezza delle immagini, ma può introdurre artefatti e rumore, specialmente se il fattore di forza α è troppo elevato. Per evitare questi problemi, è possibile personalizzare la procedura di Unsharp Masking, ad esempio applicando un filtro di sfocatura diverso, regolando il fattore di forza o utilizzando tecniche di *edge-aware filtering* per preservare i dettagli [12].

In questo progetto è stata implementata una versione personalizzata dell'unsharp masking che combina la riduzione del rumore tramite DnCNN e l'uso di una maschera basata sul gradiente dell'immagine.

L'applicazione di DnCNN alle immagini lunari ha portato a risultati a prima vista deludenti. Infatti, l'immagine risultante era molto più sfocata rispetto all'originale.

Questo è dovuto al fatto che DnCNN è stato addestrato per rimuovere il rumore, ma non per preservare i dettagli. Per ovviare a questo problema, come riportato nell'[Algoritmo 2.9](#), è stata utilizzata una maschera basata sul gradiente dell'immagine originale, che enfatizza i dettagli e i bordi in sezioni dell'immagine con valori del gradiente maggiori, mentre applica una riduzione del rumore maggiore nelle zone più uniformi (con valori del gradiente minori).

Algoritmo 2.9 Unsharp Masking personalizzato:

Data un'immagine I , un, l'algoritmo restituisce l'immagine nitida Out .

```

function CUSTOM_UNSHARP( $I, \alpha, \beta, thr$ )
   $I_D \leftarrow \text{DENOISE\_IMAGE}(I)$            ▷ Riduzione del rumore con l'Algoritmo 2.7
   $I_D \leftarrow (I_D \times \alpha) + I \times (1 - \alpha)$    ▷ Attenuazione del denoising
   $G \leftarrow \text{Gradient}(I)$                    ▷ Calcolo del gradiente
   $D_M \leftarrow \text{Get\_Mask}(G, thr)$            ▷ Maschera di denoising
   $D_M \leftarrow \text{Gaussian\_Blur}(D_M)$          ▷ Sfoca leggermente la maschera
   $S_M \leftarrow 1 - D_M$                        ▷ Maschera di sharpening
   $D \leftarrow (I - I_D) \times \beta$                ▷ Estrazione dei dettagli
   $Out \leftarrow (I_D \times D_M) + (I + D) \times S_M$  ▷ Unsharpping con dettagli amplificati
  return  $Out$ 
end function

```

dove $\text{Get_Mask}(I, thr)$ è una funzione che restituisce una maschera M tale che:

$$M(i, j) = \begin{cases} 1 & \text{se } G(i, j) < thr \\ 0 & \text{altrimenti} \end{cases}$$

Il processo consiste dunque in diversi passaggi fondamentali. Si inizia con la riduzione del rumore, applicando il modello DnCNN all'immagine originale per ottenere una versione con rumore ridotto (I_D). Successivamente, si effettua un *blend* delle immagini, combinando l'immagine denoised e quella originale con un fattore α , per preservare parte dei dettagli originali. Il passo successivo prevede il calcolo del gradiente (G) dell'immagine originale per individuare le aree con dettagli significativi. A questo punto, vengono create delle maschere (D_M , S_M , rispettivamente *Denoise Mask*, *Sharpen Mask*) basate sul gradiente per distinguere le zone in cui applicare maggior denoising o amplificare i dettagli. Per l'amplificazione dei dettagli (D), si calcola la differenza tra l'immagine originale e quella denoised, amplificandola con un fattore β nelle zone con dettagli significativi. Infine, si combinano le immagini utilizzando le maschere, ottenendo un'immagine finale (Out) in cui i dettagli sono stati amplificati e il rumore ridotto.

2.3.3 Rimozione dello sfondo

Se il cielo negli scatti non è perfettamente uniforme, potrebbero essere presenti gradienti di luminosità o rumore che influenzano negativamente il processo di stacking. Per ovviare a questo problema, è possibile rimuovere lo sfondo dalle immagini, enfatizzando i dettagli e riducendo l'effetto di gradienti di luminosità.

Una tecnica molto semplice per rimuovere lo sfondo consiste nell'applicare una maschera all'immagine, mantenendo solo i pixel con intensità al di sotto di una soglia specificata. Questo metodo è efficace per rimuovere lo sfondo uniforme, ma potrebbe non funzionare correttamente se il cielo presenta gradienti di luminosità o rumore.

Per superare queste limitazioni, è possibile applicare una trasformazione non lineare all'immagine, che enfatizza i dettagli e riduce il rumore. In questo progetto è stata utilizzata una trasformazione logistica, che comprime i valori dei pixel in un intervallo limitato, enfatizzando i dettagli e riducendo il rumore. L'algoritmo per la rimozione dello sfondo con trasformazione logistica è riportato nell'[Algoritmo 2.10](#).

Algoritmo 2.10 - Rimozione dello sfondo:

Data un'immagine I , l'algoritmo restituisce l'immagine con lo sfondo rimosso Out .

```

1: function REMOVE_BACKGROUND( $I$ , threshold,  $\alpha$ )
2:    $mask \leftarrow I < \text{threshold}$                                  $\triangleright$  Maschera per lo sfondo
3:    $smooth\_image \leftarrow I$ 
4:    $smooth\_image[mask] \leftarrow I[mask] \times (1 / (1 + \exp(-\alpha \times (I[mask] - \text{threshold}))))$ 
5:   return  $smooth\_image$ 
6: end function

```

2.4 Stacking delle immagini

Lo **stacking** è una tecnica fondamentale nell'astrofotografia che consiste nel combinare più immagini dello stesso soggetto per migliorare il rapporto segnale-rumore e rivelare dettagli altrimenti invisibili.

2.4.1 Algoritmi di stacking

Questa tecnica sfrutta il fatto che il rumore è un processo stocastico, mentre il segnale è deterministico. Quando si combinano più immagini dello stesso soggetto, il segnale rimane costante, mentre il rumore si riduce proporzionalmente alla radice quadrata del numero di immagini combinate. Questo significa che, se si combinano N immagini, il rapporto segnale-rumore migliora di un fattore \sqrt{N} [8].

Esistono diversi metodi per combinare le immagini durante lo stacking. Nel progetto sono stati implementati quattro metodi di stacking:

- **Mean:** calcola la media pixel per pixel delle immagini. Questo metodo è utile per ridurre il rumore gaussiano e migliorare la qualità dell'immagine finale.

$$I_{stacked} = \frac{1}{N} \sum_{i=1}^N I_i$$

- **Median:** calcola il valore mediano pixel per pixel delle immagini. Questo metodo è utile per ridurre il rumore impulsivo e rimuovere gli outlier.

$$I_{stacked} = \text{median}(I_1, I_2, \dots, I_N)$$

- **Sigma clipping:** calcola la media pixel per pixel delle immagini, escludendo i pixel con valori al di fuori di un intervallo di soglia. Questo metodo è utile per ridurre l'effetto di outlier e artefatti.

$$I_{stacked}(x, y) = \frac{1}{N} \sum_{i=1}^N I_i(x, y) \quad \text{dove} \quad |I_i(x, y) - \text{mean}(I(x, y))| < k \cdot \sigma(x, y)$$

- **Weighted mean:** calcola la media pesata pixel per pixel delle immagini, assegnando pesi diversi in base ad una metrica di qualità, come la nitidezza.

$$I_{stacked} = \frac{\sum_{i=1}^N W_i \cdot I_i}{\sum_{i=1}^N W_i}$$

L'algoritmo che ha portato a risultati migliori è la media pesata, molto comune in astrofotografia [3], in particolare pesati in base alla nitidezza dell'immagine. L'algoritmo per il calcolo della media pesata è riportato nell'**Algoritmo 2.11**.

Algoritmo 2.11 – Stacking con media pesata:

Dato un insieme di immagini I e i pesi W , l'algoritmo restituisce l'immagine combinata Out .

```

1: function WEIGHTED_MEAN( $I, W$ )
2:    $N \leftarrow$  numero di immagini in  $I$ 
3:    $H, W_{\text{sum}} \leftarrow 0$ 
4:   for  $i \leftarrow 1$  to  $N$  do
5:      $H \leftarrow H + W[i] \times I[i]$                                  $\triangleright$  Somma pesata
6:      $W_{\text{sum}} \leftarrow W_{\text{sum}} + W[i]$                              $\triangleright$  Somma dei pesi
7:   end for
8:    $Out \leftarrow H / W_{\text{sum}}$                                         $\triangleright$  Normalizzazione
9:   return  $Out$ 
10: end function

```

Questo metodo è particolarmente efficace quando si combinano immagini con differenti livelli di nitidezza, contrasto e rumore, perchè, con una piccola modifica, permette anche di scartare una percentuale di immagini con qualità inferiore, riducendo l'effetto di artefatti e outlier.

2.5 Post-Processing delle immagini

Il post-processing delle immagini è l'ultima fase dell'elaborazione delle immagini lunari, necessaria per migliorare la qualità dell'immagine finale e renderla pronta per l'analisi e la visualizzazione. Questa fase comprende principalmente la regolazione del contrasto e della luminosità, la rimozione di artefatti e la correzione del bilanciamento del colore.

Questa fase viene generalmente svolta su programmi di editing professionali come photoshop o lightroom, ma è possibile automatizzare il processo utilizzando algoritmi di elaborazione delle immagini.

2.5.1 Miglioramento di nitidezza e contrasto

Per quanto riguarda l'aumento della nitidezza, l'[tradizionale](#)) rimane uno dei metodi più efficaci, in quanto enfatizza i dettagli e i bordi dell'immagine. Tuttavia, è importante regolare attentamente i parametri per evitare artefatti e rumore.

L'**equalizzazione dell'istogramma** è una tecnica che distribuisce uniformemente i valori dei pixel nell'intervallo dinamico dell'immagine, migliorando il contrasto e la luminosità. Tuttavia, l'equalizzazione standard può amplificare il rumore in aree omogenee e ridurre i dettagli in aree con contrasto già elevato. Inoltre, se l'immagine ha un contrasto limitato, l'equalizzazione dell'istogramma può produrre un'immagine sovraesposta o sottoposta, generando artefatti e perdita di dettagli.

Per superare queste limitazioni, è stata utilizzata la tecnica **CLACHE** (Contrast Limited Adaptive Histogram Equalization). CLACHE suddivide l'immagine in piccole regioni chiamate *tile* e applica l'equalizzazione dell'istogramma a ciascuna regione separatamente. Inoltre, limita il contrasto di ciascuna regione per evitare l'amplificazione del rumore e la sovraesposizione. L'algoritmo per l'equalizzazione dell'istogramma con CLACHE è riportato nell'[Algoritmo 2.12](#).

Algoritmo 2.12 - Equalizzazione dell'istogramma con CLACHE:

Data un'immagine I , l'algoritmo restituisce l'immagine equalizzata Out .

```

function ENHANCE_CONTRAST( $I$ , clip_limit, tile_grid_size)
     $I_{CLAHE} \leftarrow$  Inizializza un'immagine vuota
     $I_{LAB} \leftarrow$  Converti  $I$  in spazio colore LAB
     $L, A, B \leftarrow$  Estrai i canali L, A, B da  $I_{LAB}$ 
     $C \leftarrow$  Inizializza un'oggetto CLACHE con tile_grid_size e clip_limit
     $L_{CLAHE} \leftarrow C.apply\_CLAHE(L)$  ▷ Applica CLACHE al canale L
     $I_{CLAHE} \leftarrow$  Combina  $L_{CLAHE}$ ,  $A$  e  $B$  in un'immagine
    return  $I_{CLAHE}$ 
end function

```

Lo spazio LAB è un modello di colore che separa la luminosità (L) dal colore (A e B), rendendo più semplice l'applicazione di tecniche di miglioramento del contrasto e della luminosità. L'equalizzazione dell'istogramma viene applicata solo al canale L , che rappresenta la luminosità dell'immagine, mentre i canali A e B rimangono inalterati. Questo permette di migliorare il contrasto e la luminosità dell'immagine senza influenzare il colore.

Tramite `clip_limit` e `tile_grid_size` è possibile regolare il contrasto e la dimensione delle regioni su cui applicare l'equalizzazione dell'istogramma. Un `clip_limit` più alto aumenta il contrasto, mentre un `tile_grid_size` più piccolo permette di equalizzare dettagli più fini.

2.5.2 Bilanciamento del colore

Nel contesto dell'astrofotografia lunare, il bilanciamento del colore è generalmente meno critico rispetto ad altre forme di fotografia, poiché la luna appare preva-

lentemente in toni di grigio. Tuttavia, può essere utile per correggere tonalità dell'immagine e migliorare la qualità visiva, evidenziando lievi variazioni cromatiche dovute alla composizione minerale sulla superficie. Esistono diversi metodi per il bilanciamento del colore:

- **Correzione automatica del bilanciamento del bianco:** utilizza algoritmi di bilanciamento del bianco automatici per correggere il colore dell'immagine in base alla temperatura del colore. Questo metodo è utile per correggere il colore in modo rapido e semplice.
- **Correzione manuale del bilanciamento del bianco:** permette di regolare manualmente il bilanciamento del bianco dell'immagine, selezionando un punto neutro o una zona grigia come riferimento. Questo metodo è utile per ottenere un controllo preciso sul colore dell'immagine.
- **Correzione del colore selettiva:** permette di regolare selettivamente i toni, la saturazione e la luminosità di specifici colori nell'immagine. Questo metodo è utile per migliorare il colore e il contrasto di parti specifiche dell'immagine.

Per questo progetto, è stato adottato un metodo di bilanciamento del colore automatico basato sull'assunzione delle **Shades of Gray** (tonalità di grigio). Questo approccio presuppone che l'immagine abbia una distribuzione equilibrata delle tonalità di grigio, permettendo di correggere le deviazioni cromatiche tramite la normalizzazione dei canali di colore [4]. I passaggi principali sono i seguenti:

1. **Calcolo della media normalizzata per ciascun canale:** Per ogni canale di colore (RGB), si calcola la media dei valori dei pixel elevati a una potenza specificata (p). Questa operazione enfatizza i valori più alti, riducendo l'influenza dei pixel più scuri.

$$S[c] = \left(\frac{1}{M \times N} \sum_{x=1}^M \sum_{y=1}^N I(x, y, c)^p \right)^{1/p}$$

2. **Calcolo della media delle `norm_values`:** Si calcola la media delle medie normalizzate ottenute per ciascun canale. Questo valore rappresenta il target di normalizzazione per ogni canale.

$$\mu = \frac{1}{3} \sum_{c=1}^3 S[c]$$

3. **Calcolo dei fattori di scala:** Per ogni canale, si determina un fattore di scala che normalizza la media del canale rispetto alla media complessiva. Questo assicura che ciascun canale contribuisca equamente alla tonalità di grigio finale.

$$f[c] = \frac{\mu}{S[c]}$$

4. **Applicazione dei fattori di scala:** Si moltiplicano i valori dei pixel di ogni canale per il rispettivo fattore di scala, correggendo le deviazioni cromatiche.

$$Out(x, y, c) = I(x, y, c) \times f[c]$$

5. **Clipping dei valori:** Infine, si assicura che i valori dei pixel rimangano nel range $[0, 1]$ per evitare distorsioni cromatiche.

$$Out = \text{clip}(Out, 0, 1)$$

Questa tecnica garantisce che l'immagine risultante abbia una distribuzione equilibrata delle tonalità di grigio, correggendo automaticamente le deviazioni cromatiche senza richiedere interventi manuali. Lo pseudocodice che racchiude questi passaggi è riportato nell'[Algoritmo 2.13](#).

Algoritmo 2.13 - Shades of Gray - bilanciamento del colore:

Data un'immagine I , l'algoritmo restituisce l'immagine bilanciata Out .

```

1: function SHADES_OF_GRAY( $I, p = 6$ )
2:    $M, N \leftarrow$  dimensioni di  $I$ 
3:    $S \leftarrow$  inizializza vettore per le norme
4:   for  $c \leftarrow 1$  to 3 do                                     ▷ Per ogni canale RGB
5:      $S[c] \leftarrow \left( \frac{1}{M \times N} \sum_{x=1}^M \sum_{y=1}^N I(x, y, c)^p \right)^{1/p}$ 
6:   end for
7:    $\mu \leftarrow \frac{1}{3} \sum_{c=1}^3 S[c]$                                      ▷ Media delle norme
8:   for  $c \leftarrow 1$  to 3 do
9:      $f[c] \leftarrow \mu / S[c]$                                      ▷ Fattori di scala
10:    for  $x \leftarrow 1$  to  $M$  do
11:      for  $y \leftarrow 1$  to  $N$  do
12:         $Out(x, y, c) \leftarrow I(x, y, c) \times f[c]$              ▷ Applica bilanciamento
13:      end for
14:    end for
15:  end for
16:   $Out \leftarrow \text{clip}(Out, 0, 1)$                                ▷ Limita valori tra 0 e 1
17:  return  $Out$ 
18: end function

```

Capitolo 3

Implementazione

In questo capitolo verrà descritta l'implementazione del software sviluppato per l'elaborazione di immagini lunari. In particolare, verranno presentate le scelte progettuali e le soluzioni adottate per la realizzazione delle funzionalità di calibrazione, allineamento, pre-processing, stacking e post-processing affrontate nel [Capitolo 2](#). I frammenti di codice riportati in questo capitolo rappresentano solo le parti più significative del software; in molti casi sono state omesse linee di controllo o operazioni di supporto per ragioni di brevità. Per una visione completa si rimanda al repository [GitHub](#) del progetto.

3.1 Architettura del software

Il software è stato sviluppato in linguaggio Python, utilizzando principalmente le librerie OpenCV e NumPy per la manipolazione delle immagini.

Il progetto segue una struttura modulare, con un file principale `main.py` che coordina l'esecuzione delle funzioni dei vari moduli. L'inizializzazione delle variabili e dei parametri avviene nel file `config.py`, che contiene la definizione di alcuni flag (come `DEBUG`), i percorsi delle cartelle utilizzate (`input`, `output`, `bias`, `dark`, `flat`, ecc.) e alcune funzioni per l'inizializzazione delle variabili e delle metriche. La pipeline di elaborazione delle immagini è descritta nel file `process.py`, che si occupa di chiamare le funzioni di calibrazione (`calibration.py`), allineamento (`align.py`), denoising (`denoise.py`), stacking (`stacking.py`) e, infine, le operazioni di pre-processing e post-processing (`enhancement.py`).

```
src/
|-- align.py
|-- analysis.py
|-- calibration.py
|-- config.py
|-- denoise.py
|-- enhancement.py
|-- grid_search.py
|-- image.py
|-- main.py
|-- metrics.py
|-- model.pth
|-- process.py
|-- stacking.py
'-- utils.py
```

Il modulo `utils.py` contiene alcune funzioni di ausiliarie (ad esempio, per creare cartelle di destinazione se non esistenti). All'interno di `image.py` sono presenti funzioni di utilità per la manipolazione delle immagini, come il caricamento, il salvataggio e la visualizzazione. Inoltre, nel file `metrics.py` sono presenti funzioni per il calcolo di metriche come il contrasto e la luminosità di un'immagine, e metriche di qualità come BRISQUE, NIQE e LIQE, affrontate più nel dettaglio nel [Capitolo 4](#)

Di seguito vengono descritte più nel dettaglio le implementazioni delle funzionalità principali del software.

3.1.1 Calibrazione

La fase di calibrazione è stata implementata nel file `calibration.py`, dove sono definiti i metodi per calcolare i *master bias*, *dark* e *flat*, e quello per applicare la calibrazione alle immagini.

I metodi per calcolare i master seguono la procedura descritta negli algoritmi [2.1](#), [2.2](#) e [2.3](#) e sono riportati di seguito:

```

1  [label={lst:calculate_masters}]
2  # Function to calculate the master bias
3  def calculate_master_bias(bias):
4      # Calculate the mean
5      return np.mean(bias, axis=0)
6
7  # Function to calculate the master dark
8  def calculate_master_dark(dark, master_bias=None):
9      if master_bias is None: master_bias = np.zeros_like(flat[0])
10     # Subtract the master bias, then calculate the mean
11     return np.mean(dark - master_bias, axis=0)
12
13 # Function to calculate the master flat
14 def calculate_master_flat(flat, master_bias=None, master_dark=
    None):
15     if master_bias is None: master_bias = np.zeros_like(flat[0])
16     if master_dark is None: master_dark = np.zeros_like(flat[0])
17     # Subtract master bias and dark, then calculate the mean
18     master_flat = np.mean(flat - master_bias - master_dark, axis=0)
19     # Normalize the master flat
20     mean_flat = np.mean(master_flat)
21     if mean_flat != 0: master_flat /= mean_flat
22     return master_flat

```

Queste funzioni calcolano rispettivamente i master bias, dark e flat, utilizzando la media su tutti i frame di calibrazione disponibili. Se i master bias o dark non sono disponibili, vengono inizializzati a matrici di zeri della stessa dimensione delle immagini di calibrazione. Nel caso di `calculate_master_flat`, dopo aver sottratto il master bias e il master dark, si normalizza il master flat dividendo per la sua media, a condizione che questa sia diversa da zero.

Le funzioni tre funzioni sopra definite vengono chiamate da una funzione ausiliaria (`calculate_masters`) che, prima di calcolare i masters, verifica se i frame di calibrazione sono stati caricati correttamente e se hanno le stesse dimensioni delle

immagini da calibrare; in caso contrario, viene sollevata un'eccezione. Inoltre, se per un tipo di frames di calibrazione il numero di immagini disponibili è inferiore a una certa soglia `MIN_CALIBRATION` specificata nel file di configurazione, il master corrispondente non viene calcolato e viene stampato un messaggio di avviso, poiché con troppi pochi frames il calcolo potrebbe non essere affidabile e introdurre nuovi artefatti.

Questa fase risulta computazionalmente onerosa, in quanto richiede il caricamento di tutti i frame di calibrazione in memoria, e il calcolo dei master implica l'elaborazione di ogni singolo pixel di ogni frame. Per questo motivo, è stata implementata la possibilità di salvare i master calcolati su file, in modo da poterli riutilizzare senza doverli ricalcolare ogni volta.

Il metodo per calibrare una singola immagine fa riferimento all'[Algoritmo 2.4](#). L'implementazione è la seguente:

```

1  # Function to calibrate a single image
2  def calibrate_single_image(image, master_bias = None, master_dark
    = None, master_flat = None):
3      if master_bias is None and master_dark is None and
        master_flat is None:
4          print("No calibration masters available: skipping")
5          return image
6      if master_bias is None: master_bias = np.zeros_like(flat[0])
7      if master_dark is None: master_dark = np.zeros_like(flat[0])
8      if master_flat is None: master_flat = np.ones_like(flat[0])
9      # Calibrate the image
10     calibrated = (image - master_bias - master_dark)/master_flat
11     calibrated = np.clip(calibrated, 0, 1) # Clip to valid range
12     return calibrated

```

Nella funzione `calibrate_single_image`, si procede alla calibrazione dell'immagine sottraendo il master bias e il master dark, e dividendo per il master flat. Se uno dei master non è disponibile, viene inizializzato a una matrice di zeri (per bias e dark) o di uni (per flat) della stessa dimensione dell'immagine. In questo modo, sia nel calcolo dei master, sia nella calibrazione di una singola immagine, è possibile non utilizzare uno o più set di frame di calibrazione nel caso in cui non siano disponibili. Infine, si utilizza la funzione `np.clip` di NumPy per assicurarsi che i valori dell'immagine risultante siano compresi nell'intervallo $[0, 1]$.

3.1.2 Allineamento

La fase di allineamento delle immagini è stata implementata nel file `align.py`. Qui sono definite le funzioni necessarie per allineare le immagini utilizzando algoritmi di *feature detection* e *feature matching* come ORB, SIFT e SURF, e RANSAC per il calcolo della trasformazione omografica, come già affrontato nella [Sezione 2.2.1](#). L'allineamento è fondamentale per compensare eventuali spostamenti o rotazioni tra gli scatti, garantendo una sovrapposizione precisa delle immagini durante la fase di *stacking*.

Per migliorare i risultati dell'allineamento, le immagini passano prima attraverso

una funzione di pre-processing che ne ottimizza la qualità prima dell'estrazione delle feature. Questa funzione è stata implementata nel file `enhancement.py` e svolge essenzialmente tre compiti:

1. **Convertire la foto in scala di grigi** (se non lo è già): riducendo l'immagine a un solo canale, si alleggerisce il carico computazionale per l'estrazione dei keypoint e dei descrittori, oltre a rendere il processo più robusto a variazioni cromatiche.
2. **Rimozione dello sfondo**: per facilitare l'estrazione delle features, si è scelto di rimuovere lo sfondo dell'immagine, che potrebbe contenere rumore o dettagli non rilevanti. Questo passaggio è stato implementato tramite la funzione `soft_threshold` (Algoritmo 2.10), che applica una sogliatura morbida all'immagine, mantenendo solo i pixel con intensità superiore ad una certa soglia `thr`.
3. **Miglioramento del contrasto**: questo passaggio, effettuato tramite l'applicazione dell'Algoritmo 2.12, è utile per migliorare la qualità dell'immagine, rendendo più nitidi i dettagli e facilitando l'estrazione delle features.

Inoltre, poiché gli algoritmi di OpenCV richiedono immagini in formato a 8 bit, è stata implementata una funzione `to_8bit` che converte l'immagine in un formato a 8 bit, normalizzando i valori dei pixel nell'intervallo $[0, 255]$.

L'implementazione dell'algoritmo è riportata di seguito, mentre le implementazioni di `soft_threshold` e `enhance_contrast` sono riportate nella Sezione 3.1.3.

```

1  def pre_align_enhance(image, clip_limit = 0.8, tile_grid_size =
    (20,20), thr = 0.05):
2      # Convert image to grayscale
3      enhanced = image.copy()
4      if len(image.shape) == 3:
5          enhanced = cv2.cvtColor(enhanced, cv2.COLOR_RGB2GRAY)
6      # Remove background using a threshold
7      enhanced = soft_threshold(enhanced, thr)
8      # Enhance contrast using CLACHE
9      enhanced = enhance_contrast(enhanced, clip_limit,
        tile_grid_size)
10     enhanced = to_8bit(enhanced)
11     return enhanced

```

La funzione principale per l'allineamento è `align_images`. Tale funzione prende in input una lista di immagini `images` e restituisce una lista di immagini allineate `aligned`. I due parametri opzionali `method` e `nfeatures` permettono di specificare l'algoritmo di feature detection da utilizzare (tra `orb`, `sift` e `surf`) e il numero di features da estrarre. I valori di default sono rispettivamente `orb` e 5000.

La funzione `align_images` inizialmente crea un oggetto `detector` e un oggetto `matcher` in base all'algoritmo scelto per effettuare l'estrazione e il matching delle feature. Inizializza la lista `aligned_images` con la prima immagine della lista, che fungerà da immagine di riferimento. Si procede quindi a migliorare quest'ultima

tramite la funzione `pre_align_enhance`, sopra riportata. Si estraggono quindi i keypoint e i descrittori dell'immagine di riferimento, che saranno necessari per calcolare l'allineamento di tutte le altre immagini utilizzando la funzione `align_image`. Infine, si restituisce la lista di immagini allineate.

L'implementazione è la seguente:

```

1  def align_images(images, method='orb', nfeatures=5000):
2      # Choose the feature detection algorithm
3      if method == 'orb':
4          detector = cv2.ORB_create(nfeatures=nfeatures)
5      elif method == 'sift':
6          detector = cv2.SIFT_create(nfeatures=nfeatures)
7      elif method == 'surf':
8          detector = cv2.xfeatures2d.SURF_create()
9      # Choose the norm for the matcher
10     norm = cv2.NORM_HAMMING if method == 'orb' else cv2.NORM_L2
11     # Create a matcher object
12     matcher = cv2.BFMatcher.create(norm)
13
14     # The first image is the reference image
15     ref_image = images[0]
16     aligned_images = [ref_image]
17
18     # Enhance the reference image to improve alignment
19     enhanced_ref = pre_align_enhance(ref_image)
20
21     # Detect keypoints and descriptors for the reference image
22     ref_kp, ref_des = detector.detectAndCompute(enhanced_ref,
23                                                None)
24
25     ref_shape = (ref_image.shape[1], ref_image.shape[0])
26
27     # Align each image to the reference image
28     for idx, image in enumerate(images[1:]):
29         aligned_image = align_image(image, enhanced_ref, ref_kp,
30                                    ref_des, ref_shape, detector, matcher)
31         aligned_images.append(aligned_image)
32
33     return aligned_images

```

La funzione `align_image` prende in input l'immagine da allineare `image`, i keypoint `ref_kp` e i descrittori `ref_des` dell'immagine di riferimento, l'oggetto `detector` e il `matcher`, e restituisce l'immagine allineata con quella di riferimento.

Per allineare l'immagine, si procede come segue:

1. Si applica la funzione `pre_align_enhance` all'immagine da allineare, per migliorarne la qualità.
2. Si estraggono i keypoint e i descrittori dell'immagine da allineare utilizzando il `detector` scelto.
3. Si effettua il matching dei descrittori tra l'immagine di riferimento e quella da allineare.

4. Si applica il *ratio test* per filtrare i buoni match.

$$\text{matches} = \{m \in \text{matches} \mid m.\text{distance} < 0.75 \cdot n.\text{distance}\}$$

5. Si calcola l'omografia tra i keypoints dell'immagine di riferimento e quelli dell'immagine da allineare. Nel mio caso ho scelto di utilizzare interpolazione INTER_LANCZOS4, molto utilizzata in astrofotografia per la sua capacità di preservare i dettagli.
6. Si applica la trasformazione omografica all'immagine da allineare.

L'implementazione di `align_image` è riportata qui sotto:

```

1  def align_image(image, ref_kp, ref_des, detector, matcher):
2      # Enhance the image for alignment process
3      aligned_image = pre_align_enhance(image)
4
5      # Find keypoints and descriptors for the image
6      kp, des = detector.detectAndCompute(aligned_image, None)
7
8      # Match the descriptors and filter the matches
9      matches = matcher.knnMatch(ref_des, des, k=2)
10     matches = [m for m, n in matches if m.distance < 0.75 * n.
11                distance]
12
13     # Compute the homography
14     ref_pts = np.float32([ref_kp[m.queryIdx].pt for m in matches
15                           ]).reshape(-1, 1, 2)
16     img_pts = np.float32([kp[m.trainIdx].pt for m in matches]).
17     reshape(-1, 1, 2)
18     H, _ = cv2.findHomography(img_pts, ref_pts, cv2.RANSAC, 10.0,
19                               maxIters=3000, confidence=0.995)
20     shape = image.shape[1], image.shape[0]
21
22     # Warp the original image using the final homography
23     aligned_image = cv2.warpPerspective(image, H, shape, flags=
24                                         cv2.INTER_LANCZOS4)
25     return aligned_image

```

I parametri `maxIters` e `confidence` nell'invocazione di `cv2.findHomography` permettono di regolare il numero massimo di iterazioni e la confidenza richiesta per il calcolo della trasformazione omografica. Questi parametri sono stati scelti in modo da ottenere un buon compromesso tra accuratezza e velocità di esecuzione.

3.1.3 Pre-processing

La fase di pre-processing è stata implementata nel file `enhancement.py`. Questa fase comprende operazioni volte a migliorare la qualità delle immagini calibrate e allineate, preparandole per il processo di stacking. Le operazioni implementate sono:

- **Ritaglio dell'immagine** per centrare il soggetto ed eliminare bordi esterni indesiderati

- **Riduzione del rumore (denoising)** tramite la rete neurale DnCNN;
- **Miglioramento della nitidezza** attraverso l'uso di tecniche di Unsharp Mask personalizzate;
- **Rimozione dello sfondo** tramite la sogliatura morbida;

Di seguito sono descritte le funzioni implementate per effettuare tali operazioni.

Riduzione del rumore e Unsharp Masking con DnCNN

Come descritto nel [Sezione 2.3.1](#) e nel [Sezione 2.3.2](#), è stata implementata una versione personalizzata dell'Unsharp Masking che combina la riduzione del rumore tramite DnCNN e l'utilizzo di una maschera basata sul gradiente dell'immagine.

Iniziamo col descrivere l'operazione di denoising con il modello preaddestrato. Per importarlo nel progetto è stato necessario scaricare il modello preaddestrato e salvarlo in un file `.pth`. Il modello è stato poi caricato utilizzando la libreria `torch` e la funzione `torch.load`. Il modello è stato quindi utilizzato per ridurre il rumore delle immagini, come mostrato nel seguente codice.

```

1  DNCNN_MODEL_PATH = './src/model.pth'
2
3  class DnCNN(nn.Module):
4      def __init__(self, depth=17, n_channels=64, image_channels=1):
5          super().__init__()
6          self.dncnn = nn.Sequential(
7              nn.Conv2d(image_channels, n_channels, 3, padding=1),
8              nn.ReLU(inplace=True),
9              *[layer_block(n_channels) for _ in range(depth-2)],
10             nn.Conv2d(n_channels, image_channels, 3, padding=1)
11         )
12         self._initialize_weights()
13
14     def forward(self, x):
15         return x - self.dncnn(x)
16
17     def _initialize_weights(self):
18         for m in self.modules():
19             if isinstance(m, nn.Conv2d):
20                 init.orthogonal_(m.weight)
21                 if m.bias is not None: init.constant_(m.bias, 0)
22             elif isinstance(m, nn.BatchNorm2d):
23                 init.constant_(m.weight, 1); init.constant_(m.bias, 0)
24
25     def layer_block(n_channels):
26         return nn.Sequential(
27             nn.Conv2d(n_channels, n_channels, 3, padding=1, bias=False),
28             nn.BatchNorm2d(n_channels, eps=1e-4, momentum=0.95),
29             nn.ReLU(inplace=True)
30         )
31
32     def model_init(model_path=DNCNN_MODEL_PATH):
33         model = torch.load(model_path)
34         model.eval()
35         return model

```

La classe `DnCNN` implementa una rete neurale convoluzionale profonda progettata specificamente per la riduzione del rumore nelle immagini. Questa rete, denominata **Denoising Convolutional Neural Network (DnCNN)**, è caratterizzata da una struttura composta da 17 strati convoluzionali, progettati per stimare e rimuovere il rumore presente nell'immagine di input, come visto nella [Sezione 2.3.1](#).

La funzione `model_init` si occupa di caricare il modello preaddestrato da un file `.pth` e restituirlo pronto per l'elaborazione.

Per garantire la compatibilità con il modello `DnCNN`, le immagini devono essere trasformate in tensori e normalizzate. Dopo il denoising, è necessario riconvertire i tensori in immagini nel formato `np.float32`. Queste operazioni sono state incapsulate in due funzioni:

1. `prepare_tensor_for_model`: converte un'immagine nel formato tensoriale e la normalizza.
2. `convert_tensor_to_image`: riconverte il tensore prodotto dal modello in un'immagine `np.float32` utilizzabile nei flussi di elaborazione successivi.

L'implementazione di queste funzioni non è riportata per questioni di brevità.

La funzione per effettuare il denoising, basata sull'[Algoritmo 2.7](#) è quindi la seguente:

```

1  def perform_denoising(model, image):
2      shape = image.shape
3
4      if len(shape) < 3:
5          image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
6          channels = list(cv2.split(color.rgb2lab(image)))
7
8          # convert image to tensor
9          l = prepare_tensor_for_model(channels[0], 100).to(device)
10
11         if torch.cuda.is_available(): device = torch.device('cuda')
12         else: torch.device('cpu')
13         model = model.to(device).eval()
14         with torch.no_grad():
15             denoised_l = model(l).cpu()
16
17         # convert tensor to image
18         denoised_l = convert_tensor_to_image(denoised_l, 100)
19         channels[0] = denoised_l
20
21         # Merge the channels
22         denoised = cv2.merge(channels)
23         denoised = color.lab2rgb(denoised)
24
25         if len(shape) < 3:
26             denoised = cv2.cvtColor(denoised, cv2.COLOR_RGB2GRAY)
27
28         return denoised

```

3.1.4 Stacking

3.1.5 Post-processing

- **Miglioramento della nitidezza** utilizzando l'algoritmo Unsharp Mask tradizionale;
- **Miglioramento del contrasto** applicando l'algoritmo CLAHE;
- **Bilanciamento del colore** per correggere eventuali dominanti cromatiche;

3.2 Sfide affrontate e soluzioni adottate

- Alto utilizzo di RAM
- No reference image
- Sfondo non completamente nero

Capitolo 4

Valutazione dei risultati e metriche di qualità

4.1 Metriche di valutazione con riferimento

4.1.1 SSIM (Structural Similarity Index Measure)

4.1.2 SNR (Signal-to-Noise Ratio)

4.2 Metriche di valutazione senza riferimento

4.2.1 NIQE (Naturalness Image Quality Evaluator)

4.2.2 BRISQUE (Blind/Referenceless Image Spatial Quality Evaluator)

4.2.3 LIQE (Language-Image Quality Evaluator)

4.2.4 Motivazione della scelta di LIQE come metrica di riferimento

4.3 Analisi e miglioramenti ottenuti

4.3.1 Effetti della calibrazione

4.3.2 Impatto del denoising

4.3.3 Benefici dello stacking

4.3.4 Miglioramenti con sharpening e contrasto

Conclusions

TODO

Acknowledgements

TODO

Bibliografia

- [1] Herbert Bay et al. «Speeded-Up Robust Features (SURF)». In: *Computer Vision and Image Understanding* 110.3 (2008). Similarity Matching in Computer Vision and Multimedia, pp. 346–359. ISSN: 1077-3142. DOI: <https://doi.org/10.1016/j.cviu.2007.09.014>. URL: <https://www.sciencedirect.com/science/article/pii/S1077314207001555>.
- [2] Tanmoy Bhowmik et al. *Image Processing and Analysis of Multiple Wavelength Astronomical Data Using Python Tools*. 2024. DOI: 10.48550/ARXIV.2410.06573. URL: <https://arxiv.org/abs/2410.06573>.
- [3] Roger N. Clark. «Image Stacking Methods». In: (). URL: <https://clarkvision.com/articles/image-stacking-methods/>.
- [4] Graham D. Finlayson et al. «Shades of Gray and Colour Constancy». In: *Color and Imaging Conference* 12.1 (2004), pp. 37–37. DOI: 10.2352/CIC.2004.12.1.art00008. URL: <https://library.imaging.org/cic/articles/12/1/art00008>.
- [5] Martin A. Fischler et al. «Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography». In: *Commun. ACM* 24.6 (giu. 1981), 381–395. ISSN: 0001-0782. DOI: 10.1145/358669.358692. URL: <https://doi.org/10.1145/358669.358692>.
- [6] Jonathan P. Gardner et al. «The James Webb Space Telescope». In: *Space Science Reviews* 123.4 (apr. 2006), 485–606. ISSN: 1572-9672. DOI: 10.1007/s11214-006-8315-7. URL: <http://dx.doi.org/10.1007/s11214-006-8315-7>.
- [7] Tom C Ireland. «Improving Photometry and Astrophotography by Eliminating Dark Frames and Flat Fields». In: (2019).
- [8] Peter Kurczynski et al. «A simultaneous stacking and deblending algorithm for astronomical images». In: *The Astronomical Journal* 139.4 (2010), p. 1592.
- [9] Matthew D. Lallo. «Experience with the Hubble Space Telescope: 20 years of an archetype». In: *Optical Engineering* 51.1 (feb. 2012), p. 011011. ISSN: 0091-3286. DOI: 10.1117/1.oe.51.1.011011. URL: <http://dx.doi.org/10.1117/1.OE.51.1.011011>.
- [10] David G. Lowe. «Distinctive Image Features from Scale-Invariant Keypoints». In: *International Journal of Computer Vision* 60.2 (nov. 2004), 91–110. ISSN: 0920-5691. DOI: 10.1023/b:visi.0000029664.99615.94. URL: <http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94>.

- [11] Diganta Misra et al. *Advanced Image Processing for Astronomical Images*. 2018. DOI: 10.48550/ARXIV.1812.09702. URL: <https://arxiv.org/abs/1812.09702>.
- [12] S.K. Mitra et al. «A new class of nonlinear filters for image enhancement». In: *[Proceedings] ICASSP 91: 1991 International Conference on Acoustics, Speech, and Signal Processing*. 1991, 2525–2528 vol.4. DOI: 10.1109/ICASSP.1991.150915.
- [13] Moon - Thierry Legault — *astrophoto.fr*. <http://www.astrophoto.fr/moon.html>.
- [14] NASA. *NASA's James Webb Space Telescope*. <https://webbtelescope.org/>. 2024.
- [15] Nikhil Padmanabhan et al. «An Improved Photometric Calibration of the Sloan Digital Sky Survey Imaging Data». In: *The Astrophysical Journal* 674.2 (feb. 2008), 1217–1233. ISSN: 1538-4357. DOI: 10.1086/524677. URL: <http://dx.doi.org/10.1086/524677>.
- [16] Paolo Padovani et al. «The Extremely Large Telescope». In: (2023). DOI: 10.1080/00107514.2023.2266921. eprint: [arXiv:2312.04299](https://arxiv.org/abs/2312.04299).
- [17] Rahul Raguram et al. «A Comparative Analysis of RANSAC Techniques Leading to Adaptive Real-Time Random Sample Consensus». In: *Computer Vision – ECCV 2008*. A cura di David Forsyth et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 500–513. ISBN: 978-3-540-88688-4.
- [18] Ethan Rublee et al. «ORB: An efficient alternative to SIFT or SURF». In: *2011 International Conference on Computer Vision*. IEEE, nov. 2011, 2564–2571. DOI: 10.1109/iccv.2011.6126544. URL: <http://dx.doi.org/10.1109/ICCV.2011.6126544>.
- [19] William P. Sheehan et al. *Epic Moon*. Willmann-Bell, Inc., 2001. ISBN: 9780943396725.
- [20] Chris Solomon et al. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. John Wiley & Sons, 2011.
- [21] Nick Strobel. *Naked Eye Astronomy by Nick Strobel*. <https://people.umass.edu/wqd/strobel/nakedeye/nakedeyc.htm>. 2024.
- [22] *The History of Astrophotography* / High Point Scientific — *highpointscientific.com*. <https://www.highpointscientific.com/astrometry-hub/post/astro-photography-guides/history-of-astrophotography>.
- [23] *The Moon's Orbit and Rotation – Moon: NASA Science* — *moon.nasa.gov*. <https://moon.nasa.gov/resources/429/the-moons-orbit-and-rotation/>.
- [24] Sebastian Täubert. *Bias, Flats, Darks, Darkflats*. 2024. URL: <https://astrobasics.de/en/basics/bias-flats-darks-darkflats/>.
- [25] Xin Wang et al. «Noise Attenuation for CSEM Data via Deep Residual Denoising Convolutional Neural Network and Shift-Invariant Sparse Coding». In: *Remote Sensing* 15.18 (set. 2023), p. 4456. ISSN: 2072-4292. DOI: 10.3390/rs15184456. URL: <http://dx.doi.org/10.3390/rs15184456>.

- [26] Kai Zhang et al. «Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising». In: *IEEE Transactions on Image Processing* 26.7 (lug. 2017), 3142–3155. ISSN: 1941-0042. DOI: 10.1109/tip.2017.2662206. URL: <http://dx.doi.org/10.1109/TIP.2017.2662206>.