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

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

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 potettero 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 2.3.2 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 fotocamra:

  - **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

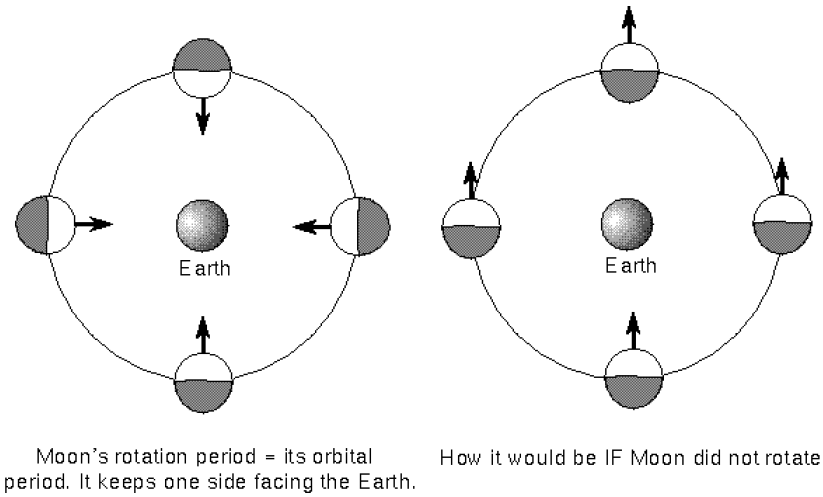

Figura 1.2. Dimostrazione grafica del moto di rotazione e rivoluzione della Luna [21]

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 le 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

function CALCULATE_MASTER_BIAS(B_f)
    $N \leftarrow$ numero di bias frames
    $M_b \leftarrow 0$
    for $i \leftarrow 1$ to N **do**
        $M_b \leftarrow M_b + \frac{B_f[i]}{N}$ ▷ Media dei bias frames
    end for
    return M_b
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 fames. 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.

function CALCULATE_MASTER_DARK(D_f, M_b)
    $N \leftarrow$ numero di dark frames
    $M_d \leftarrow 0$
    for $i \leftarrow 1$ to N **do**
        $D_c \leftarrow D_f[i] - M_b$ ▷ rimozione del bias
        $M_d \leftarrow M_d + \frac{D_c}{N}$ ▷ media dei dark frames
    end for
    return M_d
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 **rumoe di vignettatura** introdotto dal sistema ottico, ovvero la diminuzione dell'illuminazione verso i bordi dell'immagine.

Per correggere queste imperfezioni si calcula 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.

function CALCULATE_MASTER_FLAT(F_f, M_b, M_d)
    $N \leftarrow$ numero di flat frames
    $M_f \leftarrow 0$
    for $i \leftarrow 1$ to N **do**
        $F_c \leftarrow F_f[i] - M_b - M_d$ ▷ rimozione di bias e dark
        $M_f \leftarrow M_f + \frac{F_c}{N}$ ▷ media dei flat frames
    end for
    $M_f \leftarrow \frac{M_f}{\text{mean}(M_f)}$ ▷ normalizzazione
    return M_f
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: $\quad Out \leftarrow \dfrac{Img - M_b - M_d}{M_f}$
3: $\quad$ **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.

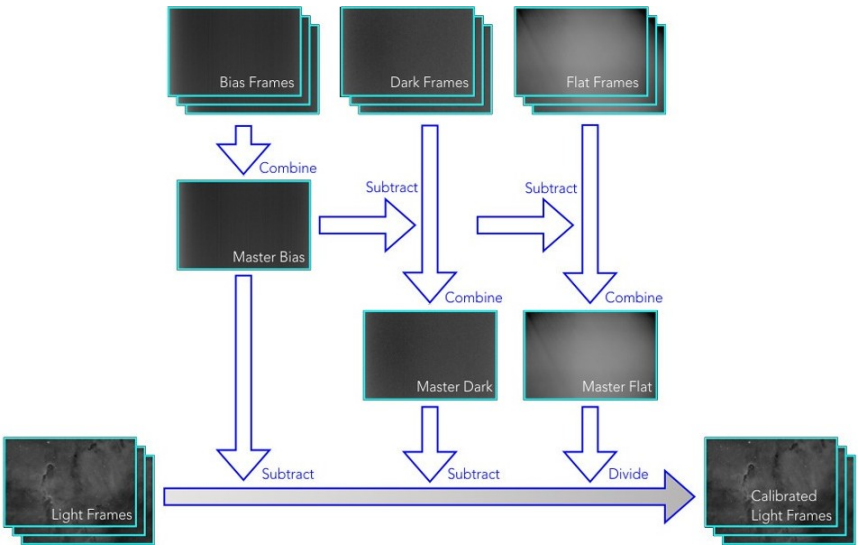

Figura 2.1. Illustrazione schematica dell'uso di immagini di calibrazione [24]

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 contestro 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.

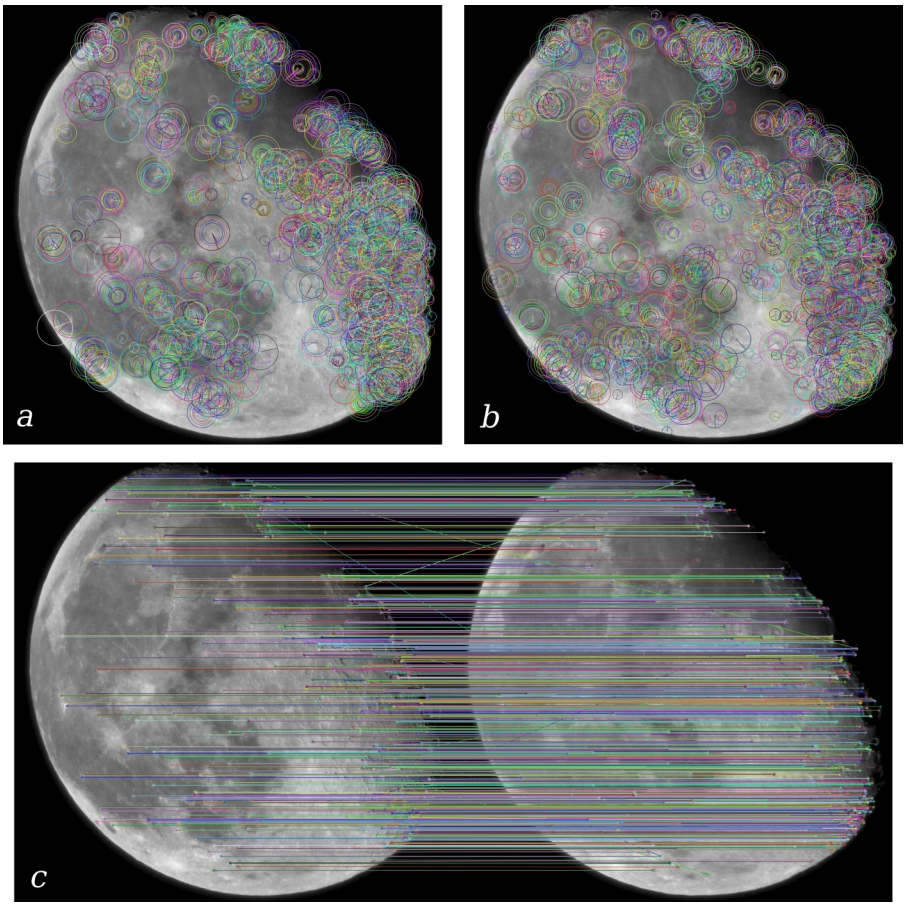

Figura 2.2. Applicazione di ORB (**a**-**b**) e RANSAC (**c**) su due scatti da me acquisiti.
a-**b**: visualizzazione di 5000 *keypoint* estratti da due immagini
c: visualizzazione dei *matches* tra i keypoints delle due immagini

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 connessività 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':

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

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.

```
function ESTIMATE_HOMOGRAPHY(matches, k, thr)
    H* ← null                                   ▷ Miglior omografia stimata
    score* ← 0                                  ▷ Miglior punteggio
    for i ← 1 to k do
        R ⊆ matches con |R| = n                 ▷ Seleziona n corrispondenze casuali
        H ← Stima l'omografia usando R
        score ← 0
        for m ∈ matches do
            if errore(H, m) < thr then          ▷ Errore come distanza euclidea
                score ← score + 1               ▷ Conta gli inlier
            end if
        end for
        if score > score* then                  ▷ Seleziona il modello migliore
            Aggiorna H* e score*
        end if
    end for
    return H*
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 3.2.2).

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

function ALIGN_IMAGES(I)
    Seleziona un riferimento r ▷ Tipicamente la più nitida
    $f \leftarrow$ un algoritmo tra ORB, SIFT, SURF
    $k_r, d_r \leftarrow$ f.calculate_descriptors(r) ▷ Calcola keypoints e descrittori di r
    for ogni i in I **do**
        $k, d \leftarrow$ f.calculate_descriptors(i) ▷ Calcola keypoints e descrittori di i
        $m \leftarrow$ match_descriptors(d_r, d) ▷ Trova i match tra i descrittori
        $H \leftarrow$ ESTIMATE_HOMOGRAPHY (k_r, k, m) ▷ Calcola l'omografia
        $a \leftarrow$ apply_transformation(i, H) ▷ Applica l'omografia
        Aggiungi a a A
    end for
    return A
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 Ritaglio delle immagini

Nel processo di pre-processing, dopo l'allineamento delle immagini, è utile ritagliare le immagini per focalizzarsi sulla Luna, riducendo la dimensione complessiva dell'immagine. Questo passaggio ha un duplice vantaggio: riduce notevolmente il carico

computazionale delle operazioni successive e migliora l'aspetto estetico del risultato finale eliminando porzioni inutili di cielo.

Il processo di ritaglio automatico si basa sull'identificazione del soggetto principale (la Luna) attraverso una sogliatura binaria, seguita dal calcolo di un'area di ritaglio quadrata centrata sul soggetto. L'algoritmo è illustrato in dettaglio nell'Algoritmo 2.7.

Algoritmo 2.7 - `Ritaglio automatico delle immagini`:
Dato un insieme di immagini allineate I e un margine m, l'algoritmo restituisce l'insieme di immagini ritagliate Out.

function CROP_TO_CENTER(I, m)
  $F \leftarrow I[0]$ ▷ Prima immagine come riferimento
  $G \leftarrow$ Converti F in scala di grigi
  $T \leftarrow$ Applica soglia binaria a G ▷ Separa il soggetto dallo sfondo
  $C \leftarrow$ Trova i contorni in T
  $B \leftarrow$ Calcola il rettangolo delimitatore del contorno più grande
  $x_c, y_c \leftarrow$ Calcola il centro di B
  $s \leftarrow \max(\text{larghezza}(B), \text{altezza}(B)) + 2m$ ▷ Dimensione lato
  $x_1 \leftarrow \max\left(x_c - \frac{s}{2}, 0\right)$ ▷ Assicura che sia entro i limiti
  $y_1 \leftarrow \max\left(y_c - \frac{s}{2}, 0\right)$
  $x_2 \leftarrow \min\left(x_1 + s, \text{larghezza}(F)\right)$
  $y_2 \leftarrow \min\left(y_1 + s, \text{altezza}(F)\right)$
  $Out \leftarrow []$
  for ogni immagine i in I **do**
    $c \leftarrow i[y_1 : y_2, x_1 : x_2]$ ▷ Ritaglia l'immagine
    Aggiungi c a Out
  end for
  return Out
end function

È importante notare che, dopo aver calcolato le coordinate di ritaglio (x_1, y_1) e (x_2, y_2), si effettuano controlli per assicurarsi che queste rientrino nei limiti dell'immagine originale, evitando così errori dovuti a coordinate fuori dai bordi.

Per ritagliare un insieme di immagini, si applica la funzione `crop_to_center` con un margine m appropriato. In questo progetto, si è utilizzato un margine compreso tra 10 e 25 pixel, che rappresenta un buon compromesso tra l'inclusione di dettagli periferici e la riduzione di porzioni inutili dell'immagine.

Questa funzione richiede che le immagini siano allineate, poiché le informazioni sul ritaglio vengono calcolate sulla prima immagine, che funge da riferimento, e le altre immagini vengono ritagliate utilizzando le stesse coordinate. Questo assicura che il soggetto principale sia centrato in tutte le immagini ritagliate, facilitando le operazioni successive come lo stacking.

2.3.2 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:

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

  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 Gaussiane per ogni pixel dell'immagine.

Questi filtri riducono il rumore, ma tendono a sfocare l'immagine e a ridurre la nitidezza dei dettagli. 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.

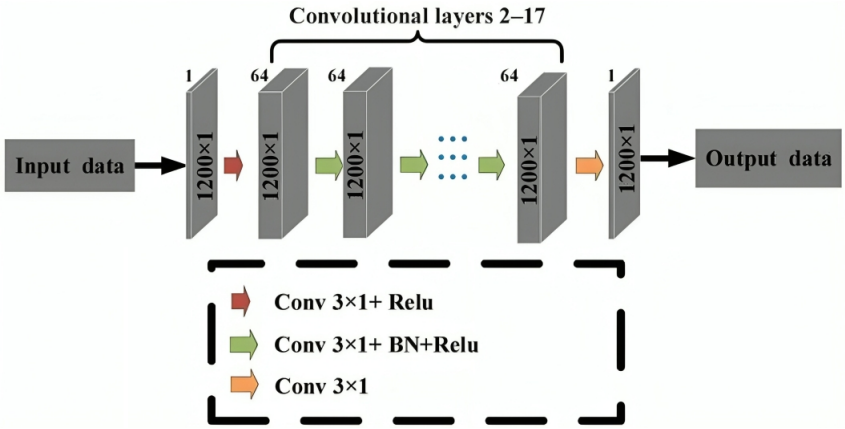

Figura 2.3. Architettura di DnCNN [25]

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 2.8.

Algoritmo 2.8 - `Riduzione del rumore con DnCNN`:
Data un'immagine I, l'algoritmo restituisce l'immagine denoised Out.

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

2.3.3 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 2.9 `Unsharp Mask`:
Data un'immagine I, l'algoritmo restituisce l'immagine nitida Out.

function UNSHARP_MASK(I, α)
  $I_{\text{B}} \leftarrow$ Gaussian_Blur(I) ▷ Sfocatura
  $M \leftarrow I - I_{\text{B}}$ ▷ 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 immaginni 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 2.10, 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 2.10 `Unsharp Masking personalizzato`:
Data un'immagine I, un, l'algoritmo restituisce l'immagine nitida Out.

function CUSTOM_UNSHARP(I, α, β, thr)
  $I_{\mathrm{D}} \leftarrow$ DENOISE_IMAGE(I) ▷ Riduzione del rumore con l'Algoritmo 2.8
  $I_{\mathrm{D}} \leftarrow (I_{\mathrm{D}} \times \alpha) + I \times (1 - \alpha)$ ▷ Attenuazione del denoising
  $G \leftarrow \mathrm{Gradient}(I)$ ▷ Calcolo del gradiente
  $D_{\mathrm{M}} \leftarrow \mathrm{Get\_Mask}(G, thr)$ ▷ Maschera di denoising
  $D_{\mathrm{M}} \leftarrow \mathrm{Gaussian\_Blur}(D_{\mathrm{M}})$ ▷ Sfoca leggermente la maschera
  $S_{\mathrm{M}} \leftarrow 1 - D_{\mathrm{M}}$ ▷ Maschera di sharpening
  $D \leftarrow (I - I_{\mathrm{D}}) \times \beta$ ▷ Estrazione dei dettagli
  $Out \leftarrow (I_{\mathrm{D}} \times D_{\mathrm{M}}) + (I + D) \times S_{\mathrm{M}}$ ▷ Unsharping con dettagli amplificati
  return Out
end function

dove $Get\_Mask(I, thr)$ 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 un 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.

2.3.4 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 2.11.

Algoritmo 2.11 - `Rimozione dello sfondo`:
Data un'immagine I e i parametri thr e α, l'algoritmo restituisce l'immagine con lo sfondo rimosso Out.

function REMOVE_BACKGROUND(I, thr, α)
    $M \leftarrow I < thr$ ▷ Calcola la maschera dello sfondo
    $Out \leftarrow I$
    for ogni pixel p in M dove $M[p] = \text{True}$ **do**
        $Out[p] \leftarrow I[p] \times \dfrac{1}{1 + \exp(-\alpha \times (I[p] - thr))}$ ▷ Trasformazione logistica
    end for
    return Out
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.

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

- **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 2.12.

Algoritmo 2.12 - `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 ← numero di immagini in I
3:      H, W_sum ← 0
4:      for i ← 1 to N do
5:          H ← H + W[i] × I[i]                ▷ Somma pesata
6:          W_sum ← W_sum + W[i]               ▷ Somma dei pesi
7:      end for
8:      Out ← H / W_sum                        ▷ 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'equalizzaziome 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 2.13.

Algoritmo 2.13 - `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)
  $I_{\text{CLAHE}} \leftarrow$ Inizializza un'immagine vuota
  $I_{\text{LAB}} \leftarrow$ 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
  $L_{\text{CLAHE}} \leftarrow C.apply\_CLAHE(L)$ ▷ Applica CLACHE al canale L
  $I_{\text{CLAHE}} \leftarrow$ 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 prevalentemente 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 2.14.

Algoritmo 2.14 - `Shades of Gray - bilanciamento del colore`:
Data un'immagine I, l'algoritmo restituisce l'immagine bilanciata Out.

function SHADES_OF_GRAY($I, p = 6$)
  $M, N \leftarrow$ dimensioni di I
  $S \leftarrow$ inizializza vettore per le norme
  for $c \leftarrow 1$ to 3 **do** ▷ Per ogni canale RGB
    $S[c] \leftarrow \left(\frac{1}{M \times N} \sum_{x=1}^{M} \sum_{y=1}^{N} I(x, y, c)^p \right)^{1/p}$
  end for
  $\mu \leftarrow \frac{1}{3} \sum_{c=1}^{3} S[c]$ ▷ Media delle norme
  for $c \leftarrow 1$ to 3 **do**
    $f[c] \leftarrow \mu / S[c]$ ▷ Fattori di scala
    for $x \leftarrow 1$ to M **do**
      for $y \leftarrow 1$ to N **do**
        $Out(x, y, c) \leftarrow I(x, y, c) \times f[c]$ ▷ Applica bilanciamento
      end for
    end for
  end for
  $Out \leftarrow \text{clip}(Out, 0, 1)$ ▷ Limita valori tra 0 e 1
  return Out
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

Nella prossima sezione verrà illustrata la pipeline completa di elaborazione delle immagini.

3.2 Pipeline di elaborazione

La pipeline di elaborazione completa è stata implementata nel file `process.py`. Questo file contiene la funzione `process_images`, che esegue l'intera pipeline di elaborazione delle immagini, dalla calibrazione all'allineamento, dallo stacking al post-processing. La funzione prende in input una lista di immagini `images` e un dizionario di parametri `params` contenente i parametri di configurazione per la pipeline, e restituisce l'immagine finale elaborata.

```
def process_images(images=None, params={}):
    det_str    = params.get('det_str',    1.3)
    grad_thr   = params.get('grad_thr',   0.008)
    dns_str    = params.get('dns_str',    1.2)
    stack_alg  = params.get('stack_alg',  'weighted_avg')
    avg_alg    = params.get('avg_alg',    'sharpness')
    ush_str    = params.get('ush_str',    2.35)
    tile_size  = params.get('tile_size',  (19, 19))
    clip_limit = params.get('clip_limit', 0.8)

    # Calibration
    calibrated = calibrate_images(images)

    # Alignment
    aligned = align_images(images)

    # Pre-processing
    cropped = crop_to_center(aligned)
    denoised = custom_unsharp_mask(aligned, det_str, grad_thr,
        dns_str)
    no_bg = [soft_threshold(img, 0.05, 50) for img in denoised]

    # Stacking
    stacked = stack_images(no_bg, stack_alg, average_alg)

    # Post-processing
    contrasted = enhance_contrast(stacked, clip_limit, tile_size)
    unsharped = unsharp_mask(contrasted, unsharp_strength)
    final_image = shades_of_gray(unsharped)

    return final_image
```

I parametri di default sono stati ottenuti eseguendo una grid search (implementata in `grid_search.py`) su un paio di set di immagini di test, al fine di trovare i valori

ottimali che massimizzano la qualità dell'output finale secondo le metriche implementate in `metrics.py`. Tale ricerca ha permesso di identificare una configurazione bilanciata di parametri che garantisce risultati accettabili sui diversi set.

Nelle sezioni seguenti verranno illustrate in dettaglio le implementazioni delle singole funzionalità della pipeline, esaminando le scelte progettuali e gli algoritmi utilizzati per ciascun modulo.

3.2.1 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:

```
[label={lst:calculate_masters}]
# Function to calculate the master bias
def calculate_master_bias(bias):
    # Calculate the mean
    return np.mean(bias, axis=0)

# Function to calculate the master dark
def calculate_master_dark(dark, master_bias=None):
    if master_bias is None: master_bias = np.zeros_like(flat[0])
    # Subtract the master bias, then calculate the mean
    return np.mean(dark - master_bias, axis=0)

# Function to calculate the master flat
def calculate_master_flat(flat, master_bias=None, master_dark=
    None):
    if master_bias is None: master_bias = np.zeros_like(flat[0])
    if master_dark is None: master_dark = np.zeros_like(flat[0])
    # Subtract master bias and dark, then calculate the mean
    master_flat = np.mean(flat-master_bias-master_dark, axis=0)
    # Normalize the master flat
    mean_flat = np.mean(master_flat)
    if mean_flat != 0: master_flat /= mean_flat
    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 compiutazionalmente 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:

```
# Function to calibrate a single image
def calibrate_single_image(image, master_bias = None, master_dark = None, master_flat = None):
    if master_bias is None and master_dark is None and master_flat is None:
        print("No calibration masters available: skipping")
        return image
    if master_bias is None: master_bias = np.zeros_like(flat[0])
    if master_dark is None: master_dark = np.zeros_like(flat[0])
    if master_flat is None: master_flat = np.ones_like(flat[0])
    # Calibrate the image
    calibrated = (image - master_bias - master_dark)/master_flat
    calibrated = np.clip(calibrated, 0, 1)  # Clip to valid range
    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].

3.2.2 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 affrontatp 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 2.11), 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 2.13, 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 3.2.3.

```
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:

```
def align_images(images, method='orb', nfeatures=5000):
    # Choose the feature detection algorithm
    if method == 'orb':
        detector = cv2.ORB_create(nfeatures=nfeatures)
    elif method == 'sift':
        detector = cv2.SIFT_create(nfeatures=nfeatures)
    elif method == 'surf':
        detector = cv2.xfeatures2d.SURF_create()
    # Choose the norm for the matcher
    norm = cv2.NORM_HAMMING if method == 'orb' else cv2.NORM_L2
    # Create a matcher object
    matcher = cv2.BFMatcher.create(norm)

    # The first image is the reference image
    ref_image = images[0]
    aligned_images = [ref_image]

    # Enhance the reference image to improve alignment
    enhanced_ref = pre_align_enhance(ref_image)

    # Detect keypoints and descriptors for the reference image
    ref_kp, ref_des = detector.detectAndCompute(enhanced_ref,
        None)
    ref_shape = (ref_image.shape[1], ref_image.shape[0])

    # Align each image to the reference image
    for idx, image in enumerate(images[1:]):
        aligned_image = align_image(image, enhanced_ref, ref_kp,
            ref_des, ref_shape, detector, matcher)
        aligned_images.append(aligned_image)

    return aligned_images
```

La funzione `align_image` funzione 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 keypoints 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:

```
def align_image(image, ref_kp, ref_des, detector, matcher):
    # Enhance the image for alignment process
    aligned_image = pre_align_enhance(image)

    # Find keypoints and descriptors for the image
    kp, des = detector.detectAndCompute(aligned_image, None)

    # Match the descriptors
    matches = matcher.knnMatch(ref_des, des, k=2)

    # Apply the ratio test
    matches = [m for m, n in matches if m.distance < 0.75 * n.
        distance]

    # Compute the homography
    ref_pts = np.float32([ref_kp[m.queryIdx].pt for m in matches
        ]).reshape(-1, 1, 2)
    img_pts = np.float32([kp[m.trainIdx].pt for m in matches]).
        reshape(-1, 1, 2)
    H, _ = cv2.findHomography(img_pts, ref_pts, cv2.RANSAC, 10.0,
         maxIters=3000, confidence=0.995)
    shape = image.shape[1], image.shape[0]

    # Warp the original image using the final homography
    aligned_image = cv2.warpPerspective(image, H, shape, flags=
        cv2.INTER_LANCZOS4)
    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.

3.2.3 Pre-processing

La fase di pre-processing affrontata nella Sezione 2.3 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 vengono eseguite a cascata e sono:

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

- **Miglioramento della nitidezza** attraverso l'uso di tecniche di Unsharp Mask personalizzate e DnCNN;
- **Rimozione dello sfondo** tramite la sogliatura morbida;

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

Ritaglio dell'immagine

Il ritaglio dell'immagine, basato sull'Algoritmo 2.7, è stato implementato nella funzione `crop_to_center`. Questa funzione prende in input una lista di immagini allineate `images` e un margine `margin` (di default 10 pixel) e restituisce una lista di immagini ritagliate in modo che il soggetto sia centrato e i bordi esterni siano eliminati.

```
def crop_to_center(images, margin=10):
cropped_images = []

# Process the first image to get the cropping parameters
ref = images[0]
gray = ref
if len(ref.shape) == 3:
    gray = cv2.cvtColor(gray, cv2.COLOR_RGB2GRAY)

# Binary thresholding
_, thresh = cv2.threshold(gray, 0.1, 1.0, cv2.THRESH_BINARY)

# Find contours
contours, _ = cv2.findContours(to_8bit(thresh), cv2.RETR_EXTERNAL
    , cv2.CHAIN_APPROX_SIMPLE)
contour = max(contours, key=cv2.contourArea)

# Get the bounding rectangle of the selected contour
x, y, w, h = cv2.boundingRect(contour)
center_x, center_y = x + w//2, y + h//2 # Calculate the center
size = max(w, h) + 2 * margin               # Determine the size

# Calculate the top-left corner of the square
start_x = max(center_x - size//2, 0)
start_y = max(center_y - size//2, 0)

# Ensure the square fits within the image boundaries
end_x = min(start_x + size, ref.shape[1])
end_y = min(start_y + size, ref.shape[0])

# Crop all images using the same parameters
for image in images:
    cropped_image = image[start_y:end_y, start_x:end_x]
    cropped_image = np.clip(cropped_image, 0, 1)
    cropped_images.append(cropped_image)

return cropped_images
```

Da notare che la funzione `crop_to_center` ritaglia tutte le immagini della lista `images` utilizzando i parametri di ritaglio calcolati sull'immagine di riferimento.

Questo approccio richiede che il soggetto sia centrato in tutte le immagini, altrimenti potrebbero verificarsi problemi di allineamento durante il processo di stacking.

Riduzione del rumore e Unsharp Masking con DnCNN

Come descritto nelle sezioni 2.3.2 e 2.3.3, 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),
                   nn.ReLU(inplace=True),
8              *[layer_block(n_channels) for _ in range(depth-2)],
9              nn.Conv2d(n_channels, image_channels, 3, padding=1)
10         )
11         self._initialize_weights()
12
13     def forward(self, x):
14         return x - self.dncnn(x)
15
16     def _initialize_weights(self):
17         for m in self.modules():
18             if isinstance(m, nn.Conv2d):
19                 init.orthogonal_(m.weight)
20                 if m.bias is not None: init.constant_(m.bias, 0)
21             elif isinstance(m, nn.BatchNorm2d):
22                 init.constant_(m.weight, 1); init.constant_(m.
                       bias, 0)
23
24 def layer_block(n_channels):
25     return nn.Sequential(
26         nn.Conv2d(n_channels,n_channels,3,padding=1,bias=False),
27         nn.BatchNorm2d(n_channels, eps=1e-4, momentum=0.95),
28         nn.ReLU(inplace=True)
29     )
30
31 def model_init(model_path=DNCNN_MODEL_PATH):
32     model = torch.load(model_path)
33     model.eval()
34     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 2.3.2.

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 2.8 è quindi la seguente:

```
def perform_denoising(model, image):
    shape = image.shape

    if len(shape) < 3:
        image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)
    channels = list(cv2.split(color.rgb2lab(image)))

    # convert image to tensor
    l = prepare_tensor_for_model(channels[0], 100).to(device)

    if torch.cuda.is_available(): device = torch.device('cuda')
    else: torch.device('cpu')
    model = model.to(device).eval()
    with torch.no_grad():
        denoised_l = model(l).cpu()

    # convert tensor to image
    denoised_l = convert_tensor_to_image(denoised_l, 100)
    channels[0] = denoised_l

    # Merge the channels
    denoised = cv2.merge(channels)
    denoised = color.lab2rgb(denoised)

    if len(shape) < 3:
        denoised = cv2.cvtColor(denoised, cv2.COLOR_RGB2GRAY)

    return denoised
```

Come è possibile notare, il denoising viene eseguito solo sul canale L dell'immagine, in quanto è il canale che contiene la maggior parte delle informazioni di luminosità. Dopodiché l'immagine viene riconvertita in RGB e, se necessario, in scala di grigi.

Possiamo ora descrivere l'algoritmo di Unsharp Masking personalizzato, basato sull'Algoritmo 2.10, che è stato implementato nella funzione `custom_unsharp_mask`.

Questa implementazione combina il denoising tramite DnCNN con una maschera basata sul gradiente dell'immagine per preservare i dettagli importanti.

```
def custom_unsharp_mask(images, model, det_str=1.3, thr=0.09,
    dns_str=1.2):
    sharpened_images = []

    for idx, image in enumerate(images):
        # Apply denoising with DnCNN
        denoised = denoise(image, model)

        # Blend original and denoised image
        denoised = denoised * dns_str + image * (1 - dns_str)

        # Convert to grayscale for gradient calculation if needed
        gray_image = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY) if
            image.ndim == 3 else image

        # Calculate image gradient magnitude
        grad_mag = np.sqrt(np.gradient(gray)**2, axis=0)

        # Create denoise mask: 1 where gradient is low
        mask = np.where(grad_mag < thr, 1, 0).astype(np.float32)
        # Smooth mask edges
        mask = cv2.GaussianBlur(mask, (3,3), 0.5)

        # Expand mask to 3 channels for RGB images
        if image.ndim == 3:
            mask = np.repeat(mask[:, :, np.newaxis], 3, axis=2)

        # Create detail mask (inverse of denoise mask)
        det_mask = 1 - mask

        # Combine denoised and original image using masks
        final_image = (denoised * mask) + (image * det_mask)

        # Amplify details in high gradient areas
        amplified_details = (image - denoised) * det_str * det_mask
        sharpened_image = final_image + amplified_details

        # Ensure valid pixel range
        sharpened_image = np.clip(sharpened_image, 0, 1)
        sharpened_images.append(sharpened_image)

    return sharpened_images
```

La funzione `custom_unsharp_mask` prende in input una lista di immagini `images`, il modello DnCNN `model` e i parametri per controllare il processo: `det_str` regola l'intensità dell'amplificazione dei dettagli, `thr` definisce la soglia per distinguere aree di dettaglio da aree uniformi, e `dns_str` indica l'intensità del denoising sul risultato.

Per ciascuna immagine, viene applicato il denoising tramite il modello DnCNN, viene poi calcolato il gradiente, da cui si ricavano due maschere che separano zone ad alto dettaglio da quelle più uniformi. La fase finale combina l'immagine originale con quella denoised utilizzando le maschere create, amplificando i dettagli nelle aree ad alto gradiente e privilegiando l'effetto del denoising nelle altre.

3.2.4 Stacking

La fase di stacking delle immagini è stata implementata nel file `stacking.py`. Come descritto nel Sezione 2.4, lo stacking è una tecnica fondamentale in astrofotografia per migliorare il rapporto segnale-rumore combinando più immagini dello stesso soggetto. Nel progetto sono stati implementati tre dei metodi di stacking più comuni:

- **Media Pesata (Weighted Mean)**: calcola la media pesata delle immagini, assegnando pesi basati su una metrica di qualità, come la nitidezza.
- **Mediana (Median Stacking)**: calcola il valore mediano pixel per pixel delle immagini.
- **Sigma Clipping**: calcola la media pixel per pixel escludendo i pixel che deviano oltre una certa soglia dalla media.

Il metodo che ha prodotto i risultati migliori è la media pesata, dove i pesi sono determinati in base alla qualità di ciascuna immagine. L'algoritmo è stato implementato nella funzione `weighted_average_stack`.

```
1  def weighted_average_stack(images, method='sharpness'):
2      # Calculate the weights for each image
3      weights = calculate_weights(images, method)
4      # Zero out the lowest 10% of the weights
5      weights[np.argsort(weights)[:len(weights)//10]] = 0
6      weights = weights / np.sum(weights)
7
8      n_ch = 1 if len(images[0].shape) == 2 else images[0].shape[2]
9      stacked_channels = []
10     for i in range(n_ch):
11         # Extract the i-th channel from each image
12         channel_stack = np.array([cv2.split(img)[i] for img in
               images])
13
14         # Calculate the weighted sum of the channels
15         stacked_channel = np.zeros_like(channel_stack[0], dtype=
               np.float64)
16         for channel, weight in zip(channel_stack, weights):
17             # Weighted sum
18             stacked_channel += channel * weight
19
20         stacked_channels.append(stacked_channel)
21
22     # Combine the channels into a single image
23     stacked_image = cv2.merge(stacked_channels)
24     return stacked_image.astype(np.float32)
```

La funzione `weighted_average_stack` calcola i pesi per ciascuna immagine in base alla metrica specificata (chiamando la funzione `calculate_weights`), normalizza i pesi e calcola la media pesata dei pixel per ciascun canale. Infine, combina i canali in un'unica immagine.

La metrica che ha portato a risultati migliori è stata la nitidezza, ma è possibile utilizzare anche altre metriche.

La funzione `calculate_weights` per calcolare i pesi è riportata di seguito:

```
def calculate_weights(images, method='sharpness'):
    if method == 'contrast':
        weights = [calculate_contrast(img) for img in images]
    elif method == 'sharpness':
        weights = [calculate_sharpness(img) for img in images]
    elif method == 'brisque':
        weights = [1 - normalize(calculate_metric(img, '
            brisque_matlab'), 0, 100) for img in images]
    elif method == 'liqe':
        weights = [calculate_metric(img, 'liqe') for img in
            images]
    else:
        raise ValueError("Unknown method: {}".format(method))
    return weights
```

Le metriche disponibili includono la nitidezza `sharpness`, il contrasto `contrast`, e le metriche `BRISQUE` e `LIQE` di PyIqa. Tutte queste metriche vengono calcolate utilizzando le funzioni definite nel file `metrics.py`, non riportate per brevità.

Il metodo `median_stack` implementa lo stacking per mediana, calcolando il valore mediano pixel per pixel delle immagini. Questo metodo è utile per ridurre l'effetto di outlier e rumore impulsivo.

```
def median_stack(images):
    n_ch = 1 if len(images[0].shape) == 2 else images[0].shape[2]
    stacked_channels = [np.median([cv2.split(img)[i] for img in
        images], axis=0) for i in range(n_ch)]
    stacked_image = cv2.merge(stacked_channels)
    return stacked_image.astype(np.float32)
```

Il metodo `sigma_clipping` implementa lo stacking utilizzando il sigma clipping, che esclude i pixel che deviano oltre una certa soglia (espressa in termini di deviazione standard) dalla media.

```
def sigma_clipping(images, sigma=3):
    n_ch = 1 if len(images[0].shape) == 2 else images[0].shape[2]
    stacked_channels = []
    for i in range(n_ch):
        channel_stack = np.array([cv2.split(img)[i] for img in
            images])
        mean = np.mean(channel_stack, axis=0)
        std = np.std(channel_stack, axis=0)
        # Mask for values within sigma × std from the mean
        mask = np.abs(channel_stack - mean) < sigma * std
        # Clip the values outside the mask to the mean
        clipped = np.where(mask, channel_stack, mean)
        # Calculate the mean of the clipped values
        stacked_channel = np.mean(clipped, axis=0)
        stacked_channels.append(stacked_channel)
    stacked_image = cv2.merge(stacked_channels)
    return stacked_image.astype(np.float32)
```

Questo metodo è efficace nel ridurre l'impatto di valori anomali e artefatti.

3.2.5 Post-processing

La fase di post-processing è stata implementata nel file `enhancement.py` e comprende diverse operazioni finalizzate a migliorare ulteriormente la qualità dell'immagine dopo lo stacking. Le principali operazioni eseguite sono:

- **Miglioramento della nitidezza** utilizzando l'algoritmo tradizionale di Unsharp Mask.
- **Miglioramento del contrasto** applicando l'algoritmo CLAHE (Contrast Limited Adaptive Histogram Equalization).
- **Bilanciamento del colore** per correggere eventuali dominanti cromatiche presenti nell'immagine finale.

Di seguito sono descritte le implementazioni di queste operazioni.

Miglioramento della nitidezza con Unsharp Mask

Dopo lo stacking, può essere utile aumentare ulteriormente la nitidezza dell'immagine per enfatizzare i dettagli superficiali della Luna. L'algoritmo di Unsharp Mask tradizionale è stato utilizzato per questo scopo, implementato nella funzione `unsharp_mask`.

```
1  def unsharp_mask(image, strength):
2      # Apply a Gaussian filter to blur the image
3      blurred_image = cv2.GaussianBlur(image, (3, 3), 0.5)
4
5      # Combine the blurred image with the original image
6      sharpened_image = cv2.addWeighted(image, 1 + strength,
           blurred_image, -strength, 0)
7
8      # Clip the values to the valid range
9      sharpened_image = np.clip(sharpened_image, 0, 1)
10
11     return sharpened_image
```

In questa funzione, viene applicato un filtro Gaussiano per ottenere una versione sfocata dell'immagine. La differenza tra l'immagine originale e quella sfocata viene poi amplificata di un fattore `strength` e sommata all'originale. Questo approccio, descritto in dettaglio nella Sezione 2.3.3, incrementa efficacemente la nitidezza dell'immagine preservando al contempo la naturalezza dei dettagli e minimizzando l'introduzione di artefatti.

Il parametro `strength` consente un controllo preciso sull'intensità dell'effetto di sharpening. Attraverso test empirici condotti nell'ambito di questo progetto, è stato determinato che valori intorno a 2.35 forniscono il miglior compromesso tra miglioramento della nitidezza e preservazione della qualità dell'immagine.

Miglioramento del contrasto con CLAHE

Per migliorare il contrasto locale e rendere più visibili le variazioni tonali nell'immagine, è stato utilizzato l'algoritmo CLAHE (Contrast Limited Adaptive Histogram Equalization), descritto nell'Algoritmo 2.13.

L'implementazione è stata realizzata nella funzione `enhance_contrast`.

```
def enhance_contrast(image, clip_limit = 0.07, tile_size = (20, 20)):
    if len(image.shape) < 3: image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)

    # Convert the image to LAB color space
    lab = cv2.cvtColor(image, cv2.COLOR_RGB2LAB)
    l_channel = lab[:, :, 0]

    # Apply CLAHE to the L channel
    clahe = cv2.createCLAHE(clipLimit=clip_limit, tileGridSize=tile_size)
    l_channel_equalized = clahe.apply(l_channel)

    # Update the L channel
    lab[:, :, 0] = l_channel_equalized

    # Convert the image back to RGB color space
    enhanced_image = cv2.cvtColor(lab, cv2.COLOR_LAB2RGB)

    return enhanced_image
```

L'algoritmo migliora il contrasto locale dell'immagine, rendendo più evidenti i dettagli nelle aree scure o chiare. I parametri `clip_limit` e `tile_grid_size` permettono di regolare il livello di contrasto e la dimensione dei blocchi utilizzati per il calcolo dell'istogramma. I valori di default sono stati scelti in modo da ottenere risultati ottimali.

Bilanciamento del colore

Per correggere eventuali dominanti cromatiche e bilanciare i colori dell'immagine finale, è stato applicato nuovamente l'algoritmo `Shades of Gray`, già descritto nella Sezione 2.5.2. L'implementazione è la seguente:

```
def shades_of_gray(image, power=6):
    # Calculate norm values for each channel using Minkowski norm
    norm_values = np.power(np.mean(np.power(image, power), axis=(0, 1)), 1 / power)

    # Calculate the mean of the norm values
    mean_norm = np.mean(norm_values)

    # Calculate scaling factors to balance the color channels
    scale_factors = mean_norm / norm_values

    # Apply the scaling factors to balance the image
    balanced_image = image * scale_factors

    # Clip values to valid range [0,1]
    balanced_image = np.clip(balanced_image, 0, 1)

    return balanced_image
```

Questa funzione normalizza i canali di colore dell'immagine, correggendo le deviazioni cromatiche e garantendo una resa cromatica più naturale.

Le operazioni di post-processing vengono eseguite in sequenza per ottenere un'immagine finale ottimizzata. Dopo lo stacking, l'immagine passa attraverso l'Unsharp Mask per migliorare la nitidezza, CLAHE per aumentare il contrasto e infine Shades of Gray per bilanciare i colori. Questa sequenza garantisce che i dettagli siano ben definiti, il contrasto elevato e i colori equilibrati, migliorando significativamente la qualità visiva dell'immagine finale.

Per assicurare che le operazioni di post-processing funzionino correttamente indipendentemente dalle condizioni iniziali dell'immagine, è stata implementata una gestione delle immagini in scala di grigi e a colori. Ad esempio, se l'immagine è in scala di grigi, viene convertita temporaneamente in formato RGB prima di applicare CLAHE, per evitare errori di dimensionamento dei canali.

3.3 Sfide affrontate e soluzioni adottate

Durante la fase di progettazione e implementazione del progetto, sono state affrontate diverse sfide. Alcune di esse sono state risolte, mentre altre, a causa di limitazioni di tempo o risorse, rimangono aperte per futuri sviluppi.

3.3.1 Alto utilizzo di RAM

Un problema significativo riscontrato durante lo sviluppo è stato l'elevato utilizzo di memoria RAM da parte del programma. Tale problema è stato causato principalmente dall'uso di numerose immagini ad medio-alta risoluzione, che richiedono molta memoria per essere elaborate. La causa principale risiede nell'implementazione della funzione per la lettura e il caricamento delle immagini in memoria. Questo metodo prende in input il percorso della cartella contenente le foto e le apre tutte contemporaneamente (utilizzando metodi diversi in base al formato), inserendole in una lista.

```
1  def read_image(file_path):
2      if file_path.lower().endswith(('.tiff', '.tif')):
3          return imageio.imread(file_path).astype(np.float32)
4      if file_path.lower().endswith(raw_formats):
5          with rawpy.imread(file_path) as raw:
6              # convert to rgb image using the camera white balance
7              return to_float32(raw.postprocess(use_camera_wb=True,
                   no_auto_bright=True, output_bps=16))
8      else:
9          return to_float32(imageio.imread(file_path))
10
11 def read_images(folder_path, max_img=MAX_IMG):
12     img_paths = read_folder(folder_path, max_img)
13     #  ------↓ here is where the problem lies ↓------
14     images = [read_image(path) for path in img_paths]
```

In questo modo, ogni immagine viene caricata in RAM in formato a 32 bit. Lavorando con insiemi di 20-30 di immagini l'uno (ad esempio nella fase di calibrazione sono stati utilizzati 30 bias frames e altrettanti dark frames, per un totale di circa 90

immagini caricate in memoria contemporaneamente) spesso il programma termina in modo anomalo a causa dell'uso eccessivo di memoria. Per ovviare a questo problema, è stata suddivisa la pipeline in stadi intermedi nei quali vengono salvati i risultati parziali in formato TIFF a 32 bit. In questo modo, è possibile riavviare l'esecuzione del programma senza dover ricalcolare tutti gli step precedenti al crash; tuttavia questa non rappresenta una soluzione ottimale.

Una soluzione più adatta a questo problema è quella dell'uso di generatori per leggere le immagini. Ciò permette di caricare in memoria solo un'immagine alla volta, riducendo notevolmente l'uso di RAM. Una possibile implementazione è la seguente:

```
def read_images_generator(folder_path, max_imgs=None):
    loaded = 0
    for filename in sorted(os.listdir(folder_path)):
        if max_imgs is not None and loaded >= max_imgs:
            break
        img_path = os.path.join(folder_path, filename)
        if os.path.isfile(img_path):
            if img_path.lower().endswith(('.tiff', '.tif')):
                yield imageio.imread(img_path).astype(np.float32)
            if img_path.lower().endswith(raw_formats):
                with rawpy.imread(img_path) as raw:
                    # convert to rgb image using the camera wb
                    yield to_float32(raw.postprocess(
                        use_camera_wb=True, no_auto_bright=True,
                        output_bps=16))
            else:
                yield to_float32(imageio.imread(img_path))
            loaded += 1
```

Questo metodo è stato testato per generare le immagini di analisi e ha portato a un notevole risparmio di memoria. Si propone quindi come soluzione per futuri sviluppi.

L'utilizzo intensivo di memoria ha spesso influenzato le scelte di progetto, portando a preferire soluzioni più efficienti dal punto di vista della memoria, anche a discapito della velocità di esecuzione. Un esempio si ha nella scelta dell'immagine di riferimento per l'allineamento; inizialmente si era pensato di selezionare tra le immagini acquisite quella con la migliore qualità, ma ciò avrebbe richiesto il caricamento in memoria di tutte le immagini per confrontarle, con un conseguente aumento dell'uso di RAM. Si è quindi preferito utilizzare la prima immagine come riferimento, anche se ciò potrebbe portare a risultati di allineamento meno precisi nel caso in cui la qualità dell'immagine di riferimento fosse inferiore rispetto alle altre.

3.3.2 Ricerca dei parametri ottimali

Una delle maggiori sfide di questo progetto è stata la ricerca dei parametri ottimali per l'elaborazione delle immagini. Molti parametri dipendono fortemente dalle immagini in ingresso, come la soglia `grad_thr` per generare la maschera dal gradiente, o `dns_str`, che regola l'intensità del denoising da applicare alla foto, entrambi nella funzione `custom_unsharp_mask`. È quindi necessario effettuare una ricerca avanzata

dei parametri per rendere i risultati ottimali indipendentemente da fattori come la luminosità, il contrasto e la quantità di rumore, presenti nelle immagini di partenza.

Un'alternativa per ottenere risultati più precisi consiste nell'estrarre i valori di tali parametri direttamente dalle caratteristiche delle immagini di partenza. Ad esempio, il parametro `tile_size` dipende principalmente dalle dimensioni dell'immagine, più che da caratteristiche della foto in sé; dopo diverse prove, è stato notato che si ottengono risultati migliori per valori che si aggirano attorno a 1/70 delle dimensioni dell'immagine dopo la fase di ritaglio.

Un ulteriore problema riguarda la valutazione dei risultati nella ricerca dei parametri. Quando si ha a che fare anche solo con una decina di parametri da testare, provando tre valori per ciascuno di essi, si ottengono migliaia di foto in output, rendendo l'analisi visiva impossibile. Inoltre, spesso si osservano variazioni tra le immagini non sempre distinguibili ad occhio nudo. È quindi necessario stabilire una metrica di riferimento che rifletta il miglioramento visivo dell'immagine. Da ciò sorge un ulteriore problema: l'assenza di un'immagine di riferimento.

3.3.3 Assenza di immagine di riferimento

Come verrà illustrato nel Capitolo 4, la metrica di valutazione maggiormente adottata nel contesto dell'astrofotografia è il rapporto segnale-rumore (SNR). Questa metrica, per essere calcolata, richiede la presenza di un'immagine di riferimento ideale a cui il programma deve tendere. Poiché le immagini sono state acquisite durante il progetto, non era disponibile un'immagine di riferimento ideale. È stato quindi tentato di utilizzare come riferimento un'immagine (scaricabile dai siti INAF o NASA) acquisita dal Lunar Reconnaissance Orbiter (LRO), un satellite in orbita intorno alla Luna. Questo satellite ha compiuto diversi scatti, da cui è stata elaborata l'immagine finale tramite la tecnica dello stitching. Tuttavia, ciò causa problemi di compatibilità con le immagini acquisite. Infatti, la foto di riferimento presenta, nelle aree periferiche della Luna, dettagli che non compaiono nelle immagini acquisite, rendendo impossibile l'allineamento. Sono state inoltre esplorate altre immagini di riferimento, ma quelle ad uso libero trovate su internet spesso non presentano una qualità superiore a quelle utilizzate nel progetto, rendendole quindi inadatte come riferimento "ideale", oppure sono immagini delle cosiddette "mineral moon", risultanti dalla sovrapposizione di scatti ottenuti attraverso filtri di frequenza, con una colorazione artificiale che le rende non adatte al confronto.

Una tecnica adottata comunemente per risolvere questo problema consiste nel prendere un'immagine di alta qualità e, da essa, generare diversi set di immagini con rumore aggiunto, in modo da simulare le condizioni reali. Questo metodo, chiamato *data augmentation*, è stato adottato in diversi progetti di machine learning, e potrebbe essere una soluzione per questo progetto. Tuttavia, vista la mancanza di tempo e di risorse, non è stato possibile implementare questa soluzione, che si rimanda a sviluppi futuri.

Viste le difficoltà affrontate, è stato deciso di utilizzare delle metriche di valutazione senza riferimento, le quali però, come vedremo nel dettaglio nel prossimo capitolo, non sempre sono in grado di percepire il miglioramento effettivo della foto.

3.3.4 Pochi dati per testing

Come già specificato nel capitolo introduttivo, gli scatti sono stati acquisiti tramite una fotocamera bridge Fujifilm Finepix S1, un dispositivo di media gamma, senza l'ausilio di strumentazione professionale come telescopi o montature robotizzate (descritte nella Sezione 1.2.1).

Il problema principale è stato rappresentato dalle condizioni meteorologiche, che nei mesi precedenti la realizzazione del progetto sono state raramente favorevoli, permettendo di acquisire pochi scatti e, soprattutto, con poca variazione dell'illuminazione tra i vari set. Questa limitazione ha complicato la ricerca dei parametri, aumentando il rischio di overfitting sui dati disponibili per i test.

Un altro problema riscontrato è la difficoltà nell'acquisizione dei bias frames. Questi infatti, come già specificato nella Sezione 2.1.3, richiedono di catturare (nelle stesse condizioni di temperatura dei light frames) una fonte luminosa omogenea, il problema sorge quando si ha a che fare con una fonte non omogenea, come una lampada a incandescenza o a LED, che possono causare vignettatura o illuminazione non uniforme, che, invece di migliorare i risultati, li peggiorerebbero.

Capitolo 4

Valutazione dei risultati e metriche di qualità

Questo capitolo presenta le metriche comunemente utilizzate in astrofotografia, nonché quelle impiegate in questo progetto per valutare la qualità delle immagini ottenute attraverso il processo di elaborazione descritto nei capitoli precedenti.

La selezione delle metriche è fondamentale per stabilire criteri oggettivi di valutazione e per confrontare i risultati ottenuti con diverse configurazioni di parametri. Verranno discusse sia metriche con riferimento, che richiedono un'immagine di riferimento ideale, sia metriche senza riferimento, che valutano autonomamente la qualità dell'immagine.

Successivamente, verranno presentati i risultati ottenuti e analizzati i miglioramenti introdotti dalle varie tecniche di elaborazione delle immagini.

4.1 Metriche di valutazione con riferimento

Le metriche con riferimento confrontano l'immagine elaborata con un'immagine ideale, fornendo una misura della similarità o della qualità relativa tra le due. Nel contesto di questo progetto, l'utilizzo di metriche con riferimento ha presentato alcune sfide, principalmente a causa dell'assenza di un'immagine di riferimento adeguata, come discusso nella Sezione 3.3.

4.1.1 SSIM

SSIM (Structural Similarity Index Measure) è una metrica che misura la similarità strutturale tra due immagini. In astrofotografia, dopo aver applicato tecniche di denoising, la metrica SSIM viene utilizzata per valutare quanto efficacemente il rumore è stato ridotto mantenendo intatte le strutture originali dell'immagine, come stelle, galassie e nebulose. SSIM può essere impiegata anche per perfezionare la tecnica dello stacking, aiutando a determinare quali combinazioni di immagini offrono la migliore preservazione delle strutture dettagliate rispetto all'immagine di

riferimento.

La formula generale per il calcolo dell'SSIM tra due immagini x e y è:

$$\text{SSIM}(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

dove μ_x e μ_y sono le medie delle immagini x e y, σ_x^2 e σ_y^2 sono le varianze, σ_{xy} è la covarianza tra x e y e, infine, C_1 e C_2 sono due costanti che stabilizzano la divisione in caso di valori molto bassi di $\mu_x^2 + \mu_y^2$ e $\sigma_x^2 + \sigma_y^2$. Il valore dell'SSIM è compreso tra -1 e 1, dove 1 indica una similarità perfetta tra le due immagini, e -1 indica una dissimilarità totale.

SSIM tiene conto della percezione umana, valutando la luminanza, il contrasto e la struttura delle immagini. Tuttavia, è sensibile a piccoli spostamenti e rotazioni tra le immagini e non è in grado di valutare la qualità di immagini con diverse dimensioni o scale.

Nel progetto, a causa dell'assenza di un'immagine di riferimento adeguata (come discusso nella Sezione 3.3), l'utilizzo dell'SSIM non è stato possibile.

4.1.2 SNR

Il **Rapporto Segnale-Rumore** (SNR: Signal-to-Noise Ratio) è una metrica fondamentale nell'ambito dell'astrofotografia, utilizzata per quantificare la qualità delle immagini astronomiche. Esso misura la relazione tra il segnale utile (le informazioni desiderate, come stelle, galassie o, in questo caso, la Luna e i suoi dettagli) e il rumore di fondo presente nell'immagine. Un alto valore di SNR indica che il segnale è dominante rispetto al rumore, risultando in immagini più nitide e dettagliate.

$$SNR = 10 \cdot \log_{10}\left(\frac{\text{Potenza del Segnale}}{\text{Potenza del Rumore}}\right) = 10 \cdot \log_{10}\left(\frac{\mu_s^2}{\mu_n^2}\right)$$

dove μ_s è la media del segnale e σ_n^2 è la varianza del rumore.

L'SNR è una metrica semplice e intuitiva, ma non tiene conto delle caratteristiche percettive del sistema visivo umano. Inoltre, come SSIM, è sensibile a piccoli spostamenti e rotazioni tra le immagini e non è in grado di valutare la qualità di immagini con diverse dimensioni o scale, richiedendo dunque un perfetto allineamento tra le immagini di riferimento e quelle elaborate.

Algoritmi di denoising avanzati, inclusi quelli basati su reti neurali, vengono valutati utilizzando l'SNR per garantire che il segnale sia preservato efficacemente mentre il rumore viene minimizzato. Tuttavia, anche l'SNR non è stato utilizzato in questo progetto a causa dell'assenza di un'immagine di riferimento adeguata.

4.2 Metriche di valutazione senza riferimento

La mancanza di un'immagine di riferimento ideale ha reso necessario l'utilizzo di metriche di valutazione senza riferimento per valutare la qualità delle immagini elaborate.

Le metriche senza riferimento valutano la qualità dell'immagine in maniera autonoma, senza bisogno di confrontarla con un'immagine ideale,analizzando dunque l'immagine stessa per stimare la sua qualità basandosi su modelli statistici o percettivi.

4.2.1 NIQE

NIQE (Naturalness Image Quality Evaluator) è una metrica di qualità delle immagini senza riferimento, sviluppata per valutare autonomamente la qualità delle immagini digitali, senza necessità di un'immagine di riferimento. NIQE è basata su un modello di percezione visiva umana, che valuta la qualità dell'immagine basandosi su caratteristiche statistiche e percettive, come la nitidezza, il contrasto, la luminosità e la presenza di artefatti [**niqe_paper**].

NIQE calcola la distanza tra le statistiche dell'immagine e quelle di un campione di immagini di riferimento, fornendo un punteggio di qualità che indica quanto l'immagine si discosti dalla media delle immagini di riferimento.

Nel progetto, NIQE è stata testata come possibile metrica di valutazione, ma ha mostrato limitazioni nel catturare miglioramenti specifici apportati dalle tecniche di elaborazione applicate alle immagini lunari. Ciò è probabilmente dovuto alla natura specifica delle immagini astronomiche, che possono differire significativamente dalle immagini di riferimento utilizzate per addestrare il modello.

4.2.2 BRISQUE

BRISQUE (Blind Referenceless Image Spatial Quality Evaluator) è una metrica di qualità delle immagini senza riferimento, che utilizza caratteristiche statistiche naturali e un modello di regressione addestrato su un dataset di immagini distorte.

BRISQUE è basata su un modello di percezione visiva umana, che valuta la qualità dell'immagine basandosi su caratteristiche statistiche e percettive, come la nitidezza, il contrasto, la luminosità e la presenza di artefatti [**brisque_paper**].

Nel progetto, l'utilizzo di BRISQUE ha fornito risultati migliori rispetto a NIQE, ma ancora non pienamente soddisfacenti per valutare le immagini lunari elaborate.

4.2.3 LIQE

LIQE (Learning-based Image Quality Evaluator) è una metrica di qualità delle immagini senza riferimento basata su apprendimento automatico, che utilizza tecniche di *deep learning* per valutare la qualità delle immagini [**liqe_paper**]. LIQE combina informazioni statistiche a livello globale e locale ed è in grado di catturare in maniera

più efficace le distorsioni presenti nelle immagini.

Il modello assegna un punteggio che va da 1 a 5, dove 1 indica una qualità molto bassa e 5 una qualità molto alta. Questa scala è adottata nell'implementazione della libreria `PyIQA` utilizzata nel progetto.

Rispetto a NIQE e BRISQUE, LIQE è stato progettato per essere più robusto e generale e può essere utilizzato per valutare una vasta gamma di immagini. LIQE è stato addestrato su un dataset di immagini distorte e utilizza una rete neurale convoluzionale per valutare autonomamente la qualità delle immagini.

4.2.4 Considerazioni sulle metriche

Nel progetto, sono state implementate e testate diverse metriche di valutazione senza riferimento, tra cui NIQE, BRISQUE e LIQE, utilizzando la libreria `PyIQA`, che fornisce implementazioni pronte all'uso di diverse metriche di qualità delle immagini.

Dopo aver valutato i risultati ottenuti, è emerso che LIQE è la metrica più adatta per valutare la qualità delle immagini lunari elaborate.

La scelta di LIQE come metrica principale per la valutazione dei risultati è stata motivata da diversi fattori. Innanzitutto, a causa dell'assenza di un'immagine di riferimento ideale, le metriche con riferimento non erano applicabili nel contesto del progetto. Tra le metriche senza riferimento, LIQE ha dimostrato di essere più sensibile ai miglioramenti apportati dalle tecniche di elaborazione implementate.

Durante lo sviluppo, è emerso che metriche come NIQE e BRISQUE non sempre riflettevano accuratamente la percezione umana della qualità delle immagini. Ad esempio, in alcuni casi, un'immagine che visivamente appariva migliorata presentava un punteggio peggiore secondo queste metriche. LIQE, invece, ha mostrato una maggiore correlazione con la qualità percepita.

È importante notare che LIQE non è specificamente progettato per valutare immagini astronomiche. Infatti, il modello importato da PyIQA, normalizzato tra 1 e 5, assegna valori che si aggirano intorno a 2, indicando una qualità molto bassa. Probabilmente ciò è dovuto al fatto che il modello è stato addestrato su un dataset di immagini a colori, mentre la Luna è pressocché monocromatica. Tuttavia, nonostante queste limitazioni, è stato notato che per miglioramenti visivi delle immagini corrispondevano incrementi nei punteggi LIQE.

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

Sviluppi futuri Il progetto ha portato allo sviluppo di una pipeline completa per l'elaborazione delle immagini lunari, ottenendo risultati soddisfacenti. Attraverso l'analisi del lavoro svolto e dei risultati ottenuti, sono state identificate diverse aree chiave che potrebbero beneficiare di ulteriori sviluppi e miglioramenti:

- **Data augmentation**: come già specificato, l'implementazione di un sistema di data augmentation per generare immagini con rumore aggiunto a partire da un'immagine di riferimento di alta qualità potrebbe migliorare la qualità dei risultati e la precisione della pipeline.
- **Ottimizzazione dei parametri**: l'implementazione di un sistema di ricerca automatica dei parametri ottimali, basato su tecniche di ottimizzazione come l'ottimizzazione bayesiana o l'ottimizzazione genetica, potrebbe migliorare la qualità dei risultati e ridurre la dipendenza da parametri manuali.
- **Metriche di valutazione più avanzate**: l'implementazione di metriche di valutazione più avanzate, come l'indice SSIM (Structural Similarity Index) o l'indice PSNR (Peak Signal-to-Noise Ratio) potrebbe fornire una valutazione più accurata della qualità delle immagini, alternativamente si potrebbe pensare di addestrare un modello come LIQE o BRISQUE direttamente con immagini lunari, per ottenere una valutazione più precisa.
- **Interfaccia grafica (GUI)**: l'implementazione di un'interfaccia grafica per la pipeline potrebbe semplificare l'uso del programma. Ciò potrebbe includere funzionalità come la selezione delle immagini, la visualizzazione dei risultati e la regolazione dei parametri mostrando i loro effetti in tempo reale.
- **Parallelizzazione e ottimizzazione del codice**: l'implementazione di tecniche di parallelizzazione e ottimizzazione del codice potrebbe migliorare le prestazioni della pipeline e ridurre i tempi di esecuzione.

Implementando queste funzionalità, si potrebbe ottenere una pipeline più efficiente, precisa e facile da utilizzare, consentendo di ottenere risultati migliori e più affidabili nell'elaborazione di immagini astronomiche, oltre che aumentare la portabilità del programma, rendendolo più accessibile a utenti non esperti.

È importante notare che alcuni di questi miglioramenti, come l'implementazione di generatori per la lettura delle immagini, sono già in fase di sviluppo iniziale.

Tuttavia, necessitano di ulteriore lavoro di implementazione, testing e ottimizzazione per essere integrati completamente nella pipeline operativa.

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.

[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/astronomy-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.