

Politechnika Wrocławskiego
Wydział Informatyki i Telekomunikacji

Kierunek: **Informatyka Techniczna (ITE)**
Specjalność: **Systemy i Sieci Komputerowe (ISK)**

**PRACA DYPLOMOWA
INŻYNIERSKA**

**Desktopowa aplikacja wspomagająca amatorów
astronomii w obróbce zdjęć głębokiego nieba**

Szymon Kłoskowski

Opiekun pracy
prof. dr hab. inż. Jan Magott

Słowa kluczowe: Astrofotografia, Stacking, C++, OpenCV, LibRaw

WROCŁAW 2025

Streszczenie

Dodaj streszczenie pracy w języku polskim. Staraj się uwzględnić wymienione na stronie tytułowej słowa kluczowe. Uwaga przedstawiony rekomendowany szablon dotyczy pracy dyplomowej pisanej w języku angielskim. W przeciwnym wypadku, student powinien samodzielnie zmienić nazwy „Chapter” na „Rodział” itp stosując odpowiednie pakiety systemu L^AT_EXoraz ustawienia w pliku *latex-settings.tex*.

Abstract

Streszczenie w języku angielskim.

Spis treści

1 Wprowadzenie	1
1.1 Opis problemu	1
1.2 Cel pracy	1
1.3 Zarys pracy	1
2 Stan dziedziny	3
2.1 Przegląd literatury	3
2.2 Istniejące narzędzia	3
3 Przetwarzanie zdjęć	5
3.1 Kalibracja	5
3.2 Rejestracja	6
3.3 Stacking	7
4 Opis aplikacji	11
4.1 Wymagania oraz technologie	11
4.2 Architektura systemu	12
4.3 Interfejs użytkownika	12
4.4 Struktura kodu	13
4.5 Testowanie aplikacji	21
5 Zbieranie danych	23
5.1 Opis otoczenia i montażu	23
5.2 Procedura i zestawy danych	24
6 Analiza implementacji algorytmów	27
6.1 Metodyka eksperymentów	27
6.2 Wyniki eksperymentów	29
6.3 Porównanie wyników	29
7 Wnioski	41
Bibliografia	44
Spis rysunków	45
Spis tabel	47

1. Wprowadzenie

Fotografia obiektów głębokiego nieba takich jak galaktyki, mgławice i gromady gwiazd często nazywana astrofotografią jest coraz popularniejszym hobby wśród pasjonatów i amatorów astronomii. Uzyskanie podstawowych fotografii największych i najjaśniejszych obiektów głębokiego nieba nie jest zaawansowaną procedurą, którą możliwe jest wykonać korzystając z telefonu czy prostego aparatu fotograficznego. W celu osiągnięcia lepszej jakości zdjęć i możliwości fotografowania ciemniejszych i mniejszych obiektów głębokiego nieba, wielu amatorów zaczyna korzystać z bardziej zaawansowanych technik, algorytmów oraz sprzętu.

1.1. Opis problemu

1.2. Cel pracy

Niniejsza praca ma na celu wspomaganie tych amatorów zapewniając nowe rozwiązanie programowe, które ułatwia proces obróbki zdjęć obiektów głębokiego nieba. Aplikacja została stworzona z myślą o użytkownikach początkujących i średnio-zaawansowanych, dla których obecne rozwiązania na rynku są zbyt skomplikowane lub kosztowne. Praca omawia również podstawowe techniki obróbki zdjęć obiektów głębokiego nieba oraz praktyczne aspekty tworzenia aplikacji desktopowej.

1.3. Zarys pracy

2. Stan dziedziny

2.1. Przegląd literatury

2.2. Istniejące narzędzia

Na rynku jest dostępnych wiele narzędzi przeznaczonych do obróbki zdjęć obiektów głębokiego nieba. Wiele z nich oferuje wiele zaawansowanych funkcjonalności, ponieważ zostały stworzone z myślą o zaawansowanych użytkownikach. Obsługa tych programów bywa skomplikowana a poznawanie interfejsu użytkownika jest czasochłonne. Poniżej zostaną przedstawione najpopularniejsze z programów używane do obróbkę astrofotografii.

- **DeepSkyStacker**

DeepSkyStacker [2] jest jednym z najbardziej popularnych darmowych narzędzi przeznaczonych do łączenia zdjęć obiektów głębokiego nieba. Pozwala na automatyczne wyrównania klatek, kalibrację przy użyciu klatek kalibracyjnych i łączenie obrazów różnymi metodami. Nie umożliwia jednak użytkownikowi na przetwarzanie zdjęć planet czy bezpośrednią obróbkę w aplikacji. Program oferuje graficzny interfejs użytkownika, który przez dużą ilość swoich możliwości może być mało intuicyjny dla początkujących.

- **PixInsight**

PixInsight [9] to zaawansowane komercyjne narzędzie używane przez profesjonalistów. Zapewnia bardzo szerokie możliwości przetwarzania danych, kalibracji i analizy fotometrycznej. Głównymi wadami programu są wysoka cena oraz wysoka ilość wymaganej wiedzy do prawidłowego obsługiwania programu.

- **Siril**

Siril [10] to darmowe i wieloplatformowe narzędzie do przetwarzania astrofotografii. Oferuje prostszy interfejs niż PixInsight. Jego główną zaletą dla bardziej zaawansowanych użytkowników jest możliwość importowania skryptów do przetwarzania fotografii, dla początkujących fotografów jest to niepotrzebne i zwiększa tylko wymaganą wiedzę do korzystania z podstawowych funkcji programu.

3. Przetwarzanie zdjęć

W celutworzenia obrazu lepszej jakości oraz z niskim poziomem szumu fotografie obiektów głębokiego należy odpowiednio przetworzyć. Przetwarzanie zdjęć można podzielić na trzy główne etapy: kalibrację, rejestrację oraz stacking.

Prawidłowe opisanie i porównanie źródeł szumu na różnych urządzeniach oraz przy różnych warunkach wymaga ustalenia odpowiedniej jednostki. Każdy foton światła padający na piksel detektora w zależności od długości światła powoduje pewną szansę na wygenerowanie elektronu - prawdopodobieństwo uwolnienia się takiego elektronu nazywa się wydajnością kwantową (*ang. Quantum Efficiency*). Standardowo ilość szumu na zdjęciu producenci określają za pomocą ilości elektronów wygenerowanych przy pobieraniu informacji z matrycy fotograficznej [17]. Przydatne informacje dotyczące szumu generowanego na popularnych matrycach aparatów można znaleźć na stronie Photons to Photos [8].

3.1. Kalibracja

Jednym z niezbędnych elementów przetwarzania zdjęć obiektów głębokiego nieba jest zastosowanie klatek kalibracyjnych. Istnieje kilka rodzajów klatek kalibracyjnych, które odpowiadają za korekcję różnych niedoskonałości teleskopu i matrycy. Poniżej opisano najczęściej używane klatki kalibracyjne. Dokładny opis założeń korzystania z klatek kalibracyjnych opisano w książce [17]. Dotyczy ona głównie astrofotografii na aparatach CCD, lecz podstawowe założenia pozostają niezmienne dla nowocześniejszych matryc CMOS.

- **Klatki Bias**

Klatki bias, nazywane też klatkami offset, wykonywane są przy całkowicie zakrytej matrycy i czasie ekspozycji możliwie najkrótszym dla wykorzystywanej matrycy. Należy pamiętać o potrzebie wykonania tych samych klatek dla tych samych ustawień ISO lub Gain (w zależności od zastosowanego rodzaju aparatu) jak główne fotografie obiektu. Stosuje się je w celu usunięcia stałego szumu generowanego przez elektronikę w trakcie odczytu informacji z matrycy. Klatki bias stanowią minimalny poziom szumu zawartego w każdej wykonanej fotografii.

- **Klatki Dark**

Klatki dark wykonuje się przy zakrytej matrycy, w tej samej temperaturze, takim samym czasem ekspozycji i ustawieniami ISO/Gain jak klatki obiektu. Ich głównym zadaniem jest pozbycie się prądu ciemnego (*ang. dark current*), redukcja zjawiska „amp glow”, pozbycie się gorących pikseli i wpływu promieniowania kosmicznego.

- **Klatki Flat**

Klatki flat służą do usunięcia nierównomiernego oświetlenia matrycy (zjawisko winiety) i różnic czułości poszczególnych pikseli. Klatki flat przydają się dodatkowo przy usunięciu artefaktów spowodowanych zanieczyszczeniami na matrycy oraz teleskopie. Wykonywane są przy równomiernym oświetleniu całego pola widzenia - kierując teleskop lub obiektyw na jasne jednolite tło. Podczas kalibracji warto zwrócić uwagę na potrzebę odjęcia bias, przed użyciem klatek flat na głównych fotografiach obiektu.

W celu osiągnięcia wymaganych efektów wykonuje się wiele klatek kalibratorycznych każdego rodzaju a następnie przetwarza się je obliczając średnią wartość na każdym pikselu. Ostateczny teoretyczny opis nałożenia klatek można opisać tak jak w równaniu (3.1)

$$I_{\text{calibrated}} = \frac{I_{\text{light}} - I_{\text{dark}}}{I_{\text{flat}} - I_{\text{bias}}} \quad (3.1)$$

gdzie:

I_{light} – surowa klatka obiektu

I_{dark} – klatka dark

I_{bias} – klatka bias

I_{flat} – klatka flat

3.2. Rejestracja

Rejestracja jest procesem przetwarzania zdjęć głębokiego nieba, która polega na odpowiedniej transformacji fotografii przed nakładaniem zdjęć tak żeby przedstawiały one ten sam fragment nieba, niezależnie od przesunięć spowodowanych różnymi czynnikami mechanicznymi. Proces rejestracji można podzielić na dwa etapy.

Pierwszym z nich będzie identyfikacja elementów, które są pozostają niezmienne na obu fotografiach. Dla astrofotografii elementy te są proste do określenia i są to zazwyczaj gwiazdy widoczne w tle fotografowanego obiektu głębokiego nieba. Do wykrywania tych punktów można wykorzystać algorytmy bazujące na właściwościach lokalnych obiektów. Algorytmy te wykrywają punkty o jednolitej jasności różniące się od otoczenia poprzez analizę jasności, kształtu i rozmiaru obiektów. Oblicza się następnie punkt centralny wykrytych obiektów, żeby można było je stosować jako punkty odniesienia w następnym etapie.

Następnym etapem będzie określenie jaką transformację trzeba dokonać, żeby pożądane zdjęcie nałożyć prawidłowo na fotografię referencyjną. Jednym z algorytmicznych sposobów rozwiązania tego problemu będzie określenie stosunków długości boków dla wszystkich zestawów trzech gwiazd znajdujących się na obu zdjęciach. Należy wtedy znaleźć najbliższe pasujące stosunki trójkątów pomiędzy zdjęciem referencyjnym i pożądanym oraz oszacowanie macierzy transformacji pomiędzy zdjęciami. Jest to metoda uproszczona, inspirowana na bardziej zaawansowanych rozwiązaniach zaproponowanych przez Lin i Xu w artykule [18].

Alternatywą dla tego podejścia jest astrometryczne rozwiązywanie obrazu (ang. *Plate Solving*), polegające na dopasowaniu układu gwiazd z obrazu do katalogów gwiazdnych w celu uzyskania dokładnych parametrów orientacji, skali oraz położenia pola widzenia obrazu względem nocnego nieba [19].

3.3. Stacking

Stacking zdjęć jest procesem przetwarzania cyfrowych zdjęć polegającym na połączeniu wielu fotografii tego samego wycinka nocnego nieba lub wskazanego obiektu głębokiego nieba w celu osiągnięcia większego stosunku sygnału do zakłóceń (ang. *Signal-to-Noise Ratio*, SNR) oraz ogólnej redukcji szumu na końcowym zdjęciu [16]. Relację sygnału do szumu dla pojedynczej ekspozycji przedstawia równanie (3.2):

$$\text{SNR} = \frac{S}{\sqrt{S + \text{Sky} + \text{Dark} + N_{\text{RON}}^2}} \quad (3.2)$$

gdzie:

$S = s \cdot t = s \cdot N_{\text{DIT}} \cdot \text{DIT}$ – całkowity sygnał pochodzący od obiektu [e^-]

Sky – szum tła nieba [e^-]

Dark – szum prądu ciemnego [e^-]

N_{RON} – szum odczytu detektora [e^-]

N_{DIT} – liczba ekspozycji

DIT – czas pojedynczej ekspozycji [s]

s – liczba elektronów zgromadzonych na sekundę od obiektu [e^-]

Dla wielu połączonych ekspozycji SNR wzrasta w przybliżeniu jak pierwiastek liczby klatek, co pokazuje równanie (3.3), gdzie N oznacza liczbę użytych klatek w procesie stackingu.

$$\text{SNR}_{\text{stack}} \approx \sqrt{N} \cdot \text{SNR}_{\text{single}} \quad (3.3)$$

Programy do przetwarzania fotografii obiektów głębokiego nieba oferują wiele algorytmów do tego procesu. Przeanalizowano jednak cztery metody, które reprezentują różne podejścia do poprawy jakości obrazu.

- **Średnia**

Metoda średniej jest jedną z najczęściej stosowanych ze względu na prostotę implementacji oraz niskie wymagania sprzętowe, co stanowi istotną zaletę przy przetwarzaniu dużych zbiorów danych. Każdy piksel wynikowego obrazu jest obliczany jako średnia arytmetyczna wartości odpowiadających pikseli ze wszystkich klatek, jak przedstawiono w pseudokodzie 3.1.

```

1 for each pixel (x, y):
2     sum = 0
3     for each frame in frames:
4         sum = sum + frame[x, y]
5     output[x, y] = sum / number_of_frames

```

Kod Źródłowy 3.1: Pseudokod metody Średnia

- **Medianą**

Metoda mediany, pokazana w pseudokodzie 3.2, opiera się na wybraniu mediany z wartości pikseli w tym samym miejscu dla wszystkich klatek. Dzięki temu skutecznie usuwa anomalie takie jak ślady po satelitach czy pojedyncze piksele zakłóceń.

```

1 for each pixel (x, y):
2     values = []
3     for each frame in frames:
4         values.append(frame[x, y])
5     sort(values)
6     output[x, y] = median(values)

```

Kod Źródłowy 3.2: Pseudokod metody Medianu

- **Kappa-Sigma Clipping**

Metoda Kappa-Sigma Clipping polega na iteracyjnym odrzucaniu wartości pikseli, które odbiegają od średniej o więcej niż κ -krotność odchylenia standardowego. Proces ten pozwala na ograniczenie wpływów odchyleń takich jak samoloty oraz satelity kosmiczne zachowując dobry stosunek sygnału do szumu. Pseudokod tej metody przedstawiono w [3.3.](#)

```

1 for each pixel (x, y):
2     values = []
3     for each frame in frames:
4         values.append(frame[x, y])
5
6     for iteration = 1 to max_iterations:
7         mean = average(values)
8         stddev = standard_deviation(values)
9         inliers = []
10        for each v in values:
11            if abs(v - mean) <= kappa * stddev:
12                inliers.append(v)
13            if inliers = values or inliers is empty:
14                break
15            values = inliers

```

Kod Źródłowy 3.3: Pseudokod metody Kappa-Sigma Clipping

- **Auto Adaptive Weighted Average**

Metoda Auto Adaptive Weighted Average jest algorytmem iteracyjnym, która przypisuje każdemu pikselowi wagę zależną od odchylenia od bieżącej średniej. Wagi są wyznaczane za pomocą funkcji adaptacyjnej kontrolowanej parametrami α oraz β . Metoda pozwala na większą swobodę w implementacji - funkcja adaptacyjna może być zależna nie tylko od wartości pikseli na fotografii ale również od statystycznie obliczonej jakości zdjęcia czy innych wymaganych parametrów. Założenia algorytmu przedstawiono w kodzie [3.4.](#)

```

1 for each pixel (x, y):
2     values = []
3     for each frame in frames:
4         values.append(frame[x, y])
5     mu = median(values)
6
7     for iteration = 1 to max_iterations:
8         weights = []
9         for i = 1 to length(values):
10            w = adaptive_weight(values[i], mu, alpha, beta)
11            weights.append(w)
12
13            total_weight = sum(weights)
14            for i = 1 to length(weights):
15                weights[i] /= total_weight

```

```
17     mu = 0
18     for i = 1 to length(values):
19         mu += weights[i] * values[i]
20
21     output[x, y] = mu
```

Kod Źródłowy 3.4: Pseudokod metody Auto Adaptive Weighted Average

4. Opis aplikacji

W niniejszym rozdziale przedstawiono opis wymagań, architektury i implementacji aplikacji i algorytmów zastosowanych do stworzenia programu desktopowego do obróbki zdjęć obiektów głębokiego nieba.

4.1. Wymagania oraz technologie

Biorąc pod uwagę istniejące technologie i narzędzia wykorzystywane w branży zdecydowano się na określenie następujących wymagań.

Wymagania funkcjonalne

1. Tworzenie nowego obszaru roboczego zawierającego podkatalogi wymagane do kalibracji i stackowania - bias, darks, flats, lights i masters.
2. Tworzenie głównych klatek kalibracyjnych na podstawie dostarczonych przez użytkownika zdjęć kalibracyjnych.
3. Wczytywanie zdjęć w wybranym formacie RAW oraz TIFF.
4. Korekcja zdjęć light z użyciem wcześniej utworzonych głównych klatek kalibracyjnych.
5. Automatyczne wyrównywanie zdjęć na podstawie wykrytych gwiazd.
6. Automatyczne nakładanie zdjęć obiektu głębokiego nieba jedną z czterech wybranych metod: średnia, mediana, kappa-sigma clipping lub auto adaptive weighted average.
7. Zapisywanie obrazu wynikowego w domyślnej lokalizacji, bądź innej wybranej przez użytkownika.

Wymagania niefunkcjonalne

1. Kod źródłowy powinien być napisany w nowoczesnym i popularnym języku programowania.
2. Kod źródłowy powinien być modularny i umożliwiać łatwe rozszerzanie i nowe algorytmy stackowania.
3. Aplikacja powinna być odporna na błędy takie jak brakujące pliki, nieprawidłowe formaty czy niepoprawne ścieżki.
4. Aplikacja powinna działać prawidłowo przy łączeniu sześćdziesięciu zdjęć i poniżej na spręcie średniej klasy.

5. Aplikacja powinna być desktopowa oraz mieć interfejs konsolowy.
6. Aplikacja powinna być przyjazna dla użytkowników mało-zaawansowanych - powinna pozwalać na uzyskanie obrazu wynikowego przy zastosowaniu trzech lub mniejszej ilości komend bez utraty funkcjonalności.
7. Aplikacja powinna umożliwiać łączenie przynajmniej pięciu zdjęć wraz z zbudowanymi głównymi klatkami kalibracyjnymi na dowolnym z dostępnych algorytmów poniżej dziewięćdziesięciu sekund.

Rozważając wymagania programu, zdecydowano się na korzystanie z języka C++ do programowania, gdyż oferuje on wysoką wydajność oraz szerokie możliwość optymalizacji wymagane przy przetwarzaniu dużej ilości danych. Do obsługi i wczytywania fotografii w formacie RAW zostanie użyta biblioteka LibRaw [5], do manipulacji obrazami i utworzenia funkcji kalibracji, detekcji gwiazd, rejestracji i nakładania zdjęć biblioteka OpenCV [7].

4.2. Architektura systemu

4.3. Interfejs użytkownika

Prosty i czytelny interfejs użytkownika jest wymaganą częścią programów zaprojektowanych dla użytkowników początkujących. Poniżej został opisany interfejs użytkownika wraz z krótką instrukcją użytkowania.

Interfejs użytkownika został zaprojektowany upraszczając interakcję pomiędzy użytkownikiem a programem, nie pozabawiając go jednak podstawowych funkcjonalności programu. Na potrzeby obecnego działania programu zdecydowano się na wykorzystanie interfejsu konsolowego. Pozwoliło to na skupienie się na prawidłowym działaniu najważniejszych funkcjonalności programu.

Funkcję programu podzielone na trzy główne elementy:

- Inicjalizacja obszaru roboczego.
- Tworzenie klatek kalibracyjnych.
- Stacking zdjęć obiektów głębokiego nieba.

Proces inicjalizacji programu polega na utworzeniu wykorzystywanej przez program struktury katalogów w wybranym folderze roboczym. Do inicjalizacji stosuje się poniższe polecenie.

```
.\FluxStack --init <workspace>
```

W procesie kalibracji zdjęć łączy się dostarczone przez użytkownika fotografie kalibracyjne w klatki główne. Po wykonaniu tego procesu w katalogu `<workspace>\masters` pojawią się, odpowiednio dla wybranych flag, gotowe klatki kalibracyjne. W przypadku braku flag program spróbuje utworzyć wszystkie trzy główne klatki kalibracyjne. Utworzenie klatek kalibracyjnych nie jest wymagane w dalszych etapach działania programu, przynoszą one

jednak znaczne korzyści dla końcowej jakości wygenerowanego zdjęcia. Do kalibracji używa się poniższe polecenie. Flagi **-d** **-b** **-f** odnoszą się kolejno do utworzenia klatek dark, bias i flat. Prawidłowy sposób sporządzenia tych klatek przybliżono w rozdziale [3](#).

```
.\FluxStack.exe --calibrate <workspace> [-d] [-b] [-f]
```

W procesie stackowania zdjęcia z katalogu **lights** są łączone w obraz wynikowy działania programu korzystając z jednego z możliwych algorytmów. Algorytm wybiera się poprzez flagę **--method**. Dostępne algorytmy to średnia (**average**), mediana (**median**), kappa-sigma clipping (**kappasigma**) oraz auto adaptive weighted average (**adaptive**). Bez podania flagi **--method** program automatycznie wybiera metodę średniej. Dodatkowo użytkownik ma możliwość zapisania pliku wynikowego w wybranej przez niego lokalizacji korzystając z flagi **--out**. Bez podania flagi **--out** obraz zapisuje się w lokalizacji **masters\stacked.tiff**. Poniżej znajduje się schemat polecenia pozwalającego na stacking obrazów.

```
.\FluxStack.exe --stack <workspace>
[--method=<average|median|kappasigma|adaptive>] [--out=<path>]
```

W razie podania nieprawidłowych parametrów czy flag, bądź wyłączeniu programu bez żadnych flag, program przypomina użytkownikowi o możliwych funkcjach programu.

4.4. Struktura kodu

Kod źródłowy został zorganizowany modularnie z możliwością dodania nowych komponentów i rozszerzenia funkcjonalności programu w przyszłości. Poniżej znajduje się ogólny opis struktury programu wraz z częściami rzeczywistego kodu źródłowego programu z wyjaśnieniami.

Główna część programu została rozbita na pliku źródłowe zgodnie z grafiką poniżej.

```
src/
└── utils/
    ├── ImageIO.cpp
    └── ImageIO.hpp
    ├── App.cpp
    ├── App.hpp
    ├── Calibrator.cpp
    ├── Calibrator.hpp
    ├── ImageProcessor.cpp
    └── ImageProcessor.hpp
    ├── main.cpp
    ├── Register.cpp
    ├── Register.hpp
    ├── Stacker.cpp
    └── Stacker.hpp
```

Klasa **ImageIO** jest odpowiedzialna za czytanie oraz zapisywanie plików użytych przez program. Do wczytywania plików RAW stosowanych przez aplikację użyta została biblioteka

LibRaw. Stosowane do testowania zdjęcia są w formacie 14-bitowym .NEF jednak programowo stosuje się 16-bitowe kontenery do przechowywania informacji o pikselach. Przy wczytywaniu zdjęć w celu uzyskania fotografii bez skalowania i innych domyślnych opcji mogących wpływać na wartości pikseli na zdjęciu należy zastosować odpowiednie parametry. Do wczytywania innych plików takich jak fotografie kalibracyjne, zapisywane w formacie .tiff, użyto wbudowanego narzędzia z biblioteki OpenCV. Kod Źródłowy 4.1 przedstawia fragment implementacji funkcji do wczytywania pojedynczego zdjęcia - przedstawiono parametry użyte do prawidłowego wczytywania plików RAW. Do wczytywania wielu zdjęć stworzono nową funkcję, która obsługuje wczytywanie wielu zdjęć jednocześnie dzięki wielowątkowości.

```

1 rawProcessor.imgdata.params.output_bps = 16;
2 rawProcessor.params.no_auto_bright = 1;
3 rawProcessor.params.use_camera_wb = 0;
4 rawProcessor.params.use_auto_wb = 0;
5 rawProcessor.params.gamm[0] = 1.0f;
6 rawProcessor.params.gamm[1] = 1.0f;
7 rawProcessor.params.user_mul[0] = 1.0f;
8 rawProcessor.params.user_mul[1] = 1.0f;
9 rawProcessor.params.user_mul[2] = 1.0f;
10 rawProcessor.params.user_mul[3] = 1.0f;
```

Kod Źródłowy 4.1: Fragment funkcji `load` z klasy `ImageIO`

Klasa App pełni rolę modułu sterującego działanie aplikacji. Klasa została napisana jako fasada ukrywająca wewnętrzną złożoność użytych komponentów.

Klasa Calibrator odpowiada za operacje związane z przygotowaniem klatek kalibracyjnych oraz korekcję obrazów użytych przez użytkownika. Zawiera funkcje tworzące trzy użyte w programie klatki kalibracyjne i nałożenie ich na zdjęcia przy procesie kalibracji. Klatki kalibracyjne są tworzone poprzez nakładanie fotografii metodą mediany następnie ich odpowiednią normalizacją.

Klasa ImageProcessor gromadzi funkcje związane z podstawowym przetwarzaniem obrazów. Klasa zawiera funkcje odpowiedzialne za korektę balansu bieli oraz przygotowaniem obrazu do wykrywania zdjęć poprzez wyodrębnienie tła obrazu.

Klasa Register odpowiada za elementy rejestracji obrazu jakimi są detekcja gwiazd na fotografii oraz obliczenie odpowiednich transformacji przed nałożeniem zdjęć. Do wykrywania gwiazd na fotografiach użyto wbudowanej w bibliotekę OpenCV funkcji `SimpleBlobDetector` z odpowiednimi parametrami przedstawionymi w kodzie źródłowym 4.2. Przed wykryciem gwiazd obraz należało przekonwertować na obraz ośmioróżkowy z jednym kanałem - zdecydowano się na użycie kanału zielonego, gdyż oferuje on największą ilość informacji w klasycznych matrycach z filtrem Bayer [14]. Parametry zostały dobrane empirycznie na podstawie danych testowych - planowana liczba wykrytych gwiazd wynosiła od 50 do 150 co pozwala na wysokie prawdopodobieństwo pokrycia się przynajmniej trzech obiektów wymaganych do obliczenia transformacji obrazu. Parametr `starDetectionThreshold` został ustawiony na 0,7.

```

1 params.filterByArea = true;
2 params.minArea = 30;
3 params.maxArea = 200;
4 params.filterByCircularity = true;
5 params.minCircularity = static_cast<float>(starDetectionThreshold);
6 params.filterByInertia = true;
7 params.minInertiaRatio = static_cast<float>(starDetectionThreshold);
8 params.filterByConvexity = true;
9 params.minConvexity = static_cast<float>(starDetectionThreshold);
10 params.filterByColor = true;
```

```
11 | params.blobColor = 255;
```

Kod Źródłowy 4.2: Fragment funkcji `detectStars` z klasy Register

Obliczenie odpowiednich transformacji jest drugim etapem po rozpoznaniu gwiazd pozwalający na prawidłowe nakładanie zdjęć. Do obliczenia przesunięcia zdjęć zastosowano transformację afiniczną opartą za geometrii trójkątów i ich niezmiennikach. Dla zdjęcia referencyjnego należy najpierw wyznaczyć wszystkie możliwe trójkąty punktów. Zastosowano jednak ograniczenie do maksymalnie 5000 trójkątów w celu ograniczenia wymagań sprzętowych - większa ilość nie wpływa znacząco na jakość końcowej transformacji. Dla każdego trójkąta wyznaczane są dwa niezmienniki - stosunek średniego boku do najkrótszego oraz stosunek najdłuższego boku do najkrótszego. Następnie te same informacje są przetwarzane dla każdego zdjęcia docelowego. Porównuje się następnie trójkąty referencyjne z tymi z obrazu docelowego. Uznaje się, że trójkąt docelowy jest zgodny z trójkątem docelowym jeśli jego niezmienniki różnią się o mniej niż ustalona tolerancja `ratioTol = 0,03`. Do oszacowania transformacji po dopasowaniu trójkątów wyznaczana jest transformacja affine jak przedstawiono w równaniu (4.1). Programowo wykorzystano implementację `estimateAffinePartial2D` z biblioteki OpenCV.

$$T(x) = A \cdot x + b \quad (4.1)$$

W celu ustalenia jakości transformacji program wykonuje tą transformację na punktach a następnie porównuje położenie gwiazd. Jeżeli odpowiadające sobie gwiazdy leżą po transformacji w odległości mniejszej niż tolerancja `inlierTol2 = 1.0` pikseli to program zalicza je jaką prawidłową transformację. Poprawność transformacji bada się dla każdego punktu i jeżeli ilość odpowiadających gwiazd jest większa niż 80% gwiazd znalezionych na obrazie referencyjnym uznaje się, że transformacja jest prawidłowa. Takie podejście zapewnia dobrą jakością obraz końcowy jest jednak wymagające obliczeniowo. Prawidłowe oszacowanie transformacji jest jednak niezbędnym elementem to uzyskania wysokiej jakości zdjęcia.

W kodzie źródłowym 4.3 przedstawiono fragment kodu użytego w programie odpowiadający za dopasowanie trójkątów pomiędzy obrazem referencyjnym a obrazem docelowym oraz oszacowanie transformacji affine na podstawie zgodynych trójkątów gwiazd.

```

1 for (const auto& tri : refTriangles) {
2     if (std::abs(tri.ratio1 - r1t) < ratioTol && std::abs(tri.ratio2 - r2t) < ratioTol) {
3         // estimate affine from these three correspondences
4         std::array<cv::Point2f,3> refTri = { refStars[tri.i], refStars[tri.j], refStars[tri.k] };
5         std::array<cv::Point2f,3> tgtTri = { targetStars[i], targetStars[j], targetStars[k] };
6         std::vector<cv::Point2f> refVec(refTri.begin(), refTri.end());
7         std::vector<cv::Point2f> tgtVec(tgtTri.begin(), tgtTri.end());
8         cv::Mat affine = cv::estimateAffinePartial2D(tgtVec, refVec);
9         if (affine.empty()) continue;
10
11         double a00 = affine.at<double>(0,0), a01 = affine.at<double>(0,1), a02 =
12             ↪ affine.at<double>(0,2);
13         double a10 = affine.at<double>(1,0), a11 = affine.at<double>(1,1), a12 =
14             ↪ affine.at<double>(1,2);
15
16         int inliers = 0;
17         for (const auto& pt : targetStars) {
18             double tx = a00 * pt.x + a01 * pt.y + a02;
19             double ty = a10 * pt.x + a11 * pt.y + a12;
20             for (const auto& refPt : refStars) {
21                 double dx = tx - refPt.x;
22                 double dy = ty - refPt.y;
23             }
24         }
25     }
26 }
```

```

21     if (dx*dx + dy*dy < inlierTol2) { ++inliers; break; }
22   }
23 }
24
25 if (inliers > bestScore) {
26   bestScore = static_cast<float>(inliers);
27   bestAffine = affine;
28   if (inliers > 0.8f * static_cast<float>(refStars.size())) {
29     return bestAffine;
30   }
31 }
32 }
33 }
```

Kod Źródłowy 4.3: Fragment funkcji `estimateTransformWithRefTriangles` z klasy Register

Klasa Stacker zawiera wszystkie dostępne metody nakładania zdjęć - średnią, medianę, kappa-sigma clipping oraz auto adaptive weighted average. Każda z metod nakładania zdjęć została rozbita na osobną funkcję.

Podstawową metodą nakładania zdjęć jest średnia arytmetyczna, której implementację przedstawiono w kodzie źródłowym 4.4. Każde zdjęcie jest programowo konwertowane do 64-bitowej głębi koloru w celu zminimalizowania utraty danych podczas sumowania. Obraz wynikowy jest następnie przekształcany do formatu 16-bitowego, który stanowi finalny efekt procesu. Metoda średniej, choć najmniej wymagająca obliczeniowo spośród omawianych, przy niewielkiej liczbie klatek nie zapewnia skutecznego odrzucania niepożądanych pikseli, na przykład pochodzących od przelatujących samolotów lub innych artefaktów. Stanowi jednak dobry wzór dla uzyskania dobrego stosunku sygnału do szumu. Złożoność obliczeniowa tego algorytmu jest liniowa.

```

1 cv::Mat Stacker::stackAverage(const std::vector<cv::Mat>& images) {
2   if (images.empty()) return cv::Mat();
3
4   cv::Mat sum = cv::Mat::zeros(images[0].size(), CV_64FC3);
5   for (const auto& img : images) {
6     cv::Mat img64;
7     img.convertTo(img64, CV_64FC3);
8     sum += img64;
9   }
10  sum /= static_cast<double>(images.size());
11
12  cv::Mat result;
13  sum.convertTo(result, CV_16UC3);
14  return result;
15 }
```

Kod Źródłowy 4.4: Funkcja `stackAverage` z klasy Stacker

Drugą z zastosowanych metod nakładania obrazów jest mediana, której implementację przedstawiono w kodzie źródłowym 4.5. W przeciwnieństwie do metody średniej charakteryzuje się skuteczniejszym odrzucaniem artefaktów. Dla każdego piksela i każdego kanału koloru zbierane są wartości ze wszystkich klatek i następnie wybierana jest wartość środkowa z nich. Wymaganie posortowania w naiwnym algorytmie odnalezienia mediany powoduje, że złożoność obliczeniowa tego algorytmu jest logarytmiczna zgodnie z równaniem (4.2).

$$T(M, N) = O(M \cdot N \log N) \quad (4.2)$$

gdzie,

$M = w \cdot h \cdot c$ - liczba próbek (pixeli dla każdego kanału zdjęcia)

N – liczba klatek

```

1 cv::Mat Stacker::stackMedian(const std::vector<cv::Mat>& images) {
2     if (images.empty()) return cv::Mat();
3
4     int rows = images[0].rows;
5     int cols = images[0].cols;
6     int channels = images[0].channels();
7     int nImg = static_cast<int>(images.size());
8
9     cv::Mat medianImage(images[0].size(), images[0].type());
10
11    // For each channel
12    for (int c = 0; c < channels; ++c) {
13        // Prepare a stack for this channel
14        std::vector<cv::Mat> channelStack(nImg);
15        for (int i = 0; i < nImg; ++i) {
16            cv::extractChannel(images[i], channelStack[i], c);
17        }
18
19        // Stack into a 3D array for median computation
20        cv::Mat stack3d(rows, cols, CV_16UC(nImg));
21        for (int i = 0; i < nImg; ++i) {
22            for (int y = 0; y < rows; ++y) {
23                const ushort* src = channelStack[i].ptr<ushort>(y);
24                ushort* dst = stack3d.ptr<ushort>(y) + i;
25                for (int x = 0; x < cols; ++x) {
26                    dst[x * nImg] = src[x];
27                }
28            }
29        }
30
31        // Compute median for each pixel
32        cv::Mat medianChannel(rows, cols, CV_16U);
33        std::vector<ushort> pixelVals(nImg);
34        for (int y = 0; y < rows; ++y) {
35            for (int x = 0; x < cols; ++x) {
36                for (int i = 0; i < nImg; ++i) {
37                    pixelVals[i] = stack3d.at<ushort>(y, x * nImg + i);
38                }
39                std::nth_element(pixelVals.begin(), pixelVals.begin() + nImg / 2, pixelVals.end());
40                medianChannel.at<ushort>(y, x) = pixelVals[nImg / 2];
41            }
42        }
43
44        cv::insertChannel(medianChannel, medianImage, c);
45    }
46
47    return medianImage;
48}

```

Kod Źródłowy 4.5: Funkcja `stackMedian` z klasy `Stacker`

Kolejną metodą wykorzystywaną w programie, przedstawioną w kodzie źródłowym 4.6, jest metoda nakładania zdjęć Kappa-Sigma Clipping. Zaimplementowana funkcja stosowana jest do odrzucania wartości odstających na podstawie danych statystycznych wartości sygnału. Dla każdego piksela obliczana jest średnia oraz odchylenie standardowe, po czym wartości znajdujące się poza zakresem przedstawionym na równaniu (4.3). Algorytm jest iteracyjny, więc po każdym odrzuceniu obliczane są nowe wartości średniej i odchylenia standardowego stabilizując wynik.

$$\mu \pm \kappa \cdot \sigma \quad (4.3)$$

gdzie,

μ - średnia jasność piksela

κ – parametr odchyleń od wartości średniej

σ – odchylenie standardowe jasności piksela

Metoda pozwala na efektywne usunięcie artefaktów ze zdjęcia, lecz w przeciwnieństwie do algorytmu mediany nie traci przy tym na końcowym stosunku ilości sygnału do szumu. Dodatkową zaletą jest również brak ograniczeń precyzji do pojedynczej wartości lecz uśrednienie pikseli, które zostały po odrzuceniu. W przeciwnieństwie do poprzednich dwóch algorytmów należy podać odpowiednie parametry działania programu - dopuszczalne odchylenie standardowe oraz ilość wymaganych iteracji na obrazie.

Implementacja programu posiada dodatkową optymalizację - w razie braku odcięcia pikseli algorytm zaprzestaje działanie przedwcześnie, unika tym niepotrzebnego liczenia bez zmiany końcowego wyniku. Ostatecznym zabezpieczeniem jest ochrona przed odcięciem wszystkich wartości - program wraca wtedy do poprzednich wartości i zaprzestaje odcinania. Podczas działania algorytmu ustawiono wartości `kappa = 3` oraz `maxIterations = 5`.

```

1 cv::Mat Stacker::stackKappaSigmaClipping(const std::vector<cv::Mat>& images, float kappa,
2   ↪ int maxIterations) {
3     if (images.empty()) return cv::Mat();
4
5     int rows = images[0].rows;
6     int cols = images[0].cols;
7     int channels = images[0].channels();
8     int nImgs = static_cast<int>(images.size());
9
10    cv::Mat result(images[0].size(), images[0].type());
11
12    // For each channel
13    for (int c = 0; c < channels; ++c) {
14      // Prepare a stack for this channel
15      std::vector<cv::Mat> channelStack(nImgs);
16      for (int i = 0; i < nImgs; ++i)
17        cv::extractChannel(images[i], channelStack[i], c);
18
19      cv::Mat outChannel(rows, cols, CV_16U);
20
21      // For each pixel
22      for (int y = 0; y < rows; ++y) {
23        for (int x = 0; x < cols; ++x) {
24          std::vector<float> vals(nImgs);
25          for (int i = 0; i < nImgs; ++i)
26            vals[i] = channelStack[i].at<ushort>(y, x);
27
28          for (int iter = 0; iter < maxIterations; ++iter) {
29            // Compute mean and stddev
30            float sum = 0, sum2 = 0;
31            for (float v : vals) { sum += v; sum2 += v * v; }
32            float mean = sum / vals.size();
33            float stddev = std::sqrt(sum2 / vals.size() - mean * mean);
34
35            // Clip values outside mean +- kappa * stddev
36            std::vector<float> newVals;
37            for (float v : vals)
38              if (std::abs(v - mean) <= kappa * stddev)
39                newVals.push_back(v);
40
41        }
42      }
43    }
44  }

```

```

40     if (newVals.size() == vals.size()) break; // No more outliers
41     if (newVals.empty()) break; // All clipped, fallback to previous
42     vals = std::move(newVals);
43   }
44
45   // Output mean of remaining values
46   double sum = 0;
47   for (double v : vals) sum += v;
48   outChannel.at<ushort>(y, x) = static_cast<ushort>(sum / vals.size() + 0.5);
49 }
50 }
51
52 cv::insertChannel(outChannel, result, c);
53 }
54
55 return result;
56 }
```

Kod Źródłowy 4.6: Funkcja `stackKappaSigmaClipping` z klasy `Stacker`

Ostatnią implementowaną metodą jest auto adaptive weighted average, której implementacje przedstawiono w kodzie źródłowym 4.7. Metoda została oparta na procedurach opisanych w pracy Stetsona poświęconej technikom obróbki fotografii na matrycach CCD [20].

W implementacji każdy wykonany pomiar (wartość piksela) otrzymuje wagę zależną od tego, jak bardzo odbiega od wartości oczekiwanej. Metoda adaptacyjna stopniowa zmniejsza wpływ pikseli odstających poziomem szumu czy takich, które posiadają inne artefakty. Podstawowym elementem działania programu jest zastosowana funkcja ważenia przedstawiona w równaniu (4.4).

$$w_i = \frac{1}{1 + (\frac{|r_i|}{\alpha})^\beta} \quad (4.4)$$

gdzie,

α - parametr regulujący czułość na odchylenia

β – parametr regulujący szybkość spadku

r_i – różnica pomiędzy wartością piksela a aktualnym oszacowaniem sygnału

w_i - przydzielona waga

Jako początkowe oszacowanie wartości pikseli używa się mediany wartości pikseli na każdym kanale. Dla każdej iteracji działania programu obliczana jest waga i aktualizowana jest nowa wartość. Dzięki braku odrzucania wartości jak w algorytmie Kappa-Sigma Clipping zmniejszanie wpływu nietypowych pikseli jest łagodniejsze a zachowanych jest więcej szczegółów niż w przypadku użycia algorytmów mediany. Istnieje również większa elastyczność, ponieważ można używać różne funkcje ważenia i stosować parametry α oraz β .

Wymaganie pamięciowe podstawowej implementacji algorytmu są jednak nieakceptowalne i wpływają znacząco dla czas działania programu spowodowane narzutami korzystania z pamięci wirtualnej komputera. Żeby spełnić wymaganie niefunkcjonalne 7 i zwiększyć stabilność działania programu dla większej ilości zdjęć jak opisano w wymaganiu niefunkcjonalnym 4 należało zaimplementować optymalizacje pamięciowe programu. W tym celu zastosowano technikę ograniczającą ilość załadowanych jednocześnie elementów do pamięci RAM dzięki podziale obrazu na mniejsze fragmenty i przetwarzaniu ich sekwencyjnie. W implementacji obliczana jest maksymalna liczba wierszy `tileH` zgodnie z ograniczeniem pamięciowym

`maxBytes` wyznaczonym na 64MiB. Każdy kawałek jest konwertowany do dokładniejszego formatu i zwalniany przed wczytaniem kolejnego do pamięci.

```

1 cv::Mat Stacker::stackAutoAdaptiveWeightedAverage(const std::vector<cv::Mat>& frames, float
2   ↪ alpha, float beta, int iterations) {
3     if (frames.empty()) return cv::Mat();
4
5     const int rows = frames[0].rows;
6     const int cols = frames[0].cols;
7     const int nImgs = static_cast<int>(frames.size());
8     const int channels = 3;
9
10    // bytes per row across all images
11    const uint64_t bytesPerRowAllImgs = (uint64_t)cols * (uint64_t)channels * sizeof(float) *
12      ↪ (uint64_t)nImgs;
13    const uint64_t maxBytes = 64ULL * 1024ULL * 1024ULL;
14
15    int tileH = static_cast<int>(std::max<uint64_t>(1, std::min<uint64_t>((uint64_t)rows,
16      ↪ maxBytes / std::max<uint64_t>(1, bytesPerRowAllImgs))));
17    if (tileH <= 0) tileH = 1;
18
19    bool tiled = (tileH < rows);
20    cv::Mat result(frames[0].size(), CV_32FC3);
21
22    // Reusable temporaries
23    std::vector<float> vals(nImgs);
24    std::vector<float> residuals(nImgs);
25    std::vector<float> weights(nImgs);
26    std::vector<float> tmpVec;
27    tmpVec.reserve(nImgs);
28    std::vector<cv::Mat> tile32;
29    tile32.resize(nImgs);
30
31    // Process tiles
32    for (int y0 = 0; y0 < rows; y0 += tileH) {
33      int y1 = std::min(rows, y0 + tileH);
34      int curH = y1 - y0;
35
36      for (int i = 0; i < nImgs; ++i) {
37        cv::Mat srcTile = frames[i].rowRange(y0, y1);
38        srcTile.convertTo(tile32[i], CV_32FC3);
39        if (!tile32[i].isContinuous()) tile32[i] = tile32[i].clone();
40      }
41
42      // process each row within the current tile
43      for (int ty = 0; ty < curH; ++ty) {
44        std::vector<const float*> rowPtrs(nImgs);
45        for (int i = 0; i < nImgs; ++i) rowPtrs[i] = tile32[i].ptr<float>(ty);
46
47        float* outRow = result.ptr<float>(y0 + ty);
48
49        for (int x = 0; x < cols; ++x) {
50          const int base = x * channels;
51
52          for (int c = 0; c < channels; ++c) {
53            int m = 0;
54            for (int i = 0; i < nImgs; ++i)
55              vals[m++] = rowPtrs[i][base + c];
56
57            // median initial guess
58            tmpVec.assign(vals.begin(), vals.begin() + m);
59            std::nth_element(tmpVec.begin(), tmpVec.begin() + (m / 2), tmpVec.end());
60            float mu = tmpVec[m / 2];
61
62            // iterative reweighting
63            for (int it = 0; it < iterations; ++it) {
64              float wsum = 0.0f;

```

```
63     for (int i = 0; i < m; ++i) {
64         residuals[i] = vals[i] - mu;
65         float w = 1.0f / (1.0f + std::powf(std::abs(residuals[i]) / alpha, beta));
66         weights[i] = w;
67         wsum += w;
68     }
69
70     if (wsum == 0.0f) break;
71
72     float new_mu = 0.0f;
73     for (int i = 0; i < m; ++i) {
74         weights[i] /= wsum;
75         new_mu += weights[i] * vals[i];
76     }
77
78     mu = new_mu;
79 }
80
81     outRow[base + c] = mu;
82 }
83 }
84 }
85
86 // release tile buffers early if large
87 for (int i = 0; i < nImg; ++i) tile32[i].release();
88 }
89
90 // convert back to original type
91 cv::Mat out;
92 result.convertTo(out, frames[0].type());
93 return out;
94 }
```

Kod Źródłowy 4.7: Funkcja `stackAutoAdaptiveWeightedAverage` z klasy Stacker

4.5. Testowanie aplikacji

5. Zbieranie danych

Dane potrzebne do prawidłowego testowania zostały wykonane samodzielnie, poniżej opisano warunki w jakich wykonano zdjęcia oraz przybliżono pojęcia przydatne przy zbieraniu danych. Pokazano również jak przygotowano zestawy danych do badania działania aplikacji.

5.1. Opis otoczenia i montażu

Zebranie przydanych danych w trakcie sesji fotograficznej wymaga odpowiednich warunków obserwacyjnych. Najważniejszymi aspektami jest stan pogody w miejscu fotografowania, jasność nieba oraz jakość widzenia. Sesje zostały przeprowadzone podczas bezchmurnych godzin dnia podczas nocy astronomicznej. Nocą astronomiczną określa się porę dnia, podczas której Słońce jest poniżej 15 stopni pod horyzontem. W celu ograniczenia osiadania się wilgoci na przedniej części obiektywu sesje zostały wykonywane w warunkach poniżej 80 procent wilgotności powietrza zgodnie z lokalną prognozą pogody. Dodatkowym zabezpieczeniem było użycie ocieplacza przedniego elementu obiektywu.

W celu określenia jasności nieba wielu amatorów astronomii odnosi się to tak zwanej Skali Bortle'a. Treść oryginalnego artykułu można znaleźć na stronie [13]. Skala Bortle'a to skala, która określa jasność nocnego nieba w od 1 do 9, gdzie 1 to najciemniejsze nocne nieba a 9 to najjaśniejsze. Skala opisuje jakie są najciemniejsze elementy nocnego nieba, które człowiek jest w stanie zaobserwować gołym okiem.

Kolejnym elementem ważnym dla astrofotografii jest widzenie astronomiczne. Widzenie astronomiczne jest miarą określającą poziom deformacji obrazu spowodowanego turbulencjami powietrza w atmosferze [12]. Obiekty takie jak ELT (*Extremely Large Telescope*) czy inne profesjonalne teleskopy są budowane na wysokościach geograficznych ograniczający wpływ gęstej atmosfery Ziemi na jakość badań.

W celu zminimalizowania wpływu szumu odczytu oraz zmniejszenia ilości danych do przetworzenia stosuje się śledzenie i sterowanie zestawu optycznego aparatu. Źeby móc prawidłowo wykorzystać narzędzia służące do śledzenia i sterowania, zestaw optyczny trzeba ustawić równolegle do osi obrotu Ziemi. Takie wyrównanie z biegunem (*ang. polar alignment*) w przeciwie dla początkujących przeprowadza się samemu korzystając z teleskopu [4], jednak proces ten w głowicach wyższej klasy został zautomatyzowany.

Śledzenie w kontekście astrofotografii odnosi się do śledzenia ruchu gwiazd na nocnym niebie - pozwala to na wykonanie fotografii o dłuższym czasie naświetlania. Bez śledzenia, fotografie o odpowiednio długim czasie naświetlania, będą powodowały rozmycie się gwiazd na fotografii. Sterowanie tym śledzeniem przy zastosowaniu drugiego teleskopu pozwala na wprowadzanie odpowiednich korekt do głowicy tym samym zwiększać precyzję śledzenia.

5.2. Procedura i zestawy danych

Do wykonania fotografii do badania i testowania aplikacji zastosowano następujący sprzęt:

- Lustrzanka Nikon D800 - z modyfikacją pozwalającą na zwiększenie sygnału docierającego do matrycy z zakresu bliskiego IR.
- Obiektyw Samyang 135mm F2.0.
- Główica paralaktyczna Sky-Watcher Star Adventurer z przeciwwagą.
- Statyw fotograficzny.
- Ocieplacz przedniego elementu obiektywu.

Procedura wykorzystywana do przechwytywania zdjęć obiektów głębokiego nieba wyglądała następująco.

1. Przeniesienie sprzętu fotograficznego wraz z zasileniem w miejsce niedostępne dla okolicznych zanieczyszczeń świetlnych.
2. Założenie głowicy na statyw fotograficzny oraz dołączenie aparatu z obiektywem do głowicy z przeciwwagą.
3. Wstępne ustawienie prawidłowego wyrównania z biegunem głowicy.
4. Ustawienie aparatu z przeciwwagą do zbalansowanego stanu.
5. Ustawienie aparatu na określoną część nieba.
6. Nałożenie poprawek na wyrównanie z biegunem.
7. Końcowe ustawienie prawidłowych opcji aparatu. Znalezienie ostrości aparatu na gwiazdach.
8. Wykonanie fotografii testowych.
9. Poprawki po fotografii testowej.
10. Włączenie aparatu na wykonanie odpowiedniej ilości zdjęć.
11. Po zakończeniu głównej części wykonywano zdjęcia kalibracyjne (opcjonalnie).

Proces ten zapewnił, że zdjęcia będą w odpowiedniej jakości. Przy nieprawidłowym wyrównaniu głowicy z biegunem lub innych niedopracowaniach mechanicznych programy obsługujące obróbkę zdjęć mogą nie przetworzyć zdjęć prawidłowo.

Korzystając z wyżej wymienionej procedury przygotowano dwa główne zestawy zdjęć, które zostaną następnie podzielone do badań. Dla jednoznacznego nazwania obiektów korzystano z katalogu NGC (*New General Catalogue*) w najnowszej udostępnionej wersji [11]. Podzielone zestawy zostały nazwane zgodnie z podanym schematem: *Zestaw numer z katalogu NGC_czas*

integracji zdjęcia [min]. W tabeli 5.1 opisano podzielone zestawy zdjęć. Określenie jasności nieba w skali Bortle'a zostało wykonane korzystając ze strony [6] i następnie zaokrąglone do pełnej wartości. Czasem integracji określa się całkowity czas naświetlania wszystkich połączonych fotografii.

Zestawy rozpoczynające się od NGC224 oznaczają fotografie obiektu NGC224 znanego jako galaktyka Andromedy znajdującym się w gwiazdozbiorze Andromedy. Fotografie obiektu zostały wykonane pod niebem o jasności 3 w skali Bortle'a. Zostało wykonanych 30 fotografii, każda z ekspozycji trwała 120 sekund. Przesłona obiektywu została ustawiona na F2.0. ISO aparatu fotograficznego zostało ustawione na 800. Nie wykonano zdjęć kalibracyjnych.

Zestawy rozpoczynające się od NGC1499 oznaczają fotografie obiektu NGC1499 znanego jako mgławica Kalifornia znajdującym się w gwiazdozbiorze Perseusza. Fotografie obiektu zostały wykonane pod niebem o jasności 6 w skali Bortle'a. Zostało wykonanych 60 fotografii, każda z ekspozycji trwała 60 sekund. ISO aparatu fotograficznego zostało ustawione na 800. Przesłona obiektywu została ustawiona na F2.0. Dodatkowo wykonano pełny zestaw zdjęć kalibracyjnych.

Nazwa	Obiekt	Czas integracji [s]	Zdjęcia kalibracyjne
Zestaw224_10	NGC224	600	NIE
Zestaw224_20	NGC224	1200	NIE
Zestaw224_30	NGC224	1800	NIE
Zestaw224_40	NGC224	2400	NIE
Zestaw224_50	NGC224	3000	NIE
Zestaw224_60	NGC224	3600	NIE
Zestaw1499_10	NGC1499	600	TAK
Zestaw1499_20	NGC1499	1200	TAK
Zestaw1499_30	NGC1499	1800	TAK
Zestaw1499_40	NGC1499	2400	TAK
Zestaw1499_50	NGC1499	3000	TAK
Zestaw1499_60	NGC1499	3600	TAK

Tabela 5.1: Opis podzielonych zestawów fotografii obiektów głębokiego nieba

6. Analiza implementacji algorytmów

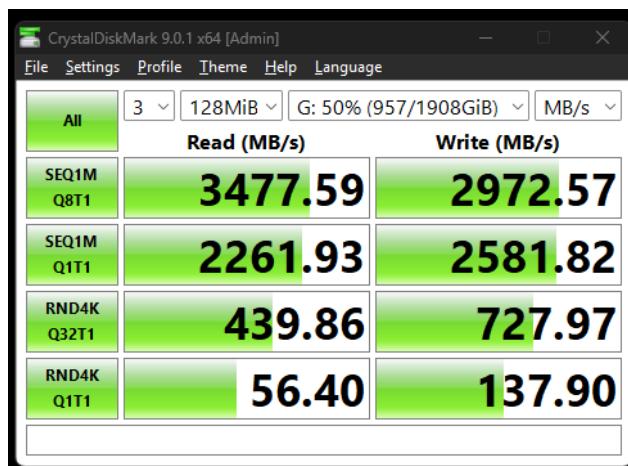
Zostały przeprowadzone badania dotyczące jakości zdjęć końcowych dla zastosowanych algorytmów nakładania zdjęć oraz ich czasu działania. Celem analizy jest określenie, który z algorytmów należy stosować dla uzyskania najlepszej ostrości zdjęcia oraz czasu działania.

6.1. Metodyka eksperymentów

Badania przeprowadzano na komputerze stacjonarnym z systemem operacyjnym Windows 11. Przed wykonywaniem badań zakończono wszystkie zbędne procesy w celu zwiększenia stabilności działania programu. Badania zostały przeprowadzone na komputerze stacjonarnym z podanymi podzespołami:

- Intel Core i7-8700
- Pamięć RAM DDR4 2x8GB 2666MT/s
- Dysk SSD NVME PCI-Express x4 gen. 3.

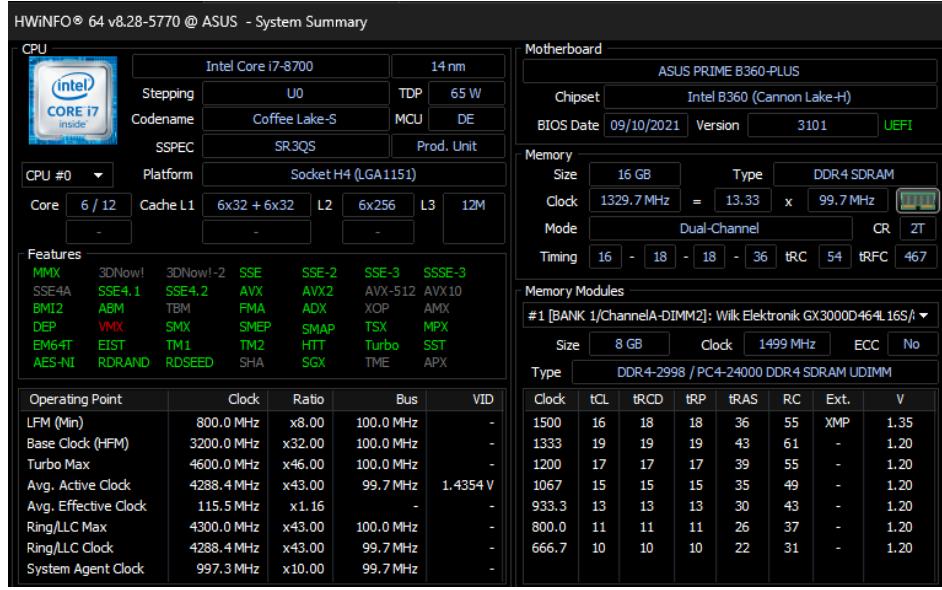
Przekroś odczytu danych z dysku może znacząco wpływać na czas działania programu. W celu realistycznego przedstawienia czasu działania programu zdecydowano się więc na przeprowadzenie badań dotyczących prędkości używanego dysku. Do badań dysku SSD użyto darmowego programu CrystalDiskMark dostępnego na oficjalnej stronie producenta [1]. Wyniki badań prędkości dysku przedstawiono na rysunku 6.1. Program bada prędkość sekwencyjnego i losowego odczytu i zapisu w różnych warunkach.



Rysunek 6.1: Wyniki testu wydajności dysku SSD.

W trakcie badań monitorowano stan podzespołów komputera przy użyciu aplikacji HWiNFO [3], nie zaobserwowano żadnych ograniczeń termicznych komponentów podczas

działania programu w trakcie badań. Pozostałe informacje na temat systemu umieszczone na rysunku 6.2.



Rysunek 6.2: Zrzut ekranu przedstawiający użyty sprzęt.

Z dwóch obiektów astronomicznych wykonano serie fotografii, sposób wykonania fotografii i ich dokładny opis znajduje się w rozdziale 5. Fotografie podzielono na zestawy do badań według tabeli 5.1 opisanej w rozdziale 5. Każdy z zestawów przetwarzano czterema zaimplementowanymi algorytmami - ich dokładna implementacja znajduje się w podrozdziale 4.4. W każdym z badanych algorytmów badano czas działania przy użyciu biblioteki `std::chrono`. Badania jakości wykonano na podstawie zaimplementowanej metryki - szerokości połówkowej gwiazd.

Szerokość połówkowa (*full width at half maximum, FWHM*) to metryka opisująca szerokość profilu gwiazdy w połowie jej maksymalnej jasności, w przypadku implementacji wyznaczana w pikselach. Niższe wartości FWHM oznaczają ostrzejsze gwiazdy i tym samym większą ilość widocznych szczegółów. Według literatury profil natężenia gwiazdy można oszacować funkcją Gaussa [15]. W pracy zastosowano algorytm obliczający FWHM na podstawie dwuwymiarowego rozkładu Gaussa. Profil jasności gwiazdy na obrazie można oszacować funkcją przedstawioną w równaniu (6.1). Po przyjęciu średniej wartości wariancji rozmycia obrazu można obliczyć FWHM zgodnie z zależnością przedstawioną na równaniu (6.2).

$$I(x, y) = I_0 \exp \left(-\frac{(x - x_c)^2}{2\sigma_x^2} - \frac{(y - y_c)^2}{2\sigma_y^2} \right) \quad (6.1)$$

gdzie,

I_0 - maksymalne natężenie

(x_c, y_c) – środek gwiazdy

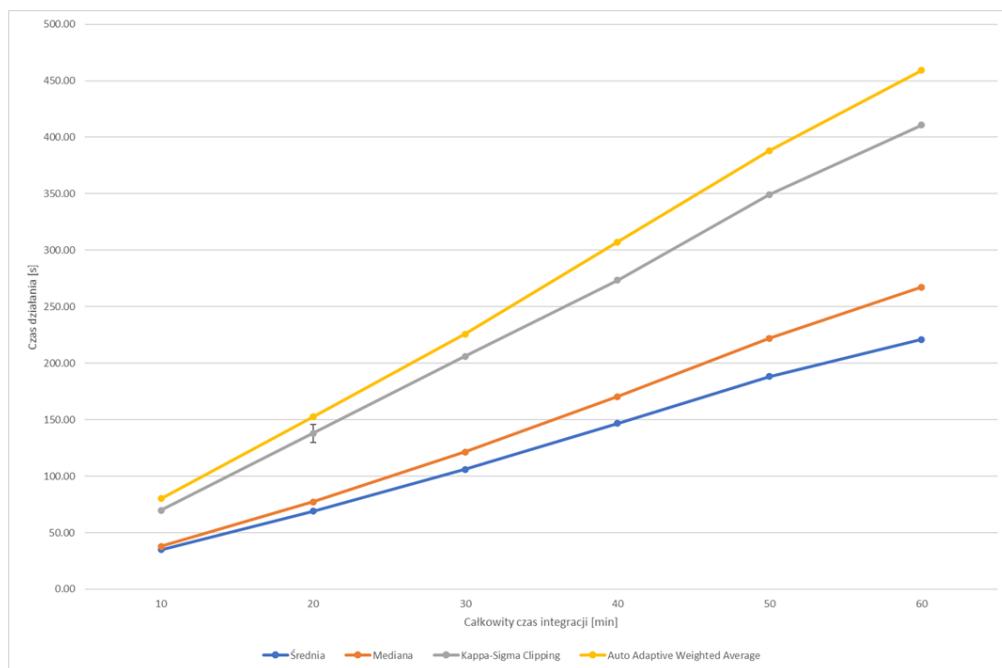
σ_x, σ_y – rozmycie obrazu

$$\text{FWHM} = 2\sqrt{2 \ln 2} \cdot \sigma \quad (6.2)$$

Przy implementacji stosuje się parametr wielkości wycinka nieba przy obliczaniu FWHM - dobranie zbyt dużego okna może spowodować artefakty spowodowane pozycją okolicznych gwiazd, lecz przy zbyt małym oknie wyniki będą nieprawidłowe dla gwiazd większych niż wybrane okno. W późniejszych badaniach wartości FWHM będą przedstawione dla różnych wielkości okna. Implementacja oblicza FWHM dla wszystkich wykrytych gwiazd, jednak wynikiem końcowym będzie mediana FWHM wszystkich wykrytych gwiazd.

6.2. Wyniki eksperymentów

Czas działania zestawów 244



Rysunek 6.3: Porównanie średnich czasów nakładania zdjęć w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 244

Czas działania zestawów 1499

FWHM dla zestawów 244

FWHM dla zestawów 1499

porównanie wizualne zestawów 60 244

porównanie wizualne zestawów 60 1499

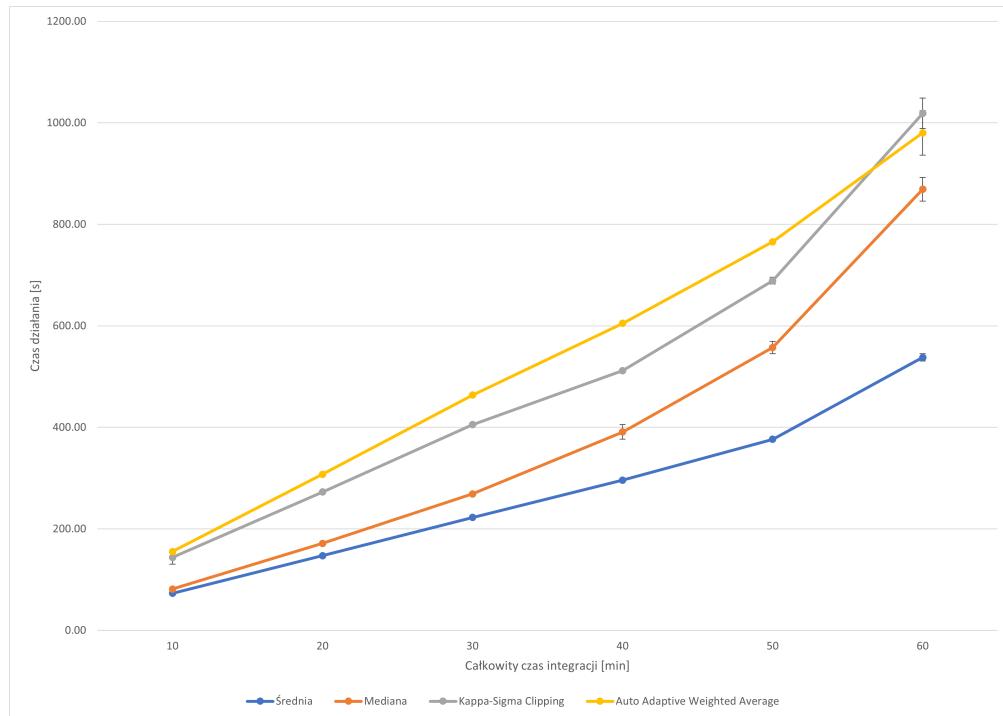
porównanie zestawów 10-60 dla 244

porównanie zestawów 10-60 dla 1499

6.3. Porównanie wyników

Metoda	\bar{t} [s]	Czas integracji [min]					
		10	20	30	40	50	60
Średnia	\bar{t} [s]	34.93	68.94	106.03	146.79	188.19	221.05
Średnia	σ [s]	0.14	0.18	0.27	0.83	0.76	0.94
Mediana	\bar{t} [s]	37.81	77.30	121.38	170.48	222.07	267.22
Mediana	σ [s]	0.87	0.15	0.63	0.75	1.24	1.73
Kappa-Sigma Clip.	\bar{t} [s]	69.82	137.88	206.07	273.56	349.24	410.79
Kappa-Sigma Clip.	σ [s]	0.27	7.82	1.06	0.33	0.69	0.77
AA Weighted Avg.	\bar{t} [s]	80.13	152.66	225.90	307.11	388.05	459.12
AA Weighted Avg.	σ [s]	0.09	1.34	0.22	0.89	0.61	0.92

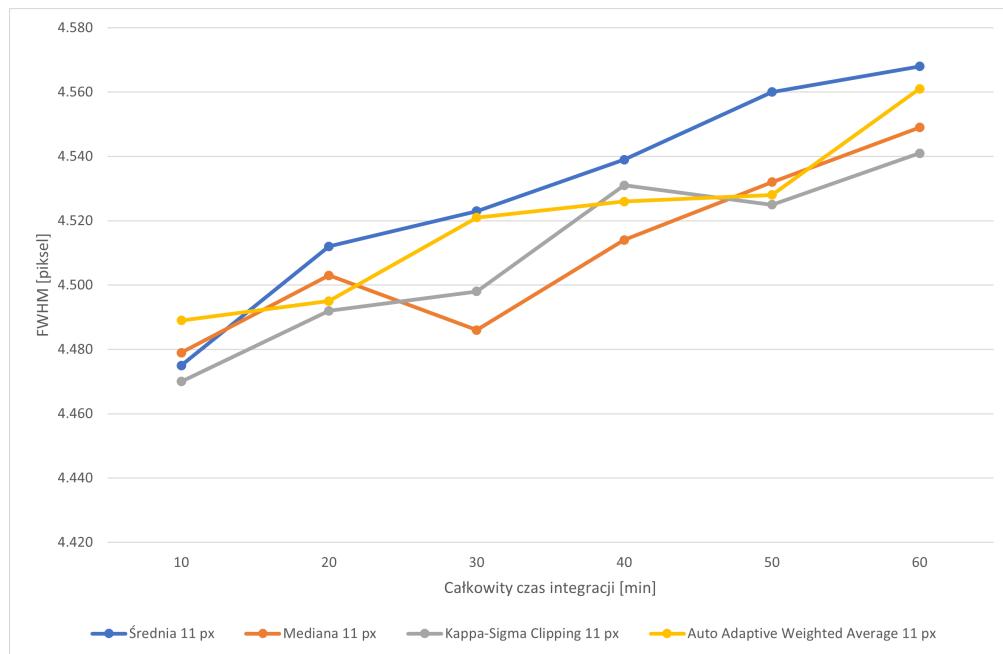
Tabela 6.1: Porównanie średnich czasów nakładania zdjęć w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 244.



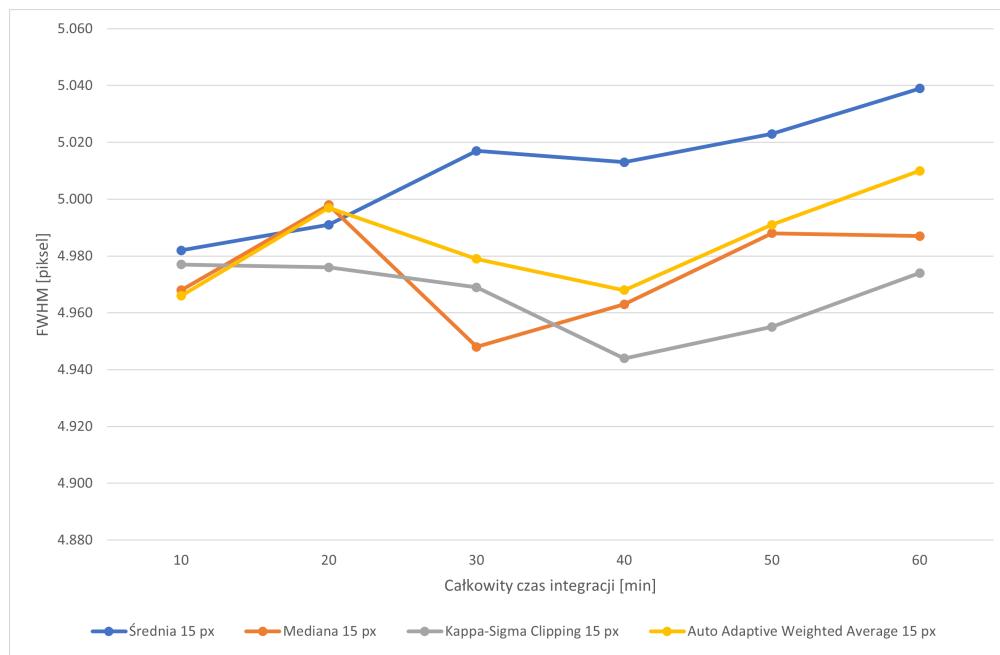
Rysunek 6.4: Porównanie średnich czasów nakładania zdjęć w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 1499

Metoda	Czas integracji [min]					
	10	20	30	40	50	60
Średnia	\bar{t} [s]	34.93	68.94	106.03	146.79	188.19
	σ [s]	0.14	0.18	0.27	0.83	0.76
Mediana	\bar{t} [s]	37.81	77.30	121.38	170.48	222.07
	σ [s]	0.87	0.15	0.63	0.75	1.24
Kappa-Sigma Clip.	\bar{t} [s]	69.82	137.88	206.07	273.56	349.24
	σ [s]	0.27	7.82	1.06	0.33	0.69
AA Weighted Avg.	\bar{t} [s]	80.13	152.66	225.90	307.11	388.05
	σ [s]	0.09	1.34	0.22	0.89	0.61

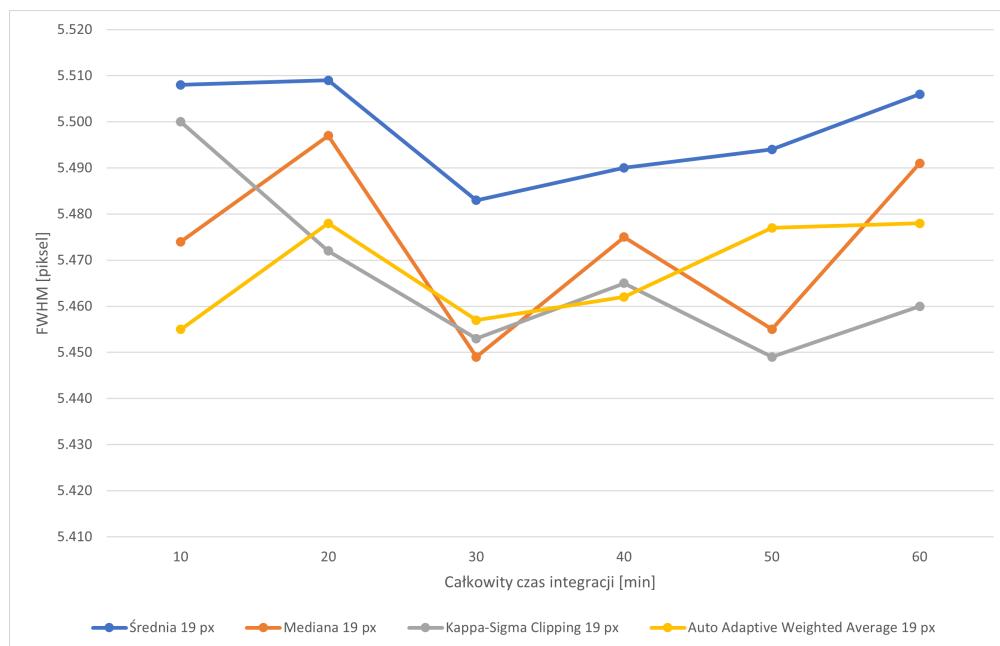
Tabela 6.2: Porównanie średnich czasów nakładania zdjęć w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 1499.



Rysunek 6.5: Porównanie FWHM zestawów 244 w zależności od metody nakładania zdjęć i czasu integracji dla okna 11 pikseli.



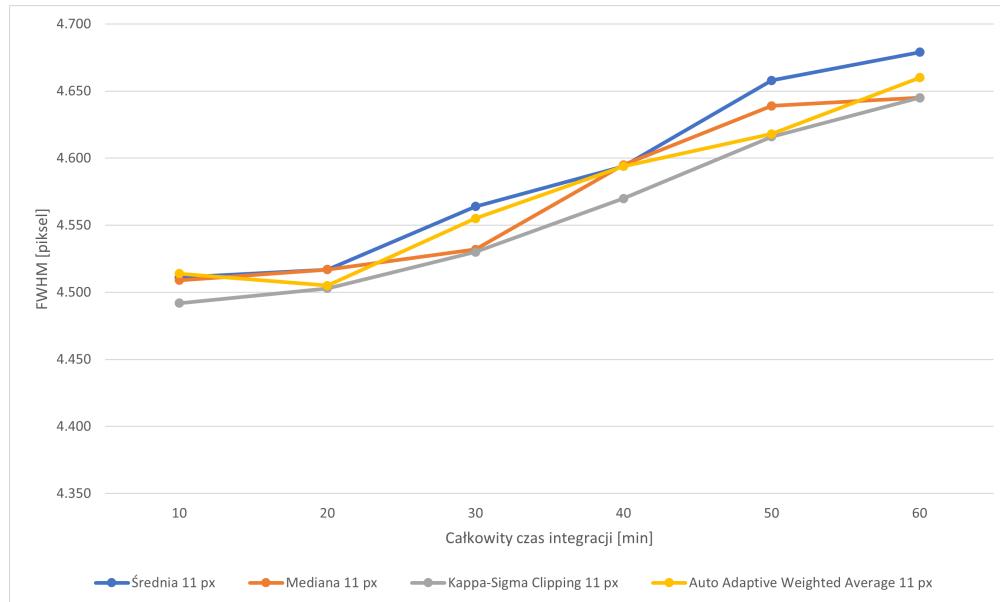
Rysunek 6.6: Porównanie FWHM zestawów 244 w zależności od metody nakładania zdjęć i czasu integracji dla okna 15 pikseli.



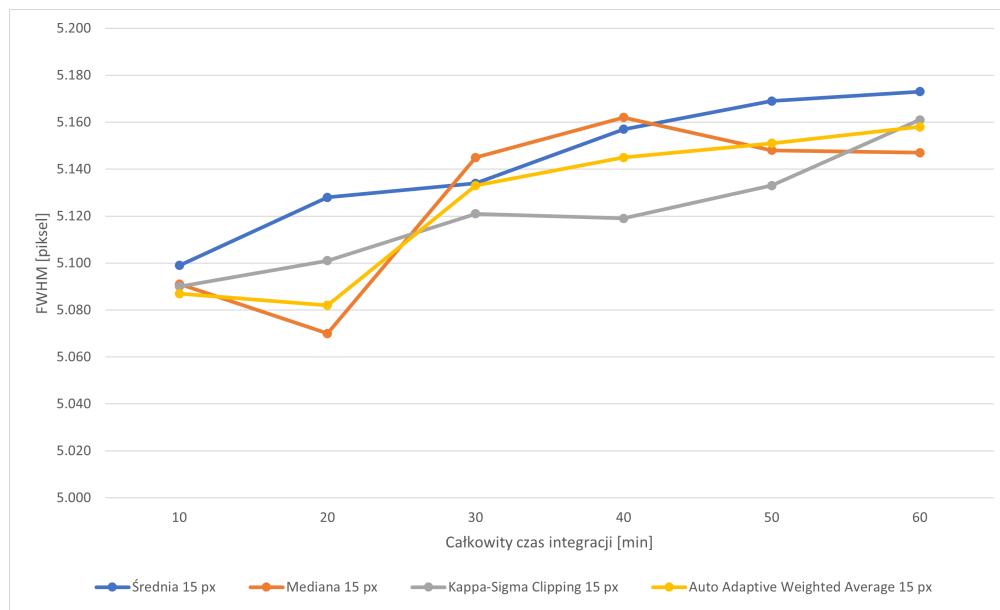
Rysunek 6.7: Porównanie FWHM zestawów 244 w zależności od metody nakładania zdjęć i czasu integracji dla okna 19 pikseli.

Metoda		Czas integracji [min]					
		10	20	30	40	50	60
Średnia	11 px [px]	4.475	4.512	4.523	4.539	4.560	4.568
	15 px [px]	4.982	4.991	5.017	5.013	5.023	5.039
	19 px [px]	5.508	5.509	5.483	5.490	5.494	5.506
Mediania	11 px [px]	4.479	4.503	4.486	4.514	4.532	4.549
	15 px [px]	4.968	4.998	4.948	4.963	4.988	4.987
	19 px [px]	5.474	5.497	5.449	5.475	5.455	5.491
Kappa–Sigma Clip.	11 px [px]	4.470	4.492	4.498	4.531	4.525	4.541
	15 px [px]	4.977	4.976	4.969	4.944	4.955	4.974
	19 px [px]	5.500	5.472	5.453	5.465	5.449	5.460
AA Weighted Avg.	11 px [px]	4.489	4.495	4.521	4.526	4.528	4.561
	15 px [px]	4.966	4.997	4.979	4.968	4.991	5.010
	19 px [px]	5.455	5.478	5.457	5.462	5.477	5.478

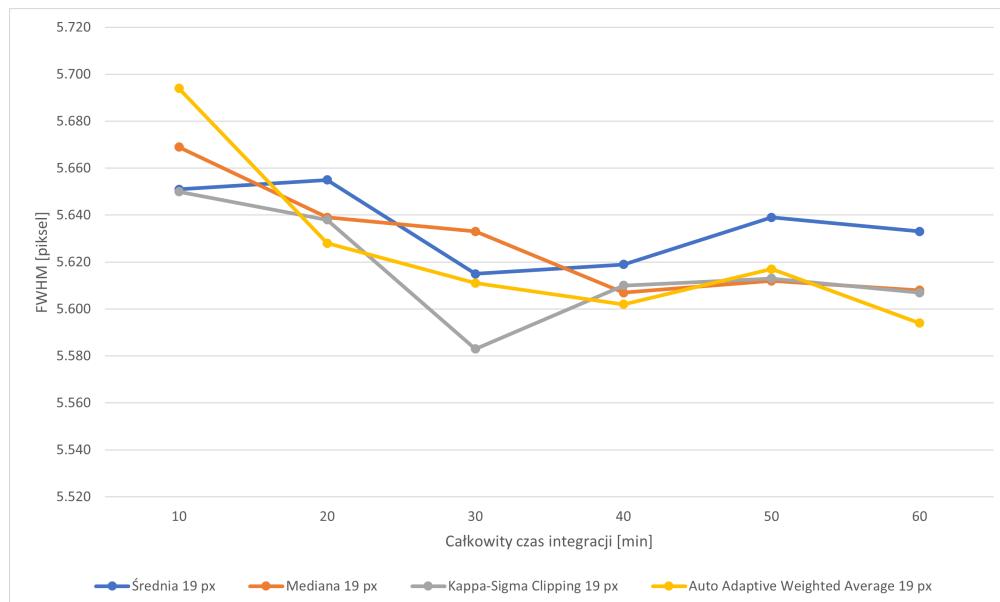
Tabela 6.3: Porównanie wartości FWHM w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 244.



Rysunek 6.8: Porównanie FWHM zestawów 1499 w zależności od metody nakładania zdjęć i czasu integracji dla okna 11 pikseli.



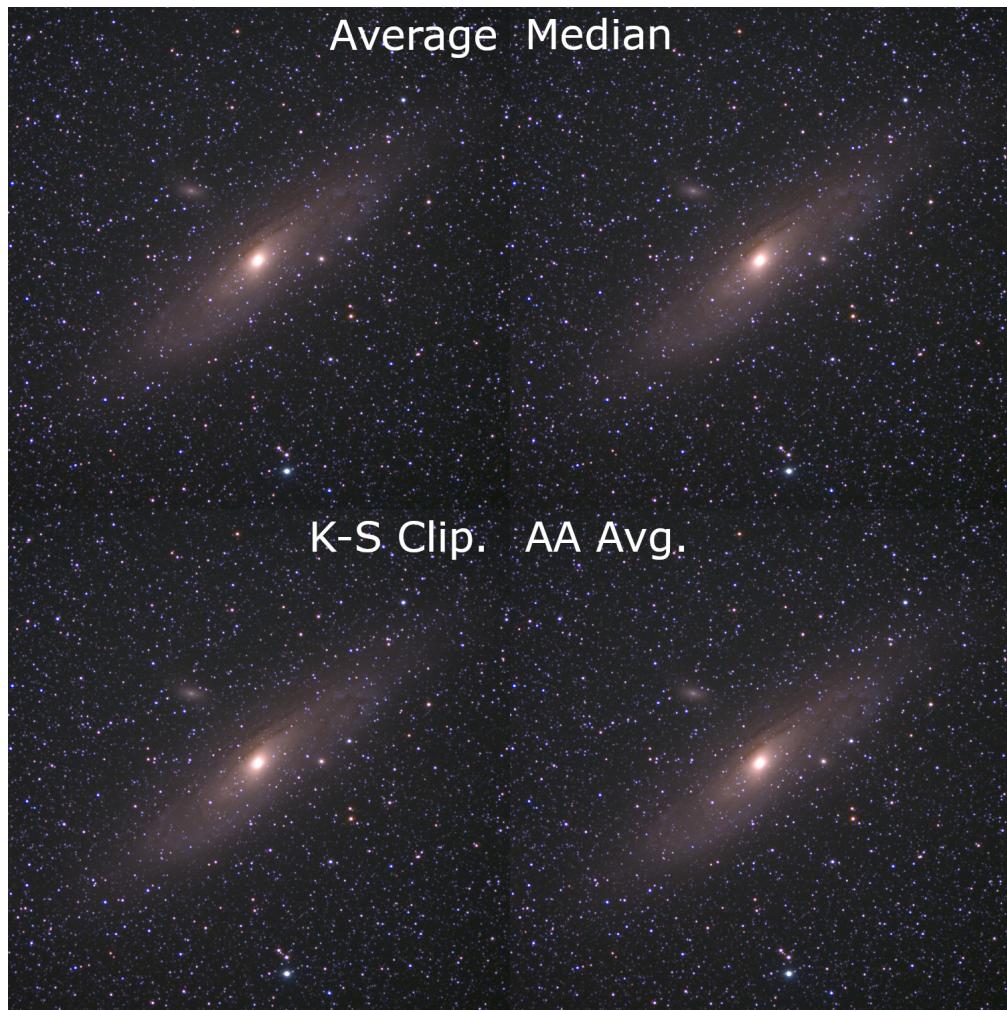
Rysunek 6.9: Porównanie FWHM zestawów 1499 w zależności od metody nakładania zdjęć i czasu integracji dla okna 15 pikseli.



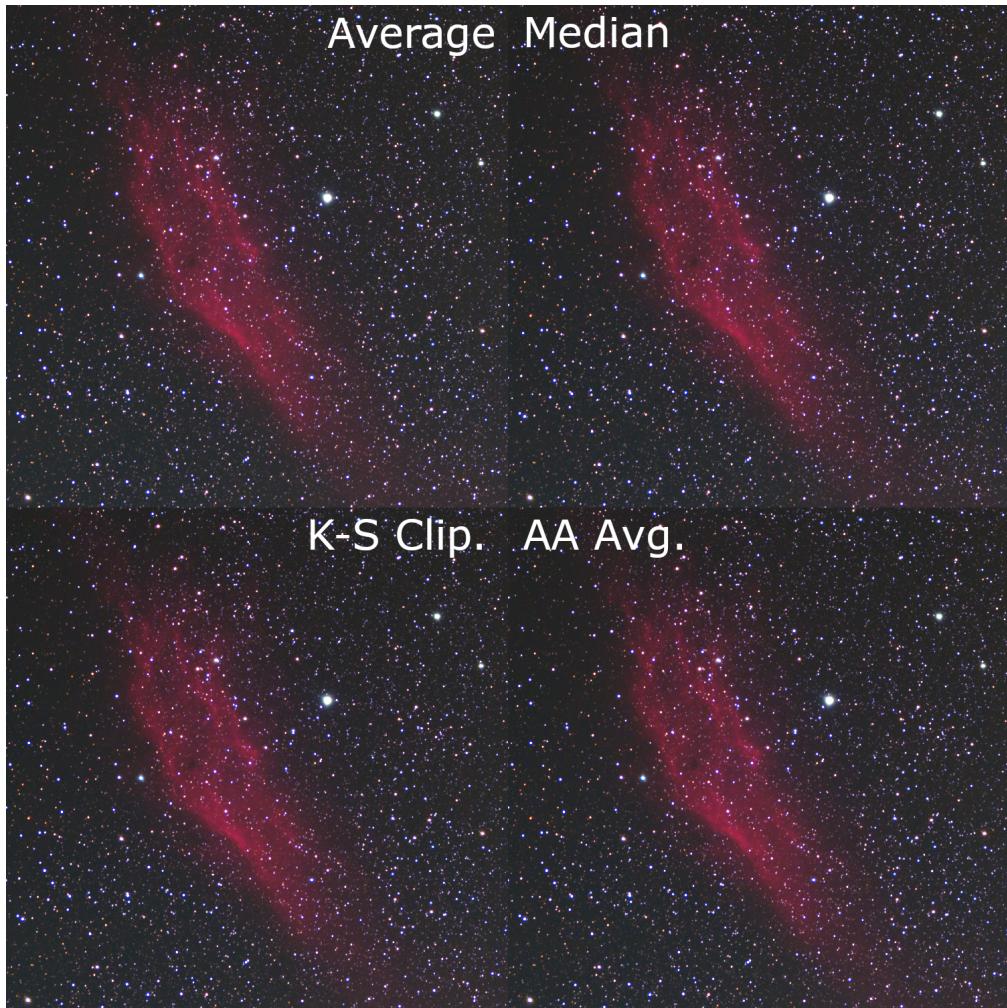
Rysunek 6.10: Porównanie FWHM zestawów 1499 w zależności od metody nakładania zdjęć i czasu integracji dla okna 19 pikseli.

Metoda		Czas integracji [min]					
		10	20	30	40	50	60
Średnia	11 px [px]	4.475	4.512	4.523	4.539	4.560	4.568
	15 px [px]	4.982	4.991	5.017	5.013	5.023	5.039
	19 px [px]	5.508	5.509	5.483	5.490	5.494	5.506
Medianą	11 px [px]	4.479	4.503	4.486	4.514	4.532	4.549
	15 px [px]	4.968	4.998	4.948	4.963	4.988	4.987
	19 px [px]	5.474	5.497	5.449	5.475	5.455	5.491
Kappa–Sigma Clip.	11 px [px]	4.470	4.492	4.498	4.531	4.525	4.541
	15 px [px]	4.977	4.976	4.969	4.944	4.955	4.974
	19 px [px]	5.500	5.472	5.453	5.465	5.449	5.460
AA Weighted Avg.	11 px [px]	4.489	4.495	4.521	4.526	4.528	4.561
	15 px [px]	4.966	4.997	4.979	4.968	4.991	5.010
	19 px [px]	5.455	5.478	5.457	5.462	5.477	5.478

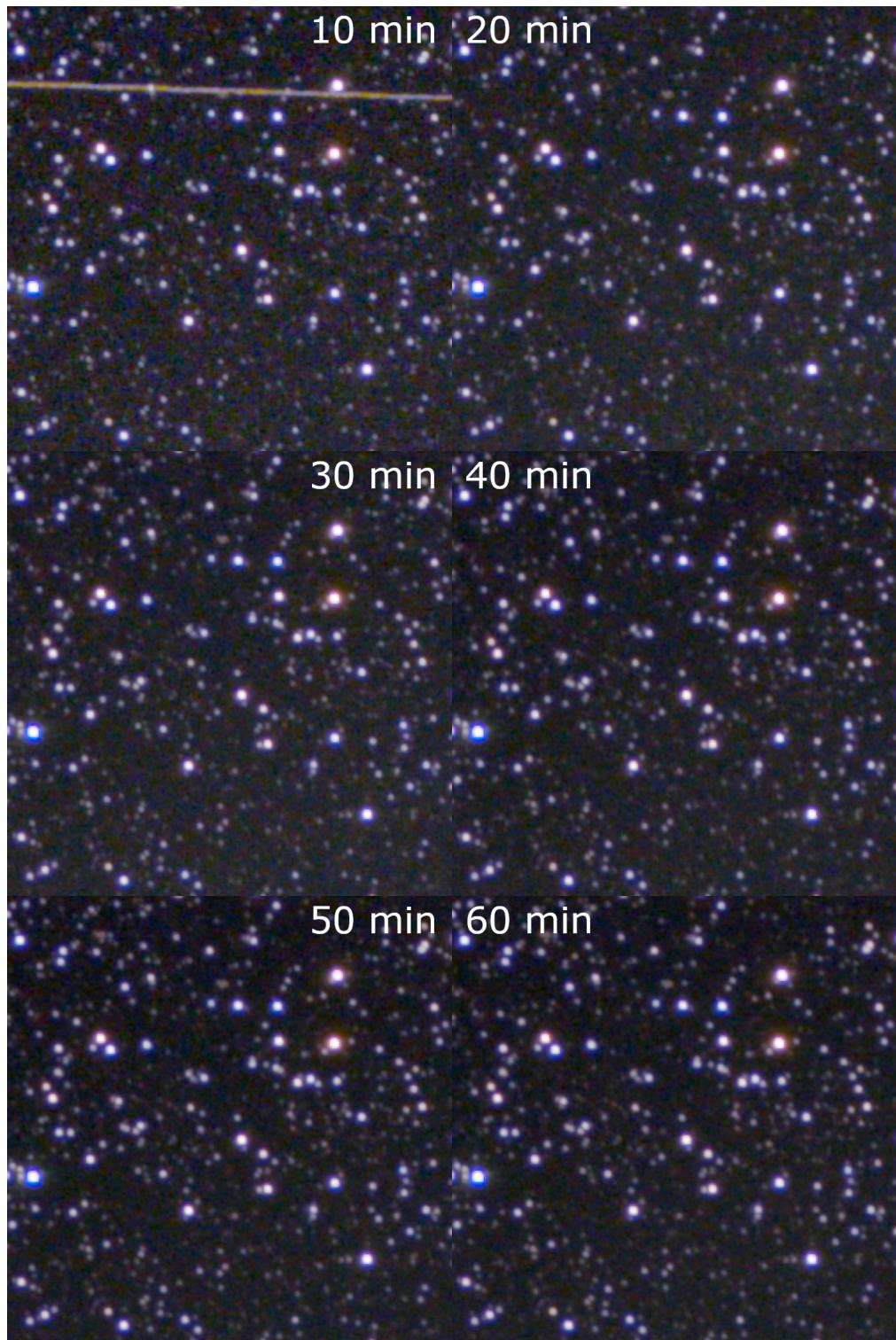
Tabela 6.4: Porównanie wartości FWHM w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 1499.



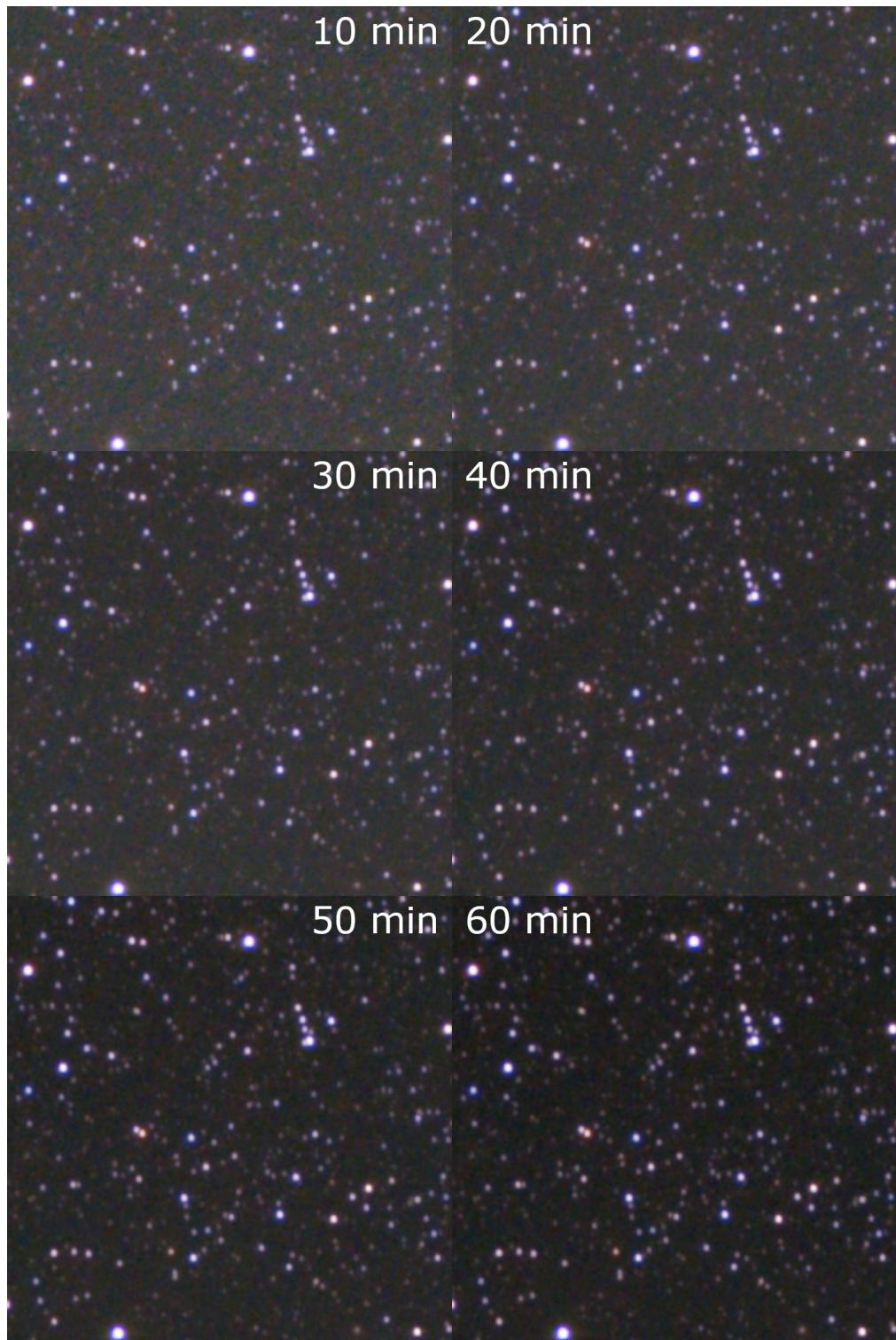
Rysunek 6.11: Porównanie wizualne obiektu z zestawu 244 o całkowitym czasie integracji 60 min. dla różnych algorytmów.



Rysunek 6.12: Porównanie wizualne obiektu z zestawu 1499 o całkowitym czasie integracji 60 min. dla różnych algorytmów.



Rysunek 6.13: Porównanie wizualne szumu dla różnego czasu integracji algorytmu Kappa-Sigma Clipping zestawów 244.



Rysunek 6.14: Porównanie wizualne szumu dla różnego czasu integracji algorytmu Kappa-Sigma Clipping zestawów 1499.

7. Wnioski

Bibliografia

- [1] CrystalMark - oficjalna strona. <https://crystalmark.info>. Data Dostępu: 21 listopada 2025.
- [2] DeepSkyStacker - oficjalna strona. <http://deepskystacker.free.fr>. Data Dostępu: 6 listopada 2025.
- [3] HWiNFO - oficjalna strona. <https://www.hwinfo.com/>. Data Dostępu: 21 listopada 2025.
- [4] Instrukcja obsługi montażu Star Watcher Star Adventurer. https://inter-static.skywatcher.com/downloads/StarAdv_manual_150722V2_updateds.pdf. Data Dostępu 11 Listopada 2025.
- [5] LibRaw - oficjalna strona. <https://www.libraw.org/>. Data Dostępu: 21 listopada 2025.
- [6] Light Pollution Map - strona aplikacji. <https://lightpollutionmap.app>. Data Dostępu: 9 listopada 2025.
- [7] OpenCV - oficjalna strona. <https://opencv.org/>. Data Dostępu: 21 listopada 2025.
- [8] Photons to Photos - oficjalna strona. <https://www.photonstophotos.net>. Data Dostępu: 8 listopada 2025.
- [9] PixInsight - oficjalna strona. <https://pixinsight.com/>. Data Dostępu: 6 listopada 2025.
- [10] Siril - oficjalna strona. <https://siril.org/>. Data Dostępu: 6 listopada 2025.
- [11] Sky Publishing Corporation and NASA Goddard Space Flight Center, New General Catalogue (NGC). <https://heasarc.gsfc.nasa.gov/W3Browse/all/ngc2000.html>, 2024. Data Dostępu: 9 listopada 2025.
- [12] J. Beish. Astronomical seeing. <https://apps.dtic.mil/sti/tr/pdf/ADA391637.pdf>, 2001. Technical Report ADA391637. Data dostępu: 11 listopada 2025.
- [13] J. E. Bortle. Gauging light pollution: The bortle dark-sky scale. <https://skyandtelescope.org/astronomy-resources/light-pollution-and-astronomy-the-bortle-dark-sky-scale/>, 2001. Sky & Telescope. Data dostępu: 9 listopada 2025.
- [14] P. A. Cheremkhin, V. V. Lesnichii, and N. V. Petrov. Use of spectral characteristics of dslr cameras with bayer filter sensors. *Journal of Physics: Conference Series*, 2014.
- [15] G. Coupinot, J. Hecquet, M. Auriere, and R. Futauly. Photometric analysis of astronomical images by the wavelet transform. *A&A*, 259:701–710, June 1992.

- [16] O. Hainaut. Signal, noise and detection. <https://www.eso.org/~ohainaut/ccd/sn.html>, 2005. Data dostępu: 31 Października 2025.
- [17] S. Howell and A. Tavackolimehr. *Handbook of CCD Astronomy*. Cambridge University Press, 2nd edition, 2006.
- [18] B. Lin, X. Xu, Z. Shen, X. Yang, L. Zhong, and X. Zhang. A registration algorithm for astronomical images based on geometric constraints and homography. *Remote Sensing*, 15(7), 2023.
- [19] C. U. C. Nguyen and T. A. Lovell. *A Practical Approach to Plate-Solving of Astronomical Images Based on Least-Squares Estimation*. 2025.
- [20] P. B. Stetson. The techniques of least squares and stellar photometry with ccds. https://ned.ipac.caltech.edu/level5/Stetson/Stetson_contents.html, 1994. Data Dostępu: 20 listopada 2025.

Spis rysunków

6.1	Wyniki testu wydajności dysku SSD.	27
6.2	Zrzut ekranu przedstawiający użyty sprzęt.	28
6.3	Porównanie średnich czasów nakładania zdjęć w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 244	29
6.4	Porównanie średnich czasów nakładania zdjęć w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 1499	30
6.5	Porównanie FWHM zestawów 244 w zależności od metody nakładania zdjęć i czasu integracji dla okna 11 pikseli.	31
6.6	Porównanie FWHM zestawów 244 w zależności od metody nakładania zdjęć i czasu integracji dla okna 15 pikseli.	32
6.7	Porównanie FWHM zestawów 244 w zależności od metody nakładania zdjęć i czasu integracji dla okna 19 pikseli.	32
6.8	Porównanie FWHM zestawów 1499 w zależności od metody nakładania zdjęć i czasu integracji dla okna 11 pikseli.	33
6.9	Porównanie FWHM zestawów 1499 w zależności od metody nakładania zdjęć i czasu integracji dla okna 15 pikseli.	34
6.10	Porównanie FWHM zestawów 1499 w zależności od metody nakładania zdjęć i czasu integracji dla okna 19 pikseli.	34
6.11	Porównanie wizualne obiektu z zestawu 244 o całkowitym czasie integracji 60 min. dla różnych algorytmów.	36
6.12	Porównanie wizualne obiektu z zestawu 1499 o całkowitym czasie integracji 60 min. dla różnych algorytmów.	37
6.13	Porównanie wizualne szumu dla różnego czasu integracji algorytmu Kappa-Sigma Clipping zestawów 244.	38
6.14	Porównanie wizualne szumu dla różnego czasu integracji algorytmu Kappa-Sigma Clipping zestawów 1499.	39

Spis tabel

5.1	Opis podzielonych zestawów fotografii obiektów głębokiego nieba	25
6.1	Porównanie średnich czasów nakładania zdjęć w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 244.	30
6.2	Porównanie średnich czasów nakładania zdjęć w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 1499.	31
6.3	Porównanie wartości FWHM w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 244.	33
6.4	Porównanie wartości FWHM w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 1499.	35