

**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**

Celem niniejszej pracy było opracowanie i implementacja aplikacji desktopowej do przetwarzania obrazów głębokiego nieba z wykorzystaniem podstawowych technik obróbki zdjęć astronomicznych. Rozwiązanie skierowane jest do użytkowników początkujących i rozpoczynających pracę z astrofotografią, oraz osób ceniących możliwość automatyzacji procesu przetwarzania zdjęć, a także możliwość wykonywania badań wydajnościowych i jakościowych. W ramach pracy zaimplementowano aplikację desktopową korzystając z języka C++, bibliotek OpenCV oraz LibRaw z istotnymi funkcjonalnościami przetwarzania zdjęć głębokiego nieba: kalibracją, rejestracją oraz nakładaniem zdjęć (stacking). Dbano również o stworzenie modularnej architektury umożliwiającą łatwą rozbudowę aplikacji o dodatkowe techniki obróbki i funkcjonalności badawcze w przyszłości. Przeprowadzone zostały testy wydajnościowe aplikacji oraz jakościowe plików wynikowych, na podstawie rzeczywistych danych astronomicznych. Analiza otrzymanych wyników potwierdziła poprawność działania programu oraz przydatność aplikacji w środowisku amatorskiej astrofotografii. Przedstawione zostały również propozycje zastosowania zaimplementowanych technik, kierunki dalszego rozwoju aplikacji i przeprowadzanych badań w przyszłości.

## **Abstract**

The aim of this thesis was to design and implement a desktop application for deep-sky image processing using basic astronomical image processing techniques, aimed at amateur astrophotographers. The solution is intended for beginners starting their work with astrophotography and people valuing automation of the image processing workflow, as well as the possibility of performing performance and quality benchmarks. As part of the work a desktop application was implemented using C++ and OpenCV and LibRaw libraries with essential deep-sky image processing functionalities: calibration, registration and stacking. Attention was also paid to creating a modular architecture enabling easy expansion of the application with additional processing techniques and research functionalities in the future. Performance benchmarks of the application and quality tests of the output files were carried out based on real astronomical data. The analysis of the obtained results confirmed the correctness of the implementation and usefulness of the application in the context of amateur astrophotography. The thesis also presents proposals for the usage of the implemented techniques and directions for further development as well as research in the future.



# Spis treści

<b>1 Wprowadzenie</b>	<b>1</b>
1.1 Opis problemu . . . . .	1
1.2 Cel pracy . . . . .	1
1.3 Zarys pracy . . . . .	2
<b>2 Stan dziedziny</b>	<b>3</b>
2.1 Przegląd literatury . . . . .	3
2.2 Istniejące narzędzia . . . . .	3
<b>3 Przetwarzanie zdjęć</b>	<b>5</b>
3.1 Kalibracja . . . . .	5
3.2 Rejestracja . . . . .	6
3.3 Stacking . . . . .	7
<b>4 Opis aplikacji</b>	<b>11</b>
4.1 Wymagania oraz technologie . . . . .	11
4.2 Architektura systemu . . . . .	12
4.3 Interfejs użytkownika . . . . .	14
4.4 Struktura kodu . . . . .	15
<b>5 Zbieranie danych</b>	<b>25</b>
5.1 Opis otoczenia i montażu . . . . .	25
5.2 Procedura eksperymentów i zestawy danych . . . . .	26
<b>6 Analiza implementacji algorytmów</b>	<b>29</b>
6.1 Metodyka eksperymentów . . . . .	29
6.2 Wyniki eksperymentów . . . . .	31
6.3 Analiza wyników . . . . .	43
<b>7 Wnioski</b>	<b>45</b>
<b>Bibliografia</b>	<b>48</b>
<b>Spis rysunków</b>	<b>49</b>
<b>Spis tabel</b>	<b>51</b>



# 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 fotografií największych i najjaśniejszych obiektów głębokiego nieba nie jest zaawansowaną procedurą. Możliwe jest wykonanie ich 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

Przetwarzanie zdjęć głębokiego nieba stanowi istotny element amatorskiej astrofotografii. Obiekty głębokiego nieba znajdują się poza Układem Słonecznym, więc ich znaczna odległość powoduje, że ich pozorna jasność na nocnym niebie nie pozwala na dostrzeżenie ich gołym okiem, zwłaszcza w obszarach o dużym zanieczyszczeniu świetlnym. Do pokonania tych ograniczeń amatorzy astrofotografii wykorzystują fotografię z dłuższym czasem naświetlania, które pozwalają na zebranie większej ilości światła z odległych obiektów. Takie podejście niesie ze sobą szereg wymogów dotyczących sprzętu oraz odpowiedniej techniki. Alternatywą do tego podejścia jest wykonanie serii krótkich ekspozycji, które są następnie łączone w jeden plik wynikowy - pozwala to na uproszczenie procesu zbierania danych. Możliwe jest łączenie tych technik i wykonanie serii dłuższych ekspozycji, często w zakresach kilku sekund do kilku minut, które są następnie łączone w jeden plik wynikowy. Niezależnie od podejścia, zebrane dane wymagają odpowiedniego przetworzenia.

## 1.2. Cel pracy

Obecnie na rynku dostępne są komercyjne i darmowe narzędzia do przetwarzania zdjęć głębokiego nieba. Wiele z tych aplikacji jest skierowana jednak do zaawansowanych użytkowników. Oferuje obłusgę wyłącznie przez interfejs graficzny nie pozwalając na automatyzację procesu przetwarzania zdjęć. Nie posiadają również przystosować do wykonywania badań wydajnościowych i jakościowych. Celem niniejszej pracy jest stworzenie aplikacji desktopowej, która umożliwi przetwarzanie zdjęć głębokiego nieba z wykorzystaniem podstawowych technik obróbki obrazów astronomicznych. Głównymi zaletami będzie prosta obsługa dla użytkowników początkujących i średnio-zaawansowanych, użytkowanie za pomocą linii komend i możliwość przeprowadzania badań wydajnościowych i jakościowych. Dodatkowo aplikacja będzie dopuszczała prostą rozbudowę o dodatkowe techniki obróbki obrazów, utworzenie interfejsu graficznego aplikacji oraz dodanie nowych funkcjonalności badawczych w przyszłości.

### 1.3. Zarys pracy

Niniejsza praca została podzielona na siedem rozdziałów. Rozdział pierwszy wprowadza czytelnika w tematykę, cele tworzonej aplikacji oraz strukturę pracy. Drugi rozdział zawiera omówienie rozwiązań obecnych na rynku oraz przybliża prace naukowe związane z przetwarzaniem obrazów astronomicznych. Trzeci rozdział przybliża pojęcia związane z przetwarzaniem obrazów astronomicznych, takie jak kalibracja, rejestracja oraz stacking, opisując algorytmy wykorzystane do ich realizacji. Wyjaśnia również w jakim celu stosuje się przetwarzanie zdjęć i jak wpływa to na teoretyczną jakość obrazu końcowego. Rozdział czwarty opisuje wymagania, które należy spełnić w trakcie implementacji aplikacji oraz technologie wykorzystane do jej stworzenia. Opisane są również moduły aplikacji, ich funkcjonalności, interfejs użytkownika oraz jak wygląda rzeczywista implementacja najważniejszych elementów kodu. W rozdziale piątym został opisany proces zbierania danych wraz z procedurą testową oraz warunki w jakich wykonano fotografie. Rozdział szósty przedstawia metodykę przeprowadzonych testów wydajnościowych i jakościowych, wraz z wynikami przeprowadzonych badań i analizą otrzymanych rezultatów. Rozdział siódmy zawiera podsumowanie i wnioski końcowe dotyczące przeprowadzonej pracy. Dodatkowo przedstawiono propozycje dalszych kierunków rozwoju aplikacji oraz dodatkowych badań.

## 2. Stan dziedziny

### 2.1. Przegląd literatury

Literatura związana z przetwarzaniem obrazów astronomicznych, niezbędna do wykonania pracy, obejmuje charakterystykę szumu, zależności stosunku sygnału do szumu od czasu integracji i źródeł szumu w badanych aparatach DSLR, i układach CCD używanych w astrofotografii. Do wymaganych podstaw teoretycznych odniesiono się do prac [17, 18]. W pozycji [18] opisano również metody kalibracji obrazów astronomicznych przy użyciu dedykowanych klatek kalibracyjnych oraz ich wpływ na jakość obrazu wynikowego. Informacje dotyczące procesu rejestracji obrazów astronomicznych i dopasowania transformacji afanicznej pomiędzy obrazami zostały zaczerpnięte z prac [19, 20].

Do uzyskania informacji na temat warunków wykonywania zdjęć astronomicznych wykorzystano prace [13, 14, 15], które opisują wpływ zanieczyszczenia świetlnego, warunków atmosferycznych na jakość uzyskiwanych obrazów oraz informacje dotyczące charakterystyki spektralnej aparatów DSLR. Informacje dotyczące obiektów astronomicznych zaczerpnięto z katalogu NGC [12]. Dodatkowo posłużono się źródłami udostępniającymi mapy zanieczyszczenia świetlnego [7] oraz informacji dotyczących wartości szumu dla różnych ustawień aparatu dostępnych na stronie [9].

W trakcie implementacji aplikacji wykorzystano dostępne informacje na temat używanych w środowisku algorytmów nakładania obrazów astronomicznych z dostępnych narzędzi takich jak DeepSkyStacker [2], Siril [11] i PixInsight [10]. Dodatkowe informacje na temat algorytmu Auto Adaptive Weighted Average zaczerpnięto z [21]. Do implementacji wykorzystano również dostępne biblioteki OpenCV [8] oraz LibRaw [6], umożliwiające kolejno przetwarzanie obrazów oraz odczyt plików RAW.

### 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 doświadczonych 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 [10] to zaawansowane komercyjne narzędzie używane przez profesjonalistów. Zapewnia bardzo szerokie możliwości przetwarzania danych, kalibracji i analizy fotometrycznej. Głównymi wadami programu jest jej cena oraz duża ilość wymaganej wiedzy do prawidłowej obsługi programu.

- **Siril**

Siril [11] 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 celu wytworzenia obrazu lepszej jakości, z niskim poziomem szumu, fotografie obiektów głębokiego nieba należy odpowiednio przetworzyć. Można podzielić to 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*). Standardową wartością określającą szum jest ilość elektronów wygenerowanych przy pobieraniu informacji z matrycy fotograficznej [18]. Przydatne informacje dotyczące szumu generowanego na popularnych matrycach aparatów można znaleźć na stronie Photons to Photos [9].

## 3.1. Kalibracja

Jednym z niezbędnych elementów przetwarzania zdjęć jest zastosowanie klatek kalibracyjnych, które odpowiadają za korekcję różnych niedoskonałości teleskopu i matrycy. Poniżej opisano najczęściej używane klatki kalibracyjne. Dokładny opis założeń korzystania z nich opisano w książce [18]. 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 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 przydadzą się dodatkowo przy zniwelowaniu 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 kalibracyjnych 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:

$I_{\text{light}}$  - surowa klatka obiektu

$I_{\text{dark}}$  - klatka dark

$I_{\text{bias}}$  - klatka bias

$I_{\text{flat}}$  - 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 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 lokalnych właściwościach 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.

Następnym etapem będzie określenie jaką transformację trzeba dokonać, żeby surową klatkę 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, a surową klatką i oszacowanie macierzy transformacji pomiędzy zdjęciami. Jest to metoda uproszczona, inspirowana na bardziej zaawansowanych rozwiązaniach zaproponowanych przez Lin i Xu w artykule [19].

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 [20].

### 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 [17]. Równania (3.2) i (3.3) zostały zaczerpnięte z [17]. 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 obiektów astronomicznych [ $\text{e}^-$ ]
- Sky – szum tła nieba [ $\text{e}^-$ ]
- Dark – szum prądu ciemnego [ $\text{e}^-$ ]
- $N_{\text{RON}}$  – szum odczytu detektora [ $\text{e}^-$ ]
- $N_{\text{DIT}}$  – liczba ekspozycji
- DIT – czas pojedynczej ekspozycji [s]
- $s$  – liczba elektronów zgromadzonych na sekundę od obiektów astronomicznych [ $\text{e}^-$ ]

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

- **Mediana**

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 Medianą

- **Kappa-Sigma Clipping**

Metoda Kappa-Sigma Clipping polega na iteracyjnym odrzucaniu wartości pikseli, które odbiegają od średniej o więcej niż  $\kappa$ -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. Wartości parametrów  $\kappa$  oraz maksymalnej liczby iteracji można dostosować w zależności od charakterystyki danych wejściowych - wartości stosowane w implementacji zostały opisane w rozdziale 4.4.

```

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  $\alpha$  oraz  $\beta$ . Należy jednak najpierw wybrać wartość początkową do dalszej iteracji, którą w tym przypadku jest mediana wartości pikseli. 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. Wartości parametrów oraz funkcja ważenia zostały opisane w rozdziale 4.4.

```

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
16
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, 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. Wczytywanie zdjęć obiektów oraz zdjęć kalibracyjnych z utworzonych katalogów po umieszczeniu w nich plików przez użytkownika.
3. Tworzenie głównych klatek kalibracyjnych na podstawie dostarczonych przez użytkownika zdjęć kalibracyjnych.
4. Wczytywanie zdjęć w wybranym formacie RAW oraz TIFF.
5. Korekcja zdjęć light z użyciem wcześniej utworzonych głównych klatek kalibracyjnych.
6. Automatyczne wyrównywanie zdjęć na podstawie wykrytych gwiazd.
7. Automatyczne nakładanie zdjęć obiektu głębokiego nieba jedną z czterech wybranych metod: średnia, mediana, kappa-sigma clipping lub auto adaptive weighted average.
8. 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 w utworzonych katalogach, nieprawidłowe formaty fotografii czy niepoprawne ścieżki.

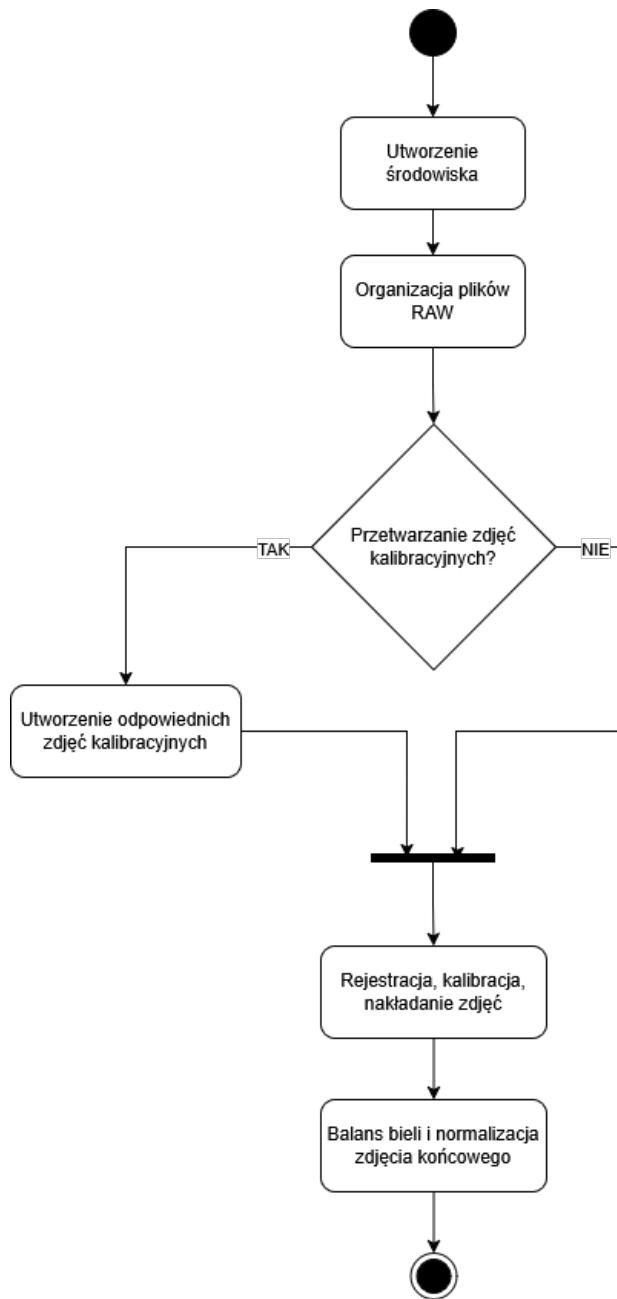
4. Aplikacja powinna działać prawidłowo przy łączeniu sześćdziesięciu zdjęć i poniżej na sprzęcie średniej klasy.
5. Aplikacja powinna być desktopowa oraz mieć interfejs konsolowy.
6. Aplikacja powinna być przyjazna dla użytkowników mało-zaawansowanych - powinna pozwalać na uzyskanie obrazu wynikowego przy zastosowaniu trzech lub mniejszej ilości komend bez utraty funkcjonalności.
7. Aplikacja powinna umożliwiać łączenie przynajmniej pięciu zdjęć wraz ze 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 [6], do manipulacji obrazami i utworzenia funkcji kalibracji, detekcji gwiazd, rejestracji i nakładania zdjęć biblioteka OpenCV [8].

## 4.2. Architektura systemu

Na rysunku 4.1 przedstawiono główny zarys działania programu. W celu realizacji zadań systemu oraz umożliwienia przyszłego rozwoju zdecydowano się na podział na odpowiednie, niezależne od siebie, moduły. System składa się z modułów inicjalizacji, przetwarzania fotografii, rejestracji, kalibracji, nakładania zdjęć, interakcji z plikami i sterującego z elementami interakcji z użytkownikiem.

- Moduł inicjalizacji odpowiada za przygotowanie środowiska pracy. Tworzy odpowiednią strukturę katalogów składającą się z odpowiednich podkatalogów na fotografie kalibracyjne, główne zdjęcia light oraz katalog na pliki wynikowe. Dzięki implementacji tego modułu użytkownik nie musi ręcznie tworzyć odpowiednich podkatalogów co ułatwi interakcję z programem oraz zmniejszy ryzyko błędów wynikających z nieporawidłowej organizacji danych.
- Moduł przetwarzania fotografii odpowiada za podstawowe przygotowanie fotografii do dalszych etapów przetwarzania jak wykrywanie gwiazd oraz jest odpowiedzialny za końcową obróbkę zdjęć po nałożeniu. Do odpowiedniego przygotowania plików przed wykrywaniem gwiazd należy przeprowadzić odpowiednią konwersję z danych surowych na dane z tylko jednym kanałem. Końcowa obróbka danych zakłada przetwarzanie zdjęć pod względem ustalenia odpowiedniego balansu bieli oraz normalizację jasności zdjęcia wynikowego.
- Moduł rejestracji odpowiada za lokalizację gwiazd na fotografiach oraz obliczenie odpowiedniej transformacji w celu nałożenia zdjęć. Do wykrywanie gwiazd wykorzystywana jest funkcja udostępniona przez bibliotekę OpenCV, która pozwala na wykrycie i ustalenie centroidów odpowiednich obiektów nazywanych jako *blob*. Odpowiednie parametry



Rysunek 4.1: Główne założenia działania programu.

pozwalaające wykrywanie gwiazd zostały opisane w rozdziale 4.4. Ustalenie transformacji będzie możliwe dzięki oszacowaniu transformacji afinicznej na podstawie dopasowanych punktów.

- Moduł kalibracji jest odpowiedzialny za przygotowanie i nakładanie zdjęć kalibracyjnych. Zdjęcia kalibracyjne obsługiwane przez program to bias, dark oraz flat.
- Moduł nakładania zdjęć odpowiada za łączenie przygotowanych fotografii w jeden obraz końcowy. Moduł będzie obsługiwał metody łączenia zdjęć: średnia, mediana,

Kappa-Sigma Clipping oraz Auto Adaptive Weighted Average.

- Moduł sterujący odpowiada za zarządzanie przepływem danych pomiędzy poszczególnymi modułami oraz interakcję z użytkownikiem.
- Moduł interakcji z plikami odpowiada za odczyt plików RAW przygotowanych przez użytkownika, odczytywanie z dysku odpowiednich plików przejściowych kalibracyjnych oraz zapisywanie plików wynikowych.

### 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 pozbawiając 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 włączeniu 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 pliki 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. Parametry `output_bps = 16` ustawia liczbę bitów na próbce na wyjściu - pozwala to przechować 14-bitowe dane z pliku NEF bez utraty rozdzielczości,

`no_auto_bright = 1` wyłącza automatyczne skalowanie jasności - przeskalowanie jasności negatywnie wpływa na późniejsze etapy kalibracji, `use_camera_wb = 0` oraz `use_auto_wb = 0` wyłączają użycie ustawień balansu bieli zapisanych w aparacie oraz automatycznego balansu bieli - pozwala to zachować dane bez korekcji kolorów narzucej przez profil aparatu. Współczynniki `gamm` oraz `user_mul` odpowiadają za zachowanie liniowych wartości pikseli bez korekcji gamma oraz bez skalowania poszczególnych kanałów kolorów. Do wczytywania wielu zdjęć stworzono nową funkcję, która obsługuje wczytywanie wielu zdjęć jednocześnie dzięki wielowątkowości.

```

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

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. Parametry `filterByArea`, `filterByCircularity`, `filterByInertia`, `filterByConvexity`, `filterByColor` za włączenie odpowiednich filtrów podczas wykrywania obiektów. Parametry `minArea` oraz `maxArea` określają minimalny i maksymalny rozmiar wykrywanych obiektów w pikselach, `minCircularity` określa minimalny poziom okrągłości obiektu, `minInertiaRatio` określa minimalny poziom intercji obiektu - miary rozłożenia obiektu względem osi głównej, `minConvexity` określa minimalny poziom wypukłości obiektu, a `blobColor` określa kolor wykrywanych obiektów (255 dla jasnych obiektów na ciemnym tle). Parametry zostały dobrane tak, żeby maksymalizować wykrycie jasnych obiektów o kształcie zbliżonym do koła na ciemnym tle, co odpowida gwiazdom na fotografiach astronomicznych. Przed wykryciem gwiazd obraz należało przekonwertować na obraz ośmioramiennego z jednym kanałem - zdecydowano się na użycie kanału zielonego, gdyż oferuje on największą ilość informacji w klasycznych matrycach z filtrem Bayer [15]. 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 referencyjnym jeśli jego niezmienniki różnią się o mniej niż ustalona tolerancja `ratioTol = 0,03`. Do oszacowania transformacji po dopasowaniu trójkątów wyznaczana jest transformacja affinecka 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 jako 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 dobrej jakości 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 affineckiej na podstawie zgodnych 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             ↘ };
6         std::array<cv::Point2f,3> tgtTri = { targetStars[i], targetStars[j], targetStars[k] };
7         std::vector<cv::Point2f> refVec(refTri.begin(), refTri.end());
8         std::vector<cv::Point2f> tgtVec(tgtTri.begin(), tgtTri.end());
9         cv::Mat affine = cv::estimateAffinePartial2D(tgtVec, refVec);
```

```

10    double a00 = affine.at<double>(0,0), a01 = affine.at<double>(0,1), a02 =
11      ↪ affine.at<double>(0,2);
12    double a10 = affine.at<double>(1,0), a11 = affine.at<double>(1,1), a12 =
13      ↪ affine.at<double>(1,2);
14
15    int inliers = 0;
16    for (const auto& pt : targetStars) {
17      double tx = a00 * pt.x + a01 * pt.y + a02;
18      double ty = a10 * pt.x + a11 * pt.y + a12;
19      for (const auto& refPt : refStars) {
20        double dx = tx - refPt.x;
21        double dy = ty - refPt.y;
22        if (dx*dx + dy*dy < inlierTol2) { ++inliers; break; }
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 przeciwieństwie 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 (pikseli dla każdego kanału zdjęcia)

$w$  - szerokość obrazu w pikselach

$h$  - wysokość obrazu w pikselach

$c$  - liczba kanałów kolorów

$N$  - 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 określonym zakresem są odrzucane. Zakres, poza którym wartości są odrzucane został przedstawiony 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,

$\mu$  - średnia jasność piksela

$\kappa$  - parametr odchyleń od wartości średniej

$\sigma$  - odchylenie standardowe jasności piksela

Metoda pozwala na efektywne usunięcie artefaktów ze zdjęcia, lecz w przeciwieństwie 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 przeciwieństwie 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 odcienia 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<float>(y, x);
27
28          float mean = vals[0];
29          float variance = 0;
30
31          for (int i = 1; i < nImgs; ++i)
32            mean += vals[i];
33            variance += (vals[i] - mean) * (vals[i] - mean);
34
35          mean /= nImgs;
36          variance /= nImgs;
37
38          if (vals[0] > mean + kappa * variance || vals[0] < mean - kappa * variance)
39            channelStack[0].at<float>(y, x) = 0;
40          else
41            channelStack[0].at<float>(y, x) = vals[0];
42
43        }
44      }
45    }
46  }
```

```

25     vals[i] = channelStack[i].at<ushort>(y, x);
26
27     for (int iter = 0; iter < maxIterations; ++iter) {
28         // Compute mean and stddev
29         float sum = 0, sum2 = 0;
30         for (float v : vals) { sum += v; sum2 += v * v; }
31         float mean = sum / vals.size();
32         float stddev = std::sqrt(sum2 / vals.size() - mean * mean);
33
34         // Clip values outside mean +- kappa * stddev
35         std::vector<float> newVals;
36         for (float v : vals)
37             if (std::abs(v - mean) <= kappa * stddev)
38                 newVals.push_back(v);
39
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 [21].

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).

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

gdzie,

$\alpha$  - parametr regulujący czułość na odchylenia

$\beta$  - 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  $\alpha$  oraz  $\beta$ .

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 liczbę 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 img = 0; img < nImgs; ++img) {
53            float val = vals[img];
54            float residual = residuals[img];
55            float weight = weights[img];
56
57            if (img == 0) {
58              val = alpha * frames[img].ptr<float>(y0 + ty + img * curH + x);
59              residual = beta * frames[img].ptr<float>(y0 + ty + img * curH + x);
60              weight = 1.0f;
61            } else {
62              val = alpha * frames[img].ptr<float>(y0 + ty + img * curH + x);
63              residual = beta * frames[img].ptr<float>(y0 + ty + img * curH + x);
64              weight = 1.0f;
65            }
66
67            val = val * weight + residual;
68            residual = residual * weight;
69            weight = weight * weight;
70
71            if (img == 0) {
72              result.ptr<float>(y0 + ty + img * curH + x) = val;
73              result.ptr<float>(y0 + ty + img * curH + x) = residual;
74              result.ptr<float>(y0 + ty + img * curH + x) = weight;
75            } else {
76              result.ptr<float>(y0 + ty + img * curH + x) = val;
77              result.ptr<float>(y0 + ty + img * curH + x) = residual;
78              result.ptr<float>(y0 + ty + img * curH + x) = weight;
79            }
80
81            if (img == 0) {
82              vals[img] = val;
83              residuals[img] = residual;
84              weights[img] = weight;
85            } else {
86              vals[img] = val;
87              residuals[img] = residual;
88              weights[img] = weight;
89            }
90
91            if (img == 0) {
92              result.ptr<float>(y0 + ty + img * curH + x) = val;
93              result.ptr<float>(y0 + ty + img * curH + x) = residual;
94              result.ptr<float>(y0 + ty + img * curH + x) = weight;
95            } else {
96              result.ptr<float>(y0 + ty + img * curH + x) = val;
97              result.ptr<float>(y0 + ty + img * curH + x) = residual;
98              result.ptr<float>(y0 + ty + img * curH + x) = weight;
99            }
100           }
101         }
102       }
103     }
104   }
105 }
```

```

49     for (int c = 0; c < channels; ++c) {
50         int m = 0;
51         for (int i = 0; i < nImg; ++i)
52             vals[m++] = rowPtrs[i][base + c];
53
54         // median initial guess
55         tmpVec.assign(vals.begin(), vals.begin() + m);
56         std::nth_element(tmpVec.begin(), tmpVec.begin() + (m / 2), tmpVec.end());
57         float mu = tmpVec[m / 2];
58
59         // iterative reweighting
60         for (int it = 0; it < iterations; ++it) {
61             float wsum = 0.0f;
62
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 // release tile buffers early if large
86 for (int i = 0; i < nImg; ++i) tile32[i].release();
87 }
88
89 // convert back to original type
90 cv::Mat out;
91 result.convertTo(out, frames[0].type());
92 return out;
93 }
94 }
```

Kod Źródłowy 4.7: Funkcja `stackAutoAdaptiveWeightedAverage` z klasy `Stacker`



# 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 [14]. 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 [13]. 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 [5], 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 eksperymentów 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 główicy na statyw fotograficzny oraz dołączenie aparatu z obiektywem do główicy z przeciwwagą.
3. Wstępne ustawienie prawidłowego wyrównania z biegunem główicy.
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 główicy 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 [12]. 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 [7] 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

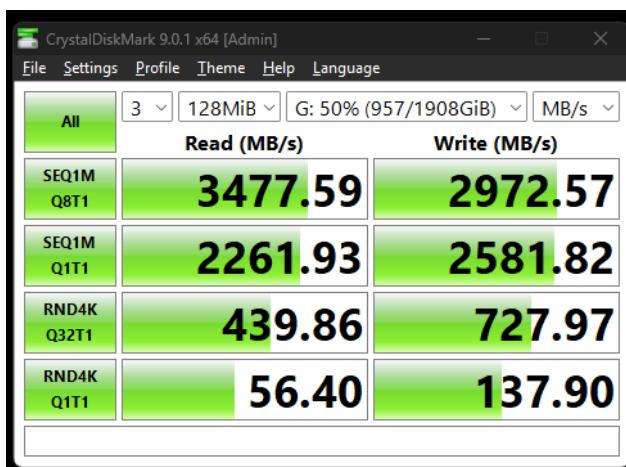
Zostały przeprowadzone badania dotyczące jakości zdjęć końcowych dla zastosowanych algorytmów nakładania zdjęć oraz ich czasu działania. Celem analizy jest określenie, który z algorytmów należy stosować dla uzyskania najlepszej ostrości zdjęcia oraz czasu działania.

## 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.

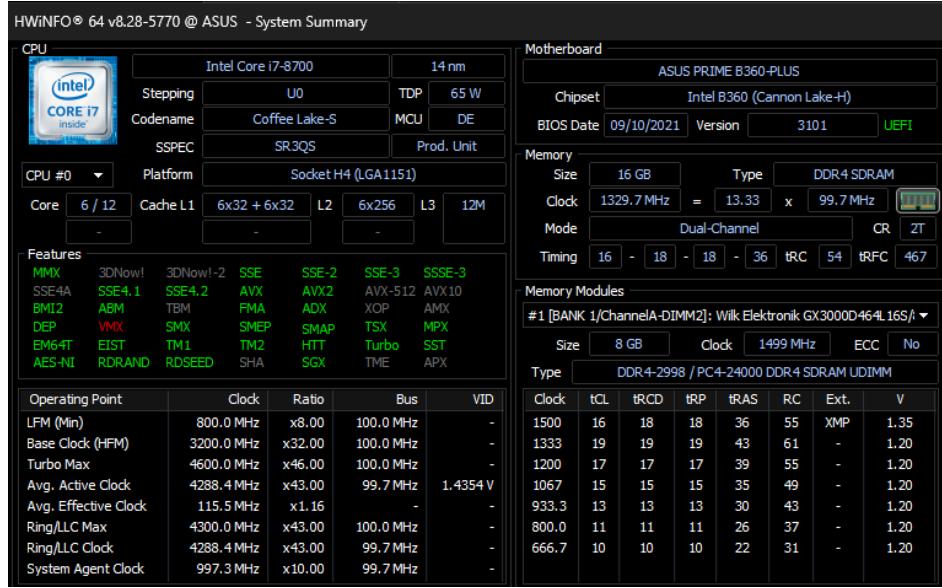
Przekrość odczytu danych z dysku może znacząco wpływać na czas działania programu. Przeprowadzone zostały testy wydajności dysku w celu realistycznego odzwierciedlenia czasu działania programu. Do badań dysku SSD użyto darmowego programu CrystalDiskMark dostępnego na oficjalnej stronie producenta [1]. 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 [4], 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: Przedstawienie użytego sprzętu.

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 [16]. 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 współczynnika 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

$\sigma_x, \sigma_y$  - współczynnik rozmycia obrazu

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

gdzie,

FWHM - szerokość połówkowa

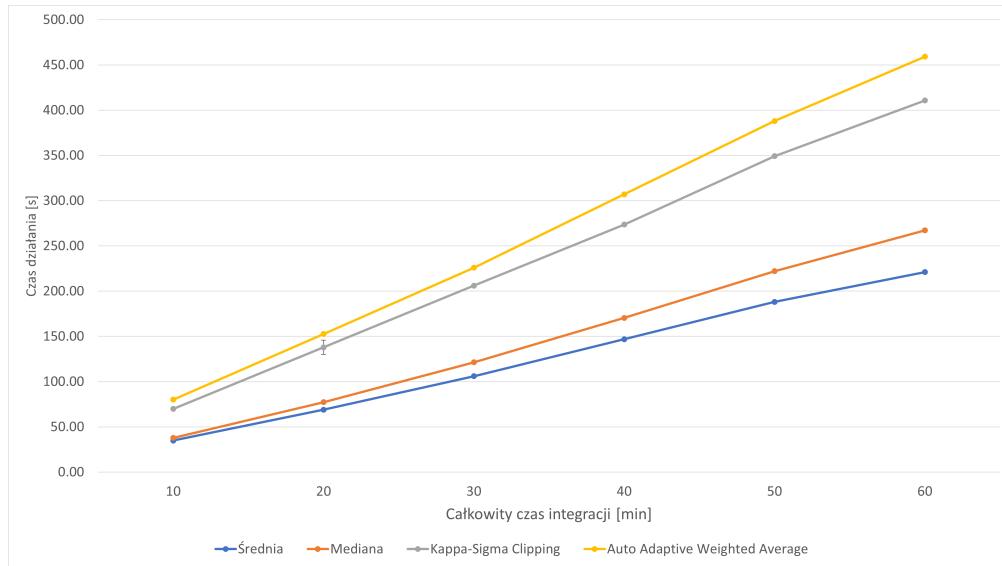
$\sigma$  - współczynnik rozmycia obrazu

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 algorytmu obliczającego wartości szerokości połówkowej oblicza FWHM dla wszystkich wykrytych gwiazd. W celu uproszczenia interpretacji, wyniki badań zostaną przedstawione jako mediana wartości FWHM wykrytych gwiazd na obrazie.

## 6.2. Wyniki eksperymentów

W tym rozdziale przedstawiono wyniki badań czasu działania programu w zależności od zastosowanej metody nakładania zdjęć oraz całkowitego czasu integracji zdjęć. Do interpretacji wyników należy pamiętać, że całkowity czas integracji nie jest porównywalny pomiędzy zestawami, ponieważ zestawy zdjęć wykorzystywały fotografie o różnym czasie naświetlania pojedynczego zdjęcia. Zestaw 1499 wykorzystywał zdjęcia o czasie naświetlania 60 sekund w porównaniu do zestawu 244, który wykorzystywał zdjęcia o czasie naświetlania 120 sekund. Dodatkowy wpływ na wyniki czasu działania Dodatkowy wpływ na wyniki czasu działania ma również wykorzystanie klatek kalibracyjnych do korekcji zdjęć przed nałożeniem w zestawie 1499. Badany został całkowity czas działania programu, który obejmował czas wczytania zdjęć, wykonania kalibracji, rejestracji oraz nałożenia zdjęć. Do czasu działania programu nie wliczano jednak czasu potrzebnego na generację klatek kalibracyjnych.

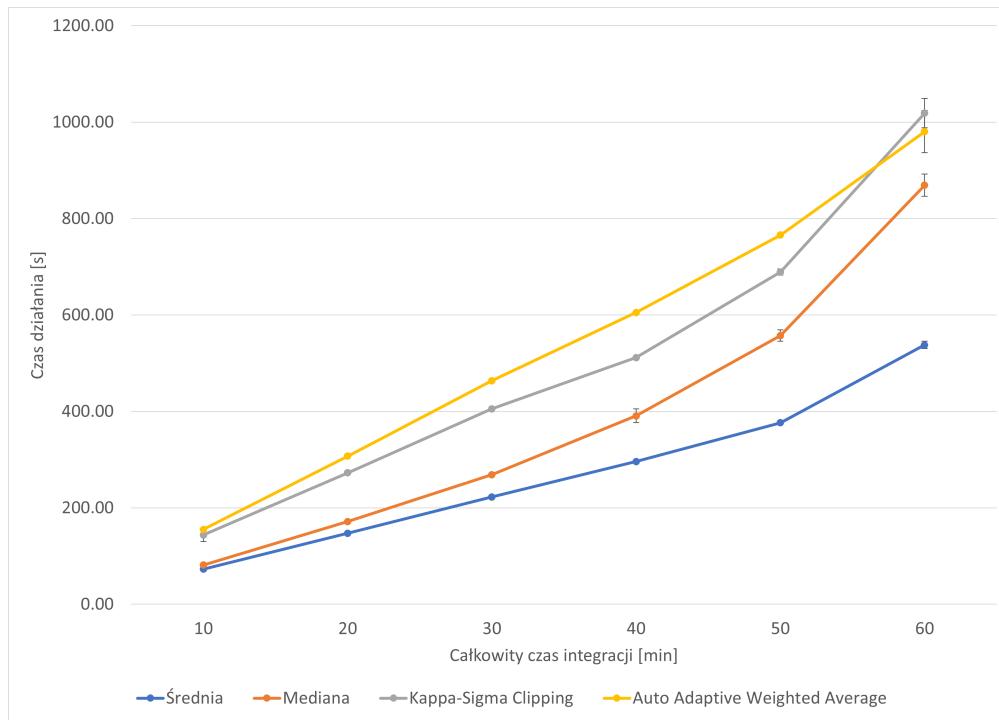
Poniżej przedstawiono wyniki eksperymentów dla czasu działania programu dla zestawów 244 oraz 1499. Czas działania programu mierzono dla czterech metod nakładania zdjęć: średnia, mediana, Kappa-Sigma Clipping oraz Auto Adaptive Weighted Average. Wskazano różnice w czasie działania programu w zależności od całkowitego czasu integracji zdjęć. Dla każdej kombinacji czasu integracji oraz badanego algorytmu wykonano pięć iteracji programu obliczono następnie średni czas działania oraz błąd. Jako błąd pomiaru przyjęto jedno odchylenie standardowe. Przed obliczeniem ostatecznych wyników zdecydowano się na inwalidację błędów grubych, które zdefiniowano na czasy odbiegające znacznie od średniego czasu działania dla badanej kombinacji. Jedyny taki błąd gruby zaobserwowano dla pojedynczego badania kombinacji zestawu 244 metodą Auto Adaptive Weighted Average przy czasie integracji 60 minut. Czas błędu grubego odstawał o ponad 50% od średniej wartości czasu działania dla tej kombinacji - po inwalidacji tego błędu grubego odchylenie standardowe pomiarów wyniosło poniżej 1 sekundy. Na wykresach 6.3 oraz 6.4 przedstawiono graficzne porównanie średnich czasów działania programu w zależności do zastosowanej metody nakładania zdjęć oraz całkowitego czasu integracji odpowiednio dla zestawu 244 oraz zestawu 1499. Dokładne wartości średnich oraz błędów przedstawiono odpowiednio w tabelach 6.1 oraz 6.2 dla zestawu 244 oraz zestawu 1499.



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

Metoda	Czas integracji [min]					
	10	20	30	40	50	60
Średnia	$\bar{t}$ [s]	34.93	68.94	106.03	146.79	188.19
	$\sigma$ [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
	$\sigma$ [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
	$\sigma$ [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
	$\sigma$ [s]	0.09	1.34	0.22	0.89	0.61

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

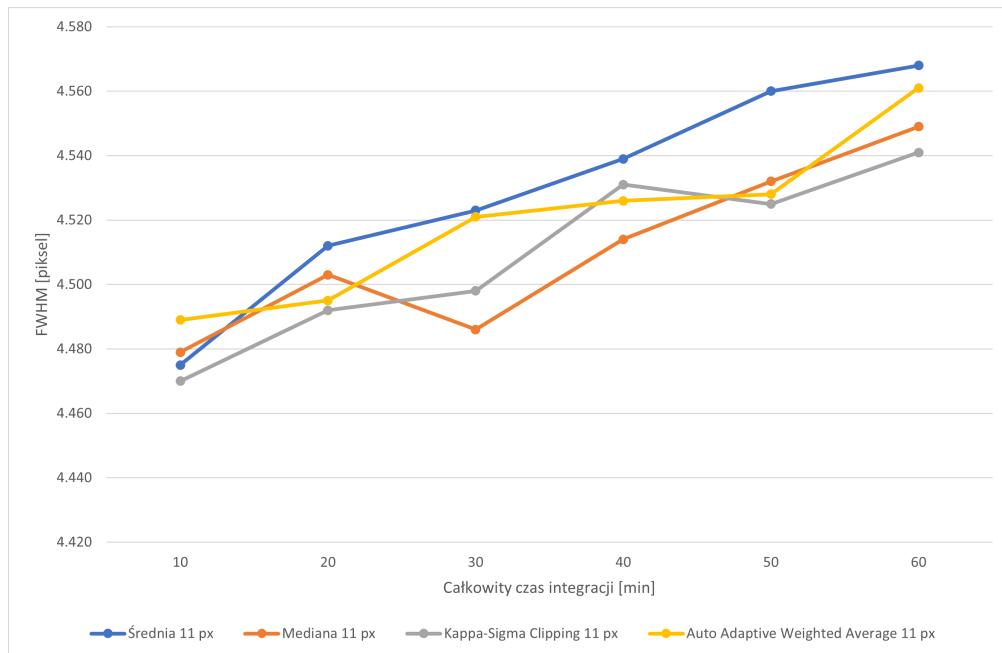


Rysunek 6.4: Porównanie średnich czasów nakładania zdjęć i odchylenia standardowego pomiarów w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 1499

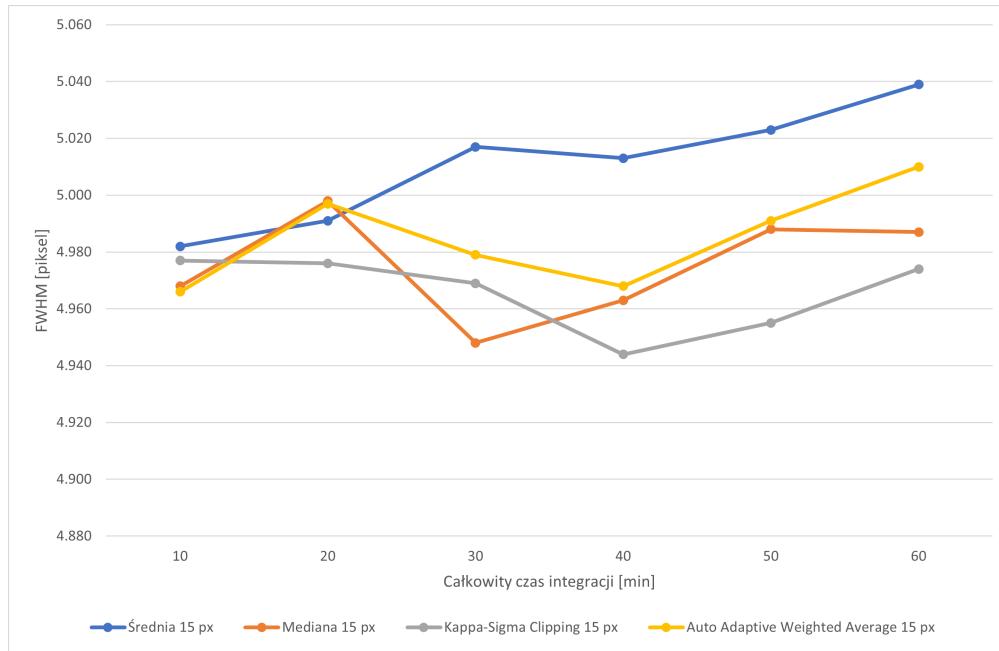
Metoda	Czas integracji [min]					
	10	20	30	40	50	60
Średnia	$\bar{t}$ [s]	72.95	147.10	222.44	296.02	376.46
	$\sigma$ [s]	0.30	0.40	0.49	2.55	0.78
Mediana	$\bar{t}$ [s]	81.47	171.42	268.77	390.84	557.23
	$\sigma$ [s]	0.41	0.40	0.43	14.39	11.88
Kappa-Sigma Clip.	$\bar{t}$ [s]	143.72	272.69	405.22	511.84	688.99
	$\sigma$ [s]	13.38	2.08	1.60	2.12	6.46
AA Weighted Avg.	$\bar{t}$ [s]	155.23	307.36	463.54	605.03	765.73
	$\sigma$ [s]	0.38	0.77	2.96	0.84	2.29

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

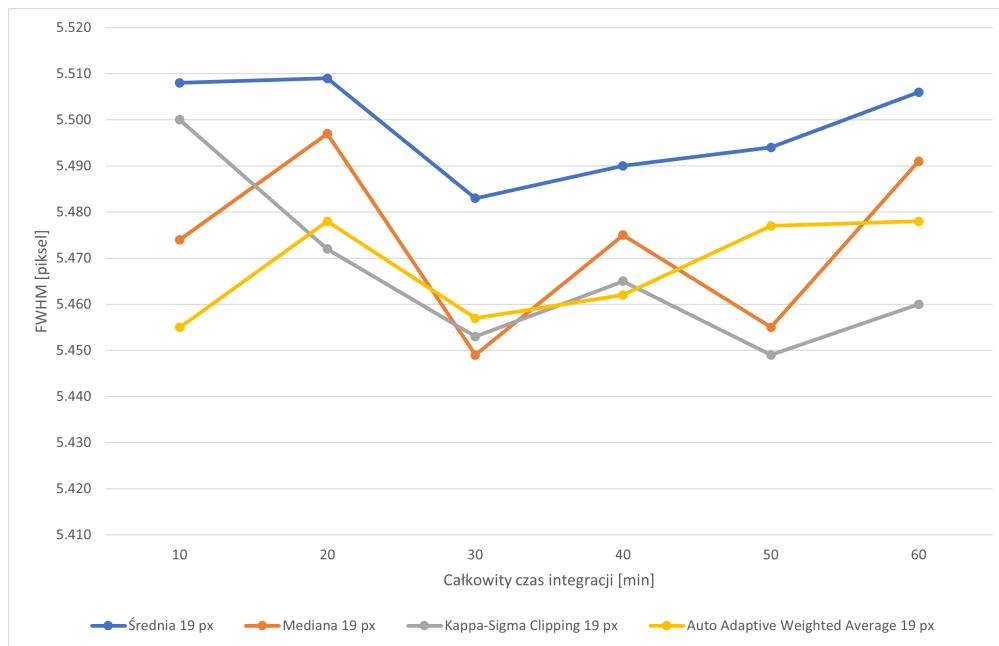
Poniżej przedstawiono wyniki eksperymentów dla FWHM zdjęć wynikowych dla zestawów 244 oraz 1499. FWHM zdjęć wynikowych mierzono dla czterech metod nakładania zdjęć: średnia, mediana, Kappa-Sigma Clipping oraz Auto Adaptive Weighted Average. Wskazano różnice w FWHM zdjęć w zależności od całkowitego czasu integracji zdjęć. Wyniki obliczenia FWHM są stałe dla pojedynczego zdjęcia i zależą tylko od określonego okna pomiarowego oraz obliczonego centrum gwiazdy. Zdecydowano się więc na przedstawienie wyników FWHM jako pojedyncznego pomiaru. Algorytm wykrywania gwiazd nie jest w pełni skuteczny, dlatego zdecydowano się na jednokrotne ustalenie pozycji gwiazd oraz ich centrum dla podstawie zdjęcia referencyjnego, którym było zdjęcie wynikowe uzyskane metodą Auto Adaptive Weighted Average przy maksymalnym badanym czasie integracji. Badanie FWHM przeprowadzone w ten sposób ma jednak możliwość wprowadzenia błędów spowodowanych innym położeniem centrum gwiazdy na zdjęciach wynikowych uzyskanych przez różne algorytmy oraz różne czasy integracji. Podany sposób badań jest jednak niezbędny do eliminacji wpływu detekcji gwiazd na wyniki FWHM. Wyniki FWHM przedstawiono dla trzech różnych rozmiarów okna pomiarowego: 11, 15 oraz 19 pikseli. Na wykresach 6.5, 6.6 oraz 6.7 przedstawiono graficzne porównanie wartości FWHM w zależności do zastosowanej metody nakładania zdjęć oraz całkowitego czasu integracji odpowiednio dla zestawu 244. Wyniki dla zestawu 1499 przedstawiono na wykresach 6.8, 6.9 oraz 6.10. Dokładne wartości wyników FWHM znajdują się odpowiednio w tabelach 6.3 oraz 6.4 dla zestawu 244 oraz zestawu 1499.



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



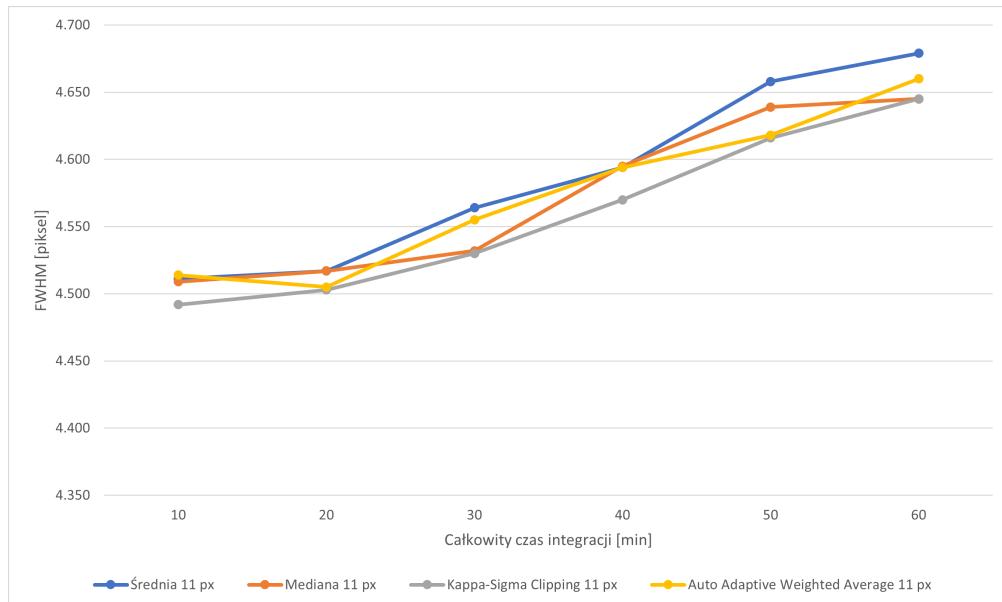
Rysunek 6.6: Porównanie FWHM zestawów 244 w zależności od metody nakładania zdjęć i czasu integracji dla okna 15 pikseli.



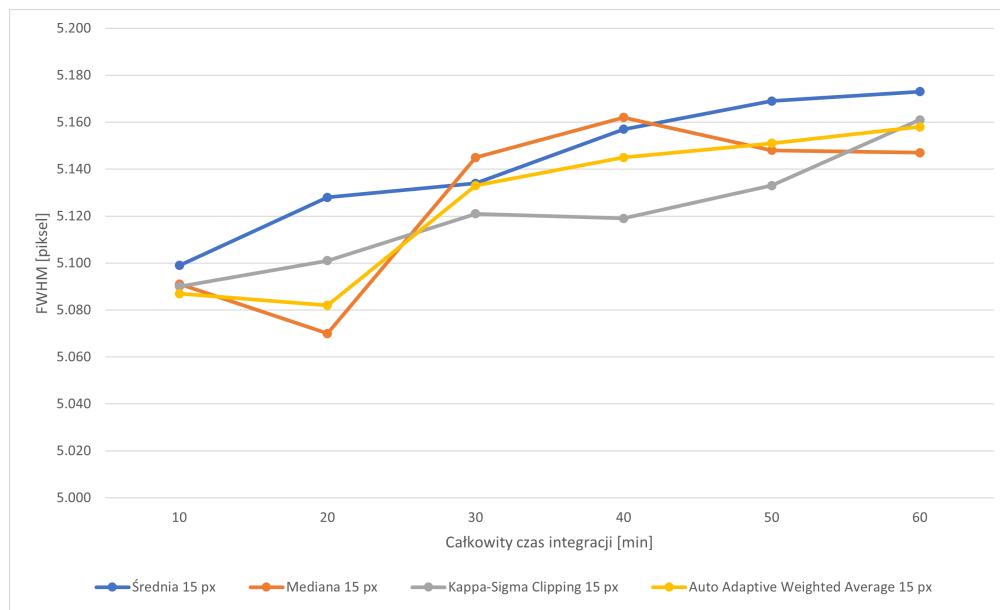
Rysunek 6.7: Porównanie FWHM zestawów 244 w zależności od metody nakładania zdjęć i czasu integracji dla okna 19 pikseli.

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
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

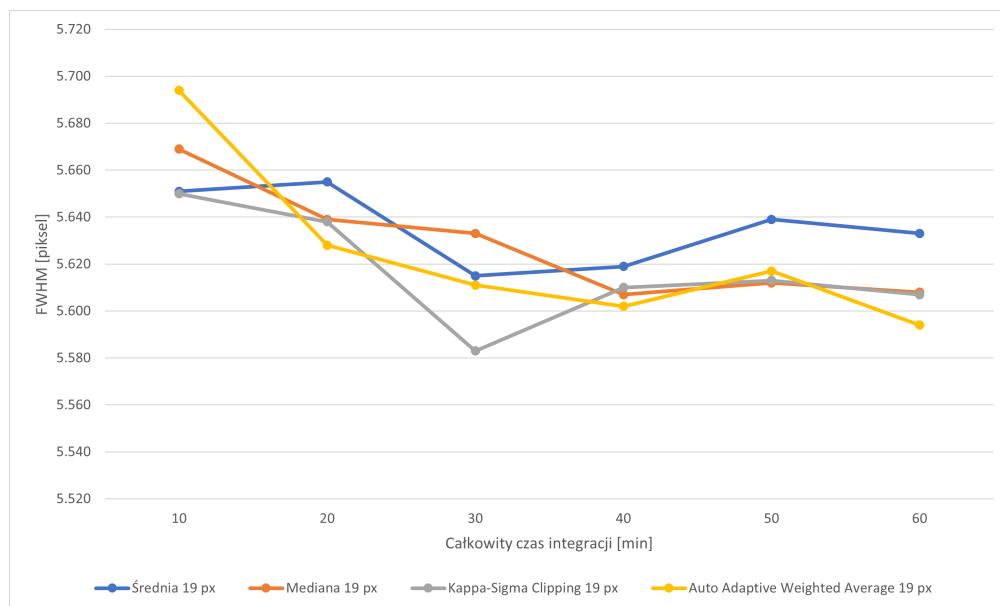
Tabela 6.3: Porównanie wartości FWHM w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 244.



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.

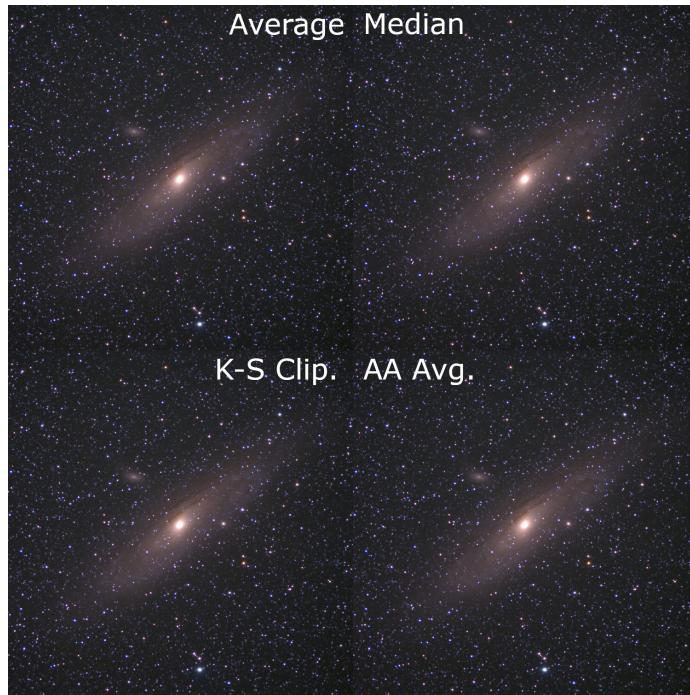


Rysunek 6.10: Porównanie FWHM zestawów 1499 w zależności od metody nakładania zdjęć i czasu integracji dla okna 19 pikseli.

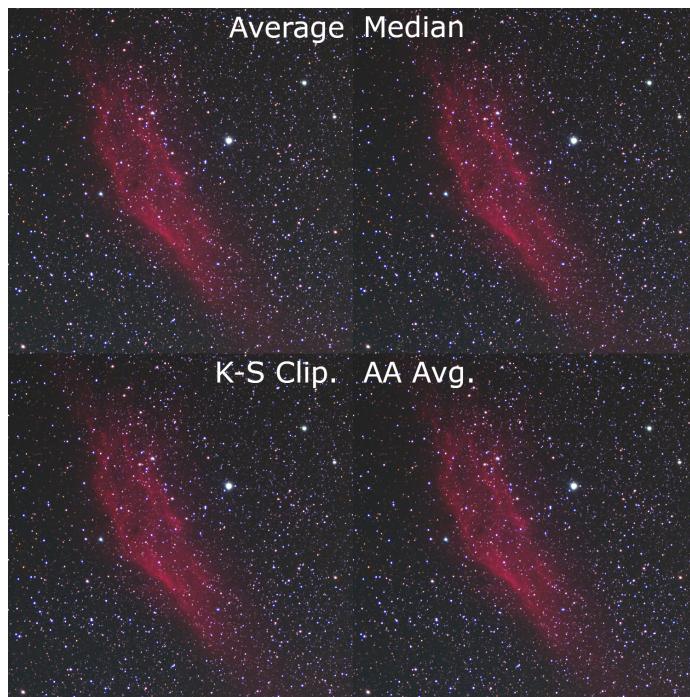
Metoda		Czas integracji [min]					
		10	20	30	40	50	60
Średnia	11 px [px]	4.511	4.517	4.564	4.594	4.658	4.679
	15 px [px]	5.099	5.128	5.134	5.157	5.169	5.173
	19 px [px]	5.651	5.655	5.615	5.639	5.619	5.633
Medianą	11 px [px]	4.509	4.517	4.532	4.595	4.639	4.645
	15 px [px]	5.091	5.070	5.145	5.162	5.148	5.147
	19 px [px]	5.669	5.639	5.633	5.607	5.612	5.608
Kappa–Sigma Clip.	11 px [px]	4.492	4.503	4.530	4.570	4.616	4.645
	15 px [px]	5.090	5.101	5.121	5.119	5.133	5.161
	19 px [px]	5.650	5.638	5.583	5.610	5.613	5.607
AA Weighted Avg.	11 px [px]	4.514	4.505	4.555	4.594	4.618	4.660
	15 px [px]	5.087	5.082	5.133	5.145	5.151	5.158
	19 px [px]	5.694	5.628	5.611	5.602	5.617	5.594

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

Poniżej przedstawiono wizualne porównanie tych samych wycinków zdjęć wynikowych przedstawiających ten sam obiekt dla różnych algorytmów nakładania zdjęć. Każde ze zdjęć wynikowych zostało uzyskane poprzez nałożenie zdjęć z całkowitym czasem integracji 60 minut. Do wykonania zdjęć wynikowych obiektu 244 nie użyto klatek kalibracyjnych, natomiast do wykonania zdjęć wynikowych obiektu 1499 użyto klatki kalibracyjne. Każda z fotografii wynikowych została zmodyfikowana w ten sam sposób w celu poprawy widoczności szczegółów na zdjęciach - zastosowano krzywe oraz zwiększeno nasycenie kolorów. Do edycji zdjęć użyto darmowy program GNU Image Manipulation Program (GIMP) możliwy do pobrania z oficjalnej strony producenta [3]. Dla zestawu 244 obiektem fotografowanym był obiekt NGC 244 - galaktyka spiralna w gwiazdozbiorze Andromedy znana jako galaktyka Andromedy. Porównanie wizualne zdjęć zestawu 244 przedstawiono na rysunku 6.11. Dla zestawu 1499 obiektem fotografowanym był obiekt NGC 1499 - mgławica emisyjna w gwiazdozbiorze Perseusza znana jako mgławica Kalifornia. Porównanie wizualne zdjęć zestawu 1499 przedstawiono na rysunku 6.12. Ze względu na różne warunki obserwacyjne, jasność nieba w zenicie oraz różny czas naświetlania pojedynczego zdjęcia, fotografie wynikowe nie powinny zostać porównywane pomiędzy zestawami.

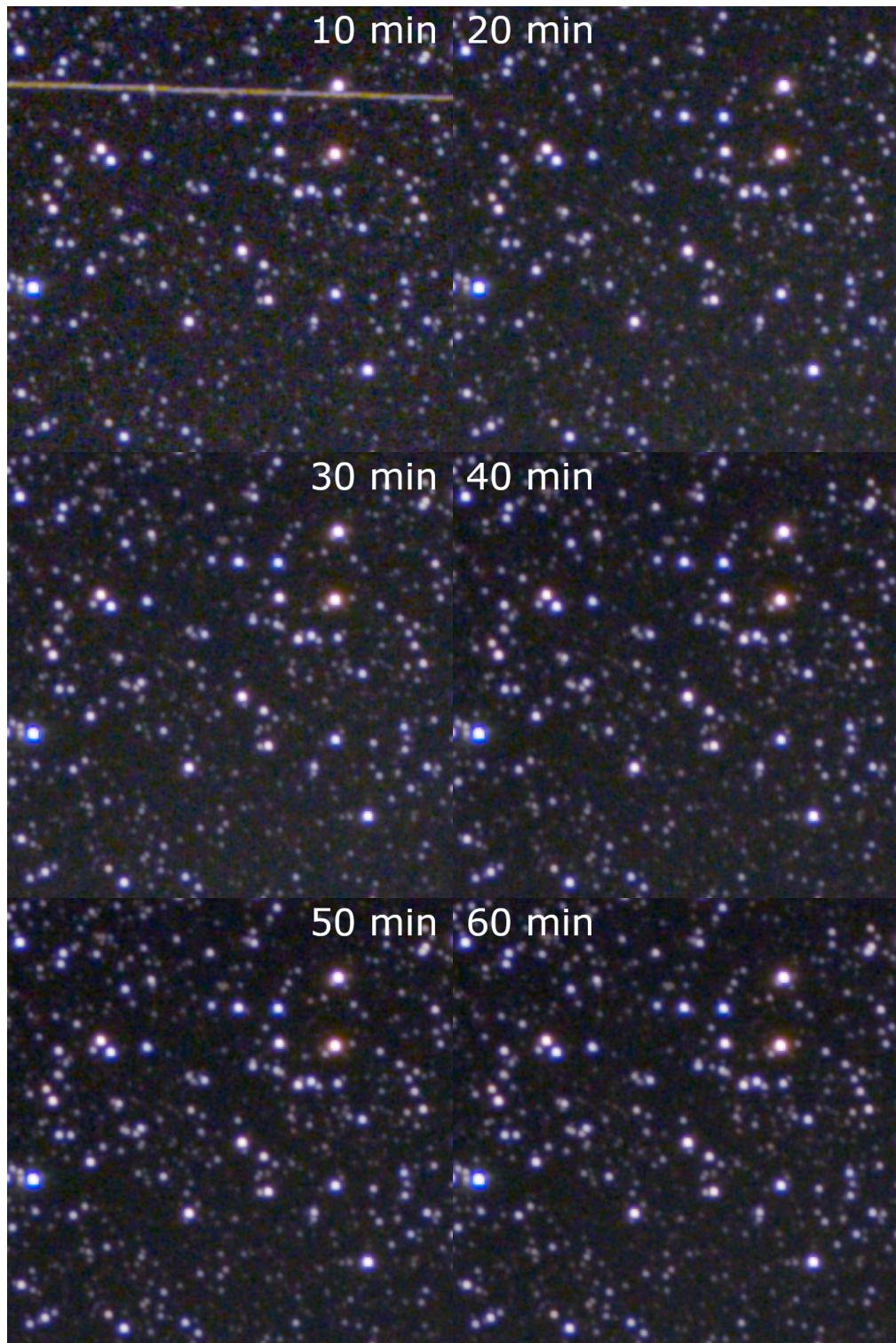


Rysunek 6.11: Porównanie wizualne obiektu z zestawu 244 o całkowitym czasie integracji 60 min. dla różnych algorytmów.

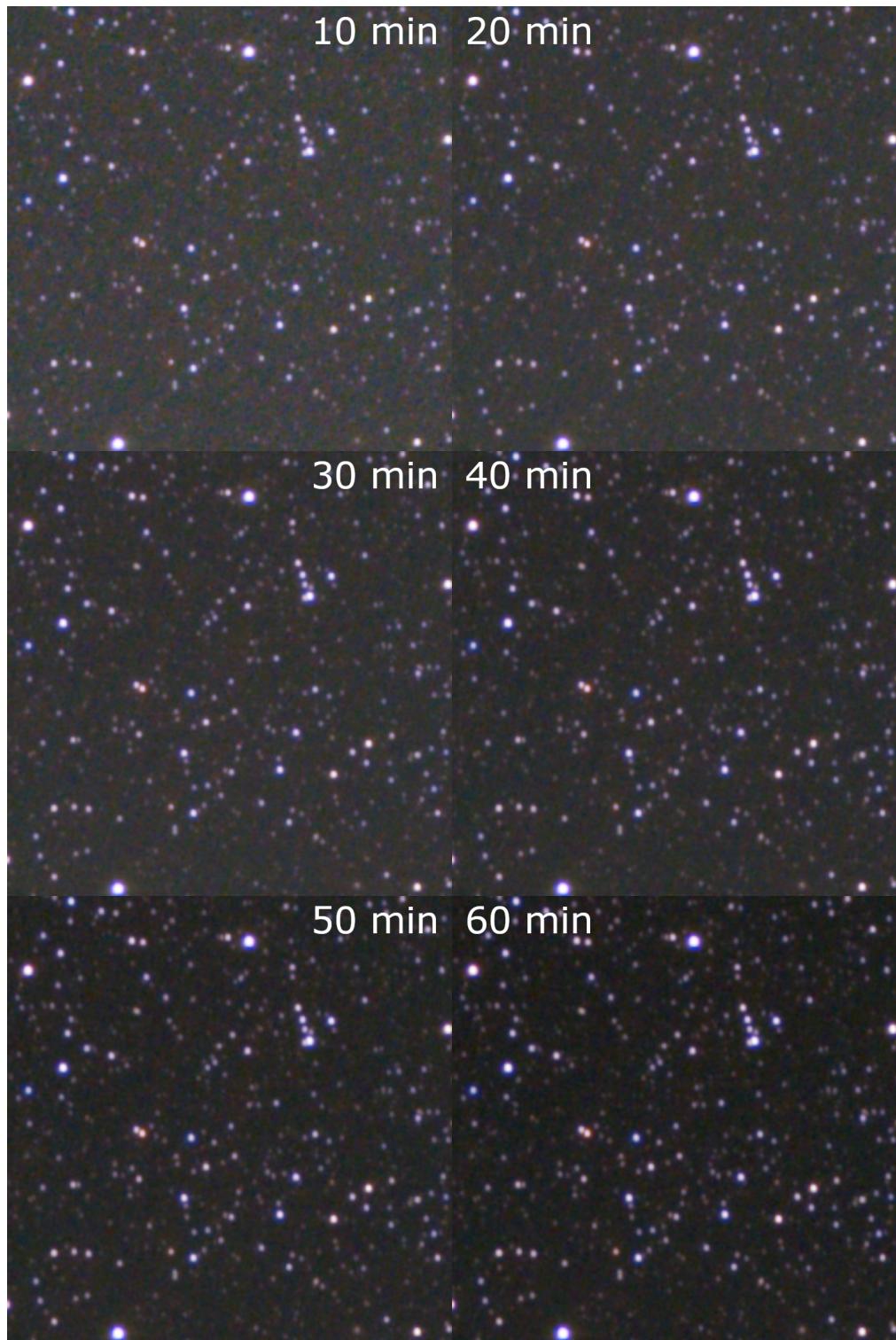


Rysunek 6.12: Porównanie wizualne obiektu z zestawu 1499 o całkowitym czasie integracji 60 min. dla różnych algorytmów.

Poniżej przedstawiono wizualne porównanie tych samych wycinków zdjęć wynikowych dla różnych czasów integracji zdjęć. Obrazy wynikowe zostały wykonane korzystając z algorytmu Kappa-Sigma Clipping. Każde ze zdjęć wynikowych zostało uzyskane poprzez nałożenie zdjęć z różnym całkowitym czasem integracji opisanym na rysunku. Każda z fotografii wynikowych została zmodyfikowana w ten sam sposób w celu poprawy widoczności szczegółów na zdjęciach - zastosowano krzywe oraz zwiększone nasycenie kolorów. Do wykonania zdjęć wynikowych obiektu 244 nie użyto klatek kalibracyjnych, natomiast do wykonania zdjęć wynikowych obiektu 1499 użyto klatki kalibracyjne. Ze względu na różne warunki obserwacyjne, jasność nieba w zenicie oraz różny czas naświetlania pojedynczego zdjęcia, fotografie wynikowe nie powinny zostać porównywane pomiędzy zestawami.



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.

### 6.3. Analiza wyników

W niniejszym rozdziale dokonano analizy wyników eksperymentów zebranych w rozdziale 6.2. Skupiono się na dwóch głównych aspektach: czasie wykonania poszczególnych metod nakładania zdjęć oraz jakości obrazu mierzonej za pomocą FWHM. Określenie czy przedstawione dane można interpretować za liniowe bądź wykładnicze określono korzystając z programu Microsoft Excel na podstawie funkcji dopasowania krzywej trendu do danych eksperymentalnych. Określenie dopasowania krzywej trendu do danych eksperymentalnych dokonano na podstawie współczynnika determinacji  $R^2$  - linia trendu z najwyższą wartością współczynnika była uznawana za najlepszą reprezentację krzywej do danych.

Analiza wyników eksperymentów dla zestawu 244 zawartych w tabeli 6.1 oraz na wykresie 6.3 pokazuje wyraźne różnice w czasie wykonania pomiędzy metodami prostymi: średnia oraz mediana, a metodami złożonymi: Kappa-Sigma Clipping oraz Auto Adaptive Weighted Average. Wyznaczona krzywa wskazuje na to, że do 30 zdjęć wszystkie z badanych algorytmów mają liniowy czas wykonania. Wzrost całkowitego czasu integracji powoduje większy wzrost czasu działania programu dla algorytmu Auto Adaptive Weighted Average w porównaniu do Kappa-Sigma Clipping, który wynosi od około 15% różnicy dla pięciu łączonych zdjęć (oznaczonych jako całkowity czas integracji równy 10 minut) do prawie 18% różnicy dla trzydziestu łączonych zdjęć. Różnice w czasie działania programu pomiędzy algorytmami średniej i mediany są znacznie większe wynosząc od 8% dla pięciu łączonych zdjęć do ponad 20% dla trzydziestu łączonych zdjęć. Największą odpornością na wzrost czasu działania programu względem całkowitego czasu integracji charakteryzuje się algorytm Auto Adaptive Weighted Average, którego czas działania programu wzrósł o 573% pomiędzy pięcioma a trzydziestoma łączonymi zdjęciami, najmniejszą odporność wykazuje algorytm mediany, którego czas działania wzrósł o ponad 706% w tym samym zakresie.

Analiza wyników eksperymentów dla zestawu 1499 zawartych w tabeli 6.2 oraz na wykresie 6.4 pokazuje na nagły wzrost czasu działania programu powyżej sześćdziesięciu łączonych zdjęć dla wszystkich badanych algorytmów. Największym wzrostem czasu działania programu od pięćdziesięciu do sześćdziesięciu łączonych zdjęć charakteryzują się algorytmy Kappa-Sigma Clipping oraz mediany, dla których średni czas działania programu wzrósł o odpowiednio 48% oraz 56%. Najmniejszym wzrostem czasu działania w tym zakresie charakteryzuje się algorytm Auto Adaptive Weighted Average, którego średni czas działania programu wzrósł o 28%. Algorytm mediany wykazuje najmniejszą odporność na wzrost czasu działania najprawdopodobniej ze względu na konieczność sortowania wartości pikseli dla każdego piksela w obrazie wejściowym. W celu zmniejszenia czasu działania programu dla tego algorytmu można zastosować algorytmu umożliwiające na znalezienie mediany w czasie liniowym. Dla metody Kappa-Sigma Clipping nie zastosowano optymalizacji pamięciowych tak jak w algorytmie Auto Adaptive Weighted Average, co może być powodem nagłego wzrostu czasu działania. W celu dostarczenia dokładniejszej analizy czasów działania należy wykonać badania zużycia pamięci przez poszczególne algorytmy oraz badania dla większej ilości danych.

Analiza eksperymentów zależności FWHM dla całkowitego czasu integracji bazowana na wynikach przedstawionych na wykresach 6.5 - 6.10 oraz w tabelach 6.3 oraz 6.4, rozmiarom okna pomiarowego oraz metody nakładania zdjęć dla obu badanych zestawów danych nie wykazała jednoznacznie przewagi żadnej z metod nakładania zdjęć na ostrość obrazu wynikowego dla wszystkich badanych rozmiarów okna. Drobne różnice są obserwowane lokalnie - algorytm

Kappa-Sigma Clipping reprezentuje najwyższe wartości FWHM dla największych czasów integracji dla wszystkich okien pomiarowych w zestawie 244 oraz dla okna 11 pikseli w zestawie 1499, wielkość tych różnic jest jednak niewielka. Algorytm średniej dla większości badanych przypadków charakteryzuje się najwyższymi wartościami FWHM. Wynika to prawdopodobnie z braku eliminacji wartości odstających, które mogą powodować rozmycie obrazu końcowego. Brak idealnego dopasowania transformacji afonicznej pomiędzy poszczególnymi zdjęciami w zestawie może również powodować rozmycie obrazu końcowego, w celu dokładniejszej weryfikacji wyników tego zjawiska należy zbadać wpływ dokładności dopasowania transformacji na wartość FWHM obrazów wynikowych - obecnie stosowana tolerancja jednego piksela może mieć znaczący wpływ na końcowe wartości FWHM zważając na fakt, że różnice pomiędzy algorytmami w badanych przypadkach są o rząd wielkości mniejsze. Dla najmniejszego okna pomiarowego wartości FWHM mają tendencję do niewielkiego wzrostu wraz ze wzrostem całkowitego czasu integracji, natomiast dla okien 15 i 19 pikseli zależność jest mniej spójna i niemożliwe jest wyciągnięcie jednoznacznych wniosków. Zależności opisane powyżej sugerują, że przyrost SNR związany z dłuższym czasem integracji nie przekłada się bezpośrednio na istotne zmniejszenie FWHM. W celu dokładniejszej analizy wyników FWHM należy przeprowadzić dodatkowe badania zależności FWHM danych wejściowych zmiany wartości FWHM obrazów wynikowych. Zmienne wartości widzenia, temperatury i ostrości zdjęć w ciągu nocy obserwacyjnej mogą mieć wpływ na ostateczne wartości FWHM obrazów wynikowych - te zmiany mogą mieć różny wpływ w zależności od badanego algorytmu.

Porównania wizualne edytowanych plików wynikowych przedstawionych na rysunkach [6.11](#) oraz [6.12](#) pozwalają na subiektywną ocenę jakości obrazu uzyskanego różnymi algorytmami pod względem poziomu szumu oraz zachowania ostrości szczegółów. Algorytmy Kappa-Sigma Clipping oraz Auto Adaptive Weighted Average wykazują nieznaczną przewagę w poziomie tła widoczną głównie w obszarach pozbawionych gwiazd oraz w zachowaniu struktur o niskim kontraście - różnice są jednak subtelne. Na porównaniach wizualnych zawartych na rysunkach [6.13](#) oraz [6.14](#) skupiono się na ocenie wpływu całkowitego czasu integracji dla wybranego algorytmu. Ze względu na wyniki wizualne oraz FWHM zdecydowano się na wykorzystanie algorytmu Kappa-Sigma Clipping jako przykład - tendencje spadkowe pozostają jednak podobne dla wszystkich algorytmów. Porównania te pokazują spadek poziomu szumu wraz ze wzrostem całkowitego czasu integracji zgodnie z oczekiwaniemi. Efekt jest bardziej widoczny w zestawie 1499 - jest to możliwe ze względu na zastosowaną kalibrację zdjęć wejściowych w przeciwieństwie do zestawu 244, gdzie nie wykorzytano zdjęć kalibracyjnych. Alternatywnie, większy spadek w zestawie 1499 można tłumaczyć wyższym poziomem zanieczyszczenia świetlnego. W takich warunkach wydłużenie czasu integracji wykazuje wyraźniejszą poprawę stosunku sygnału do szumu, ponieważ SNR wzrasta wraz z pierwiastkiem czasu. W celu obiektywnej oceny jakości obrazu należy przeprowadzić analizę algorytmiczną poziomu sygnału do szumu. Podczas tej pracy nie udało się przeprowadzić analizy SNR ze względu na wymagany czas opracowania odpowiednich narzędzi do pomiaru SNR w obrazach astronomicznych.

## 7. Wnioski

W pracy zrealizowano implementację narzędzia do nakładania zdjęć obiektów głębiokiego nieba z wykorzystaniem czterech zaimplementowanych algorytmów: średniej, mediany, Kappa-Sigma Clipping oraz Auto Adaptive Weighted Average z uwzględnieniem przesunięć pomiędzy zdjęciami oraz wykorzystaniem zdjęć kalibracyjnych. Dokonano analizy porównawczej czasów działania programu w zależności od wybranego algorytmu i wykorzystywanych zestawów danych, oraz jakości uzyskanego obrazu mierzonej za pomocą szerokości połówkowej gwiazd. Przeprowadzone badania wykazały, że system prawidłowo realizuje zadanie wykonywania obrazów wynikowych.

Metody proste - średnia oraz mediana - charakteryzują się krótszym czasem działania programu w porównaniu do metod złożonych - Kappa-Sigma Clipping oraz Auto Adaptive Weighted Average, zwłaszcza dla większej ilości łączonych zdjęć. Dla badanych zestawów zauważono nagły wzrost czasu działania programu powyżej sześćdziesięciu łączonych zdjęć. Analiza FWHM nie wykazała jednoznacznej przewagi żadnej z metod nakładania zdjęć na ostrość obrazu wynikowego. Wizualne porównanie wykorzystanych algorytmów, dla jednej godziny całkowitego czasu integracji fotografii, sugerują nieznaczne zmiany w jakości obrazu wynikowego na korzyść algorytmów złożonych (Kappa-Sigma Clipping oraz Auto Adaptive Weighted Average).

Wraz ze wzrostem całkowitego czasu integracji znacząco spada poziom szumu w obrazie wynikowym. Należy jednak zbadać dokładny wpływ zastosowanych algorytmów oraz całkowitego czasu integracji na poziom sygnału do szumu w obrazie wynikowym korzystając z obiektywnych metod miernika SNR.

Dla obecnej implementacji programowej zaleca się korzystanie z algorytmów Auto Adaptive Weighted Average lub Kappa-Sigma Clipping dla użytkowników ceniących jakość obrazu wynikowego, kosztem dłuższego czasu działania programu. Dla użytkowników preferujących krótszy czas działania aplikacji, kosztem jakości obrazu, zaleca się korzystanie z algorytmu średniej lub jeżeli występują jasne artefakty na zdjęciach wejściowych, takie jak satelity czy meteory, algorytmu mediany. Dodatkowo w celu poprawy jakości obrazu wynikowego, dla użytkowników zaawansowanych, zaleca się stosowanie odpowiednio wykonanych zdjęć kalibracyjnych.

W przyszłości planuje się rozszerzenie analizy o dokładne pomiary SNR i badanie zużycia pamięci przez poszczególne algorytmy. Dodatkowo zostaną wykonane badania wpływu parametrów dla algorytmu Kappa-Sigma Clipping i wpływu parametrów i wykorzystanej funkcji ważenia dla algorytmu Auto Adaptive Weighted Average na czas działania programu oraz jakość obrazu wynikowego korzystając z bardziej zaawansowanych technik badawczych.

Implementację programową należy rozszerzyć o obsługę większej ilości formatów plików wejściowych oraz o interfejs graficzny ułatwiający korzystanie z narzędzia. W celu zwiększenia wydajności programu można rozważyć implementację algorytmów z wykorzystaniem przetwarzania równoległego, wykorzystania wektoryzacji SIMD lub implementację programu z wykorzystaniem kart graficznych. Do rozszerzenia funkcjonalności aplikacji można rozważyć

implementację dodatkowych metod nakładania zdjęć, możliwość dostosowania parametrów algorytmów przez użytkownika i rozszerzenie procesu rejestracji zdjęć o wykorzystanie gwiazd referencyjnych z katalogów astronomicznych do potencjalnego zwiększenia dokładności rejestracji.

# 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] GIMP - oficjalna strona. <https://www.gimp.org/>. Data Dostępu: 26 listopada 2025.
- [4] HWiNFO - oficjalna strona. <https://www.hwinfo.com/>. Data Dostępu: 21 listopada 2025.
- [5] Instrukcja obsługi montażu Star Watcher Star Adventurer. [https://inter-static.skywatcher.com/downloads/StarAdv\\_manual\\_150722V2\\_updateds.pdf](https://inter-static.skywatcher.com/downloads/StarAdv_manual_150722V2_updateds.pdf). Data Dostępu 11 Listopada 2025.
- [6] LibRaw - oficjalna strona. <https://www.libraw.org/>. Data Dostępu: 21 listopada 2025.
- [7] Light Pollution Map - strona aplikacji. <https://lightpollutionmap.app>. Data Dostępu: 9 listopada 2025.
- [8] OpenCV - oficjalna strona. <https://opencv.org/>. Data Dostępu: 21 listopada 2025.
- [9] Photons to Photos - oficjalna strona. <https://www.photonstophotos.net>. Data Dostępu: 8 listopada 2025.
- [10] PixInsight - oficjalna strona. <https://pixinsight.com/>. Data Dostępu: 6 listopada 2025.
- [11] Siril - oficjalna strona. <https://siril.org/>. Data Dostępu: 6 listopada 2025.
- [12] 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.
- [13] J. Beish. Astronomical seeing. <https://apps.dtic.mil/sti/tr/pdf/ADA391637.pdf>, 2001. Technical Report ADA391637. Data dostępu: 11 listopada 2025.
- [14] 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.
- [15] 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.

- [16] 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.
- [17] O. Hainaut. Signal, noise and detection. <https://www.eso.org/~ohainaut/ccd/sn.html>, 2005. Data dostępu: 31 Października 2025.
- [18] S. Howell and A. Tavackolimehr. *Handbook of CCD Astronomy*. Cambridge University Press, 2nd edition, 2006.
- [19] 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.
- [20] C. U. C. Nguyen and T. A. Lovell. *A Practical Approach to Plate-Solving of Astronomical Images Based on Least-Squares Estimation*. 2025.
- [21] P. B. Stetson. The techniques of least squares and stellar photometry with ccds. [https://ned.ipac.caltech.edu/level5/Stetson/Stetson\\_contents.html](https://ned.ipac.caltech.edu/level5/Stetson/Stetson_contents.html), 1994. Data Dostępu: 20 listopada 2025.

# Spis rysunków

4.1 Główne założenia działania programu.	13
6.1 Wyniki testu wydajności dysku SSD.	29
6.2 Przedstawienie użytego sprzętu.	30
6.3 Porównanie średnich czasów nakładania zdjęć i odchylenia standardowego pomiarów w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 244	32
6.4 Porównanie średnich czasów nakładania zdjęć i odchylenia standardowego pomiarów w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 1499	33
6.5 Porównanie FWHM zestawów 244 w zależności od metody nakładania zdjęć i czasu integracji dla okna 11 pikseli.	34
6.6 Porównanie FWHM zestawów 244 w zależności od metody nakładania zdjęć i czasu integracji dla okna 15 pikseli.	35
6.7 Porównanie FWHM zestawów 244 w zależności od metody nakładania zdjęć i czasu integracji dla okna 19 pikseli.	35
6.8 Porównanie FWHM zestawów 1499 w zależności od metody nakładania zdjęć i czasu integracji dla okna 11 pikseli.	36
6.9 Porównanie FWHM zestawów 1499 w zależności od metody nakładania zdjęć i czasu integracji dla okna 15 pikseli.	37
6.10 Porównanie FWHM zestawów 1499 w zależności od metody nakładania zdjęć i czasu integracji dla okna 19 pikseli.	37
6.11 Porównanie wizualne obiektu z zestawu 244 o całkowitym czasie integracji 60 min. dla różnych algorytmów.	39
6.12 Porównanie wizualne obiektu z zestawu 1499 o całkowitym czasie integracji 60 min. dla różnych algorytmów.	39
6.13 Porównanie wizualne szumu dla różnego czasu integracji algorytmu Kappa-Sigma Clipping zestawów 244.	41
6.14 Porównanie wizualne szumu dla różnego czasu integracji algorytmu Kappa-Sigma Clipping zestawów 1499.	42



# Spis tabel

5.1	Opis podzielonych zestawów fotografii obiektów głębokiego nieba . . . . .	27
6.1	Porównanie średnich czasów nakładania zdjęć w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 244. . . . .	32
6.2	Porównanie średnich czasów nakładania zdjęć w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 1499. . . . .	33
6.3	Porównanie wartości FWHM w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 244. . . . .	36
6.4	Porównanie wartości FWHM w zależności od metody nakładania zdjęć i czasu integracji dla zestawów 1499. . . . .	38