

**Politechnika Warszawska**

**Wydział Matematyki i Nauk Informacyjnych**

---

**Biometria**

**Implementacja Edytora Obrazów - Dokumentacja**

---

Hubert Kowalski

Kierunek: Inżynieria i Analiza Danych

25 marca 2025

# **Spis treści**

<b>1 Wprowadzenie</b>	<b>3</b>
<b>2 Opis aplikacji</b>	<b>4</b>
2.1 Architektura systemu . . . . .	4
2.2 Opis modułów . . . . .	4
2.3 Wykorzystane technologie . . . . .	5
<b>3 Implementacja metod</b>	<b>6</b>
3.1 Operacje podstawowe . . . . .	6
3.1.1 Skala szarości . . . . .	6
3.1.2 Regulacja jasności (Brightness Adjustment) . . . . .	7
3.1.3 Regulacja kontrastu (Contrast Adjustment) . . . . .	8
3.1.4 Korekcja gamma (Gamma Correction) . . . . .	9
3.1.5 Binaryzacja obrazu . . . . .	10
3.1.6 Negatyw obrazu . . . . .	10
3.2 Filtry przestrzenne . . . . .	11
3.2.1 Filtr uśredniający (Image Averaging) . . . . .	11
3.2.2 Filtr Gaussa (Gaussian Blur) . . . . .	12
3.2.3 Filtr wyostrzający (Sharpening) . . . . .	13
3.2.4 Stosowanie własnego filtra splotowego (Custom Filter) . . . . .	13
3.2.5 Detekcja krawędzi - Operator Sobela . . . . .	14
3.2.6 Detekcja krawędzi - Krzyż Robertsa . . . . .	15
3.2.7 Detekcja krawędzi metodą Canny'ego . . . . .	16
3.3 Statystyki obrazu i projekcje . . . . .	16
3.4 Algorytm wyznaczania koloru dominującego . . . . .	18
<b>4 Interfejs użytkownika</b>	<b>20</b>
<b>5 Wnioski</b>	<b>21</b>
5.1 Wnioski z przetwarzania obrazów . . . . .	21
5.2 Trudności . . . . .	21

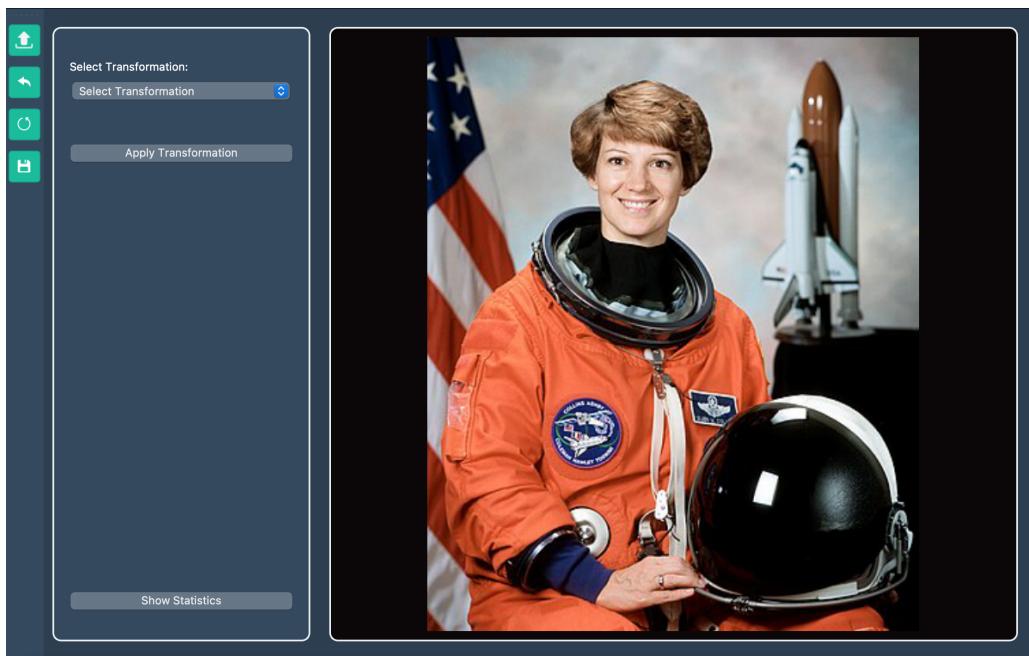
# 1 Wprowadzenie

Projekt został zrealizowany w ramach przedmiotu *Biometria* prowadzonego na Politechnice Warszawskiej. Głównym celem pracy było stworzenie aplikacji do przetwarzania obrazów z interfejsem graficznym umożliwiającym interaktywną pracę z różnymi technikami przetwarzania wizualnego.

Opracowane narzędzie stanowi kompleksowe środowisko do:

- Podstawowych operacji na obrazach (konwersja do skali szarości, negatyw, regulacja jasności i kontrastu)
- Zaawansowanych przekształceń z wykorzystaniem filtrów przestrzennych
- Analizy statystycznej obrazów poprzez generowanie histogramów i projekcji
- Detekcji krawędzi z wykorzystaniem różnych operatorów
- Eksperymentów z własnymi parametrami transformacji

Aplikacja implementuje wszystkie wymagania określone w specyfikacji projektu. Została ona zaprojektowana w sposób umożliwiający rozszerzenie o dodatkowe funkcjonalności. Wyróżnia się czystym interfejsem użytkownika, łatwością obsługi oraz możliwością sterowania znaczną ilością parametrów. Graficzny interfejs aplikacji został przedstawiony na Rysunku 1.



Rysunek 1: Główny interfejs aplikacji z widocznymi sekcjami: panel sterowania (lewa kolumna), podgląd obrazu (środek) oraz pasek stanu (dół).

## 2 Opis aplikacji

### 2.1 Architektura systemu

Struktura projektu została zaprojektowana zgodnie z zasadami modularności i separacji obowiązków. Główna organizacja plików prezentuje się następująco:

```
Biometria-Projekt-1/
├── adv_img_operations/
│   └── color_thief.py
├── image_statistics/
│   └── statistics_window.py
│       └── Class StatisticsWindow
├── static/
│   ├── css/
│   └── img/
├── transformations/
│   ├── convolution/
│   │   ├── smoothing.py
│   │   ├── sharpening.py
│   │   └── custom_filter.py
│   ├── edge_detection/
│   │   ├── sobel.py
│   │   └── roberts.py
│   └── filters/
│       ├── grayscale.py
│       └── brightness.py
└── ...
├── app.py
└── └── Class MainWindow
└── requirements.txt
```

### 2.2 Opis modułów

Opis najważniejszych modułów aplikacji zawiera Tabela 1.

Komponent	Opis funkcjonalności
<i>Class MainWindow</i>	Główna klasa zarządzająca interfejsem użytkownika. Koordynuje przepływ danych między komponentami i obsługuje zdarzenia.
<i>Class StatisticsWindow</i>	Wyspecjalizowane okno do prezentacji statystyk obrazu. Oblicza i wizualizuje histogramy oraz podstawowe metryki jakości obrazu.
<i>transformations/</i>	Centralny moduł operacji na obrazie, implementujący: <ul style="list-style-type: none"> <li>• Operacje na pojedynczych pikselach</li> <li>• Operacje splotowe</li> <li>• Wzorzec Strategy dla różnych transformacji</li> </ul>
<i>adv_img_operations/</i>	Moduł zaawansowanych algorytmów: <ul style="list-style-type: none"> <li>• Implementuje wykrywanie dominanty kolorystycznej</li> <li>• Wykorzystuje popularną bibliotekę uczenia maszynowego - <i>scikit-learn</i></li> </ul>

Tabela 1: Charakterystyka głównych komponentów systemu

## 2.3 Wykorzystane technologie

- **Python 3.10** – główny język programowania zapewniający elastyczność i bogaty ekosystem bibliotek.
- **PyQt5 (5.15.9)** – framework do budowy interfejsu graficznego (GUI), odpowiedzialny za zarządzanie oknami i widgetami, obsługę zdarzeń użytkownika i integrację wyświetlacza obrazu z operacjami przetwarzania
- **CSS** – stylizacja komponentów GUI (kolorystyka, rozmiary, efekty *hover*).
- **NumPy (1.24.3)** – operacje na macierzach pikseli i optymalizacja obliczeń obrazowych.
- **Pillow (9.4.0)** – wczytywanie/zapisywanie obrazów w różnych formatach (PNG, JPG, BMP).
- **Matplotlib (3.10.1)** – generowanie histogramów i projekcji w module statystyk.
- **scikit-learn (1.6.1)** – eksperymentalne wykorzystanie do zaawansowanej analizy obrazu (ekstrakcja dominujących kolorów).

### 3 Implementacja metod

W tym rozdziale opisane zostały zaimplementowane algorytmy przetwarzania obrazów. Transformacje te zostały podzielone na następujące kategorie: operacje podstawowe, filtry przestrzenne, statystyki obrazu oraz dodatkowo algorytm wyznaczania koloru dominującego na obrazie. Podczas implementacji algorytmów zastosowano wysokie standardy obsługi wyjątków i błędów, wszelkie nieprawidłowości wyświetlane są użytkownikowi aplikacji na pasku *status bar*. We wszystkich algorytmach zachowano następujące właściwości:

- **Automatyczne skalowanie** wartości parametrów wejściowych do zakresu danego typu obrazu
- Obsługa zarówno obrazów całkowitoliczbowych (**np.uint8**), jak i zmiennoprzecinkowych (**np.float32**)
- Zachowanie oryginalnego typu danych wyjściowych
- Jednolita modyfikacja wszystkich kanałów kolorystycznych
- **Dokumentacja** funkcji przy pomocy informatywnych *Docstringów*

#### 3.1 Operacje podstawowe

##### 3.1.1 Skala szarości

###### Wersja Average (średnia)

Algorytm konwersji do skali szarości opiera się na średniej arytmetycznej wartości wszystkich kanałów kolorystycznych. Dla każdego piksela obliczana jest średnia wartość z kanałów czerwonego, zielonego i niebieskiego, co daje jednolitą reprezentację jasności.

*Parametry wejściowe:*

- **Brak** - stałe współczynniki dla kanałów: R (0.33), G (0.33), B (0.33)

###### Wersja Luminosity (luminancja)

Metoda uwzględnia różną percepcję jasności przez człowieka, stosując ważoną średnią kanałów kolorystycznych. Współczynniki zostały dobrane eksperymentalnie zgodnie ze standardem ITU-R BT.601.

*Parametry wejściowe:*

- **Brak** - stałe współczynniki dla kanałów: R (0.299), G (0.587), B (0.114)

###### Wersja Custom (niestandardowa)

Umożliwia ręczne określenie współczynników poszczególnych kanałów kolorystycznych. Pozwala to na dostosowanie wyniku do specyficznych potrzeb użytkownika poprzez zmianę proporcji między kanałami. Dla każdego piksela wartość wynikowa jest obliczana ze wzoru:

$$I_{gray} = \frac{w_r \cdot R + w_g \cdot G + w_b \cdot B}{w_r + w_g + w_b} \quad (1)$$

*Parametry wejściowe:*

- **Waga** dla kanału czerwonego (domyślnie 3)
- **Waga** dla kanału zielonego (domyślnie 6)
- **Waga** dla kanału niebieskiego (domyślnie 1)

*Przykładowe efekty wszystkich algorytmów przedstawiono na Rysunku 2*



Rysunek 2: Przykłady konwersji do skali szarości: wersja Average (lewo), wersja Luminosity (środek), wersja Custom z wagami R (0.6) G(0.2) B(0.2) (prawo)

### 3.1.2 Regulacja jasności (Brightness Adjustment)

Algorytm addytywnej modyfikacji jasności obrazu poprzez jednolite skalowanie wartości pikseli we wszystkich kanałach kolorystycznych. Wartości wykraczające poza dopuszczalny zakres są przycinane do prawidłowych wartości. Skalowanie parametru addytywnego  $\Delta$  pozwala uzyskać pełen zakres regulacji jasności [1].

$$I_{out}(x, y) = \text{clip}\left(I_{in}(x, y) + \Delta \cdot \frac{V_{max} - V_{min}}{100}\right) \quad (2)$$

gdzie:

- $\Delta$  - Wartość regulacji (-100 do 100)
- $V_{max}$ ,  $V_{min}$  - odpowiednio maksymalna i minimalna wartość dla typu danych obrazu, przykładowo 0 oraz 255 dla obrazu 8-bitowego.
- `clip()` - Funkcja przycinająca wartości do zakresu właściwego dla typu danych obrazu

*Przykładowe wartości dla obrazu 8-bitowego:*

$$\Delta = 50 \Rightarrow \text{przyrost} = 50 \cdot \frac{255}{100} = 127.5 \approx 128$$

*Parametry wejściowe:*

- **Wartość regulacji** ( $\Delta$ ) - w zakresie -100 do 100:

- $\Delta > 0$ : zwiększa jasność obrazu
- $\Delta < 0$ : zmniejsza jasność obrazu
- $\Delta = 0$ : brak zmiany

*Przykładowe efekty przedstawiono na Rysunku 3*



Rysunek 3: Przykłady regulacji jasności: oryginał (środek),  $\Delta = -25$  (lewo),  $\Delta = +25$  (prawo)

### 3.1.3 Regulacja kontrastu (Contrast Adjustment)

Algorytm liniowej modyfikacji kontrastu obrazu poprzez skalowanie odchylenia od średniej jasności. Wartości większe niż 1 zwiększają kontrast, z automatycznym przycinaniem do dopuszczalnego zakresu.

$$I_{out}(x, y) = \text{clip}(\mu + v \cdot (I_{in}(x, y) - \mu)) \quad (3)$$

gdzie:

- $\mu$  - średnia wartość kanału
- $v$  - wartość regulacji (0.0 do 5.0)

*Parametry wejściowe:*

- **Wartość regulacji** (0.0 do 5.0):

- $v = 0.0$ : jednolite szare tło (zerowy kontrast)
- $v = 1.0$ : oryginalny obraz (bez zmian)
- $v = 5.0$ : maksymalne zwiększenie kontrastu

*Przykładowe efekty przedstawiono na Rysunku 4*



Rysunek 4: Przykłady regulacji kontrastu:  $v = 0.6$  (lewo), oryginał (środek),  $v = 1.6$  (prawo)

### 3.1.4 Korekcja gamma (Gamma Correction)

Nieliniowa transformacja jasności obrazu stosowana do kompensacji nieliniowej charakterystyki wyświetlaczy. Algorytm wykorzystuje potęgowanie wartości pikseli według parametru gamma, z optymalizacją poprzez tablicę Look-Up (*LUT*) dla obrazów całkowitoliczbowych.

$$I_{out}(x, y) = \begin{cases} \lfloor V_{max} \cdot \left( \frac{I_{in}(x, y)}{V_{max}} \right)^{\gamma} \rfloor & \text{dla obrazów całkowitoliczbowych} \\ I_{in}(x, y)^{\gamma} & \text{dla obrazów float} \end{cases} \quad (4)$$

Parametry wejściowe:

- **Wartość gamma ( $\gamma > 0$ ):**

- $\gamma < 1.0$ : rozjaśnianie obrazu (kompresja ciemnych tonów)
- $\gamma = 1.0$ : brak zmiany
- $\gamma > 1.0$ : przyciemnianie obrazu (kompresja jasnych tonów)

Przykładowe efekty przedstawiono na Rysunku 5



Rysunek 5: Przykłady korekcji gamma: oryginał (środek),  $\gamma = 0.6$  (lewo),  $\gamma = 1.6$  (prawo)

### Optymalizacja Look-Up Table

Kluczowy schemat działania algorytmu optymalizacji przy pomocy tablicy *LUT* przedstawia poniższy fragment kodu:

```

lut = ((np.arange(max_val + 1, dtype=np.float32) / max_val) ** gamma)
lut = np.round(lut * max_val).clip(0, max_val).astype(image.dtype)
return lut[image]

```

### 3.1.5 Binaryzacja obrazu

Algorytm progowania obrazu szarości do postaci binarnej (czarno-białej) z wykorzystaniem zadanego progu. W przypadku obrazów kolorowych, automatycznie konwertuje je do skali szarości przed zastosowaniem binaryzacji.

$$I_{out}(x,y) = \begin{cases} V_{max} & \text{gdy } I_{in}(x,y) \geq T_{scaled} \\ V_{min} & \text{w przeciwnym przypadku} \end{cases} \quad (5)$$

gdzie:

- $T_{scaled} = \frac{T}{255} \cdot (V_{max} - V_{min})$  - przeskalowany próg
- $T$  - wartość progu wejściowego (0-255)

*Parametry wejściowe:*

- **Próg binaryzacji** (0-255) - wartość graniczna decydująca o przypisaniu do klasy

Przykładowe efekty przedstawiono na Rysunku 6



Rysunek 6: Przykłady binaryzacji dla różnych progów: zdjęcie w odcieniach szarości (lewo),  $T = 180$  (środek),  $T = 240$  (prawo)

### 3.1.6 Negatyw obrazu

Algorytm odwracania wartości intensywności pikseli, tworzący efekt analogiczny do negatywu fotograficznego. Działa poprzez odejmowanie wartości pikseli od maksymalnej wartości danego typu danych.

$$I_{out}(x,y) = \begin{cases} V_{max} - I_{in}(x,y) & \text{dla obrazów całkowitoliczbowych} \\ 1.0 - I_{in}(x,y) & \text{dla obrazów zmienoprzecinkowych} \end{cases} \quad (6)$$

*Metody optymalizacji:*

- Dla obrazów całkowitoliczbowych: wykorzystanie tablicy LUT (Look-Up Table)
- Dla obrazów float: bezpośrednie odejmowanie wektorowe

*Przykładowe efekty przedstawiono na Rysunku 7*



Rysunek 7: Przykład transformacji negatywu: obraz oryginalny (lewo) i wynik negatywu (prawo)

## 3.2 Filtry przestrzenne

### 3.2.1 Filtr uśredniający (Image Averaging)

Filtr liniowy stosujący jednolite ważenie sąsiedztwa. Wygładza obraz poprzez zastąpienie wartości piksela średnią z jego otoczenia. Efektywnie redukuje szum typu additive white noise.

$$K_{avg} = \frac{1}{\text{kernel\_size}^2} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad (7)$$

Przykład dla jądra wymiaru 3x3:

$$\frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

*Parametry wejściowe:*

- **Rozmiar jądra** (liczba nieparzysta, np. 3, 5)

*Przykładowe efekty przedstawiono na Rysunku 8*



Rysunek 8: Przykłady działania filtru uśredniającego: oryginał (lewo),  $\text{kernel\_size} = 5$  (środek),  $\text{kernel\_size} = 9$  (prawo)

### 3.2.2 Filtr Gaussa (Gaussian Blur)

Filtr niskoczęstotliwościowy wykorzystujący rozkład normalny do ważenia sąsiedztwa. Zachowuje krawędzie lepiej niż filtr uśredniający dzięki mniejszemu ważeniu odległych pikseli.

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (8)$$

Przykład jądra wymiaru  $3 \times 3$  ( $\sigma = 1$ ) w zaokrągleniu:

$$\frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

*Parametry wejściowe:*

- **Rozmiar jądra** (liczba nieparzysta, np. 3, 5)
- **Sigma ( $\sigma$ )** - parametr rozmycia. Duże wartości parametru powodują silne rozmycie.

Przykładowe efekty przedstawiono na Rysunku 9



Rysunek 9: Przykłady działania filtru Gaussa dla jądra wymiaru  $7 \times 7$ : oryginał (lewo),  $\sigma = 1.0$  (środek),  $\sigma = 5.0$  (prawo)

### 3.2.3 Filtr wyostrzający (Sharpening)

Filtr wysokoczęstotliwościowy wzmacniający krawędzie poprzez odjęcie wygładzonej wersji obrazu od oryginału (unsharp masking). Zwiększa lokalny kontrast w obszarach szybkich zmian intensywności.

$$S(i, j) = \begin{cases} 1 + \alpha, & \text{jeśli } (i, j) = (\text{centralny element}) \\ -\frac{\alpha}{\text{kernel\_size}^2}, & \text{wpp} \end{cases}$$

(9)

Przykład jądra wymiaru 3x3 ( $\alpha = 0.2$ ):

$$\begin{bmatrix} -0.2 & -0.2 & -0.2 \\ -0.2 & 2.6 & -0.2 \\ -0.2 & -0.2 & -0.2 \end{bmatrix}$$

Parametry wejściowe:

- **Rozmiar jądra** (liczba nieparzysta, np. 3, 5)
- **Alpha ( $\alpha$ )** - parametr siły wyostrzenia. Duże wartości parametru powodują silniejszy efekt wyostrzenia.

Przykładowe efekty przedstawiono na Rysunku 10



Rysunek 10: Przykłady działania filtru wyostrzającego dla jądra wymiaru 7x7: oryginał (lewo),  $\alpha = 1.0$  (środek),  $\alpha = 5.0$  (prawo)

### 3.2.4 Stosowanie własnego filtru splotowego (Custom Filter)

Algorytm aplikacji dowolnego jądra splotowego 3x3 na obrazie z zachowaniem oryginalnego typu danych i zakresu wartości. Implementacja wykorzystuje optymalizację poprzez operacje wektorowe i refleksyjne dopełnienie brzegów. Algorytm posiada wbudowaną walidację poprawności podanego jądra, lecz nie zapewnia jego normalizacji. Ma to na celu pokazanie użytkownikowi konsekwencji użycia jądra, którego wartości nie sumują się do 1.

$$I_{out}(x, y) = \text{clip} \left( \sum_{i=-1}^1 \sum_{j=-1}^1 K(i, j) \cdot I_{in}(x+i, y+j) \right) \quad (10)$$

gdzie:

- **K** - jądro wymiaru 3x3

*Przykładowe jądro:*

$$\text{Jądro Scharra} = \begin{bmatrix} 3 & 0 & -3 \\ 10 & 0 & -10 \\ 3 & 0 & -3 \end{bmatrix}$$

*Parametry wejściowe:*

- **Jądro filtru** - lista 9 wartości (rzędami) wybranych przez użytkownika

*Przykładowe efekty przedstawiono na Rysunku 11*



Rysunek 11: Przykład zastosowania własnego filtru splotowego (w tym wypadku jądra Scharra): oryginał (lewo), własny filtr (prawo)

### 3.2.5 Detekcja krawędzi - Operator Sobela

Operator Sobela to dyskretny operator różniczkowy wykorzystujący splot z dwoma ortogonalnymi jądrami 3x3 do aproksymacji gradientu obrazu. Algorytm wykrywa krawędzie poprzez obliczenie gradientu intensywności w kierunkach poziomym i pionowym [2]. W celu zachowania rozmiaru zdjęcia wykorzystano dopełnienie refleksywne (*padding*).

$$G = \sqrt{G_x^2 + G_y^2} \quad (11)$$

gdzie:

- $G_x$  - gradient poziomy (aproksymacja pochodnej cząstkowej po x)
- $G_y$  - gradient pionowy (aproksymacja pochodnej cząstkowej po y)

*Jądra operatora Sobela:*

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Parametry wejściowe:

- **Próg binaryzacji** (0-255) - wartość określająca minimalną siłę krawędzi

Przykładowe efekty przedstawiono na Rysunku 12



Rysunek 12: Przykłady detekcji krawędzi operatorem Sobela: wynik bez progu (lewo), wynik binarny z progiem  $T = 32$  (środek), wynik binarny z progiem  $T = 128$  (prawo)

### 3.2.6 Detekcja krawędzi - Krzyż Robertsa

Jeden z najstarszych operatorów krawędziowych, wykorzystujący jądra 2x2 do aproksymacji gradienu w kierunkach przekątnych. Charakteryzuje się wysoką czułością na drobne krawędzie, ale większą podatnością na szum w porównaniu do operatora Sobela [3].

$$G = \sqrt{G_{45^\circ}^2 + G_{135^\circ}^2} \quad (12)$$

gdzie:

- $G_{45^\circ}$  - gradient w kierunku 45 stopni
- $G_{135^\circ}$  - gradient w kierunku 135 stopni

Jądra operatora Robertsa:

$$G_{45^\circ} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad G_{135^\circ} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

Parametry wejściowe:

- **Próg binaryzacji** (0-255) - minimalna wartość gradientu uznawana za krawędź

Przykładowe efekty przedstawiono na Rysunku 13



Rysunek 13: Przykłady detekcji krawędzi krzyżem Robertsa: wynik bez progu (lewo), wynik binarny z progiem  $T = 8$  (środek), wynik binarny z progiem  $T = 32$  (prawo)

### 3.2.7 Detekcja krawędzi metodą Canny'ego

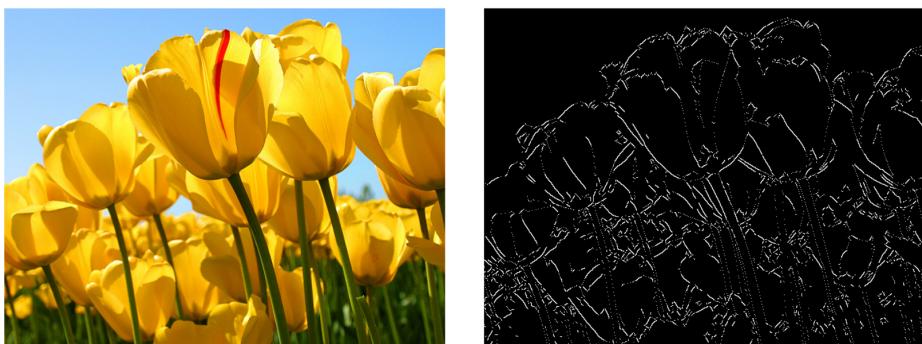
Algorytm wieloetapowej detekcji krawędzi, zapewniający wysoką jakość wyników poprzez supresję nie-maksimów i histerezę progowania. Uznawany za standard w aplikacjach wizji komputerowej.

$$\text{Kroki algorytmu: } \begin{cases} \text{1. Wygładzenie Gaussa} & G_\sigma * I \\ \text{2. Gradient Sobela} & \nabla I = \sqrt{G_x^2 + G_y^2} \\ \text{3. Supresja nie-maksimów} & \text{w kierunku gradientu} \\ \text{4. Progowanie z histerezą} & T_{low}, T_{high} \end{cases} \quad (13)$$

*Parametry wejściowe:*

- **Próg dolny** (0-255) - minimalna siła słabych krawędzi
- **Próg górny** (0-255) - minimalna siła silnych krawędzi

Przykładowe efekty przedstawiono na Rysunku 14



Rysunek 14: Przykład zastosowania algorytmu detekcji krawędzi Canny: oryginał (lewo), wynik binarny z progami  $T_{low} = 50$ ,  $T_{high} = 90$

## 3.3 Statystyki obrazu i projekcje

### Histogram intensywności

Algorytm obliczania histogramu różni się dla obrazów w skali szarości i kolorowych:

- **Dla skali szarości:**

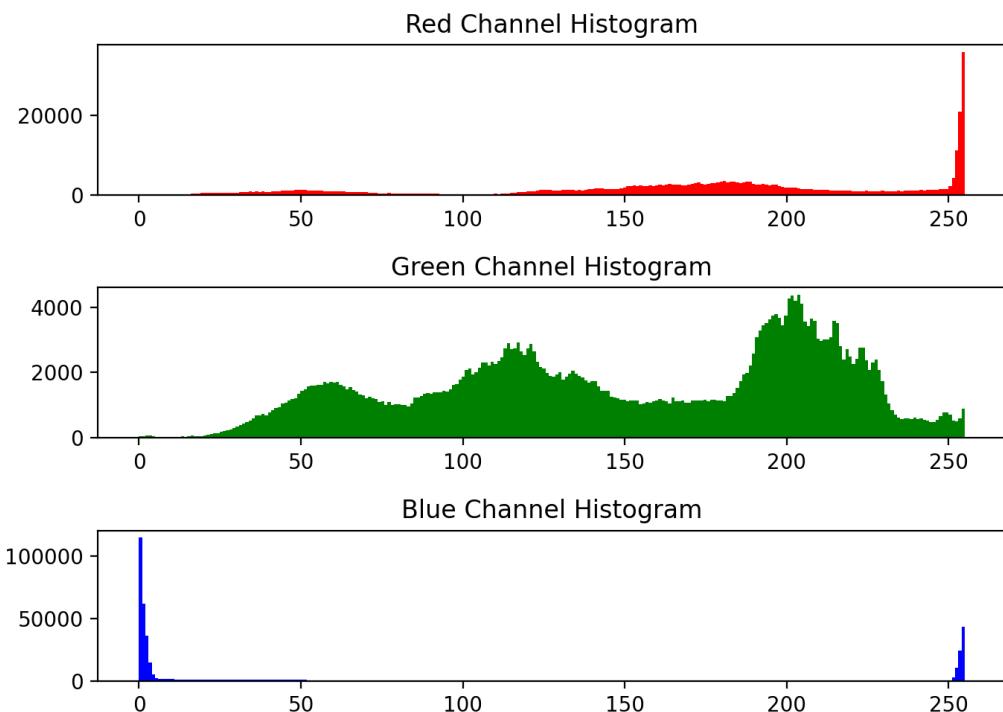
$$H_g[i] = \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \delta(I(x,y) - i), \quad i \in [0, 255] \quad (14)$$

gdzie  $\delta$  to funkcja delta Kroneckera

- **Dla obrazów RGB:**

$$H_c[i] = \sum_{x=0}^{w-1} \sum_{y=0}^{h-1} \delta(I_c(x,y) - i), \quad c \in \{R, G, B\}, i \in [0, 255] \quad (15)$$

*Przykładowe efekty przedstawiono na Rysunku 15*



Rysunek 15: Przykładowy histogram dla kanałów RGB obrazu tulipanów.

## Projekcje obrazu

Metody analizy rozkładu intensywności wzdłuż osi:

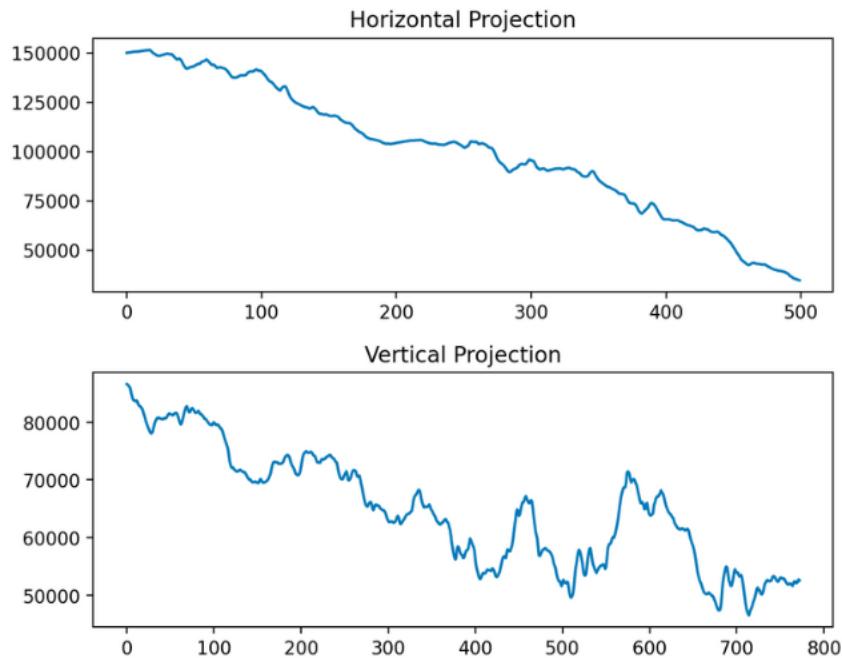
- **Projekcja pozioma** - suma wartości pikseli w wierszach:

$$P_h[y] = \sum_{x=0}^{w-1} I(x,y) \quad (16)$$

- **Projekcja pionowa** - suma wartości pikseli w kolumnach:

$$P_v[x] = \sum_{y=0}^{h-1} I(x,y) \quad (17)$$

*Przykładowe efekty przedstawiono na Rysunku 16*



Rysunek 16: Wizualizacja projekcji: (a) pozioma, (b) pionowa

*Zastosowania analityczne dla histogramów jasności oraz projekcji przedstawia Tabela 2*

Metryka	Zastosowanie
Histogram	Analiza kontrastu, wykrywanie prześwietleń
Projekcja pozioma	Detekcja linii tekstu w OCR
Projekcja pionowa	Analiza struktury kolumnowej

Tabela 2: Zastosowania statystyk obrazu

### 3.4 Algorytm wyznaczania koloru dominującego

Algorytm identyfikacji głównego koloru w obrazie wykorzystujący klasteryzację K-średnich w przestrzeni RGB. Inspirowany biblioteką Color Thief [4], służy do automatycznego dobierania koloru tła za obrazem w interfejsie graficznym.

*Parametry wejściowe:*

- **Liczba klastrów (k)** - domyślnie 5
- **Rozmiar po redukcji** - krotka (width, height), domyślnie (100, 100)

W celu zoptymalizowania szybkości działania algorytmu konieczna jest redukcja rozmiaru obrazu. W tym celu zaimplementowano algorytm redukcji rozmiaru metodą *nearest-neighbor* zgodnie ze wzorem:

$$I_{out}(i, j) = I_{in} \left( \left\lfloor i \cdot \frac{h_{in}}{h_{out}} \right\rfloor, \left\lfloor j \cdot \frac{w_{in}}{w_{out}} \right\rfloor \right) \quad (18)$$

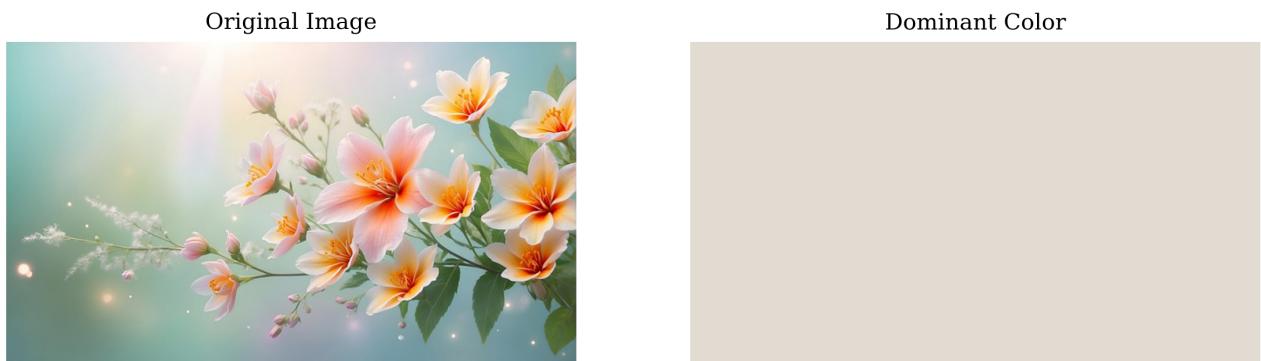
*Kroki algorytmu:*

1. Redukcja rozmiaru obrazu metodą nearest-neighbor
2. Konwersja do tablicy pikseli 3D → 2D (N×3)
3. Klasteryzacja K-średnich w przestrzeni RGB
4. Wybór klastra z największą liczbą pikseli
5. Konwersja centroidu do formatu HEX

*Właściwości:*

- Adaptacyjna obsługa różnych formatów obrazu (grayscale, RGBA)
- Automatyczne przycinanie wartości kolorów do zakresu [0,255]
- Optymalizacja poprzez redukcję rozmiaru przed klasteryzacją
- Deterministyczne wyniki dzięki stałej inicjalizacji KMeans

*Przykładowe efekty przedstawiono na Rysunku 17*



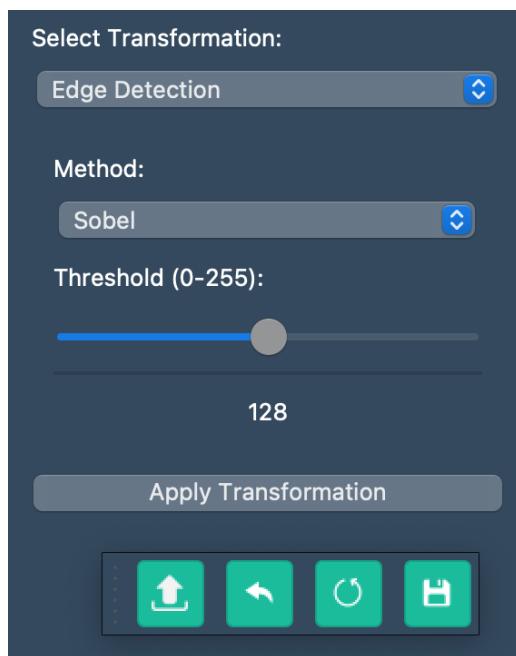
Rysunek 17: Przykład działania: oryginalny obraz (lewo), wykryty kolor dominujący (prawo)

## 4 Interfejs użytkownika

Ekran główny aplikacji, ukazujący się po jej uruchomieniu, został przedstawiony na Rysunku 1. Główną część ekranu zajmuje panel wyświetlający przetwarzany obraz. Po lewej stronie znajduje się **panel wyboru transformacji**, ukazany na Rysunku 18. Z jego poziomu użytkownik wybiera jedną z transformacji opisanych w sekcji 3 wraz z niezbędnymi parametrami. Transformacje nanoszone są na obraz po naciśnięciu przycisku "*Apply Transformation*". Do dyspozycji użytkownika jest również **pasek narzędzi**, który można umieścić w dowolnym wygodnym miejscu na ekranie (Rysunek 18). Z jego poziomu użytkownik wybiera następujące operacje:

- **Wczytaj** - operacja wczytania do aplikacji zdjęcia w jednym z obsługiwanych formatów (\*.png \*.jpg \*.jpeg \*.bmp)
- **Cofnij** - operacja cofnięcia ostatnio wykonanej transformacji
- **Reset** - operacja powrotu do zdjęcia oryginalnego, które zostało pierwotnie wczytane do aplikacji
- **Zapisz** - operacja zapisu przetworzonego zdjęcia pod wskazaną ścieżką

Na ekranie głównym znajduje się również przycisk "*Show Statistics*", który przenosi użytkownika do nowego okna zawierającego podstawowe statystyki obrazu, histogram jasności oraz projekcje pionową i poziomą. Wyświetlane statystyki dotyczą aktualnie przetwarzanego obrazu. Przykładowe wizualizacje wyświetlane w tej części aplikacji zostały ukazane na Rysunku 15 oraz 16.



Rysunek 18: Panel wyboru transformacji wraz z paskiem narzędzi.

## 5 Wnioski

W trakcie realizacji projektu udało się osiągnąć wszystkie zakładane cele oraz wyciągnąć wnioski na podstawie własnoręcznych eksperymentów.

### 5.1 Wnioski z przetwarzania obrazów

- Operacja Gamma okazała się być bardziej naturalnym sposobem na modyfikację jasności i kontrastu obrazu niż proste dodawanie lub mnożenie wartości pikseli. Pozwala to na uzyskanie bardziej estetycznych i realistycznych efektów.
- Filtr Gaussa zachowuje lepiej krawędzie w porównaniu do innych filtrów, co jest szczególnie widoczne przy redukcji szumów w obrazach.
- Detekcja krawędzi za pomocą operatora Sobela okazała się być bardziej niezawodna niż metoda Roberts Cross
- Wykonanie rozmycia za pomocą filtra Gaussa przed zastosowaniem algorytmów detekcji krawędzi prowadzi do zmniejszenia liczby niepożądanych artefaktów na zdjęciach. Uzyskane w ten sposób obrazy są mniej wrażliwe na szумy w tle.

### 5.2 Trudności

- Jedną z głównych trudności było zarządzanie interfejsem użytkownika z wykorzystaniem PyQt [5]. Specyfika tej biblioteki wymaga stworzenia centralnego *widgetu*, do którego nie dodaje się bezpośrednio kolejnych *widgetów*. Zamiast tego, należy utworzyć obiekt *layout*, a do niego dodać pozostałe elementy interfejsu. Proces ten wymagał dodatkowego czasu na zrozumienie i zaimplementowanie poprawnego układu.
- Kolejnym wyzwaniem było optymalizowanie wydajności algorytmów przetwarzania obrazów, szczególnie przy dużych rozmiarach obrazów. Zastosowanie technik takich jak tablice Look-Up oraz obliczenia zwięktryzowane okazały się kluczem do osiągnięcia satysfakcjonującej szybkości działania.

Podsumowując, projekt pozwolił na zdobycie cennych doświadczeń w zakresie przetwarzania obrazów oraz zarządzania interfejsem użytkownika. Uzyskane wnioski mogą być wykorzystane w przyszłych projektach, aby uniknąć podobnych trudności i dążyć do jeszcze lepszych rozwiązań.

## Bibliografia

- [1] NITK Surathkal. *Algorithms for Adjusting Brightness and Contrast of an Image*. <https://ie.nitk.ac.in/blog/2020/01/19/algorithms-for-adjusting-brightness-and-contrast-of-an-image/>. Accessed: 2025-03-29. Sty. 2020.
- [2] *Edge Detection in Image Processing: An Introduction*. <https://blog.roboflow.com/edge-detection/>. Accessed: 2025-03-29.
- [3] *Comparative analysis of common edge detection*. <https://arxiv.org/pdf/1405.6132.pdf>. Accessed: 2025-03-29.
- [4] Lokesh Dhakar. *Color Thief*. <https://lokeshdhakar.com/projects/color-thief/>. Accessed: 2025-03-29.
- [5] The Qt Company. *Qt for Python Documentation*. <https://doc.qt.io/qtforpython-6/>. Accessed: 2025-03-29.