

Pain't

Aplikacja do przetwarzania zdjęć
Dokumentacja

Michał Matuszyk

Marzec 2025

Streszczenie

Aplikacja do przetwarzania zdjęć, pozwalająca na zastosowanie różnych filtrów, operacji pikselowych, analizy histogramów oraz projekcji obrazu. Wspiera różne formaty plików graficznych i umożliwia szybkie zapisywanie wyników. Projekt zrealizowany w Pythonie z wykorzystaniem biblioteki Tkinter.

Spis treści

1 Wprowadzenie	3
2 Opis aplikacji	3
3 Opis Metod	5
3.1 Metody Pikselowe	7
3.1.1 grayscale	7
3.1.2 adjust brightness	7
3.1.3 adjust contrast	7
3.1.4 negative	8
3.1.5 binarize	8
3.2 Filtry konwolucyjne (filters)	8
3.2.1 averaging filter	10
3.2.2 gaussian filter	10
3.2.3 sharpening filter	10
3.2.4 edge detection filter	10
3.2.5 emboss filter	10
3.2.6 horizontal sobel filter, vertical sobel filter	11
3.2.7 motion blur filter	11
3.2.8 custom filter	11
4 Prezentacja Wyników Działania	12
4.1 Wynik Grayscale	13
4.2 Wynik Adjust Brightness	13
4.3 Wynik Adjust Contrast	14
4.4 Wynik Negative	14
4.5 Wynik Binarize	15
4.6 Wynik Averaging Filter	15
4.7 Wynik Gaussian Filter	16
4.8 Wynik Sharpening Filter	18
4.9 Wynik Edge Detection Filter	19
4.10 Wynik Emboss Filter	20
4.11 Wynik Horizontal Sobel Filter	20
4.12 Wynik Vertical Sobel Filter	21
4.13 Wynik Motion Blur Filter	22
5 Wnioski	22

1 Wprowadzenie

Celem niniejszej dokumentacji jest przedstawienie założeń oraz funkcjonalności aplikacji *Pain't* służącej do przetwarzania obrazów. Aplikacja została zaprojektowana z myślą o użytkownikach chcących w prosty i intuicyjny sposób eksperymentować z różnymi technikami filtracji obrazów.

Program umożliwia nakładanie operacji pikselowych (takich jak zmiana jasności, kontrastu czy binarizacja) oraz bardziej złożonych filtrów bazujących na macierzach konwolucyjnych (tzw. kernelach), takich jak filtry wygładzające, wyostrzające czy wykrywające krawędzie.

Tego typu przekształcenia stanowią istotny element wstępnego przetwarzania danych obrazowych, szczególnie w kontekście algorytmów uczenia maszynowego i głębokiego uczenia. Aplikacja pozwala na ich łatwe zastosowanie i – co równie ważne – umożliwia wizualną analizę efektów działania filtrów, co znaczco ułatwia zrozumienie procesów zachodzących podczas przetwarzania obrazu.

2 Opis aplikacji

Aplikacja została zaimplementowana w języku **Python**, z wykorzystaniem biblioteki **Tkinter** jako głównego narzędzia do tworzenia graficznego interfejsu użytkownika (GUI). Projekt został zaprojektowany z naciskiem na funkcjonalność i przejrzystość, stawiając użyteczność ponad aspekty wizualne.

Interfejs użytkownika umożliwia intuicyjne przeciąganie i upuszczanie filtrów, dynamiczne podglądy działania operacji oraz łatwą obsługę zarówno dla użytkowników technicznych, jak i nietechnicznych.

Interfejs składa się z siedmiu głównych części, widocznych na poniższym rysunku:



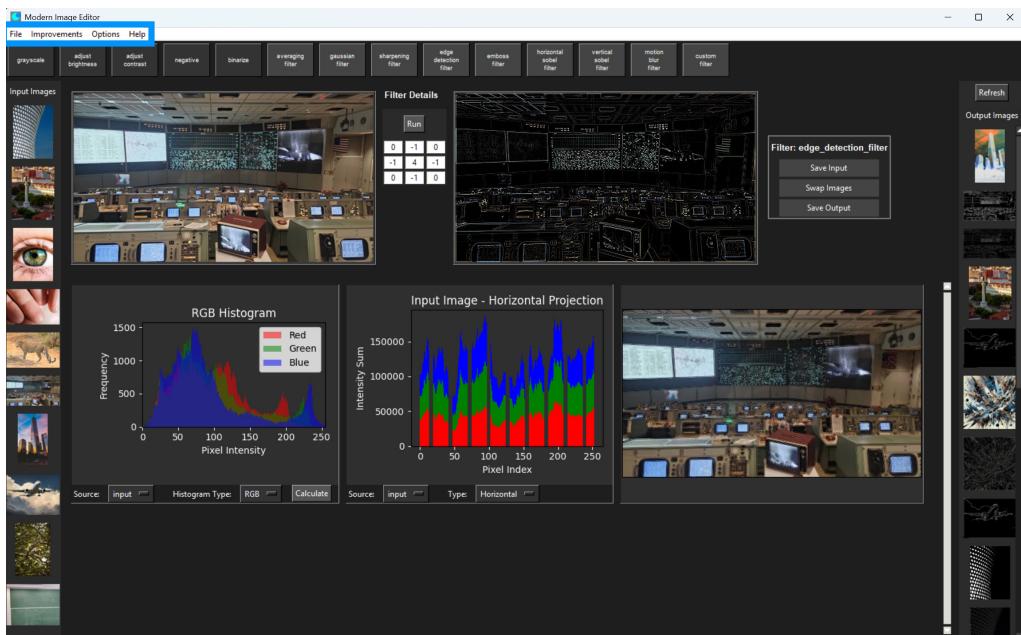
Rysunek 1: Główne elementy interfejsu edytora

Są to:

1. **Sample images / Output images** – zestawy miniaturow obrazów. Zdjęcia wejściowe są wczytywane z katalogu `sample_images`, a zapisane trafiają do `saved_images`.
2. **Menu operacji** – umożliwia wybór operacji pikselowej lub filtru konwolucyjnego, które mogą zostać przeciągnięte na obraz.
3. **Zdjęcie wejściowe** – obszar, na którym prezentowany jest oryginalny obraz, poddawany dalszej obróbce.

4. **Zdjęcie wynikowe** – prezentuje efekt zastosowania wybranych operacji lub filtrów.
5. **Panel analizy** – zawiera histogram, projekcję (poziomą lub pionową) oraz różnicę między obrazem wejściowym a wynikowym.
6. **Menu akcji** – umożliwia zapis obrazów wejściowych i wyjściowych oraz ich zamianę miejscami.
7. **Szczegóły operacji** – zawiera suwaki, pola numeryczne i macierze filtrów, które pozwalają użytkownikowi dostosować parametry wybranej operacji.

Dodatkowo użytkownik ma możliwość wczytania własnych zdjęć poprzez opcję File → Open Image, a także może włączyć tryb przyspieszonego nakładania filtrów w zakładce Improvements (opcja Fast Apply).



Rysunek 2: Menu aplikacji

Biblioteki użyte w projekcie

GUI i interfejs użytkownika

- `tkinter` – podstawowa biblioteka GUI w Pythonie.
- `ttk` – zestaw stylizowanych komponentów (przyciski, paski przewijania).
- `messagebox` – do wyświetlania komunikatów dla użytkownika.
- `filedialog` – do obsługi otwierania i zapisywania plików graficznych (np. otwiera Explorer w Windows)

Obsługa obrazów

- `PIL (Pillow)` – biblioteka do obsługi grafiki rastrowej, wspierająca wiele formatów plików:
 - `Image`, `ImageTk` – obiekty obrazu zgodne z Tkinterem.

Wizualizacja danych

- `matplotlib.pyplot` – generowanie wykresów, takich jak histogramy czy projekcje.
- `FigureCanvasTkAgg` – pozwala na umieszczenie wykresu matplotlib bezpośrednio w oknie Tkinter.

Praca z plikami i danymi

- `os` – operacje na systemie plików, m.in. zapisywanie zdjęć.
- `json` – przechowywanie preferencji użytkownika.
- `datetime` – generowanie znaczników czasu dla zapisanych plików.
- `threading` – wykonywanie czasochłonnych operacji w tle (np. obliczenia różnic, histogramów).

Przetwarzanie obrazów

- `numpy` – operacje numeryczne na obrazach w formie macierzy pikseli, m.in. wykorzystywane w szybkim nakładaniu filtrów (`Fast Apply`).

3 Opis Metod

W implementowanych metodach korzystam z własnej klasy - `CustomImageClass`.

Zaimplementowane metody przetwarzania obrazu można podzielić na dwie główne kategorie:

- **Metody pikselowe (pixelwise)** – operacje wykonywane bezpośrednio na wartościach pojedynczych pikseli, bez uwzględniania ich sąsiedztwa.
- **Filtry konwolucyjne (filters)** – operacje bazujące na analizie sąsiadujących pikseli przy pomocy masek (jąder konwolucyjnych).

Metody pikselowe (pixelwise)

- `grayscale` – konwersja obrazu do skali szarości.
- `adjust_brightness` – regulacja jasności obrazu.
- `adjust_contrast` – regulacja kontrastu.
- `negative` – negatyw obrazu.
- `binarize` – binaryzacja obrazu przy zadanym progu.

Filtry konwolucyjne (filters)

- `averaging_filter` – uśrednianie lokalnych wartości pikseli (filtr wygładzający).
- `gaussian_filter` – wygładzanie z wykorzystaniem rozkładu Gaussa.
- `sharpening_filter` – wyostrzanie krawędzi w obrazie.
- `edge_detection_filter` – detekcja krawędzi.

- `emboss_filter` – filtr tworzący efekt wytłoczenia.
- `horizontal_sobel_filter` – detekcja krawędzi w kierunku poziomym (Sobel).
- `vertical_sobel_filter` – detekcja krawędzi w kierunku pionowym (Sobel).
- `motion.blur_filter` – symulacja rozmycia ruchu.
- `custom_filter` – filtr definiowany przez użytkownika (wprowadzany ręcznie).

Obsługa obrazu wejściowego – klasa `CustomImageClass`

Każda z zaimplementowanych metod przetwarzania obrazu jako argument wejściowy przyjmuje obiekt klasy `CustomImageClass`, który pełni rolę kontenera na dane obrazu oraz zapewnia szereg funkcji pomocniczych, takich jak:

- wczytywanie obrazu z pliku lub tworzenie pustego obrazu o zadanych wymiarach,
- dostęp do danych pikseli w postaci listy dwuwymiarowej `pixel_data`,
- funkcje do pobierania i ustawiania wartości pikseli,
- aktualizację obrazu na podstawie zmodyfikowanych danych pikseli,
- zapis obrazu na dysk, generowanie miniatury, histogramu lub konwersję do obiektu `numpy.ndarray`.

Aby uniknąć bezpośredniej modyfikacji oryginalnego obrazu (co mogłoby prowadzić do błędów i niepożądanych efektów), każda metoda przetwarzania obrazu zaczyna się od utworzenia kopii wejściowego obiektu za pomocą metody `copy()`:

```
output_image = image.copy()
```

Listing 1: Tworzenie kopii obrazu wejściowego

Metoda `copy()` tworzy nowy obiekt `CustomImageClass` o takich samych wymiarach i głęboko skopiowanych danych pikseli:

```
def copy(self):
    """Create a fast copy of this object."""
    return CustomImageClass(
        height=self.height,
        width=self.width,
        pixel_data=[row[:] for row in self.pixel_data]
    )
```

Listing 2: Implementacja metody `copy`

Dzięki temu możliwe jest niezależne modyfikowanie obrazu wynikowego bez wpływu na oryginał. Takie podejście pozwala zachować pełną kontrolę nad przepływem danych w aplikacji, umożliwia porównywanie obrazów przed i po operacji, a także późniejsze odwrócenie operacji (np. zamiana obrazów miejscami).

Wszystkie metody opisane w tej dokumentacji oczekują na wejściu obiektu klasy `CustomImageClass`, co zapewnia spójność i jednolity interfejs do pracy z obrazami w całej aplikacji.

3.1 Metody Pikselowe

3.1.1 grayscale

Metoda `grayscale` przekształca obraz kolorowy do skali szarości poprzez uśrednienie wartości kanałów R, G i B dla każdego piksela. W wyniku działania funkcji wszystkie trzy kanały otrzymują tę samą wartość, co skutkuje odcieniem szarości.

Poniżej przedstawiono implementację funkcji w języku Python:

```
def black_to_white(image: CustomImageClass):
    """Convert an image to grayscale by averaging RGB values."""
    output_image = image.copy()
    for y in range(image.height):
        for x in range(image.width):
            r, g, b = output_image.pixel_data[y][x]
            avg = (r + g + b) // 3 # Compute grayscale value
            output_image.pixel_data[y][x] = (avg, avg, avg) # Set grayscale pixel
    return output_image
```

Listing 3: Funkcja `black_to_white` – konwersja do skali szarości

3.1.2 adjust brightness

Metoda `adjust_brightness` umożliwia zwiększenie lub zmniejszenie jasności obrazu poprzez przemnożenie wartości każdego kanału R, G i B przez zadany współczynnik (`factor`). Wartości są przycinane do przedziału [0, 255].

```
def adjust_brightness(image: CustomImageClass, factor: float):
    """Adjust image brightness by multiplying RGB values by a factor"""
    output_image = image.copy()
    for y in range(image.height):
        for x in range(image.width):
            r, g, b = output_image.pixel_data[y][x]
            new_pixel = (
                max(0, min(255, int(r * factor))),
                max(0, min(255, int(g * factor))),
                max(0, min(255, int(b * factor))))
        output_image.pixel_data[y][x] = new_pixel
    return output_image
```

Listing 4: Regulacja jasności obrazu

3.1.3 adjust contrast

Funkcja `adjust_contrast` zwiększa lub zmniejsza kontrast obrazu względem jego średniego poziomu jasności. Nowe wartości pikseli są obliczane względem tej średniej i skalowane przez parametr `factor`.

```
def adjust_contrast(image: CustomImageClass, factor: float):
    """Adjust image contrast by scaling pixel values."""
    output_image = image.copy()
    avg_brightness = sum(sum(pixel) // 3 for row in output_image.pixel_data for pixel
        in row) // (
            image.width * image.height)

    for y in range(image.height):
        for x in range(image.width):
```

```

r, g, b = output_image.pixel_data[y][x]
new_pixel = (
    max(0, min(255, int(avg_brightness + (r - avg_brightness) * factor)))
    ,
    max(0, min(255, int(avg_brightness + (g - avg_brightness) * factor)))
    ,
    max(0, min(255, int(avg_brightness + (b - avg_brightness) * factor)))
)
output_image.pixel_data[y][x] = new_pixel
return output_image

```

Listing 5: Regulacja kontrastu obrazu

3.1.4 negative

Metoda `negative` tworzy negatyw obrazu poprzez odwrócenie wartości każdego kanału RGB (tzn. 255 – wartość).

```

def negative(image: CustomImageClass):
    """Convert an image to its negative."""
    output_image = image.copy()
    for y in range(image.height):
        for x in range(image.width):
            r, g, b = output_image.pixel_data[y][x]
            output_image.pixel_data[y][x] = (255 - r, 255 - g, 255 - b)
    return output_image

```

Listing 6: Tworzenie negatywu obrazu

3.1.5 binarize

Funkcja `binarize` przekształca obraz do postaci czarno-białej w oparciu o próg jasności (`threshold`). Piksele jaśniejsze lub równe progowi stają się białe, pozostałe czarne.

```

def binarize(image: CustomImageClass, threshold: int = 128):
    """Convert an image to black and white (binarization) based on a threshold."""
    output_image = image.copy()
    for y in range(image.height):
        for x in range(image.width):
            r, g, b = output_image.pixel_data[y][x]
            avg = (r + g + b) // 3 # Compute grayscale value
            output_image.pixel_data[y][x] = (255, 255, 255) if avg >= threshold else (0, 0, 0)
    return output_image

```

Listing 7: Binarizacja obrazu na podstawie progu

3.2 Filtry konwolucyjne (filters)

Filtry konwolucyjne to kluczowy mechanizm wykorzystywany w przetwarzaniu obrazów, umożliwiający m.in. wygładzanie, wyostrzanie, wykrywanie krawędzi czy symulację efektów specjalnych. W ramach aplikacji *Pain't*, operacje te zostały zaimplementowane w klasie `ImageFilters`, która współpracuje z obiektami typu `CustomImageClass`.

Zasada działania

Dla każdego piksela (x, y) w obrazie wynikowym, nowa wartość RGB jest obliczana jako suma ważona sąsiadujących pikseli w obrazie wejściowym przy użyciu jądra (macierzy) konwolucyjnego:

$$R'(x, y) = \sum_{i=-k}^k \sum_{j=-k}^k K(i, j) \cdot R(x + i, y + j)$$

Analogicznie obliczane są kanały G i B , gdzie:

- $K(i, j)$ – waga jądra konwolucyjnego,
- k – połowa długości boku jądra (np. dla jądra 3×3 : $k = 1$),
- $R(x + i, y + j)$ – wartość kanału czerwonego w sąsiedztwie.

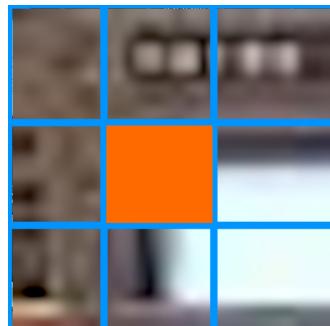
Na poniższym rysunku proces konwolucji jest zilustrowany. Dla każdego piksela widocznego na obrazie wejściowym jest wyliczana wartość po uwzględnieniu sąsiednich pikseli (na ilustracji jest pokazane kilka pikseli, w rzeczywistości jest to pojedynczy piksel).



(a) Zdjęcie wejściowe



(b) Wybór piksela



(c) Wyliczanie wartości pomarańczowego piksela

Rysunek 3: Porównanie obrazu oryginalnego i po filtrze krawędziowym

Dostępne filtry

W aplikacji zaimplementowano następujące filtry:

- `averaging_filter` – klasyczne wygładzanie przez uśrednianie.
- `gaussian_filter` – filtr Gaussa z opcjonalnym parametrem σ i rozmiarem jądra.
- `sharpening_filter` – wyostrzanie konturów poprzez wzmacnienie różnic między pikselami.
- `edge_detection_filter` – detekcja krawędzi (Laplacian).
- `emboss_filter` – imitacja wypukłości (efekt 3D).
- `horizontal_sobel_filter`, `vertical_sobel_filter` – klasyczne operatory Sobela wykrywające krawędzie w danym kierunku.

- `motion_blur_filter` – rozmycie symulujące ruch.
- `custom_filter` – filtr użytkownika z własnym jądrem.

Przykład jądra konwolucyjnego – filtr wyostrzający:

$$K_{\text{sharpen}} = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Tryb przyspieszony (`fast_apply`)

Dla wydajności zastosowano również opcję `fast_apply`, która wykorzystuje bibliotekę NumPy do przeprowadzenia operacji konwolucji w sposób wektorowy – poprzez przesunięcie i zsumowanie odpowiednio przemnożonych wersji obrazu tzn. wyliczane jest $k \times k$ macierzy z oryginalnego obrazka - każda wymnożona o wartość elementu w macierzy, następnie są sumowane z odpowiednim przesunięciem i powstaje obraz o rozmiarze $(n + 2k + 1) \times (m + 2k + 1)$, następnie wyciągany jest środek. Metoda ta znacznie przyspiesza obliczenia, szczególnie dla dużych obrazów i filtrów - w badaniach przyspieszenie wynosiło 12x.

3.2.1 averaging filter

Filtr uśredniający (*averaging filter*) działa poprzez zastąpienie każdego piksela wartością średnią jego sąsiednich pikseli (stosuje się maskę o rozmiarze 3×3). Efektem działania tego filtra jest wygładzanie obrazu i redukcja szumów, ale jednocześnie może powodować utratę szczegółów.

3.2.2 gaussian filter

Filtr Gaussa (*Gaussian filter*) wykorzystuje funkcję Gaussa do określenia wag pikseli w otoczeniu, co daje większą wagę pikselom bliżej centrum maski. Filtr charakteryzuje się dwoma parametrami: rozmiarem jądra (określającym zakres działania filtra) oraz parametrem σ (określającym stopień rozmycia). Efektem jest łagodniejsze i bardziej naturalne wygładzenie obrazu niż w przypadku klasycznego filtra uśredniającego.

3.2.3 sharpening filter

Filtr wyostrzający (*sharpening filter*) działa przez uwydatnienie różnic między pikselami oraz zwiększenie kontrastu krawędzi. Jest realizowany przez odejmowanie uśrednionych (rozmytych) wartości od oryginalnego obrazu, co powoduje wzmacnianie konturów. Efekt wizualny to wyraźniejsze i ostrzejsze krawędzie obiektów.

3.2.4 edge detection filter

Filtr detekcji krawędzi (oparty na operatorze Laplace'a, *Laplacian filter*) wykrywa szybkie zmiany intensywności pikseli w obrazie, pozwalając uzyskać wizualizację samych krawędzi obiektów.

3.2.5 emboss filter

Filtr imitacji wypukłości (*emboss filter*) nadaje obrazowi efekt pseudo-trójwymiarowy. Działa poprzez uwydatnienie krawędzi obiektów, symulując padające światło i cień. Efektem jest obraz wyglądający jak płaskorzeźba, sprawiający wrażenie głębi przestrzennej.

3.2.6 horizontal sobel filter, vertical sobel filter

Filtry Sobela (*Sobel filters*) wykrywają krawędzie w obrazie w konkretnych kierunkach (poziomym lub pionowym). Polegają na zastosowaniu gradientów intensywności pikseli w danym kierunku, co pozwala podkreślić krawędzie prostopadłe do tego kierunku. Wynikiem są obrazy przedstawiające wyraźne linie poziome (horizontal Sobel) lub pionowe (vertical Sobel).

3.2.7 motion blur filter

Filtr rozmycia ruchowego (*motion blur filter*) symuluje efekt ruchu kamery lub poruszających się obiektów. Działa poprzez rozmywanie pikseli wzduż konkretnego kierunku, co daje efekt "ciągnących się" śladów. Efekt ten może być kontrolowany przez długość oraz kierunek maski.

3.2.8 custom filter

Filtr użytkownika (*custom filter*) umożliwia zdefiniowanie własnego jądra (maski) konwolucyjnego przez użytkownika. Pozwala na dowolne kształtowanie efektu wizualnego przez ręcznie dobranie wartości współczynników maski, co umożliwia tworzenie specjalistycznych, eksperymentalnych lub niestandardowych efektów.

Ograniczenia i uwagi

- Brak przetwarzania pikseli brzegowych: operacje są wykonywane tylko tam, gdzie jądro w pełni mieści się w obrębie obrazu. Obszary przy brzegach pozostają niezmienione.
- Wartości pikseli są ograniczane do przedziału $[0, 255]$ – wszystkie wartości są przycinane po konwolucji.
- Dla niestandardowych jąder należy zachować ostrożność – błędne wartości mogą prowadzić do przesterowania lub artefaktów

Efekt działania

Każdy filtr zwraca nowy obiekt obrazu, nie modyfikując oryginału. Operacja ta pozwala użytkownikowi swobodnie porównywać obraz przed i po przekształceniu.

4 Prezentacja Wyników Działania

Poniższe wyniki, są efektem operacji na poniższym zdjęciu wejściowym (NASA Mission Operations Control Room from 1969 Apollo 11 Mission zdj. własne. Ciekawostka: jest to pomieszczenie kontrolne które odegrało kluczową rolę w postawieniu pierwszych kroków przez człowieka na księżycu):



Rysunek 4: Zdjęcie wejściowe

4.1 Wynik Grayscale

Obraz przekształcony do skali szarości poprzez uśrednienie wartości kanałów R, G i B. Kolor został całkowicie usunięty, zachowując jedynie informacje o jasności, co pozwala skupić się na strukturze obrazu bez wpływu koloru.



Rysunek 5: Wynik Grayscale

4.2 Wynik Adjust Brightness

Jasność obrazu została zwiększa przez przemnożenie wartości pikseli przez dodatni współczynnik. Efektem jest rozjaśnienie sceny – ciemniejsze obszary stały się lepiej widoczne, a cały obraz wygląda na jaśniejszy, warto zwrócić uwagę na ekranie komputerów - ich zawartość staje się niewidoczna.



Rysunek 6: Wynik Adjust Brightness

4.3 Wynik Adjust Contrast

Kontrast obrazu został zwiększony względem średniego poziomu jasności. Piksele jaśniejsze od średniej zostały rozjaśnione, a ciemniejsze przyciemnione, co podkreśliło różnice tonalne i uwypukniło szczegóły. Tutaj również warto zwrócić uwagę na ekranы.



Rysunek 7: Wynik Adjust Contrast

4.4 Wynik Negative

Negatyw obrazu – każda wartość kolorystyczna została odwrócona względem 255. Obszary jasne stały się ciemne, a ciemne – jasne. Efekt często używany do analizy struktury obrazu w nietypowy sposób.



Rysunek 8: Wynik Negative

4.5 Wynik Binarize

Obraz został zbinaryzowany przy użyciu ustalonego progu jasności. Piksele powyżej progu stały się białe, a poniżej – czarne. Obraz zyskał formę silnego kontrastu czerni i bieli, przydatną np. w segmentacji lub OCR. Doskonale widać miejsca pracy pracowników - poprzez ekranы komputerów.



Rysunek 9: Wynik Binarize

4.6 Wynik Averaging Filter

Filtr uśredniający (ang. *Averaging Filter*) to jedno z podstawowych narzędzi do wygładzania obrazu. Jego działanie polega na zastąpieniu wartości piksela średnią arytmetyczną wartości jego najbliższych sąsiadów w określonym oknie (jadrze), co skutkuje zredukowaniem szumów i wygładzeniem przejść tonalnych.

Efektem zastosowania filtra uśredniającego jest wyraźne zmiękczenie obrazu – ostre krawędzie zostają rozmyte, a szczegóły zatracone. Jest to przydatne w przypadku obrazów zaszumionych, jednak należy pamiętać, że nadmierne uśrednianie prowadzi do utraty istotnych informacji wizualnych.



Rysunek 10: Wynik Averaging Filter

4.7 Wynik Gaussian Filter

Na załączonych obrazkach widać doskonale, jak zwiększa się efekt "rozmycia" wraz z wzrastającą wielkością jądra.

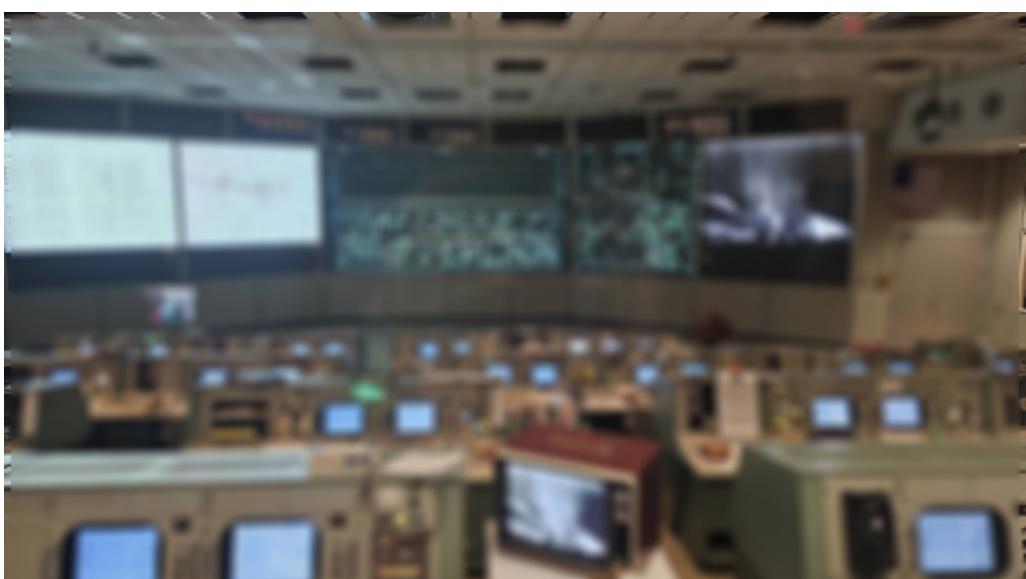
Tutaj jest możliwe edycja parametrów, zatem Wynik są 3, na pierwszym rysunku jest zastosowany filtr o jadrze 3x3, na drugim 5x5 a na trzecim 7x7:



Rysunek 11: Gaussian 3x3



Rysunek 12: Gaussian 5x5



Rysunek 13: Gaussian 7x7

4.8 Wynik Sharpening Filter

Filtr wyostrzajacy uwyydatnił krawędzie oraz detale na obrazie, wzmacniając różnice pomiędzy sąsiadującymi pikselami. Efekt sprawia, że obraz wygląda na bardziej szczegółowy i ostry.



Rysunek 14: Wynik Sharpening Filter

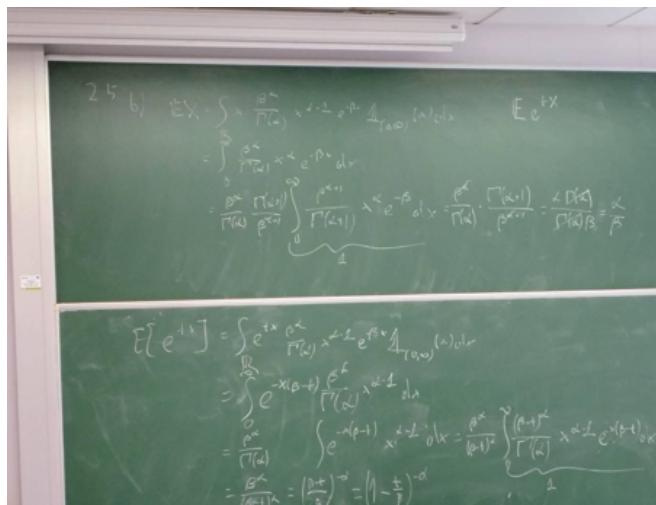
4.9 Wynik Edge Detection Filter

Filtr detekcji krawędzi (oparty na operatorze Laplace'a) zidentyfikował miejsca gwałtownej zmiany intensywności – głównie kontury i granice obiektów. Obraz zawiera głównie informacje strukturalne.

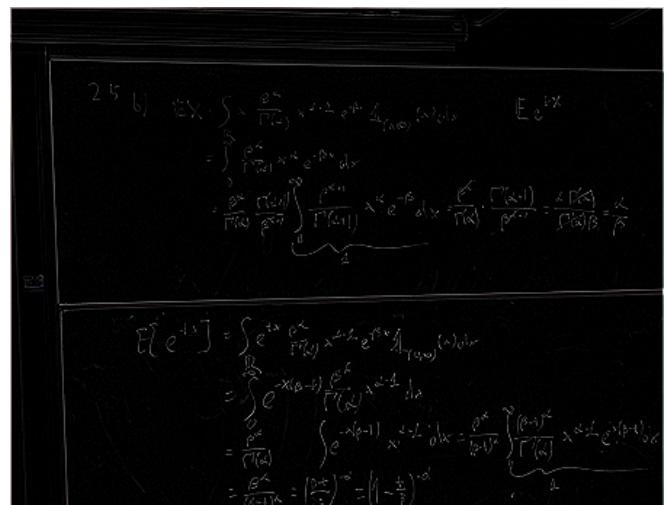


Rysunek 15: Wynik Edge Detection Filter

Na uwagę również zasługuje poniższy przykład:



(a) Obraz wejściowy tablicy



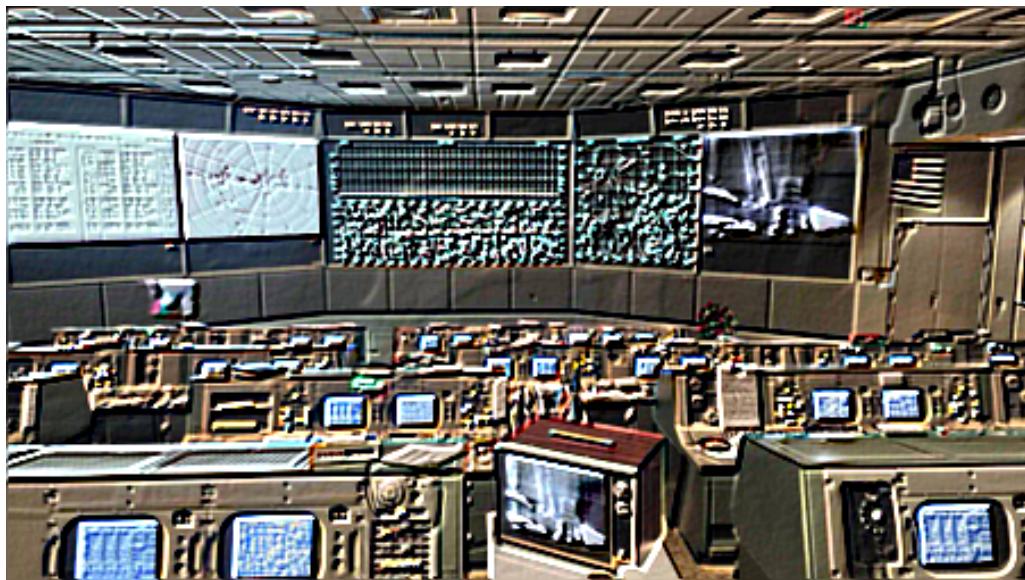
(b) Wynik edge detection na tablicy

Rysunek 16: Porównanie obrazu oryginalnego i po filtrze krawędziowym

Przeprowadzenie OCR (Optical Character Recognition) na pierwszym obrazie może okazać się trudne, natomiast na drugim obrazie doskonale widać tekst przez co algorytm OCR ma większe szanse na powodzenie.

4.10 Wynik Emboss Filter

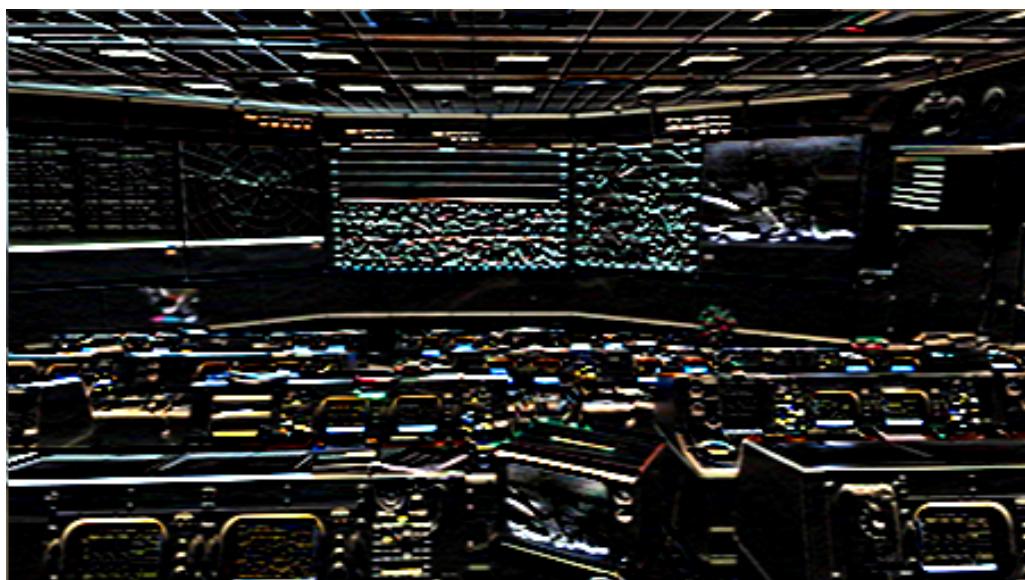
Zastosowanie filtra emboss nadało obrazowi efekt wytłoczenia 3D. Krawędzie zostały przekształcone w sposób sugerujący głębię, a tło przybrało formę zbliżoną do płaskorzeźby.



Rysunek 17: Wynik Emboss Filter

4.11 Wynik Horizontal Sobel Filter

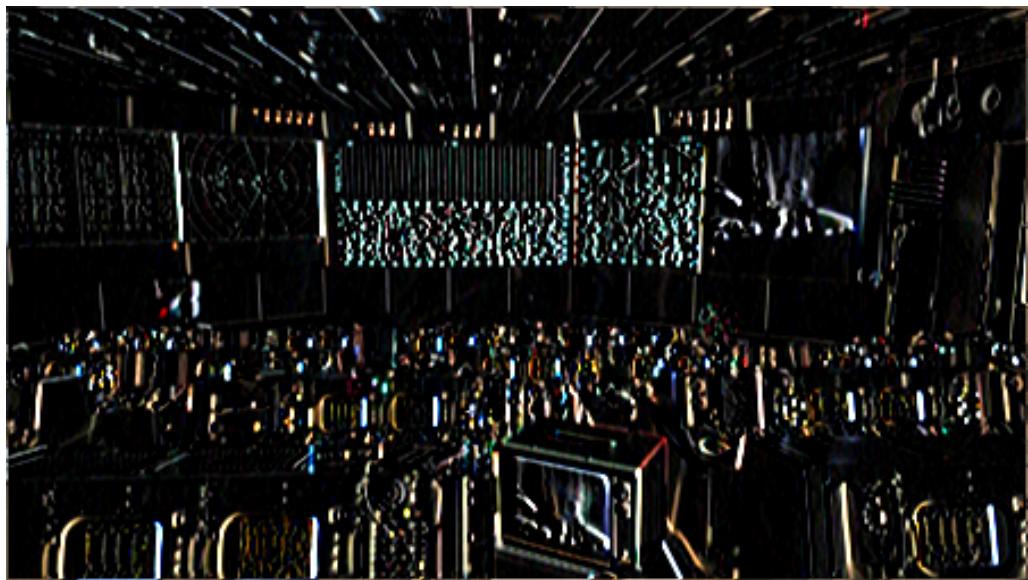
Filtr Sobela w kierunku poziomym wykrywa pionowe krawędzie w obrazie. Linie i struktury pionowe stały się bardziej wyraziste, natomiast poziome granice zostały wygaszone.



Rysunek 18: Wynik Horizontal Sobel Filter

4.12 Wynik Vertical Sobel Filter

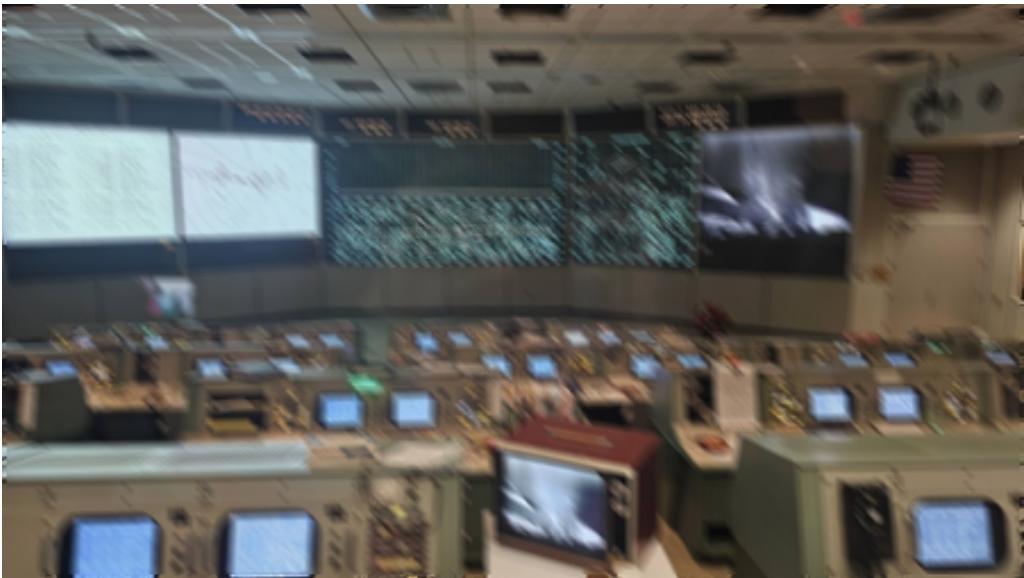
Filtr Sobela w kierunku pionowym identyfikuje poziome krawędzie. Linie horyzontalne zostały podkreślone, co uwidacznia strukturę wzdłuż osi poziomej.



Rysunek 19: Wynik Vertical Sobel Filter

4.13 Wynik Motion Blur Filter

Filtr symulujący rozmycie ruchu poprzez zastosowanie jądra z wartościami rozmieszczenymi po przekątnej. Obraz uzyskał efekt przesunięcia (rozmazania), jakby wykonano go podczas ruchu aparatu.



Rysunek 20: Wynik Motion Blur Filter

5 Wnioski

Na podstawie przeprowadzonych eksperymentów oraz zaprezentowanych wyników można sformułować następujące wnioski:

- Wszystkie zaimplementowane metody – zarówno operacje pikselowe, jak i filtry konwolucyjne – działają zgodnie z oczekiwaniami. Efekty wizualne odpowiadają ich teoretycznemu działaniu, co zostało potwierdzone na licznych przykładach.
- Szczególnie istotna była implementacja kopii obrazu wejściowego (`image.copy()`), dzięki której operacje nie wpływały na oryginalny obraz, umożliwiając porównywanie przed i po przekształceniu.
- Podczas implementacji największym wyzwaniem okazały się operacje związane z filtracją – zwłaszcza konwolucja z wykorzystaniem macierzy jądra. Należało zadbać o poprawne uwzględnienie brzegów oraz zapewnienie, że wynikowa wartość pikseli nie wykracza poza zakres [0, 255].
- Tryb przyspieszony (`fast_apply`) przyniósł zauważalne korzyści wydajnościowe, szczególnie przy filtrach o dużych jądrach (np. `7x7`). Dla większych obrazów zastosowanie NumPy skracalo czas działania nawet kilkunastokrotnie.
- W przypadku filtra binarnego, efekt końcowy silnie zależał od przyjętego progu (`threshold`).
- Niektóre filtry (np. `emboss` czy `motion blur`) szczególnie dobrze eksponują się na obrazach z wyraźnymi strukturami i kontrastami. Dla obrazów o niskim kontraste efekty mogą być mniej zauważalne.
- Rozbudowa o możliwość edycji jądra przez użytkownika (`custom_filter`) otwiera drzwi do wielu zaawansowanych zastosowań, w tym eksperymentów z filtrami własnego pomysłu, a nawet zastosowań edukacyjnych.

Projekt potwierdza, że nawet w środowisku GUI opartym na Tkinterze możliwe jest stworzenie funkcjonalnego, interaktywnego edytora graficznego, który może służyć zarówno jako narzędzie analityczne, jak i dydaktyczne. W przyszłości warto rozważyć dodanie:

- operacji morfologicznych (np. erozja, dylatacja),
- automatycznej analizy histogramu,
- oraz integracji z algorytmami rozpoznawania obiektów czy OCR.