POLITECHNIKA ŚWIĘTOKRZYSKA

Wydział Elektrotechniki, Automatyki i Informatyki

Katedra Elektrotechniki Przemysłowej i Automatyki

Zakład Urządzeń i Systemów Automatyki

Przetwarzanie obrazów i systemy wizyjne

Temat: Przetwarzanie obrazu: przekształcenia punktowe, tablice LUT i operacje na histogramie

Instrukcja laboratoryjna

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z podstawowymi technikami przetwarzania obrazów cyfrowych poprzez przekształcenia punktowe, wykorzystanie tablic LUT (ang. Look-Up Table) oraz operacje na histogramie. Ćwiczenie ma na celu naukę modyfikowania jasność i kontrast obrazu, stosownie różnych funkcji przekształceń oraz analizowanie wpływ operacji histogramowych na jakość obrazu. Pozwoli również na praktyczne wykorzystanie metod wyrównywania histogramu oraz jego modyfikacji w celu poprawy wizualnej cech obrazu.

1. Przekształcanie punktowe

1.1. Definicja i charakterystyka

Przekształcenia punktowe (zwane również przekształceniami intensywności) to operacje, w których wartość wyjściowa piksela jest określana wyłącznie na podstawie wartości tego samego piksela w obrazie wejściowym. Geometryczne relacje między pikselami pozostają bez zmian – modyfikujemy jedynie **poziom jasności** (w skali szarości) lub wartości składowych (w obrazie kolorowym) każdego pojedynczego piksela.

$$g(x,y) = f\big(s(s,y)\big),$$

gdzie $s(s,y)$ to wartości piksela wejściowego, a $g(x,y)$ to wartość piksela wyjściowego

Cechy przekształceń punktowych:

- Dotyczą wyłącznie modyfikacji jasności bądź barwy piksela (bez uwzględniania sąsiedztwa).
- Jeżeli użyta funkcja f jest **ściśle monotoniczna** (rosnąca lub malejąca), możliwe jest odwrócenie tej operacji. Gdy funkcja nie jest monotoniczna, część informacji może zostać bezpowrotnie utracona.
- Przekształcenia punktowe służą wyłącznie do lepszego uwidocznienia cech już istniejących w obrazie – **nie wprowadzają nowej informacji**.

```
import imageio.v3 as iio
import matplotlib.pyplot as plt
from skimage import img_as_float

# Wczytanie obrazu w odcieniach szarości:
```

```
image_path = 'panorama.jpg'
image_gray = iio.imread(image_path, mode='F')

plt.imshow(image_gray, cmap='gray')
plt.title("Oryginalny obraz - skala szarości")
plt.axis('off')
plt.show()
```

Listing 1: Wczytywanie obrazu.

1.2. Zastosowania

Przekształcenia punktowe stosuje się do:

- **Poprawa jakości obrazu** – np. poprawa kontrastu, jasności.
- **Negacja obrazu** – uzyskanie tzw. „negatywu".
- **Korekcja (kompensacja) charakterystyki urządzeń** – np. korekcja gamma w celu uwzględnienia nieliniowej charakterystyki monitora.
- **Wydobycie lub uwypuklenie wybranych obszarów intensywności** – np. w zastosowaniach medycznych do uwypuklania pewnych tkanek na zdjęciach rentgenowskich.
- **Arytmetyka obrazów** – dodawanie lub odejmowanie wartości pikseli (przydatne w analizie różnicowej obrazów).

1.3. Podstawowe przekształcenia punktowe

1. Zmiana jasności

$$L' = L + dL$$

Pozwala rozjaśnić jeśli $dL > 0$ lub przyciemnić jeśli $dL < 0$ cały obraz o stałą wartość.

```
def adjust_brightness(image, delta):
    """
    Zwiększa (delta>0) lub zmniejsza (delta<0) jasność obrazu
    image: obraz w formacie float [0.0, 1.0]
    delta: wartość dodawana do każdego piksela
    """
    # Dodanie wartości delta
    bright_image = image + delta

    # Obcięcie wartości poniżej 0 i powyżej 255, aby uniknąć przekroczenia
zakresu
```

```
    bright_image = np.clip(bright_image, 0.0, 225.0)
    return bright_image

# Przykładowe zastosowanie:
delta_value = 200 # np. rozjaśnienie o 200 (w zakresie 0-255)
brighter = adjust_brightness(image_gray, delta_value)

plt.imshow(brighter, cmap='gray')
plt.title(f"Zmiana jasności o +{delta_value}")
plt.axis('on')
plt.show()
```

Listing 2: Zmiana jasności.

2. Zmiana kontrastu

$$L' = k * L, \qquad (k > 0)$$

Mnożenie wartości jasności przez współczynnik k. Dla $k > 1$ zwiększamy kontrast czyli rozciągamy różnice między pikselami, natomiast dla w przedziale $0 < k < 1$ kontrast jest zmniejszany.

```
def adjust_contrast(image, factor):
    """
    Zmienia kontrast obrazu poprzez mnożenie (factor>1 => większy kontrast)
    image: obraz w formacie float [0.0, 1.0]
    factor: współczynnik kontrastu
    """
    # Środek obrazu (średnia jasność)
    mean_val = np.mean(image)

    # Operacja: (pixel - mean) * factor + mean
    # Dzięki temu podbijamy rozrzut wokół średniej
    contrast_image = (image - mean_val) * factor + mean_val

    # Obcięcie zakresu
    contrast_image = np.clip(contrast_image, 0.0, 255.0)
    return contrast_image

# Przykładowe zastosowanie:
contrast_factor = 10  # zwiększenie kontrastu
higher_contrast = adjust_contrast(image_gray, contrast_factor)

fig, ax = plt.subplots(1, 2, figsize=(15, 5))
ax[0].imshow(image_gray, cmap='gray')
ax[0].set_title('Original Image')
ax[0].axis('on')
```

```
ax[1].imshow(higher_contrast, cmap='gray')
ax[1].set_title('Image + 100')
ax[1].axis('on')

plt.show()
```

Listing 3: Zmiana kontrastu.

3. Negacja obrazu

Negacja obrazu często realizowana jest jako:

$$L' = 255 - L$$

Jest to przypadek 8-bitowego obrazu w skali szarości. Tworzymy „negatyw" obrazu – jasne obszary stają się ciemne i odwrotnie.

```
def negative(image):
    """
    Negatyw obrazu - odwrotność wartości pikseli
    Dla float: 255.0 - pixel
    """
    return 255.0 - image

neg_image = negative(image_gray)

plt.imshow(neg_image, cmap='gray')
plt.title("Negacja obrazu")
plt.axis('off')
plt.show()
```

Listing 4: Zmiana kontrastu.

4. Korekcja gamma

Oprócz nieliniowej zmiany jasności obrazu (tzw. ekspansji lub kompresji), funkcja potęgowa stanowi podstawę mechanizmu korekcji gamma. Funkcja potęgowa wykorzystywana jest do nieliniowej zmiany jasności obrazu z wąskiego obszaru w szerszy lub odwrotnie.

$$L'(x,y) = (L(x,y))^{\gamma}$$

(z odpowiednim przeskalowaniem do zakresu 0-255, jeśli pracujemy w 8 bitach.

- $\gamma > 1$ – zwiększamy kontrast w jasnych obszarach (tzw. ekspansja gamma).
- $\gamma < 1$ – wzmacnia kontrast w ciemnych obszarach (tzw. kompresja gamma).

Sam wzór potęgowy jest więc używany nie tylko do celowego zmieniania kontrastu w jasnych bądź ciemnych obszarach, lecz także do kompensacji charakterystyki urządzeń tak, by wyświetlany (lub rejestrowany) obraz odzwierciedlał rzeczywiste wrażenia luminancji.

```
def power_transformation(image, gamma):
    """
    Funkcja potęgowa L'(x) = (L(x))^gamma
    (zakładamy, że wartości obrazu są w [0,1])
    """
    # Podniesienie do potęgi gamma
    transformed = np.power(image, gamma)
    return np.clip(transformed, 0.0, 255.0)

gamma_value = 1.3  # ekspansja gamma - uwydatnienie jasnych rejonów
gamma_corrected = power_transformation(image_gray, gamma_value)

plt.imshow(gamma_corrected, cmap='gray')
plt.title(f"Korekcja gamma, gamma = {gamma_value}")
plt.axis('off')
plt.show()
```

Listing 5: Zmiana kontrastu.

5. Uwypuklanie wybranych poziomów szarości

Polega na zaprojektowaniu funkcji, która mocno „wybija” (podnosi) wybrane poziomy jasności, a inne tłumi. W konsekwencji pewne obszary na obrazie stają się bardziej widoczne, co bywa wykorzystywane np. w diagnostyce medycznej.

```
def emphasize_range(image, lower=10, upper=20):
    """
    Uwypuklanie pikseli w zakresie [lower, upper].
    Wszystkie wartości spoza tego zakresu zostaną znacznie przyciemnione.
    """
    emphasized = np.zeros_like(image)
    mask = (image >= lower) & (image <= upper)
    print(mask)
    # Możemy np. wzmocnić wartości w tym przedziale o pewien mnożnik
    emphasized[mask] = (image[mask] - lower) / (upper - lower)
    return emphasized

emphasized_image = emphasize_range(image_gray, 50, 100)

fig, ax = plt.subplots(1, 2, figsize=(15, 5))
```

```
ax[0].imshow(image_gray, cmap='gray')
ax[0].set_title('Original Image')
ax[0].axis('on')

ax[1].imshow(emphasized_image, cmap='gray')
ax[1].set_title('Emphasized Range')
ax[1].axis('on')

plt.show()
```

Listing 6: Uwypuklenie wybranych szarości.

2. Wykorzystanie tablic LUT (ang. Look-Up Table)

2.1. Idea i zaleta stosowania tablic LUT

Tablice LUT pozwalają na szybkie i wydajne wdrażanie przekształceń punktowych w sytuacjach, gdy obraz ma dużą rozdzielczość lub gdy sama funkcja przekształcająca jest złożona. Zamiast każdorazowo obliczać nową wartość piksela na podstawie formuły matematycznej, tworzy się (przed przetwarzaniem) tablicę odwzorowującą każde wejściowe natężenie (np. 0–255) w odpowiadające mu natężenie wyjściowe.

Przykład dla 8-bitowych obrazów w skali szarości:

1. Inicjalnie tworzymy tablicę o rozmiarze 256.
2. Dla każdej możliwej wartości $i \in [0,255]$ obliczamy $LUT[i] = f(i)$ (gdzie f – funkcja przekształcająca).
3. Podczas przetwarzania obrazu każdy piksel o wartości s zostaje zastąpiony wartością $LUT[s]$.

Zalety:

- Znaczna **redukcja obciążenia obliczeniowego** – nie musimy za każdym razem przeliczać funkcji.
- **Elastyczność** – wystarczy zmienić definicję tablicy LUT, aby zmienić cały sposób przekształcenia.
- **Uniwersalność** – dowolna, nawet bardzo skomplikowana, funkcja może zostać zapisana w postaci LUT (nie musi mieć postaci analitycznej).

2.2. Przykłady przekształceń realizowanych z użyciem LUT

1. **Zmiana jasności** ($L' = L + dL$) i **kontrastu** ($L' = k * L$).

2. **Funkcja potęgowa**, korekcja gamma.
3. **Negacja**, czyli $L' = 255 - L$.
4. **Uwypuklanie wybranych poziomów** – dowolna niemonotoniczna funkcja, która np. „wycina” fragmenty histogramu (zwiększając ich wartość wyjściową), a resztę tłumi.

```python
def create_lut_power_law(gamma):
    """
    Tworzy tablicę LUT dla funkcji potęgowej (gamma).
    Zakładamy 256 możliwych poziomów w skali 8-bitowej (0..255).
    """
    lut = np.arange(256, dtype=np.float32)  # [0, 1, 2, ..., 255]
    lut /= 255.0  # Normalizacja do przedziału [0,1]
    lut = lut ** gamma  # Funkcja potęgowa
    lut *= 255.0        # Powrót do przedziału [0,255]
    lut = np.clip(lut, 0, 255).astype(np.uint8)
    print(lut)
    return lut

# Przykładowe wykorzystanie:
gamma = 0.7
lut_gamma = create_lut_power_law(gamma)

# Funkcja do przekształcenia obrazu z tablicą LUT (dla obrazu w skali 0..255)
def apply_lut(image, lut):
    """
    image: numpy array z wartościami w zakresie [0,255]
    lut: tablica look-up table
    """
    # Zakładamy, że obraz jest w typie uint8
    # Każdy piksel w image[x,y] -> lut[image[x,y]]
    return lut[image]

# Przygotowanie obrazu 8-bitowego (uint8) do zastosowania LUT
image_uint8 = (image_gray * 255).astype(np.uint8)
result_lut = apply_lut(image_uint8, lut_gamma)

plt.imshow(result_lut, cmap='gray')
plt.title(f"Obraz z wykorzystaniem LUT (gamma={gamma})")
plt.axis('off')
plt.show()
```

Listing 7: Wykorzystanie tablicy LUT do przekształcenia obrazu.

3. Operacje na histogramie

3.1. Definicja i znaczenie histogramu

Histogram obrazu (w skali szarości) to funkcja, która dla każdej wartości jasności (0–255) pokazuje liczbę (lub częstość) pikseli o tej wartości w obrazie. W praktyce:

- Na osi **X** odkładamy poziomy jasności (0–255).
- Na osi **Y** – liczbę pikseli (lub procent) o danym poziomie.

Histogram pozwala szybko stwierdzić, czy obraz jest:

- „Zbyt ciemny" – większość pikseli ma niskie wartości.
- „Zbyt jasny" – większość pikseli w wysokich wartościach.
- Ma wąski zakres jasności (niski kontrast) lub szeroki.

3.2. Rozciąganie (stretching) histogramu

Rozciąganie histogramu służy do powiększenia zakresu intensywności tak, aby obejmował cały dostępny przedział (np. 0–255). Jeśli w oryginalnym obrazie piksele zajmują zakres od $\min(s)$ do $\max(s)$, to nowe wartości wyliczamy zwykle w sposób liniowy:

$$g(x,y) = \frac{s(s,y) - \min(s)}{\max(s) - \min(s)} * 255$$

- Dzięki temu obraz maksymalnie wykorzystuje cały dostępny zakres, co często poprawia widoczność detali.
- Alternatywnie można tzw. „ucinać" (clipping) pewien % pikseli w skrajnych przedziałach, jeśli zawierają jedynie szum.
-

3.3. Wyrównywanie (equalization) histogramu

Wyrównywanie histogramu (ang. *histogram equalization*) powoduje „spłaszczenie" i rozciągnięcie histogramu tak, by w miarę możliwości równomiernie rozłożyć wartości jasności.

- Intuicyjnie: jeśli jakieś poziomy jasności występują często, to są one „rozciągane", aby lepiej wydobyć różnice między tymi wartościami.
- Podstawą do wyznaczenia tej transformacji jest **skumulowany histogram** (tzw. dystrybuanta lub CDF – *cumulative distribution function*).

- W efekcie uzyskujemy obraz o zwiększonym kontraksie, zwłaszcza gdy oryginalny histogram był wąski lub miał nierównomierny rozkład.

```
def histogram_equalization(image):
    """
    Wyrównywanie histogramu dla obrazu w [0,1].
    Zwraca: obraz po equalizacji
    """
    # 1. Obliczenie histogramu (256 przedziałów)
    hist, bin_edges = np.histogram(image.flatten(), bins=256, range=(0,255))

    # 2. Obliczenie dystrybuanty skumulowanej (CDF)
    cdf = hist.cumsum()
    cdf = cdf / cdf[-1]  # normalizacja (ostatni element cdf to suma
wszystkich pikseli)

    # 3. Tworzymy mapowanie (LUT) w oparciu o cdf
    # bin_edges to przedziały (257 wartości), dlatego index = bin_index
    equalized_image = np.interp(image.flatten(), bin_edges[:-1], cdf)

    # 4. Powrót do oryginalnego kształtu
    equalized_image = equalized_image.reshape(image.shape)
    return equalized_image

equalized = histogram_equalization(image_gray)
plt.imshow(equalized, cmap='gray')
plt.title("Wyrównanie histogramu - implementacja własna")
plt.axis('off')
plt.show()
```

Listing 8: Operacje na histogramie

3.4. Zastosowanie operacji na histogramie

- **Poprawa kontrastu** – rozciąganie lub wyrównywanie histogramu, by wydobyć detale.
- **Segmentacja** – wybór wartości progowych na podstawie kształtu histogramu.
- **Analiza jakości** – histogram pozwala szybko ocenić, w jakim stopniu wykorzystywany jest dostępny zakres jasności.

```
from skimage import exposure
import imageio.v3 as iio
import matplotlib.pyplot as plt
```

```
import numpy as np

# Wczytanie obrazu w odcieniach szarości:
image_path = 'panorama.jpg'
image_gray = iio.imread(image_path, mode='F')

p2, p98 = np.percentile(image_gray, (30, 50
stretched = exposure.rescale_intensity(image_gray, in_range=(p2, p98))

fig, ax = plt.subplots(1, 4, figsize=(15, 5))
ax[0].imshow(image_gray, cmap='gray')
ax[0].set_title('Original Image')
ax[0].axis('on')

ax[2].imshow(stretched, cmap='gray')
ax[2].set_title('Stretched Image')
ax[2].axis('on')

# Histogramy
ax[1].hist(image_gray.ravel(), bins=256, range=(0.0, 255.0), fc='k', ec='k')
ax[1].set_title('Histogram - Original Image')
ax[1].set_xlim([0.0, 255.0])

ax[3].hist(stretched.ravel(), bins=256, range=(0.0, 1.0), fc='k', ec='k')
ax[3].set_title('Histogram - Stretched Image')
ax[3].set_xlim([0.0, 255.0])
ax[3].set_ylim([0, 10000])
plt.show()
```

Listing 9: Operacje na histogramie

4. Zadania do wykonania

Zadanie 1: Wczytanie i konwersja obrazów do skali szarości

- Wczytaj wybrany obraz kolorowy z użyciem biblioteki ***scikit-image*** (np. ***io.imread***) bądź ***imageio***.
- Dokonaj konwersji do obrazu w odcieniach szarości z wykorzystaniem ***color.rgb2gray***.
- Wyświetl obraz wejściowy i wynikowy w celu porównania.

Zadanie 2: Proste przekształcenia punktowe bez użycia tablic LUT

- Napisz funkcję do zmiany jasności (dodawanie stałej).

- Napisz funkcję do zmiany kontrastu (mnożenie przez współczynnik).
- Napisz funkcję do negacji obrazu (odwrotność intensywności).
- Zademonstruj działanie powyższych funkcji na obrazie w odcieniach szarości.

Zadanie 3: Przekształcenia potęgowe z wykorzystaniem różnych wartości parametru gamma

- Zaimplementuj funkcję, która realizuje przekształcenie potęgowe dla obrazu w odcieniach szarości.
- Wybierz kilka wartości parametru ***γ\gammaγ*** (np. 0.5, 1.5, 2.2) i zademonstruj efekt na obrazie.

Zadanie 4: Funkcje z tablicą LUT

- Napisz funkcję, która tworzy tablicę LUT dla zadanych operacji (np. zwiększanie jasności, kontrastu czy korekcja gamma).
- Zademonstruj jej działanie na wybranym obrazie w skali szarości.

Zadanie 5: Zmiana jasności i kontrastu dla obrazu kolorowego (kanały RGB)

- Wczytaj obraz kolorowy i rozbij go na składowe (R, G, B).
- Zastosuj prostą funkcję zmiany jasności i/lub kontrastu na każdej składowej osobno.
- Złóż obraz ponownie i wyświetl efekt.

Zadanie 6: Modyfikacje obrazu w przestrzeni HSV

- Wczytaj obraz kolorowy.
- Skonwertuj go do przestrzeni HSV (***color.rgb2hsv***).
- Zmodyfikuj wybraną składową (np. H) za pomocą funkcji zmiany jasności, kontrastu i gamma.
- Przekonwertuj z powrotem do RGB (***color.hsv2rgb***) i omów różnice wizualne.

Zadanie 7: Generowanie obrazów o zadanych zakresach jasności

- Skonwertuj wybrany obraz do [0,255] (uint8).
- Dokonaj regulacji jasności i kontrastu tak, aby końcowe piksele mieściły się w wąskich, ustalonych przedziałach (np. 0–150, 70–170, 150–255).
- Zaobserwuj efekty wizualne i skomentuj.

Przykładowe podejście:

- Wykorzystaj ***rescale_intensity*** z biblioteki ***exposure***, podając ***in_range*** i ***out_range***=(0,150) czy (70,170) itp.

Zadanie 8: Rozciąganie histogramu (implementacja własna)

- Napisz funkcję, która znajduje minimalną i maksymalną wartość piksela i wykonuje liniowe przeskalowanie do pełnego zakresu (np. 0–1).
- Zademonstruj jej działanie z wykorzystaniem exposure.histogram do wizualizacji histogramu.

Zadanie 9: Wyrównywanie histogramu (implementacja z numpy.cumsum())

- Napisz funkcję realizującą histogram equalization (jak w przykładzie w poprzednich częściach).
- Porównaj swój wynik z ***exposure.equalize_hist***.

Zadanie 10: Przykłady funkcji z scikit-image.exposure

- Wykorzystaj:
  - ***exposure.adjust_gamma***
  - ***exposure.rescale_intensity***
  - ***exposure.equalize_hist***

na wybranych obrazach (zarówno w odcieniach szarości, jak i kolorowych).

- Porównaj ich działanie z własnymi implementacjami.

Zadanie 11: Wykrywanie „nasyconych” pikseli przed rozciąganiem histogramu

- Zanim zastosujesz ***rescale_intensity***, usuń np. 1% najciemniejszych i 1% najjaśniejszych pikseli (***clipping***), by ograniczyć wpływ ekstremalnych wartości. Porównaj wyniki z rozciąganiem „bez clipowania”.

Zadanie 12: Dynamiczne tworzenie tablicy LUT na podstawie histogramu

- Zdefiniuj funkcję, która wczytuje obraz, oblicza jego histogram i na podstawie histogramu tworzy LUT rozciągającą bądź „ścięcie” jasności. Następnie zastosuj tę LUT do obrazu.

Zadanie 13: Porównanie rozciągania histogramu i wyrównywania histogramu na obrazach medycznych

- Wczytaj dostępny obraz RTG lub tomografii (w odcieniach szarości).
- Zastosuj ***stretch_histogram*** i ***equalize_hist***.
- Porównaj efekty wzrokowo, a także przeanalizuj uzyskane histogramy.

1. https://imageio.readthedocs.io/
2. https://scikit-image.org/
3. NumPy