POLITECHNIKA ŚWIĘTOKRZYSKA

Wydział Elektrotechniki, Automatyki i Informatyki

Katedra Elektrotechniki Przemysłowej i Automatyki

Zakład Urządzeń i Systemów Automatyki

Przetwarzanie obrazów i systemy wizyjne

Temat: Podstawy przetwarzania obrazów (imageio, scikit-image)

Instrukcja laboratoryjna

Cel ćwiczenia

Celem ćwiczenia jest zapoznanie się z podstawowymi operacjami przetwarzania obrazów przy użyciu języka Python w wersji 3.11 oraz bibliotek **imageio** i **scikit-image**. Ćwiczenie ma na celu zaznajomienie uczestników z podstawami cyfrowego przetwarzania obrazów, co stanowi fundament dla bardziej zaawansowanych technik analizy obrazów i komputerowego rozpoznawania wzorców.

1. Wczytywanie i zapisywanie obrazów z wykorzystaniem biblioteki *imageio*

1.1. Biblioteka *imageio* – opis i zastosowanie

Biblioteka **imageio** [1] to narzędzie do obsługi operacji wejścia/wyjścia na obrazach. Umożliwia wczytywanie i zapisywanie obrazów w różnych formatach, a także konwersję między nimi. Jest to lekkie, łatwe w użyciu rozwiązanie, które dobrze integruje się z innymi bibliotekami przetwarzania obrazów, takimi jak **scikit-image** czy **OpenCV**
Dzięki ***imageio*** możemy:

- Wczytywać obrazy w wielu popularnych formatach, takich jak **JPEG, PNG, BMP, TIFF, GIF, WebP** i inne.
- Zapisywać obrazy w różnych formatach, umożliwiając konwersję między nimi.
- Obsługiwać zarówno obrazy statyczne, jak i animowane (np. **GIF**).

1.2. Instalacja biblioteki

Jeśli biblioteka ***imageio*** nie jest jeszcze zainstalowana, można ją dodać do środowiska Python za pomocą wpisania w konsole cmd komendy:

pip install imageio

1.3. Wczytywanie obrazu za pomocą *imageio*

Aby wczytać obraz, należy użyć funkcji imageio.imread(). Przykład wczytania obrazu z pliku:

```
import imageio.v3 as iio

image = iio.imread('image1.jpg')
print (image.shape) # (300, 451, 3)
```

1.4. Obsługa różnych formatów plików

Biblioteka automatycznie rozpoznaje format pliku na podstawie rozszerzenia. Oto przykłady wczytywania różnych formatów:

```
image_jpg = iio.imread('panorama.jpg') # Wczytanie obrazu JPG

image_png = iio.imread('city.png') # Wczytanie obrazu PNG
```

1.5. Zapisywanie i konwersja obrazów na inne formaty

Dzięki ***imageio*** można łatwo zapisać obraz w innym formacie, np. konwersja z **PNG** do **JPEG**:

```
# Wczytanie obrazu PNG
image = iio.imread("city.png")

# Zapisanie obrazu w formacie JPEG
iio.imwrite("image_converted.jpg", image)
```

W podobny sposób możemy przekonwertować obraz na inne formaty:

```
# Zapis obrazu do TIFF
iio.imwrite("image_converted.tiff", image)
# Zapis obrazu do BMP
iio.imwrite("image_converted.bmp", image)
```

Biblioteka ***imageio*** jest wygodnym narzędziem do wczytywania i zapisywania obrazów w różnych formatach. Pozwala na prostą konwersję między formatami oraz integrację z innymi bibliotekami przetwarzania obrazów, co czyni ją przydatnym narzędziem w analizie i obróbce obrazów.

2. Biblioteka *scikit-image* – opis i zastosowanie

2.1. Opis biblioteki *scikit-image*

Biblioteka ***scikit-image*** [2] to zaawansowane narzędzie do przetwarzania obrazów w języku Python. Jest częścią ekosystemu ***SciPy*** i oferuje szeroki zestaw funkcji do analizy i modyfikacji obrazów, takich jak:

- Wczytywanie i zapisywanie obrazów
- Przetwarzanie kolorów i konwersje
- Operacje morfologiczne
- Filtrowanie i detekcja krawędzi
- Segmentacja i analiza cech obrazu

Dzięki ***scikit-image*** można w prosty sposób realizować złożone operacje przetwarzania obrazów przy użyciu funkcji opartych na ***NumPy***.

2.2. Instalacja biblioteki

Jeśli biblioteka ***scikit-image*** nie jest jeszcze zainstalowana, można ją dodać do środowiska Python za pomocą wpisania w konsole cmd komendy:

pip install scikit-image

2.3. Wczytywanie obrazów za pomocą *scikit-image*

Biblioteka ***scikit-image*** wykorzystuje moduł ***io*** do wczytywania obrazów w różnych formatach. Przykład wczytania obrazu:

```
from skimage import io

# Wczytanie obrazu
image = io.imread("panorama.jpg")

# Sprawdzenie wymiarów obrazu
print("Wymiary obrazu:", image.shape)
```

Podobnie jak ***imageio***, ***scikit-image*** obsługuje różne formaty plików, takie jak ***JPEG***, ***PNG***, ***TIFF***, ***BMP***.

2.4. Wbudowane obrazy w *scikit-image*

Biblioteka ***scikit-image*** zawiera kilka wbudowanych obrazów testowych, które można używać do eksperymentów i analizy. Są one dostępne w module ***skimage.data***. Lista przykładowych wbudowanych obrazów:

Obraz	Opis	Kod ładowania obrazu
camera	Czarnobiały obraz kamery	data.camera()
astronaut	Kolorowy obraz astronauty	data.astronaut()
coins	Monety w skali szarości	data.coins()
checkerboard	Szachownica	data.checkerboard()
chelsea	Kolorowy kot	data.chelsea()
coffee	Kubek kawy	data.coffee()
binary_blobs	Losowe kształty binarne	data.binary_blobs()

Przykład wczytywania wbudowanego obrazu:

```
from skimage import io, data

# Wczytanie obrazu testowego szachownica
image = data.checkerboard()

# Wyświetlenie obrazu
io.imshow(image)
io.show()
```

2.5. Zapisywanie obrazów za pomocą *scikit-image*

Biblioteka ***scikit-image*** umożliwia również zapisywanie obrazów:

```
io.imsave("panorama.png", image)
```

Jednak zapis obrazów może wymagać dodatkowych zależności, dlatego w niektórych przypadkach lepiej użyć ***imageio***.

2.6. Wyświetlanie obrazu

W scikit-image można wyświetlać obrazy na dwa sposoby:

1. **Wyświetlając za pomocą scikit-image:**

```
from skimage import io

# Wczytanie obrazu
image = io.imread("panorama.jpg")

# Wyświetlenie obrazu
io.imshow(image)
io.show()  # Wymagane w niektórych środowiskach (np. Spyder)
```

Zalety takiej metody:

- Prosta i szybka metoda do podejrzenia obrazu.
- Nie wymaga dodatkowych bibliotek.

Wady:

- Ograniczone możliwości dostosowania (np. brak łatwego dodawania tytułów, ukrywania osi).
- W niektórych środowiskach (np. Jupyter Notebook) może nie działać poprawnie.

2. **Wyświetlanie za pomocą biblioteki *matplotlib***

```
import matplotlib.pyplot as plt
from skimage import io

# Wczytanie obrazu
image = io.imread("panorama.jpg")

# Wyświetlenie obrazu z matplotlib
plt.imshow(image)
plt.axis("off")  # Ukrycie osi
plt.title("Obraz wczytany z scikit-image")  # Dodanie tytułu
plt.show()
```

Zalety:

- Większa kontrola nad wyświetlaniem obrazu (możliwość dodawania tytułów, ukrywania osi, dostosowania koloru).
- Działa dobrze w Jupyter Notebook i innych środowiskach.

Wady:

- Wymaga dodatkowej biblioteki ***matplotlib***.
- Dłuższa składnia w porównaniu do ***io.imshow()***.

2.6. Porównanie *scikit-image* i *imageio*

Cecha	imageio	scikit-image
Wczytywanie obrazów	Tak	Tak
Obsługa wielu formatów	Tak	Tak
Zapisywanie obrazów	Tak	Tak, ale z ograniczeniami
Wbudowane obrazy testowe	Nie	Tak
Operacje na obrazach	Nie	Tak
Wyświetlanie obrazów (imshow())	Nie	Tak
Wyświetlanie z matplotlib	Tak	Tak

2.7. Różnice w wydajności wczytywania i zapisywania

- ***imageio*** jest szybsze w wczytywaniu i zapisywaniu obrazów.
- **scikit-image** automatycznie konwertuje obrazy na ***NumPy***, co może spowolnić działanie.
- **scikit-image** jest lepsze, jeśli chcemy od razu przetwarzać obraz (np. filtrowanie, segmentacja).

3. Edycja obrazów – przetwarzanie obrazu z wykorzystaniem biblioteki scikit-image

Rozdział ten będzie dotyczył edycji obrazów przy użyciu ***scikit-image***. Przedstawione zostaną podstawowe operacje edycji obrazu, takie jak:

- Konwersja kolorów (np. RGB → grayscale).
- Zmiana rozmiaru (skalowanie obrazu).
- Obracanie obrazu.
- Odbicie lustrzane (flip obrazu).
- Zmiana jasności i kontrastu.

3.1. Instalacja wymaganych bibliotek

Jeśli jeszcze nie masz zainstalowanych bibliotek, możesz dodać je poleceniem:

pip install imageio scikit-image matplotlib numpy

3.2. Konwersja kolorów (RGB → skala szarości, RGB → HSV)

Obrazy cyfrowe mogą być zapisane w różnych przestrzeniach barw. W tej sekcji omówimy trzy popularne reprezentacje kolorów:

1. RGB (Red, Green, Blue)

Model RGB to najczęściej używana przestrzeń kolorów w grafice komputerowej i fotografii cyfrowej. Każdy piksel obrazu jest reprezentowany jako kombinacja trzech składowych kolorów:

- **R** (Red – czerwony)
- **G** (Green – zielony)
- **B** (Blue – niebieski)

Wartości te mogą mieć zakres od **0 do 255** (dla 8-bitowych obrazów), gdzie:

- (0, 0, 0) oznacza czarny
- (255, 255, 255) oznacza biały
- (255, 0, 0) to czysta czerwień

RGB jest przestrzenią addytywną – mieszanie składowych daje jaśniejsze kolory. Kod do wczytywania obrazu w przestrzeni RGB:

```
import imageio.v3 as iio
import matplotlib.pyplot as plt

# Wczytanie obrazu w przestrzeni RGB
image = iio.imread("panorama.jpg")

# Wyświetlenie obrazu
plt.imshow(image)
plt.axis("off")
plt.title("Oryginalny obraz (RGB)")
plt.show()
```

2. Skala szarości (Grayscale)

Obraz w skali szarości (grayscale) zawiera tylko jedną wartość jasności dla każdego piksela zamiast trzech składowych RGB.

Wartości pikseli są liczbami **od 0 do 255**, gdzie:

- 0 – czarny
- 255 – biały
- Wartości pośrednie to odcienie szarości

Konwersja RGB → Grayscale odbywa się poprzez **odpowiednie ważenie składowych RGB**. Standardowa formuła to:

$$Y = 0.2989R + 0.5870G + 0.1140B$$

gdzie ludzkie oko jest najbardziej wrażliwe na kolor zielony, więc składowa G ma najwyższą wagę. Kod do konwersji i wyświetlenia obrazu w skali szarości:

```
from skimage import io, color
import matplotlib.pyplot as plt

# Wczytanie obrazu
image = io.imread("panorama.jpg")

# Konwersja do skali szarości
gray_image = color.rgb2gray(image)

# Wyświetlenie obrazu
plt.imshow(gray_image, cmap="gray")  # cmap="gray" ustawia właściwą paletę kolorów
plt.axis("off")
plt.title("Obraz w skali szarości")
plt.show()
```

Zastosowanie:

- Przetwarzanie obrazu w uczeniu maszynowym (MNIST, OCR).
- Segmentacja obiektów w analizie obrazu.
- Redukcja danych w obróbce obrazu.

3. Przestrzeń HSV (Hue, Saturation, Value)

Model HSV to alternatywna przestrzeń barw, która bardziej odpowiada percepcji człowieka. Każdy piksel jest reprezentowany przez **trzy składowe:**

- **H (Hue – odcień koloru)**- wartości od 0 do 360° (0° – czerwony, 120° – zielony, 240° – niebieski).
- **S (Saturation – nasycenie)** - jak bardzo kolor jest "żywy", od 0 (szary) do 1 (pełny kolor).
- **V (Value – jasność)** - określa, jak jasny jest kolor, od 0 (czarny) do 1 (pełna jasność).

HSV jest używane w filtrach koloru, ponieważ separuje odcień od nasycenia i jasności, co ułatwia np. wykrywanie obiektów w określonym kolorze. Kod do konwersji i wyświetlenia obrazu w przestrzeni HSV:

```
from skimage import io, color
import matplotlib.pyplot as plt

# Wczytanie obrazu
image = io.imread("panorama.jpg")

# Konwersja do przestrzeni hsv
hsv_image = color.rgb2hsv(image)

# Wyświetlenie obrazu
plt.imshow(hsv_image)
plt.axis("off")
plt.title("Obraz w przestrzeni HSV")
plt.show()
```

3.3. Zmiana rozmiaru obrazu

Obraz można zmieniać proporcjonalnie lub na określone wymiary przy użyciu transform.resize(). Ma to zastosowanie np. do zmniejszania obrazów w celu oszczędzania pamięci bądź normalizacji rozmiaru przed sieciami neuronowymi. Kod zmieniający rozmiar obrazu:

```
from skimage import io, color
from skimage.transform import resize
import matplotlib.pyplot as plt

# Wczytanie obrazu
image = io.imread("panorama.jpg")
```

```
# Skalowanie obrazu do 50% oryginalnego rozmiaru
resized_image = resize(image, (image.shape[0] / 2, image.shape[1] / 2))

# Wyświetlenie oryginału i przeskalowanego obrazu
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].imshow(image)
ax[0].set_title("Oryginalny obraz")
ax[0].axis("on")

ax[1].imshow(resized_image)
ax[1].set_title("Zmniejszony obraz (50%)")
ax[1].axis("on")

plt.show()
```

3.4. Obracanie obrazu

Obrót obrazu jest jedną z podstawowych operacji edycji obrazu, stosowaną np. do korekcji orientacji zdjęć. W ***scikit-image*** możemy użyć funkcji ***transform.rotate()***, aby obrócić obraz o dowolny kąt. Rotację stosuje się np. do ujednolicenia pozycji obiektów w analizie obrazu. Kod obracający obraz:

```
from skimage import io, color
from skimage.transform import rotate
import matplotlib.pyplot as plt

# Wczytanie obrazu
image = io.imread("panorama.jpg")

# Obrót o 45 stopni
rotated_image = rotate(image, 45)

# Wyświetlenie oryginału i obróconego obrazu
fig, ax = plt.subplots(1, 2, figsize=(10, 5))
ax[0].imshow(image)
ax[0].set_title("Oryginalny obraz")
ax[0].axis("on")

ax[1].imshow(rotated_image)
ax[1].set_title("Obrócony o 45°")
ax[1].axis("on")

plt.show()
```

W tym przypadku można zaobserwować problem z obcinaniem obrazu po obrocie. Podczas obracania obrazu **nie zmienia się jego rozmiar**, co oznacza, że część obrazu może zostać **ucięta**, miejsca poza oryginalnym obszarem są **wypełnione czarnym tłem**. Jeśli chcemy, aby cały obrócony obraz był widoczny, należy **zwiększyć rozmiar płótna** przed obrotem. Można to zrobić poprzez dodanie większego obszaru tła dodając do funkcji parametr ***resize*** na wartość ***true***. Kod przedstawiający obrót wraz z powiększeniem obszaru obrazu:

```
# Obrót o 45 stopni ze zmiana rozmiaru obrazu
rotated_image = rotate(image, 45, resize=True)
```

3.5. Odbicie lustrzane (flip obrazu)

Odbicie można wykonać w pionie lub poziomie, zamieniając osie obrazu. Zabieg ten ma zastosowanie np. w augmentacji danych w uczeniu maszynowym, korekcja odbicia w zdjęciach. Kod wykorzystujący odbicie lustrzane:

```
from skimage import io, color
from skimage.transform import rotate
import matplotlib.pyplot as plt
import numpy as np

# Wczytanie obrazu
image = io.imread("panorama.jpg")

# Odbicie poziome (flip lewo-prawo)
flipped_lr = np.fliplr(image)

# Odbicie pionowe (flip góra-dół)
flipped_ud = np.flipud(image)

# Wyświetlenie wyników
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(image)
ax[0].set_title("Oryginalny obraz")
ax[0].axis("off")

ax[1].imshow(flipped_lr)
ax[1].set_title("Odbicie poziome")
ax[1].axis("off")

ax[2].imshow(flipped_ud)
ax[2].set_title("Odbicie pionowe")
```

```
ax[2].axis("off")

plt.show()
```

3.6. Zmiana jasności i kontrastu

Jasność i kontrast można zmieniać poprzez operacje na wartościach pikseli. Wykorzystywana jest np. do poprawy widoczności szczegółów na obrazie, przygotowania do segmentacji. Kod zmieniający jasność i kontrast obrazu:

```
from skimage import io
import matplotlib.pyplot as plt
from skimage import exposure

# Wczytanie obrazu
image = io.imread("panorama.jpg")

# Zwiększenie jasności
bright_image = exposure.adjust_gamma(image, gamma=0.5)

# Zwiększenie kontrastu
high_contrast_image = exposure.rescale_intensity(image, in_range=(50, 200))

# Wyświetlenie wyników
fig, ax = plt.subplots(1, 3, figsize=(15, 5))
ax[0].imshow(image)
ax[0].set_title("Oryginalny obraz")
ax[0].axis("off")

ax[1].imshow(bright_image)
ax[1].set_title("Jasność zwiększona")
ax[1].axis("off")

ax[2].imshow(high_contrast_image)
ax[2].set_title("Zwiększony kontrast")
ax[2].axis("off")

plt.show()
```

4. Zadania do wykonania

Zadanie 1: Uruchomić i przeanalizować skrypty z przykładów znajdujących się w instrukcji.

Zadanie 2: Wczytywanie i analiza obrazu

1. Wczytaj obraz z pliku za pomocą biblioteki imageio.
2. Wyświetl podstawowe informacje o obrazie:
    - Wymiary obrazu (wysokość, szerokość, liczba kanałów kolorów).
    - Typ danych pikseli (dtype).
3. Wczytaj ten sam obraz w formacie **PNG** i porównaj jego właściwości z obrazem w formacie JPEG.

Zadanie 3: Konwersja i zapis obrazów

1. Wczytaj obraz w formacie **JPEG** przy użyciu imageio.
2. Zapisz ten obraz w trzech różnych formatach: **PNG, BMP, TIFF**.
3. Spróbuj wczytać zapisane obrazy i porównać ich właściwości (wymiary, typ danych pikseli).

Zadanie 4: Wczytywanie i podstawowa analiza obrazu

1. Wczytaj obraz "przykladowy_obraz.jpg" za pomocą scikit-image.
2. Wyświetl podstawowe informacje o obrazie:
    - Wymiary (shape).
    - Liczba kanałów kolorów.
    - Typ danych (dtype).

Zadanie 5: Generowanie i zapis obrazu testowego

1. Wygeneruj obraz testowy z scikit-image. Możesz wybrać jeden z wbudowanych obrazów.
2. Wyświetl wygenerowany obraz.
3. Zapisz obraz w formacie **PNG** i **JPEG**.

Zadanie 6: Porównanie szybkości wczytywania i zapisywania obrazów (imageio vs scikit-image)

1. Wczytaj duży obraz (large_image.jpg) za pomocą imageio i zmierz czas operacji.

2. Powtórz operację dla scikit-image.
3. Zapisz obraz w formacie JPEG i PNG za pomocą obu bibliotek i zmierz czas każdej operacji.
4. Porównaj wyniki i zdecyduj, która biblioteka działa szybciej.

Podpowiedź:

Użyj modułu ***time*** do pomiaru czasu wykonania każdej operacji, czasy wyświetl z użyciem funkcji ***print()***.

Zadanie 7: Konwersja obrazu RGB → Grayscale

1. Wczytaj obraz "przykladowy_obraz.jpg".
2. Przekonwertuj go do skali szarości (rgb2gray).
3. Wyświetl obraz oryginalny i w skali szarości.
4. Zapisz obraz w skali szarości jako "gray_image.jpg".

Zadanie 8: Konwersja obrazu RGB → HSV

1. Wczytaj obraz "przykladowy_obraz.jpg".
2. Przekonwertuj go do przestrzeni HSV (rgb2hsv).
3. Wyświetl oryginalny obraz oraz obraz w przestrzeni HSV.
4. Przekonwertuj obraz HSV z powrotem na RGB i również go wyświetl.
5. Zapisz przekonwertowane obrazy do pliku.

Zadanie 9: Zmiana rozmiaru obrazu

1. Wczytaj obraz "przykladowy_obraz.jpg".
2. Przeskaluj go do 30%, 50%, 80% oryginalnych wymiarów.
3. Wyświetl obraz przed i po zmianie rozmiaru.
4. Zapisz edytowane obrazy do pliku.

Zadanie 10: Obracanie obrazu

1. Wczytaj obraz "przykladowy_obraz.jpg".
2. Obróć go o 30°, 45°, 60°, 90°.
3. Wyświetl obraz oryginalny i obrócony.

4. Zapisz edytowane obrazy do pliku.

Zadanie 10: Odbicie lustrzane obrazu

1. Wczytaj obraz "przykladowy_obraz.jpg".
2. Wykonaj odbicie lustrzane **w poziomi**.
3. Wykonaj odbicie lustrzane **w pionie**.
4. Wyświetl wszystkie trzy wersje.
5. Zapisz powstałe obrazy do pliku.

Zadanie 11: Zmiana jasności i kontrastu

1. Wczytaj obraz "przykladowy_obraz.jpg".
2. Zwiększ jasność (adjust_gamma).
3. Zwiększ kontrast (rescale_intensity).
4. Wyświetl oryginalny obraz oraz obie zmodyfikowane wersje.
5. Zapisz powstałe obrazy do pliku.

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