

# Zaawansowane algorytmy wizyjne

materiały do ćwiczeń laboratoryjnych

Piotr Pawlik, Tomasz Kryjak

Copyright © 2018 Piotr Pawlik, Tomasz Kryjak

PUBLISHED BY AGH

*First printing, March 2018*

# Contents

<b>1</b>	<b>Wstęp do implementacji algorytmów wizyjnych w Python 3.X ..</b>	<b>5</b>
1.1	Wykorzystywane oprogramowanie	5
1.2	Moduły Pythona wykorzystywane w przetwarzaniu obrazów	5
1.3	Operacje wejścia/wyjścia	6
1.3.1	Wczytywanie, wyświetlanie i zapisywanie obrazu z wykorzystaniem OpenCV	6
1.3.2	Wczytywanie, wyświetlanie i zapisywanie obrazu z wykorzystaniem modułu <i>Matplotlib</i>	7
1.4	Konwersje przestrzeni barw	7
1.4.1	OpenCV	7
1.4.2	Matplotlib	8
1.5	Skalowanie, zmiana rozdzielczości	8
1.5.1	OpenCV	8
1.5.2	SciPy	8
1.6	Operacje arytmetyczne: dodawanie, odejmowanie, mnożenie, moduł z różnicy	9
1.7	Wyliczenie histogramu	9
1.8	Wyrównywanie histogramu	9
1.8.1	Wyrównywanie „klasyczne”	10
1.8.2	Wyrównywanie CLAHE	10
1.9	Filtracja	10



# 1 — Wstęp do implementacji algorytmów wizyjnych

W ramach ćwiczenia zaprezentowane/przypomniane zostaną podstawowe informacje związane z wykorzystaniem języka Python do realizacji operacji przetwarzania i analizy obrazów, a także sekwencji wideo.

## 1.1 Wykorzystywane oprogramowanie

Przed rozpoczęciem ćwiczeń proszę utworzyć własny katalog roboczy w miejscu podanym przez Prowadzącego. Nazwa katalogu powinna być związana z Państwa nazwiskiem.

Do przeprowadzania ćwiczeń można wykorzystać jedno z dwóch środowisk programowania w Pythonie - Spyder3 lub PyCharm. Ich zaletą jest możliwość łatwego podglądu tablic dwuwymiarowych (czyli obrazów). Spyder jest prostszy, ale także ma mniejsze możliwości i więcej niedociągnięć. PyCharm jest oprogramowaniem komercyjnym udostępnianym także w wersji darmowej. W PyCharmie pracę należy zacząć od utworzenia projektu, Spyder pozwala na pracę na pojedynczych plikach (bez konieczności tworzenia projektu).

## 1.2 Moduły Pythona wykorzystywane w przetwarzaniu obrazów

Python nie posiada natywnego typu tablicowego. Do operacji na tablicach 2D (czyli także na obrazach) powszechnie używa się zewnętrznego modułu *NumPy*. Jest to podstawa dla wszystkich dalej wymienionych modułów wspierających przetwarzanie i wyświetlanie obrazów. Istnieją co najmniej 3 zasadnicze pakiety wspierające przetwarzanie obrazów:

- para modułów: moduł *ndimage* z biblioteki *SciPy* – zawiera funkcje do przetwarzania obrazów (także wielowymiarowych) oraz moduł *pyplot* z biblioteki *Matplotlib* – do wyświetlania obrazów i wykresów.
- moduł *PILLOW* (fork<sup>1</sup> starszego, nierozwijanego modułu *PIL*)
- moduł *cv2* będący nakładką na popularną bibliotekę *OpenCV*.

W naszym kursie oprzemy się na *OpenCV*, aczkolwiek wykorzystywane będzie także *Matplotlib* i sporadycznie *ndimage*. Głównie w sytuacjach kiedy funkcje z *OpenCV* nie mają odpowiednich funkcjonalności albo są mniej wygodne w stosowaniu (np. wyświetlanie obrazów)

---

<sup>1</sup> gałąź boczna projektu

## 1.3 Operacje wejścia/wyjścia

### 1.3.1 Wczytywanie, wyświetlanie i zapisywanie obrazu z wykorzystaniem OpenCV

1. Ze strony kursu pobierz obraz *mandril.jpg* i umieść go w we własnym katalogu roboczym.
2. Uruchom program *spyder3* lub *charm* (z konsoli lub za pomocą ikony). Stwórz nowy plik i zapisz go we własnym katalogu roboczym
3. W pliku załaduj moduł *cv2* (`import cv2`). Przetestuj wczytywanie i wyświetlanie obrazów za pomocą tej biblioteki – do wczytywania służy funkcja *imread* (przykład użycia: `I = cv2.imread('mandril.jpg')`).
4. Wyświetlenie obrazka wymaga użycia co najmniej dwóch funkcji – *imshow()* tworzy okienko i rozpoczyna wyświetlanie, a *waitKey()* pokazuje obraz przez zadany okres czasu (argument 0 oznacza czekanie na wciśnięcie klawisza). Po zakończeniu pracy okno może być automatycznie zamykane, ale pod warunkiem zakończenia przez naciśnięcie dowolnego klawisza.
5. UWAGA - Zamknięcie okna przez przycisk na belce okna spowoduje zapętlenie programu i konieczność jego przerwania:  
 Spyder - zamknięcie konsoli IPython  
 PyCharm - wybranie pozycji 'Stop' w menu lub odpowiadającego jej przycisku  
 Niepotrzebne już okno można zamknąć za pomocą *destroyWindow* z odpowiednim parametrem lub zamknąć wszystkie otwarte okna za pomocą *destroyAllWindows*.

```
I = cv2.imread('mandril.jpg')
cv2.imshow("Mandrill", I)      # wyświetlenie
cv2.waitKey(0)                 # oczekiwanie na klawisz
cv2.destroyAllWindows()       # zamknięcie wszystkich okien
```

6. Uwaga. Okno niekoniecznie musi zostać wyświetlone "na wierzchu".
7. Zapis do pliku realizuje funkcja *imwrite* (proszę zwrócić uwagę na zmianę formatu z jpg na png):

```
cv2.imwrite("m.png", I) # zapis obrazu do pliku
```

8. Wyświetlany obraz jest kolorowy, co znaczy, że składa się z 3 składowych. Innymi słowy jest reprezentowany jako tablica 3D. Często istnieje potrzeba podglądnięcia wartości tej tablicy. Można to robić 'liniowo', ale lepiej to zrobić w formie macierzy dwuwymiarowej. Spyder - w zakładce *Variable explorer* (narzędzie podobne do *Workspace* znanego z Matlaba) wystarczy kliknąć na zmienną *I*.

PyCharm - tu konieczne jest uruchomienie programu w trybie Debug i ustawienie break-pointa. Po zatrzymaniu na nim i kliknięciu w zakładkę Debugger należy kliknąć w tekst *View as Array* na końcu linii pokazującej zmienną *I*.

W obu wypadkach wyświetlona jest macierz 2D będąca 'wycinkiem' tablicy 3D, jednakże domyślnie jest to wycinek 'w złej osi' - wyświetlana jest pojedyncza linia w 3 składowych. Aby uzyskać wyświetlenie całego obrazu dla jednej składowej należy zmienić oś.

Spyder - pole Axis należy ustawić na 2

PyCharm - Wycinek *I[0]* należy zmienić na *I[:, :, 0]*

Pozostałe składowe są dostępne:

Spyder - pole Index, wartości 1 i 2

PyCharm - Wycinki *I[:, :, 1]* i *I[:, :, 2]*

9. W pracy przydatny może być dostęp do parametrów obrazu:

```
print(I.shape) # rozmiary /wiersze, kolumny, glebia/
print(I.size)  # liczba bajtów
print(I.dtype) # typ danych
```

Funkcja *print* to oczywiście wyświetlanie na konsolę.

### 1.3.2 Wczytywanie, wyświetlanie i zapisywanie obrazu z wykorzystaniem modułu *Matplotlib*

Alternatywny sposób realizacji operacji wejścia/wyjścia to użycie biblioteki *Matplotlib*. Wtedy obsługa wczytywania/wyświetlania itp. jest zbliżona do tej znanej z pakietu Matlab. Dokumentacja biblioteki dostępna jest on-line Matplotlib.

W celu zachowania pewnego porządku w kodzie, proponuje się wykorzystanie innego pliku z kodem źródłowym.

1. Załaduj moduł *pyplot*.

```
import matplotlib.pyplot as plt
```

(as pozwala skrócić nazwę, którą trzeba będzie wykorzystywać w projekcie)

2. Wczytaj ten sam obraz *mandril.jpg*

```
I = plt.imread('mandril.jpg')
```

3. Wyświetlanie realizuje się bardzo podobnie jak w pakiecie Matlab:

```
plt.figure(1)          # stworzenie figury
plt.imshow(I)          # dodanie do niej obrazka
plt.title('Mandrill')  # dodanie tytułu
plt.axis('off')         # wyłączenie wyświetlania układu współrzędnych
plt.show()             # wyświetlenie całości
```

4. Zapisywanie obrazu:

```
plt.imsave('mandril.png', I)
```

5. Podczas laboratorium przydatne może być też wyświetlanie pewnych elementów na obrazku – np. punktów, czy ramek.

6. Dodanie wyświetlania punktów:

```
x = [ 100, 150, 200, 250]
y = [ 50, 100, 150, 200]
plt.plot(x, y, 'r.', markersize=10)
```

Uwaga 1. Przy ustalaniu wartości tablicy przecinki są niezbędne (w odróżnieniu od Matlab). Uwaga 2. W poleceniu `plot` składania podobna jak w Matlabie – 'r' – kolor, '.' – kropka oraz rozmiar markera. Pełna lista możliwości w dokumentacji.

7. Dodanie wyświetlania prostokąta – rysowanie kształtów tj. prostokątów, elips, kół itp. jest dostępne w *matplotlib.patches*.

```
from matplotlib.patches import Rectangle          # dodac na
poczatku                                         #
fig, ax = plt.subplots(1)                       # zamiast plt.
figure(1)                                       #
rect = Rectangle((50,50),50,100,fill=False, ec='r'); # ec - kolor
krawedzi                                       #
ax.add_patch(rect)                             # wyswietlenie
plt.show()                                     #
```

## 1.4 Konwersje przestrzeni barw

### 1.4.1 OpenCV

Do konwersji przestrzeni barw służy funkcja *cvtColor*.

```
IG = cv2.cvtColor(I, cv2.COLOR_BGR2GRAY)
IHSV = cv2.cvtColor(I, cv2.COLOR_BGR2HSV)
```

Uwagi:

1. Proszę zauważyć, że w OpenCV odczyt jest w kolejności BGR, a nie RGB. Może to być istotne w przypadku „ręcznego” manipulowania pikselami.
2. Pełna lista dostępnych konwersji wraz ze stosownymi wzorami w dokumentacji OpenCV

Zadania:

1. Dokonać konwersji obrazu *mandril.jpg* do odcieni szarości i przestrzeni HSV. Wynik wyświetlić.
2. Wyświetlić składowe H,S,V obrazu po konwersji.

Uwaga. Przydatna składnia:

```
IH = IHSV[:, :, 0];
IS = IHSV[:, :, 1];
IV = IHSV[:, :, 2];
```

Proszę zwrócić uwagę, że w odróżnieniu np. od pakietu Matlab, tu indeksowanie jest od 0.

### 1.4.2 Matplotlib

Tu wybór dostępnych konwersji jest dość ograniczony.

- RGB do odcienie szarości. Można wykorzystać rozbieżność na poszczególne kanały i wzór:

$$G = 0.299 \cdot R + 0.587 \cdot G + 0.144 \cdot B \quad (1.1)$$

Całość można opakować w funkcję:

```
def rgb2gray(I):
    return 0.299*I[:, :, 0] + 0.587*I[:, :, 1] + 0.114*I[:, :, 2]
```

Uwaga. Przy wyświetlaniu należy ustawić mapę kolorów. Inaczej obraz wyświetli się, w domyślnej, która nie jest bynajmniej w odcieniach szarości:

```
plt.gray()
```

- RGB do HSV.

```
I_HSV = matplotlib.colors.rgb_to_hsv(I)
```

## 1.5 Skalowanie, zmiana rozdzielczości

Zadanie: przeskalować obraz *mandril*.

### 1.5.1 OpenCV

Do skalowania służy funkcja *resize*. Przykład użycia:

```
height, width = I.shape[:2] # pobranie elementów 1 i 2 tj. odpowiednio
    wysokości i szerokości
scale = 1.75 # współczynnik skalujący
Ix2 = cv2.resize(I, (int(scale*height), int(scale*width)))
cv2.imshow("Big_Mandril", Ix2)
```

### 1.5.2 SciPy

W bibliotece *SciPy* dostępna jest funkcja *imresize*. Przykład użycia:

```
I_2 = scipy.misc.imresize(I, 0.5)
```



## 1.6 Operacje arytmetyczne: dodawanie, odejmowanie, mnożenie, moduł z różnicy

Obrazy są macierzami, a zatem operacje arytmetyczne są dość proste – tak jak w Matlab. Należy oczywiście pamiętać o konwersji na odpowiedni typ danych. Zwykle dobrym wyborem będzie `double`.

1. Pobierz ze strony kursu obraz *lena*, a następnie go wczytaj (dla ustalenia uwagi do części OpenCV). Wykonaj konwersję do odcieni szarości. Dodaj „szarą” wersję mandryla i Leny. Wynik wyświetl.
2. Podobnie wykonaj odjęcie i mnożenie obrazów.
3. Zaimplementuj kombinację liniową obrazów. Uwaga. Przy wyświetleniu obraz nie będzie poprawny, ponieważ jest o typu *float64* (*double*). Stosowna konwersja:

```
import numpy as np
cv2.imshow("C", np.uint8(C))
```

4. Ważną operacją w przetwarzaniu jest moduł z różnicy obrazów. Można ją wykonać „ręcznie” – konwersja na odpowiedni typ, odjęcie, moduł (`abs`), konwersja na *uint8*. Alternatywa to wykorzystanie funkcji *absdiff* z OpenCV.

## 1.7 Wyliczenie histogramu

Wyliczanie histogramu można wykonać z wykorzystaniem funkcji `calcHist`. Jednak zanim do tego przejdziemy, przypomnimy sobie podstawowe struktury sterowania w Pythonie – funkcje i podprogramy. Proszę samodzielnie dokończyć poniższą funkcję wyliczającą histogram z obrazu w 256-ciu odcieniach szarości:

```
def hist(img):
    h=np.zeros((256,1), np.float32) # tworzy i zeruje tablice
    jednokolumnowa
    height, width =img.shape[:2] # shape - krotka z wymiarami - bierzemy
    2 pierwsze
    for y in range(height):
        .....
    return h
```

Uwaga. W Pythonie ważne są wcięcia, gdyż to one wyznaczają blok funkcji, czy pętli ! Histogram można wyświetlić wykorzystując funkcję `plt.hist` lub `plt.plot` z biblioteki *Matplotlib*.

Funkcja `calcHist` może policzyć histogram kilku obrazów (lub składowych), stąd jako parametry otrzymuje tablice (`np.` obrazów) a nie jeden obraz. Najczęściej jednak wykorzystywana jest postać:

```
hist = cv2.calcHist([IG], [0], None, [256], [0, 256])
# [IG] -- obraz wejsciowy
# [0] -- dla obrazow w odcieniach szarosci jest tylko jeden kanal
# None -- maska (mozna liczyc histogram z wybranego fragmentu obrazu)
# [256] -- liczba przedzialow histogramu
# [0 256 ] -- zakres w ktorym liczony jest histogram
```

## 1.8 Wyrównywanie histogramu

Wyrównywanie histogramu to popularna i ważna operacja przetwarzania wstępnego. Zadanie: uruchom i porównaj obie metody wyrównywania.

### 1.8.1 Wyrównywanie „klasyczne”

```
IGE = cv2.equalizeHist(IG)
```

### 1.8.2 Wyrównywanie CLAHE

Metoda klasyczna jest globalna, zatem nie działa najlepiej dla obrazów niejednorodnych. Jednym z ciekawych rozwiązań jest zastosowanie algorytmu CLAHE (ang. *Contrast Limited Adaptive Histogram Equalization*) Metoda działa następująco:

- podział obrazu na rozłączne bloki (kwadratowe),
- wyliczenie w bloku histogramu,
- wykonanie wyrównywania, przy czym ogranicza się maksymalną „wysokość” histogramu, a nadmiar re-dystrybuuje na sąsiednie przedziały,
- interpolację wartości pikseli na podstawie wyliczonych histogramów dla danych bloków (uwzględnia się cztery sąsiednie środki kwadratów).

Szczegóły na Wiki oraz tutorial.

```
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
# clipLimit - maksymalna wysokosc slupka histogramu - wartosci powyzej
#             rozdzielana sa pomiedzy sasiadow
# tileGridSize - rozmiar pojedynczego bloku obrazu (metoda lokalna, dziala
#             na rozdzielnych blokach obrazu)
I_CLAHE = clahe.apply(IG)
```

## 1.9 Filtracja

Filtracja to bardzo ważna grupa operacji na obrazach. W ramach ćwiczenia proszę uruchomić:

- filtrację Gaussa (GaussianBlur)
- filtrację Sobela (Sobel)
- Laplasjan (Laplacian)
- medianę (medianBlur)

Uwaga. Pomocna dokumentacja OpenCV Proszę zwrócić uwagę również na inne dostępne funkcje:

- filtrację bilateralną,
- filtry Gabora,
- operacje morfologiczne (wykorzystane poniżej).



## Bibliography