Zaawansowane algorytmy wizyjne

materiały do ćwiczeń laboratoryjnych

Piotr Pawlik, PhD
Tomasz Kryjak, PhD

PUBLISHED BY AGH

First printing, 2022

Spis treści

1 — Algorytmy wizyjne w Python 3.X – wstęp

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 – zalecana forma to `NazwiskoImie` bez polskich znaków.

Do przeprowadzania ćwiczeń można wykorzystać jedno z trzech środowisk programowania w Pythonie – Spyder5, PyCharm lub Visual Studio Code (VSCode). 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 (wygląd zbliżony do CLion dla C/C++). W PyCharmie pracę należy zacząć od utworzenia projektu, Spyder pozwala na pracę na pojedynczych plikach (bez konieczności tworzenia projektu).

Visual Studio Code jest darmowym, lekkim, intuicyjnym i łatwym w obsłudze edytorem kodu źródłowego. Jest to uniwersalne oprogramowanie do dowolnego języka programistycznego. Konfiguracja programu do ćwiczeń sprowadza się jedynie do wybrania odpowiedniego interpretera, a dokładnie odpowiedniego środowiska wirtualnego w języku Python.

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[1] starszego, nierozwijanego modułu PIL),

[1] gałąź boczna projektu

- 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).

1.3 Operacje wejścia/wyjścia

1.3.1 Wczytywanie, wyświetlanie i zapisywanie obrazu z wykorzystaniem OpenCV

Zadanie 1.1 Wykonaj zadanie, w którym przećwiczysz obsługę plików z wykorzystaniem OpenCV.

1. Ze strony kursu pobierz obraz *mandril.jpg* i umieść go we własnym katalogu roboczym.
2. Uruchom program – *spyder*, *pycharm*, *vscode* – 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 jest automatycznie zamykane, ale pod warunkiem zakończenia przez naciśnięcie dowolnego klawisza.

   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,
   - VSCode – zatrzymanie programu w terminalu poprzez użycie skrótu klawiszowego 'CTRL+Z' lub zamknięcie terminalu.

   Wskazówka
   W przypadku szczególnie obrazów o dużej rozdzielczości warto przed funkcją `cv2.imshow()` użyć funkcję `namedWindow()` z tą samą nazwą co w funkcji *imshow()*.

5. 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("Mandril",I)       # display
cv2.waitKey(0)                 # wait for key
destroyAllWindows()         # close all windows
```

   Wskazówka
   Okno niekoniecznie musi zostać wyświetlone na wierzchu. Dobrą praktyką jest zawsze stosowanie funkcji *cv2.destroyAllWindows()*.

6. 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
```

7. Wyświetlany obraz jest kolorowy, co oznacza, że składa się z 3 składowych. Innymi słowy jest reprezentowany jako tablica 3D. Często istnieje potrzeba podglądnięcia wartości tej tablicy. Można to robić w formie jednowymiarowej, ale lepiej to zrobić za pomocą macierzy dwuwymiarowej.
   - Spyder – w zakładce *Variable explorer* (narzędzie podobne do *Workspace* znanego z Matlaba) wystarczy kliknąć na zmienną *I*.
   - PyCharm – konieczne jest uruchomienie programu w trybie *Debug* i ustawienie breakpointa. Po zatrzymaniu na nim i kliknięciu w zakładkę *Debugger* należy kliknąć w tekst *View as Array* na końcu linii pokazującej zmienną *I*,
   - VSCode – konieczne jest uruchomienie programu w trybie *Debug* i ustawienie breakpointa. W zakładce *Run and Debug* i w sekcji *Variables* znajdziemy zmienną *I*. Aby wyświetlić wartości macierzy *I* należy wybrać opcję *View Value in Data Viewer* pod prawym klawiszem myszki. W sekcji *Watch* możliwe jest odwołanie do konkretnego elementu tablicy *I* lub wykonać odpowiednie operacje na niej.

   We wszystkich przypadkach wyświetlana 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]`,
   - VSCode – w sekcji *SLICING* wybrać odpowiednią oś – ustawić na 2 lub wybrać odpowiednie indeksowanie i nacisnąć przycisk *Apply*.

   Pozostałe składowe są dostępne:
   - Spyder – pole *Index*, wartości 1 i 2,
   - PyCharm – Wycinki `I[:,:,1]` i `I[:,:,2]`,
   - VSCode – pole *Index*.
8. W pracy przydatny może być dostęp do parametrów obrazu:

```
print(I.shape)  # dimensions /rows, columns, depth/
print(I.size)   # number of bytes
print(I.dtype)  # data type
```

   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.

Zadanie 1.2 Wykonaj zadanie, w którym przećwiczysz obsługę plików z wykorzystaniem biblioteki Matplotlib. W celu zachowania pewnego porządku w kodzie, utwórz nowy plik 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)         # create figure
plt.imshow(I)         # add image
plt.title('Mandril') # add title
plt.axis('off')       # disable display of the coordinate system
plt.show()            # display
```

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!
Przy ustalaniu wartości tablicy przecinki są niezbędne (w odróżnieniu od Matlaba). 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*. Proszę zwrócić uwagę na komentarze w poniższym kodzie.

```
from matplotlib.patches import Rectangle # add at the top of the file

fig,ax = plt.subplots(1) # instead of plt.figure(1)

rect = Rectangle((50,50),50,100,fill=False, ec='r'); # ec - edge
    colour
ax.add_patch(rect) # display
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)
```

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

Zadanie 1.3 Wykonaj konwersję przestrzeni barw podanego obrazu.

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

■

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

Przydatna składnia:

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

1.4.2 Matplotlib

Tu wybór dostępnych konwersji jest dość ograniczony.

1. RGB do odcieni szarości. Można wykorzystać rozbicie na poszczególne kanały i wzór:

$$G = 0.299 \cdot R + 0.587 \cdot G + 0.144 \cdot B \tag{1.1}$$

Całość można opakować w funkcję:

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

Wskazówka
Przy wyświetlaniu należy ustawić mapę kolorów. Inaczej obraz wyświetli się w domyślnej, która nie jest bynajmniej w odcieniach szarości: `plt.gray()`

2. RGB do HSV.

```
import matplotlib # add at the top of the file
_HSV = matplotlib.colors.rgb_to_hsv(I)
```

1.5 Skalowanie, zmiana rozdzielczości przy użyciu OpenCV

Zadanie 1.4 Przeskaluj obraz *mandril*. Do skalowania służy funkcja `resize`. ■

Przykład użycia:

```
height, width =I.shape[:2] # retrieving elements 1 and 2, i.e. the
    corresponding height and width
scale = 1.75   # scale factor
Ix2 = cv2.resize(I,(int(scale*height),int(scale*width)))
cv2.imshow("Big_Mandrill",Ix2)
```

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

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

Zadanie 1.5 Wykonaj operacje arytmetyczne na obrazie *lena*.

1. Pobierz ze strony kursu obraz *lena*, a następnie go wczytaj za pomocą funkcji z OpenCV – dodaj ten fragment kodu do pliku, który zawiera wczytywanie obrazu *mandril*. Wykonaj konwersję do odcieni szarości. Dodaj macierze zawierające mandryla i Leny w skali szarości. Wyświetl wynik.
2. Podobnie wykonaj odjęcie i mnożenie obrazów.
3. Zaimplementuj kombinację liniową obrazów.
4. Ważną operacją 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. Proszę obliczyć moduł z różnicy „szarych" wersji mandryla i Leny.

■

Wskazówka
Przy wyświetleniu obraz nie będzie poprawny, ponieważ jest on typu *float64* (*double*). Stosowna konwersja:

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

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) # creates and zeros single-column
    arrays
        height, width =img.shape[:2] # shape - we take the first 2 values
                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] -- input image
# [0] -- for greyscale images there is only one channel
# None -- mask (you can count the histogram of a selected part of the image)
# [256] -- number of histogram bins
# [0 256 ] -- the range over which the histogram is calculated
```

Proszę sprawdzić czy histogramy uzyskane obiema metodami są takie same.

1.8 Wyrównywanie histogramu

Wyrównywanie histogramu to popularna i ważna operacja przetwarzania wstępnego.

1. Wyrównywanie "klasyczne"jest metodą globalną, wykonuje się na całym obrazie. W bibliotece OpenCV znajduje się gotowa funkcja do tego typu wyrównania:

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

2. Wyrównywanie CLAHE (ang. *Contrast Limited Adaptive Histogram Equalization*) – jest to metoda adaptacyjna, która poprawia warunki oświetleniowe na obrazie. Wyrównuje histogram w poszczególnych fragmentach obrazu, a nie dla całego obrazu. Metoda działa następująco:
   - podział obrazu na rozłączne bloki (kwadratowe),
   - wyliczanie histogramu w blokach,
   - wykonanie wyrównywania histogramu, przy czym ogranicza się maksymalną „wysokość" histogramu, a nadmiar re-dystrybuuje na sąsiednie przedziały,
   - interpolacja 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 - maximum height of the histogram bar - values above are
    distributed among neighbours
# tileGridSize - size of a single image block (local method, operates on
    separate image blocks)
I_CLAHE = clahe.apply(IG)
```

Zadanie 1.6 Uruchom i porównaj obie metody wyrównywania. ■

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`)

Wskazówka
Pomocna będzie dokumentacja OpenCV.

Proszę zwrócić uwagę również na inne dostępne funkcje:

- filtrację bilateralną,
- filtry Gabora,
- operacje morfologiczne.