Zaawansowne algorytmy wizyjne

materiały do ćwiczeń laboratoryjnych

Piotr Pawlik, Tomasz Kryjak

PUBLISHED BY AGH

First printing, March 2018

Contents

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 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)
- 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)

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

O informacje związane z zalecanym IDE, a także szczegóły konfiguracyjne proszę pytać Prowadzącego (zależą od konfiguracji oprogramowania w laboratorium).

1.2 Operacje wejścia/wyjścia

1.2.1 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* (z konsoli lub za pomocą ikony). Stwórz nowy plik i zapisz go we własnym katalogu roboczym

[1] gałąź boczna projektu

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). Ponadto uruchamiane spod Pythona okno nie jest automatycznie zamykane, dlatego potrzebne jest jeszcze wywołanie funkcji zamykającej (można też zawołać `destroyWindow` z odpowiednim parametrem).

```
I = cv2.imread('mandril.jpg')
     cv2.imshow("Mandril",I)        # wyswietlenie
    cv2.waitKey(0)                 # oczekiwanie na klawisz
    cv2.destroyAllWindows()        # zamkniecie wszystkich okien
```

5. Uwaga. Okno niekoniecznie musi zostać wyświetlone "na wierzchu".
6. Zapis do pliku realizuje funkcja `imwrite` (proszę zwrócić uwagę na zmianę formatu):

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

7. Obraz wczytywany jest jako macierz. Można to sprawdzić w IDE Spyder, w zakładce *Variable explorer* (narzędzie podobne do *Workspace* znanego z Matlaba). W pracy przydatny może być dostęp do parametrów obrazu:

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

   Funkcja `print` to oczywiści wyświetlanie na konsolę.

1.2.2 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('Mandril')   # dodanie tytulu
plt.axis('off')        # wylaczenie wyswietlania ukladu wspolrzednych
plt.show()             # wyswietlnie calosci
```

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 Matlaba). 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()
```

1.3 Konwersje przestrzeni barw

1.3.1 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.

1.3.2 Matplotlib

Tu wybór dostępnych konwersji jest dość ograniczony.

- RGB do odcienie 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]
```

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)
```

1.4 Skalowanie, zmiana rozdzielczości

Zadanie: przeskalować obraz *mandril*.

1.4.1 OpenCV

Do skalowanie służy funkcja *resize*. Przykład użycia:

```
height, width =I.shape[:2]        # pobranie elementow 1 i 2 tj. odpowienio
    wysokosci i szerokosci
scale = 1.75                      # wspolczynnik skalujacy
Ix2 = cv2.resize(I,(int(scale*height),int(scale*width)))
cv2.imshow("Big Mandril",Ix2)
```

1.4.2 SciPy

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

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

1.5 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" wersją 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. Ważna 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.

1.6 Wyliczenie histogramu

Wyliczanie histogramu można wykonać z wykorzystaniem funkcji `calcHist`. Jednak zanim do tego przejdziemy, przypomnimy sobie podstawowe struktury sterowania w Pythpnie – 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
```

1.7 Wyrównywanie histogramu

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

1.7.1 Wyrównywanie „klasyczne”

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

1.7.2 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 pojedyczncznego bloku obrazu (metoda lokalna, dziala
     na rozdzielnych blokach obrazu)
I_CLAHE = clahe.apply(IG)
```

1.8 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).

1.9 Wczytywanie sekwencji wideo, odejmowanie ramek, indeksacja

W ramach części ćwiczeń będziemy przetwarzać sekwencje wideo. Dlatego warto poznać jak wczytywać kolejne pliki. Przykładem niech będzie proste odejmowanie dwóch ramek, połączone z binaryzacją, indeksacją i analizą otrzymanych obiektów.

1.9.1 Wczytywanie sekwencji

1. Pobierz ze strony kursu zamieszczoną sekwencję.
2. Wczytanie sekwencji umożliwia następujący kod:

```
for i in range(1,200):
        I = cv2.imread('input/in%06d.jpg' % i)
        cv2.imshow("I",I)
        cv2.waitKey(10)
```

Uwaga 1. Zakłada się, że obrazki są w tym samym folderze, co plik źródłowy (w innym przypadku trzeba dodać odpowiednią ścieżkę).
Uwaga 2. Należy użyć funkcji `waitKey`. Inaczej nie nastąpi odświeżenie wyświetlanego obrazka.

1.9.2 Odejmowanie ramek i binaryzacja

Odejmujemy dwie kolejne ramki. Dla uproszczenia rozważań lepiej wcześniej dokonać konwersji do odcieni szarości. Na początku trzeba „jakoś" obsłużyć pierwszą iterację. Przykładowo – wczytać pierwszą ramkę przed pętlą i uznać ją za poprzednią. Później na końcu pętli należy dodać przypisanie ramka poprzednia = ramka bieżąca. Testowo wyświetlamy wyniki odejmowania – powinny być widoczne krawędzie.

Binaryzację można wykonać wykorzystując następującą składnię:

```
B = 1*(D > 10)
```

Uwaga. W takim przypadku `1*` oznacza konwersję z typu logicznego na liczbowy. Osobną kwestią jest poprawne wyświetlenie – trzeba zmienić zakres (pomnożyć przez 255) i dokonać konwersji na *uint8*.

Próg należy dobrać, tak aby obiekty były względnie wyraźne.

Alternatywa to użycie funkcji wbudowanej w OpenCV:

```
B = cv2.threshold(D,10,255,cv2.THRESH_BINARY)
# D - obraz
# 10 - prog
# 255 - co ma byc przypisane na wyjscie jako wartosc maksymalna
# cv2.THRESH_BINARY - typ binaryzacji (tu najprostsza - za obiekty piksele
    powyzej progu)
B = B[1]
```

Uwaga. Druga linia „wydobywa" obraz z krotki, która zwraca funkcja. Pierwszy argument to prób binaryzacji. Jest to użyteczne w przypadku stosowania automatycznego wyznaczenia progu np. metodą Otsu lub trójkątów. Szczegóły w dokumentacji OpenCV.

1.9.3 Operacje morfologiczne

Uzyskany obraz jest dość zaszumiony. Proszę wykonać filtrację wykorzystując erozję i dylatację (`erode` i `dilate` z OpenCV).

Uwaga. Celem tego etapu jest uzyskanie maksymalnie widocznej sylwetki, przy minimalnych zakłóceniach. Dla poprawy efektu warto dodać jeszcze etap filtracji medianowej oraz ew. skorygować próg binryzacji.

1.9.4 Indeksacja i prosta analiza

W kolejnym etapie przeprowadzimy filtrację uzyskanego wyniku. Wykorzystamy indeksację (nadanie numerów dla grup połączonych pikseli) oraz obliczanie parametrów tychże grup. Funkcja `connectedComponentsWithStats`. Wywołanie:

```
retval, labels, stats, centroids =  cv2.connectedComponentsWithStats(BE)
# retval -- ???
# labels -- obraz z indeksami
# stats -- lewy skrajny punkt, gorny skrajny punkt, szerokosc, wysokosc,
    pole
# centroids -- srodki ciezkosci
```

Uwaga. Przy wyświetlaniu trzeba nieco pokombinować, bo i format nie ten i trzeba dodać skalowanie.

Zadanko:
Wyświetlić prostokąt otaczający, pole i indeks dla największego obiektu.

Przykładowe rozwiązanie:

```
retval, labels, stats, centroids = cv2.connectedComponentsWithStats(BE)

cv2.imshow("Labels",np.uint8(labels/stats.shape[0]*255))

if (stats.shape[0] > 1):   # czy sa jakies obiekty
        pi, p = max(enumerate(stats[1:,4]), key=(lambda x: x[1]))
        pi = pi + 1
        # wyrysownie bbox
        cv2.rectangle(I_VIS,(stats[pi,0],stats[pi,1]),(stats[pi,0]+stats[pi
    ,2],stats[pi,1]+stats[pi,3]),(255,0,0),2)
        # wypisanie informacji
        cv2.putText(I_VIS,"%f" % stats[pi,4],(stats[pi,0],stats[pi,1]),cv2.
    FONT_HERSHEY_SIMPLEX,0.5,(255,0,0))
        cv2.putText(I_VIS,"%d" %pi,(np.int(centroids[pi,0]),np.int(centroids
    [pi,1])),cv2.FONT_HERSHEY_SIMPLEX,1,(255,0,0))
```

Komentarze:

- uwaga – rozwiązanie jest przykładowe. Lepsze mile widzane.
- `stats.shape[0]` to liczba obiektów. Ponieważ funkcja zlicza też obiekt o indeksie 0 (tj. tło), w sprawdzeniu czy są obiekty warunek jest > 1
- kolejna linia to obliczenie indeksu maksimum z kolumny numer 5 (pole). Polecenie `enumerate` pozwala „ponumerować” elemnty (nadać im indeksy od 0). Polecenie `key=(lambda x: x[1])`, pozwala wskazać, który element będzie porównywany.
- uzyskany indeks należy inkrementować, ponieważ w analizie pominęliśmy element 0 (tło) – `stats[1:,4]`.
- do rysowania prostokąta otaczającego na obrazie wykorzystujemy funkcję `rectangle` z OpenCV. Składania `"%f" % stats[pi,4]` to taki `printf` „w pigułce” – pozwala wypisać tekst. Kolejne parametry to współrzędne dwóch przeciwległych wierzchołków prostokąta. Następnie kolor w formacie (B,G,R), a na końcu grubość. Szczegóły w dokumentacji funkcji.

- do wypisywanie teksty służy funkcja `putText`. Jako punkt „startowy” podaje się lewy dolny róg tekstu. Następnie czcionka (pełna lista w dokumnetacji), rozmiar i kolor.
- Uwaga. `IMG_VIS` to kopia obrazu wejściowego jeszcze przed konwersją do odcieni szarości.

1.10 Uwagi końcowe

1. W ramach dalszych ćwiczeń będziemy poznawać kolejne algorytmy i funkcjonalnści języka Python. Tym niemniej język ten traktujemy jako wygodne narzędzie, coś pomiędzy Matlabem, a C++ z bonusem, darmowości. Przedmiot ZAW nie jest kursem Pythona !
2. Przedstawione rozwiązania należy zawsze traktować jako przykładowe. Na pewno problem można rozwiązać inaczej, a czas lepiej.
3. W kolejnych ćwiczeniach „poziom szczegółowości” opisu rozwiązania będzie maleć. Zachęcamy zatem do aktywnego korzystania z dokumnetacji do Pythona, OpenCV i po prostu z „wójka Google” – umiejętnie sfomrułowane zapytanie prawie zawsze pozwoli szybko rozwiązań napotkany problem :).