Zaawansowane algorytmy wizyjne

Skrypt do ćwiczeń laboratoryjnych

Piotr Pawlik, Tomasz Kryjak, Mateusz Wąsala

Zespół Wbudowanych Systemów Wizyjnych
Laboratorium Systemów Wizyjnych
WEAIiIB, AGH w Krakowie

Wydanie trzecie, zmienione i poprawione
Kraków, luty 2024

Spis treści

Laboratorium 1

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 pozwalającego na wykonanie operacji pomiędzy elementami dwóch kontenerów (w dogodny i wydajny sposób). 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).

1.3 Operacje wejścia/wyjścia

1.3.1 Wczytywanie, wyświetlanie i zapisywanie obrazu z wykorzystaniem OpenCV

Ćwiczenie 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.

   R 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
cv2.destroyAllWindows()        # close all windows
```

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

[1] gałąź boczna projektu

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.

Rozliczenie ćwiczenia

W celu rozliczenia ćwiczenia, po wykonaniu zadania zgłoś swoją gotowość prowadzącemu zajęcia. Równocześnie możesz zgłosić swoją gotowość po wykonaniu wszystkich zadań dotyczących poruszanego tematu zajęć. Po akceptacji z jego strony, umieść skrypt o rozszerzeniu `.py` lub `.ipynb` w odpowiednim miejscu w zasobach kursu na UPeL.

■

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.

Ćwiczenie 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()
```

Rozliczenie ćwiczenia

W celu rozliczenia ćwiczenia, po wykonaniu zadania zgłoś swoją gotowość prowadzącemu zajęcia. Równocześnie możesz zgłosić swoją gotowość po wykonaniu wszystkich zadań dotyczących poruszanego tematu zajęć. Po akceptacji z jego strony, umieść skrypt o rozszerzeniu `.py` lub `.ipynb` w odpowiednim miejscu w zasobach kursu na UPeL. ■

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)
```

Uwaga! 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

Ćwiczenie 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.

Rozliczenie ćwiczenia

W celu rozliczenia ćwiczenia, po wykonaniu zadania zgłoś swoją gotowość prowadzącemu zajęcia. Równocześnie możesz zgłosić swoją gotowość po wykonaniu wszystkich zadań dotyczących poruszanego tematu zajęć. Po akceptacji z jego strony, umieść skrypt o rozszerzeniu `.py` lub `.ipynb` w odpowiednim miejscu w zasobach kursu na UPeL. ■

Uwaga! 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]
```

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

1.5 Skalowanie, zmiana rozdzielczości przy użyciu OpenCV

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

Rozliczenie ćwiczenia
W celu rozliczenia ćwiczenia, po wykonaniu zadania zgłoś swoją gotowość prowadzącemu zajęcia. Równocześnie możesz zgłosić swoją gotowość po wykonaniu wszystkich zadań dotyczących poruszanego tematu zajęć. Po akceptacji z jego strony, umieść skrypt o rozszerzeniu `.py` lub `.ipynb` w odpowiednim miejscu w zasobach kursu na UPeL. ■

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.

Ćwiczenie 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.

Rozliczenie ćwiczenia
W celu rozliczenia ćwiczenia, po wykonaniu zadania zgłoś swoją gotowość prowadzącemu zajęcia. Równocześnie możesz zgłosić swoją gotowość po wykonaniu wszystkich zadań dotyczących poruszanego tematu zajęć. Po akceptacji z jego strony, umieść skrypt o rozszerzeniu `.py` lub `.ipynb` w odpowiednim miejscu w zasobach kursu na UPeL. ■

Uwaga! 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)
```

Ćwiczenie 1.6 Uruchom i porównaj obie metody wyrównywania.

Rozliczenie ćwiczenia
W celu rozliczenia ćwiczenia, po wykonaniu zadania zgłoś swoją gotowość prowadzącemu zajęcia. Równocześnie możesz zgłosić swoją gotowość po wykonaniu wszystkich zadań dotyczących poruszanego tematu zajęć. Po akceptacji z jego strony, umieść skrypt o rozszerzeniu `.py` lub `.ipynb` w odpowiednim miejscu w zasobach kursu na UPeL. ■

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 będzie dokumentacja OpenCV.

Proszę zwrócić uwagę również na inne dostępne funkcje:

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

Ćwiczenie 1.7 Uruchom i porównaj wymienione metody.

Rozliczenie ćwiczenia
W celu rozliczenia ćwiczenia, po wykonaniu zadania zgłoś swoją gotowość prowadzącemu zajęcia. Równocześnie możesz zgłosić swoją gotowość po wykonaniu wszystkich zadań dotyczących poruszanego tematu zajęć. Po akceptacji z jego strony, umieść skrypt o rozszerzeniu `.py` lub `.ipynb` w odpowiednim miejscu w zasobach kursu na UPeL. ■

II Laboratorium 2

2. Detekcja pierwszoplanowych obiektów ruchom

Co to są obiekty pierwszoplanowe ?

Takie, które są dla nas (dla rozważanej aplikacji) interesujące. Zwykle: ludzie, zwierzęta, samochody (lub inne pojazdy), bagaże (potencjalne bomby). Zatem definicja jest ściśle związana z docelową aplikacją.

Czy segmentacja obiektów pierwszoplanowych to segmentacja obiektów ruchomych ?

Nie. Po pierwsze, obiekt, który się zatrzymał, nadal może być dla nas interesujący (np. stojący przed przejściem dla pieszych człowiek). Po drugie, istnieje cały szereg ruchomych elementów sceny, które nie są dla nas interesujące. Przykłady to płynąca woda, fontanna, poruszające się drzewa i krzaki itp. Należy również zauważyć, że zwykle obiekty ruchome są brane pod uwagę podczas przetwarzania obrazu. Zatem detekcję obiektów pierwszoplanowych można i warto wspomagać detekcją obiektów ruchomych.

Najprostsza metodą detekcji obiektów ruchomych to przeprowadzenie odejmowania kolejnych (sąsiednich) ramek. W ramach ćwiczenia zrealizujemy proste odejmowanie dwóch ramek, połączone z binaryzacją, indeksacją i analizą otrzymanych obiektów. Na koniec spróbujemy uznać wynik odejmowania jako rezultat segmentacji obiektów pierwszoplanowych i sprawdzimy jakość tej segmentacji.

2.1 Wczytywanie sekwencji obrazów

Ćwiczenie 2.1 Wczytywanie sekwencji obrazów

1. Ściągnij odpowiednią sekwencję testową z platformy `UPeL`. W ćwiczeniu skupimy się na sekwencji *pedestrians*. Wykorzystywane sekwencje pochodzą ze zbioru danych ze strony changedetection.net. Zbiór zawiera poetykietowane sekwencje, tzn. każda ramka

obrazu posiada maskę obiektów - maskę referencyjną (ang. *ground truth*). Na nich każdy z pikseli został przydzielony do jednej z pięciu kategorii: tło (0), cień (50), poza obszarem zainteresowania (85), nieznany ruch (170) oraz obiekty pierwszoplanowe (255) – w nawiasach podano odpowiadające poziomy szarości.
W ramach ćwiczenia interesować nas będzie tylko podział na obiekty pierwszoplanowe oraz pozostałe kategorie. Dodatkowo w folderze znajduje się maska obszaru zainteresowania (ROI) oraz plik tekstowy z przedziałem czasowym dla którego należy analizować wyniki (temporalROI.txt) – szczegóły w dalszej części ćwiczenia.

2. Wczytanie sekwencji umożliwia następujący, przykładowy kod:

```
for i in range(300,1100):
  I = cv2.imread('input/in%06d.jpg' % i)
  cv2.imshow("I",I)
  cv2.waitKey(10)
```

Zakłada się, że sekwencja została rozpakowana w tym samym folderze, co plik źródłowy (w innym przypadku trzeba dodać odpowiednią ścieżkę).

Należy użyć funkcji `waitKey`. Inaczej nie nastąpi odświeżenie wyświetlanego obrazka.

3. Do pętli dodaj opcję analizy co i-tej ramki – wykorzystywana funkcja *range* może mieć jako trzeci parametr: *krok*. Poeksperymentuj z jego wartością teraz (przy wyświetlaniu filmu) i później (przy detekcji obiektów ruchomych).

2.2 Odejmowanie ramek i binaryzacja

W celu detekcji elementu ruchomych, odejmujemy dwie kolejne ramki. Należy zaznaczyć, że dokładnie chodzi nam o obliczenie modułu z różnicy.

Uwaga! Aby uniknąć problemów z odejmowaniem liczb bez znaku (*uint8*), należy wykonać konwersję do typu *int* – `IG = IG.astype('int')`.

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 głównie 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* (możemy być potrzebny import biblioteki `numpy`).

Próg należy dobrać, tak aby obiekty były względnie wyraźne. Proszę zwrócić uwagę na artefakty związane z kompresją.

Alternatywa to użycie funkcji wbudowanej w OpenCV:

```
(T, thresh) = cv2.threshold(D,10,255,cv2.THRESH_BINARY)
```

```
# D -- input array
# 10 -- threshold value
# 255 -- maximum value to use with the THRESH_BINARY and THRESH_BINARY_INV
    thresholding types
# cv2.THRESH_BINARY -- thresholding type
# T - our threshold value
# thresh - output image
```

R Pierwszy argument zwracanych wartości przez funkcję `cv2.threshold` to próg binaryzacji (jest on użyteczny w przypadku stosowania automatycznego wyznaczenia progu np. metodą Otsu lub trójkątów). Szczegóły w dokumentacji OpenCV.

2.3 Operacje morfologiczne

Ćwiczenie 2.2 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 (przed morfologią) oraz ew. skorygować próg binaryzacji.

2.4 Indeksacja i prosta analiza

W kolejnym etapie przeprowadzimy filtrację uzyskanego wyniku. Wykorzystamy indeksację (nadanie etykiet dla grup połączonych pikseli) oraz obliczanie parametrów tychże grup. Służy do tego funkcja `connectedComponentsWithStats`. Wywołanie:

```
retval, labels, stats, centroids =  cv2.connectedComponentsWithStats(B)
# retval --  total number of unique labels
# labels -- destination labeled image
# stats -- statistics output for each label, including the background label.
# centroids -- centroid output for each label, including the background label.
```

R Przy wyświetlaniu obrazu *labels* trzeba w odpowiedni sposób ustawić format oraz dodać skalowanie.
Do skalowania można wykorzystać informację o liczbie znalezionych obiektów.

```
cv2.imshow("Labels", np.uint8(labels / retval * 255))
```

Następnie wyświetl prostokąt otaczający, pole i indeks dla największego obiektu. Poniżej znajduje się przykładowe rozwiązanie zadania. Proszę go uruchomić i ewentualnie spróbować go zoptymalizować.

```
I_VIS = I # copy of the input image

if (stats.shape[0] > 1):    # are there any objects

  tab = stats[1:,4] # 4 columns without first element
  pi = np.argmax( tab )# finding the index of the largest item
  pi = pi + 1 # increment because we want the index in stats, not in tab
  # drawing a 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)
# print information about the field and the number of the largest element
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 do przykładowego rozwiązania:

- `stats.shape[0]` to liczba obiektów. Ponieważ funkcja zlicza też obiekt o indeksie 0 (tj. tło), w sprawdzeniu czy są obiekty warunek jest > 1
- kolejne dwie linie to obliczenie indeksu maksimum z kolumny numer 4 (pole).
- uzyskany indeks należy inkrementować, ponieważ w analizie pominęliśmy element 0 (tło).
- do rysowania prostokąta otaczającego na obrazie wykorzystujemy funkcję `rectangle` z OpenCV. Składnia `''%f'' % stats[pi,4]` pozwala wypisać wartość w odpowiednim formacie (f - float). Kolejne parametry to współrzędne dwóch przeciwległych wierzchołków prostokąta. Następnie kolor w formacie (B, G, R), a na końcu grubość linii. Szczegóły w dokumentacji funkcji.
- do wypisywania tekstu na obrazie służy funkcja `putText`. Do określenia pozycji pola tekstowego podaje się współrzędne lewego dolnego wierzchołka. Następnie czcionka (pełna lista w dokumentacji), rozmiar i kolor.

  Uwaga! `I_VIS` to obraz wejściowy jeszcze przed konwersją do odcieni szarości.

Ćwiczenie 2.3 Wykorzystaj funkcję `cv2.connectedComponentsWithStats` do indeksacji oraz oblicz parametry otrzymanych obiektów. Wyświetl prostokąt otaczający, pole oraz indeks dla największego obiektu. Na podstawie wskazówek i przykładów zamieszczonych w tekscie powyżej spróbuj zoptymalizować rozwiązanie. ■

2.5 Ewaluacja wyników detekcji obiektów pierwszoplanowych

Aby ocenić, i to najlepiej w miarę obiektywnie, algorytm detekcji obiektów pierwszoplanowych należy wyniki przez niego zwracane tj. maskę obiektów porównać z maską referencyjną (ang. *groundtruth*). Porównywanie odbywa się na poziomie poszczególnych pikseli. Jeśli wykluczy się cienie (a tak założyliśmy na wstępie) to możliwe są cztery sytuacje:

- TP – wynik prawdziwie dodatni (ang. *true positive*) – piksel należący do obiektu z pierwszego planu jest wykrywany jako piksel należący do obiektu z pierwszego planu,
- TN – wynik prawdziwie ujemny (ang. *true negative*) – piksel należący do tła jest wykrywany jako piksel należący do tła,
- FP – wynik fałszywie dodatni (ang. *false positive*) – piksel należący do tła jest wykrywany jako piksel należący do obiektu z pierwszego planu,
- FN – wynik fałszywie ujemny (ang. *false negative*) – piksel należący do obiektu jest wykrywany jako piksel należący do tła.

Na podstawie wymienionych współczynników można policzyć szereg miar. My wykorzystamy trzy: precyzję (ang. *precision* - P), czułość (ang. *recall* - R) i tzw. $F1$. Zdefiniowane są one następująco:

$$P = \frac{TP}{TP+FP} \tag{2.1}$$

$$R = \frac{TP}{TP+FN} \tag{2.2}$$

$$F1 = \frac{2PR}{P+R} \tag{2.3}$$

Miara F1 jest z zakresu [0;1], przy czym im jej wartość jest większa, tym lepiej.

Ćwiczenie 2.4 Zaimplementuj obliczanie miar P, R i $F1$.

1. W pierwszym kroku należy zdefiniować globalne liczniki TP, TN, FP, FN, a po skończonej pętli obliczyć żądane wartości P, R i $F1$.
2. Do pętli należy dodać wczytywanie maski referencyjnej – analogicznie jak obrazka. Jest ona dostępna w folderze *groundtruth*.
3. Kolejny krok to porównanie maski obiektów i referencyjnej. Najprostsze rozwiązanie, czyli pętla `for` po całym obrazie, w każdej iteracji sprawdzamy podobieństwo pikseli. Oczywiście nie jest to zbyt wydajne podejście. Można spróbować wykorzystać możliwości Python'a w realizacji operacji na macierzach. Utwórz odpowiednie warunki logiczne np. dla TP:

```
TP_M = np.logical_and((B == 255),(GTB == 255)) # logical product of the
    matrix elements
TP_S = np.sum(TP_M)         # sum of the elements in the matrix
TP = TP + TP_S              # update of the global indicator
```

   Przy czym obliczenia wykonujemy tylko wtedy, gdy dostępna jest poprawna mapa referencyjna. W tym celu należy sprawdzić zależność licznika ramki i wartości z pliku `temporalROI.txt` – musi się on zawierać w zakresie tam opisanym. Przykładowy kod wczytujący zakres (przy okazji proszę zwrócić uwagę na sposób obsługi plików tekstowych):

```
f = open('temporalROI.txt','r')         # open file
line = f.readline()                     # read line
roi_start, roi_end = line.split()       # split line
roi_start = int(roi_start)              # conversion to int
roi_end = int(roi_end)                  # conversion to int
```

4. Przeprowadź obliczenia dla metody odejmowania kolejnych klatek. Zwróć uwagę na wartość $F1$.
5. Wykonaj obliczenia dla pozostałych dwóch sekwencji – *highway* i *office*.

Rozliczenie ćwiczenia

W celu rozliczenia ćwiczenia, po wykonaniu zadania zgłoś swoją gotowość prowadzącemu zajęcia. Równocześnie możesz zgłosić swoją gotowość po wykonaniu wszystkich zadań dotyczących poruszanego tematu zajęć. Po akceptacji z jego strony, umieść skrypt o rozszerzeniu `.py` lub `.ipynb` w odpowiednim miejscu w zasobach kursu na UPeL.

■

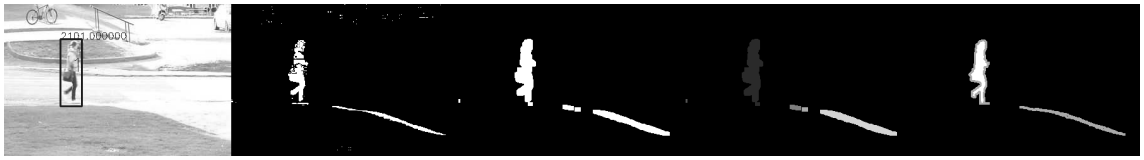
R Informacje dotyczące kolejnych zajęć.

1. W ramach dalszych ćwiczeń będziemy poznawać kolejne algorytmy i funkcjonalności języka Python. Jednak nie kładziemy szczególnej uwagi na poznawanie języka Python, ale na wykorzystaniu w praktyce kolejnych algorytmów pojawiających się na wykładach. Przedmiot Zaawansowane Algorytmy Wizyjne nie jest kursem Pythona!
2. Przedstawione rozwiązania należy zawsze traktować jako przykładowe. Na pewno problem można rozwiązać inaczej, a czasem lepiej.

3. W kolejnych ćwiczeniach poziom szczegółowości opisu rozwiązania będzie maleć. Zachęcamy zatem do aktywnego korzystania z dokumentacji do Pythona, OpenCV i materiałów/tutoriali znajdujących się w internecie :).

2.6 Przykładowe rezultaty algorytmu do detekcji ruchomych obiektów pierwszoplanowych

W celu lepszej weryfikacji poszczególnych etapów zaproponowanej metody do detekcji ruchomych obiektów pierwszoplanowych przedstawiony zostanie przykładowy wynik dla obrazu o indeksie 350.



Rysunek 2.1: Przykładowy wynik poszczególnych etapów algorytmu. Od lewej obraz wejściowy z prostokątem otaczającym, obraz po binaryzacji, obraz po zastosowaniu filtracji medianowej oraz operacji morfologicznych (erode, dilate), obraz reprezentujący etykiety po indeksacji, obraz referencyjny.