Zaawansowne algorytmy wizyjne

materiały do ćwiczeń laboratoryjnych

Piotr Pawlik, Tomasz Kryjak

PUBLISHED BY AGH

First printing, March 2018

Contents

1 — Detekcja narożników

1.1 Cel zajęć

- zapoznanie z zagadnieniem wykrywania punktów charakterystycznych
- implementacja prostego algorytmu detekcji - metoda Harrisa
- implementacja wyszukiwania punktów charakterystycznych w obrazach o różnej skali ('piramidzie' obrazów)

UWAGA: W dzisiejszym ćwiczeniu do wyświetlania obrazów proszę używać funkcji z matplotlib.pyplot a nie z opencv.

1.2 Detekcja narożników metodą Harrisa – teoria

Detekcja narożników metodą Harrisa polega na wyszukiwaniu pikseli dla których moduł gradientu pionowego i poziomego ma znaczną wartość.

Metoda opisana jest za pomocą następujących zależności:

$$H(x,y) = det(M(x,y)) - k * trace^2(M(x,y)) \tag{1.1}$$

gdzie: det – wyznacznik macierzy, $trace$ – ślad macierzy, k – stała, (x,y) – lokalizacja na obrazie.

Macierz autokorelacji M to:

$$M(x,y) = \begin{bmatrix} < I_x^2(x,y) > & < I_{xy}(x,y) > \\ < I_{xy}(x,y) > & < I_y^2(x,y) > \end{bmatrix} \tag{1.2}$$

gdzie poszczególne symbole oznaczają:

$$< I_x^2(x,y) >= I_x^2(x,y) \otimes h(x,y) \tag{1.3}$$

$$< I_y^2(x,y) >= I_y^2(x,y) \otimes h(x,y) \tag{1.4}$$

$$< I_{xy}(x,y) >= I_x I_y(x,y) \otimes h(x,y) \tag{1.5}$$

gdzie: $I_x(x,y)$ oraz $I_y(x,y)$ są pochodnymi cząstkowymi w kierunku x i y wartości (wyznaczone np. za pomocą np. gradientu Sobela), symbol $\otimes$ oznacza splot (konwolucję), a $h(x,y)$ to funkcja Gaussa:

$$h(x,y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1.6}$$

gdzie: σ – odchylenie standardowe. Filtracja Gaussa jest używana celem redukcji wpływu szumu, który szczególnie mocno wypływa na obliczanie wartości gradientu.

Wyznacznik i ślad macierzy 2×2 (dla przypomnienia):

$$det(M(x,y)) = < I_x(x,y)^2 >< I_y(x,y)^2 > - < I_{xy}(x,y) >^2 \tag{1.7}$$

.

$$trace(M(x,y)) = < I_x(x,y)^2 > + < I_y(x,y)^2 > \tag{1.8}$$

.

Ostatecznie dany piksel jest klasyfikowany jako narożnik jeśli wyliczona wartość $R(x,y)$ jest większa niż określony próg.

Jako końcową filtrację warto wykorzystać wyszukiwanie lokalnych maksimów. Za kandydata na narożnik uznajemy tylko te piksele, które są lokalnymi maksimami w swoim otoczeniu (np. 7×7).

1.3 Implementacja metody Harrisa

Na podstawie powyższej teorii proszę zaimplementować metodę Harrisa.

1. Ze strony kursu pobierz archiwum z danymi do ćwiczenia i rozpakuj je we własnym katalogu roboczym.
2. Wczytaj obrazy 'fontanna1.jpg' oraz 'fontanna2.jpg'.
3. Zaimplementuj funkcję wyliczającą wartość H z macierzy autokorelacji. Funkcja jako parametry powinna otrzymywać obraz w odcieniach jasności oraz rozmiar masek filtrów Sobela i Gaussa przez nią wykorzystywanych (może to być taki sam rozmiar dla obu filtrów). Przefiltruj obraz pionowym i poziomym filtrem Sobela (funkcja cv2.Sobel) - będzie to realizacja pochodnej obrazu po x i y. Jako typ wyniku Sobela proszę podać cv2.CV_32F. Wylicz odpowiednie iloczyny pochodnych kierunkowych i rozmyj je za pomocą filtru Gaussa (funkcja cv2.GaussianBlur - przykładowe wywołanie: cv2.GaussianBlur(image, (size, size), 0), gdzie size to rozmiar maski filtra). Rozmycie filtrem Gaussa odpowiada wyliczeniu ważonej sumy elementów z otoczenia każdego punktu macierzy (ważonej krzywą Gaussa). Do obrazu wynikowego wpisz wartości H (z wzoru wykorzystującego wyliczone wyznaczniki i ślady). Przyjmij wartość współczynnika k = 0.05. Zwróć obraz wynikowy (warto go znormalizować np. do zakresu 0-1 celem łatwiejszego doboru progu w następnym punkcie)
4. Wykorzystaj poniższą funkcję znajdującą maksima lokalne w otrzymanej jako parametr tablicy:

```
import scipy.ndimage.filters as filters

def find_max(image, size, threshold) : # size - rozmiar maski filtra
    maksymalnego
    data_max = filters.maximum_filter(image, size)
```

```
    maxima = (image==data_max)
    diff = image>threshold
    maxima[diff == 0] = 0
    return np.nonzero(maxima)
```

Maksima są tu wyszukiwane z użyciem filtra maksymalnego. Zastosowana metoda jest uproszczona (może zwrócić kilka leżących w obrębie maski takich samych maksimów), ale na nasze potrzeby wystarczająca. Funkcja zwraca znalezione maksima w postaci list współrzędnych - jedna lista dla każdego wymiaru. Dla tablicy 2D będą to listy kolejno: wpółrzędnych y i współrzędnych x. Funkcja działa także dla tablic 3D - wtedy zwróci trzy listy, przy czym pierwsza lista to współrzędne z (czyli np. wysokość w oktawie piramidy).

5. Dla obu wczytanych obrazów zastosuj powyższe funkcje (druga ma wyszukiwać maksima lokalne w wyniku zwracanym przez pierwszą). Jako rozmiar masek w obu funkcjach możesz przyjąć 7. Wyniki z drugiej funcji wyświetl jako znaki (np. *) naniesione na obrazy początkowe. Warto w tym celu stworzyć osobną funkcję rysującą, która utwozy figure, wyświetli obraz za pomocą plt.imshow, a następnie naniesie na niego znaki przy użyciu funkcji plt.plot. Przykładowo plt.plot(5, 10, '*', color='r') wyrysuje czerwoną gwiazdkę we współrzędnych (x,y) równych (5,10). Za pomocą tej funkcji można jednocześnie wyrysować kilka gwiazdek: plt.plot([1,2,3], [4,5,6], '*') wyrysuje trzy niebieskie gwiazdki we współrzędnych (x,y) równych (1,4), (2,5) i (3,6). Listy [1,2,3] i [4,5,6] można uzyskać z drugiel funkcji - ale uwaga na kolejność współrzędnych!
6. Sprawdź 'naocznie' czy wykrywane punkty na obu obrazach znajdują się (mniej więcej) w tych samych położeniach (czyli czy detektor jest powtarzalny)
7. Powtórz operacje z powyższych punktów dla obrazów 'budynek1.jpg' i 'budynek2.jpg'

1.4 Wyszukiwanie najlepszego dopasowania w obrazach różniących się skalą (nieobowiązkowe)

1. W nowym skrypcie napisz funkcję, która otrzymany obraz rozmyje wieloktrotnie filtrem Gaussa z rosnącą wartością σ - każde kolejne rozmycie ma mieć σ k razy większą od poprzedniego. Wyniki kolejnych rozmywań należy odejmować od siebie i dołączać do wynikowej tablicy obrazów różnicowych. Lokalne ekstrema w całej tablicy 3D wskażą na punkty charakterystyczne z uwzględnieniem skali w której występują.
   Funkcja jako parametry, oprócz obrazu niech otrzyma liczbę rozmyć, początkową wartość σ, k. Funkcja ma zwrócić tablicę obrazów różnicowych (DoG - Diference of Gaussians). Początek tej funkcji może wyglądać następująco:

```
def pyramid(image, blur_nbr, k, sigma):
    res_shape=(blur_nbr, image.shape[0], image.shape[1])
    res_img = np.zeros(res_shape)

    fimage = np.float64(image)
    prev_img = cv2.GaussianBlur(fimage,(0,0),sigmaX=sigma, sigmaY=sigma
)
    # dalej powinna byc petla wypelniajaca res_img
```

2. Wczytaj obrazy 'fontanna1.jpg' oraz 'fontanna_pow.jpg'. Dla obu wczytanych obrazów zastosuj napisaną funkcje (sugerowane parametry - liczba rozmyć pierwszego obrazu - 5, drugiego obrazu 10; początkowa wartość $\sigma = 1.6$, k = 1.26). W oparciu o funkcję find_max wyszukiwaj ekstrema lokalne w rezultatach napisanej funkcji.
   Proszę wyrysować znalezione ekstrema osobno dla każdej ze skal. Do wyrysowania wyników można użyć funkcji rysującej z poprzedniego zadania - należy tylko z list ekstremów uzyskać współrzędne punktów z jednej skali. Można to prosto zrobić korzystając

z własności macierzy w numpy - można je 'filtrować' tablicami boolowskimi zawierającymi True w pozycjach, które chcemy pozostawić. Przykładowo, jeżeli tablica **extrem** to wynik find_max, to:

```
x = extrem[2] # wspolrzedne x we wszystkich skalach
x[extrem[0]==i] # jako wspolrzedne x tylko w skali i
```

Czy jesteś w stanie stwierdzić o ile 'skal' różnią się oba obrazy?

3. Sprawdź jak sobie radzi z takimi obrazami Harris. Uruchom funkcje z poprzedniego zadania dla powyższych 2 obrazów.

Bibliography