# Zaawansowne algorytmy wizyjne

materiały do ćwiczeń laboratoryjnych

Piotr Pawlik, Tomasz Kryjak

PUBLISHED BY AGH

*First printing, March 2018*

# Contents

# 1 — Śledzenie obiektów

## 1.1 Mean-shift

W ramach ćwiczenia zaimplementujemy prostą wersję śledzenia obiektu o zadanym kolorze z wykorzystaniem algorytmu *mean-shift*.

### 1.1.1 Inicjalizacja

Pierwszym etapem jest inicjalizacja algorytmu. Potrzebujemy jądro i jego gradient (ang. *kernel*). Możemy założyć, że wykorzystamy dwuwymiarowy rozkład Gaussa. Ustalmy jego rozmiar na kwadrat 75.

W Python (chyba?) nie ma funkcji wprost zwracającej dwuwymiarowy rozkład Gaussa. Można go uzyskać stosując kod:

```
import math                                          # do PI
from mpl_toolkits.mplot3d import Axes3D              # do 3D

# Generowanie Gaussa

kernel_size = 75                                     # rozmiar rozkladu
sigma = 10                                  # odchylenie std
x = np.arange(0, kernel_size, 1, float)      # wektor poziomy
y = x[:,np.newaxis]                         # wektor pionowy
x0 = y0 = kernel_size // 2                           # wsp. srodka
G = 1/(2*math.pi*sigma**2) * np.exp(-0.5 * ((x-x0)**2 + (y-y0)**2) / sigma
    **2)

# Rysowanie
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x, y, G, color='b')
plt.show()
```

Proszę zwrócić uwagę na „dodatkowe” biblioteki.

Kolejny krok to obliczenie pochodnych jądra. Należy wykorzystać funkcję `numpy.diff`. Obliczamy dwie pochodne – po „x” i „y”.

```
# Pochodne
G_y = np.diff(G,1,0);
```

```
G_y = np.append(G_y,np.zeros((1,kernel_size),float),0)     # dodanie
    dodatkowego wiersza
G_y = -G_y
G_x = np.diff(G,1,1);
G_x = np.append(G_x,np.zeros((kernel_size,1),float),1)     # dodanie
    dodatkowej kolumny
G_x = -G_x
```

Dodatkowe linie kodu odpowiadają za dodanie wiersza/kolumny, tak aby macierz pozostała kwadratowa oraz zmianę znaku (wynika z zastosowanego wzoru).

### 1.1.2 Wybór obiektu do śledzenia

Obiekt, który będziemy śledzić należy wskazać – ręcznie. Po pierwsze pobieramy ze strony kursu sekwencję testową – przelot śmigłowca TOPR. Po drugie wczytujemy wybraną ramkę (proponowana to 100). Po trzecie pobieramy współrzędne prostokąta. Uwaga, poniższe rozwiązanie jest „działające”, co nie znaczy, że najlepsze.

```
kernel_size = 45                                # rozmiar rozkladu

def track_init(event,x,y,flags,param):
        global mouseX,mouseY
        if event == cv2.EVENT_LBUTTONDBLCLK:
                cv2.rectangle(I, (x-kernel_size//2, y- kernel_size//2), (x +
     kernel_size, y + kernel_size), (0, 255, 0), 2)
                mouseX,mouseY = x,y

# Wczytanie pierwszego obrazka
I = cv2.imread('seq/track00100.png')
cv2.namedWindow('Tracking')
cv2.setMouseCallback('Tracking',track_init)

# Pobranie klawisza
while(1):
        cv2.imshow('Tracking',I)
        k = cv2.waitKey(20) & 0xFF
        if k == 27:    # ESC
                break
```

Uwaga. Dla potrzeb testów wygodniej jest „tymczasowo” ustawić współrzędne na stałe. Wybór obiektu za każdym uruchomieniem aplikacji bywa irytujący oraz czasochłonny.

### 1.1.3 Inicjalizacja histogramu – funkcji gęstości prawdopodobieństwa wzorca

Po pierwsze dokonujemy przypisania do współrzędnych obiektu np. `xS = mouseX-kernelSize//2`. Po drugie konwertujemy obraz wejściowy do HSV `I_HSV = cv2.cvtColor(I, cv2.COLOR_BGR2HSV)`. W tym miejscu warto wyświetlić składową H obrazu i zobaczyć czy nasz obiekt się jakoś wyróżnia (pewnie okaże się, że mniej niż byśmy się spodziewali).

Następnie obliczamy histogram z uwzględnieniem wag. Najprościej – dwie pętle for po odpowiednim fragmencie obrazu. Przykładowy kod:

```
I_H = I_HSV[:,:,0]
hist_q = np.zeros((256,1),float)
for jj in range(0,kernel_size):
        for ii in range (0,kernel_size):
                pixel_H = I_H[yS+jj,xS+ii];
                hist_q[pixel_H] = hist_q[pixel_H] + pixel_H * G[jj,ii]
```

### 1.1.4 Śledzenie właściwe

Mając wszystko przygotowane możemy zacząć śledzić nasz obiekt. Na początek jako bieżące położenie obiektu $(xC, yC)$ przyjmuje to startowe $(xS, yS)$ Następnie, w pętli `for` wczytujemy kolejne ramki z sekwencji, zmieniamy na HSV oraz wybieramy składową H.

Dalej, w ramach pętli `for` wykonujemy kolejne iteracje algorytmu (np. 10 lub 20). W ramach iteracji:

- obliczamy histogram aktualnego obszaru (należy umiejętnie skopiować wykorzystany wcześniej kod).
- obliczamy współczynnik Bhattacharyya (bez sumowania):

$$\rho(y) = \sum_{u=1}^{m} \sqrt{(p_u(y)q_u)} \tag{1.1}$$

  Kod: `rho = np.sqrt(hist_q*hist_p)`
- obliczamy przesunięcie na podstawie wzoru:

$$y_1 = \frac{\sum_{i=1}^{N} x_i w_i g(||y - x_i||^2)}{\sum_{i=1}^{N} w_i g(||y - x_i||^2)} \tag{1.2}$$

  Uwaga. Osobno dla „x” i „yy”.

```
dx_l = 0;
dx_m = 0;
dy_l = 0;
dy_m = 0;

for jj in range(0,kernel_size):
        for ii in range (0,kernel_size):
                dx_l = dx_l + ii*rho[I_H[yC+jj,xC+ii]]*G_x[jj,ii]
                dx_m = dx_m + ii*G_x[jj,ii]
                dy_l = dy_l + jj*rho[I_H[yC+jj,xC+ii]]*G_y[jj,ii]
                dy_m = dy_m + jj*G_y[jj,ii]

dx = dx_l/dx_m
dy = dy_l/dy_m

# Obliczanie nowych wspolrzednych
xC = np.int(np.floor(xC + dx))
yC = np.int(np.floor(yC + dy))
```

- na koniec warto oczywiście dodać wizualizację sposobu działania algorytmu.

Rozwiązanie powinno działać, aczkolwiek czasem śledzenie „gubi” obiekt. Można poeksperymentować z rozmiarem okna i parametrem sigma. Znaczenie ma też punkt startowy metody.