Cyfrowe przetwarzanie sygnałów i obrazów



Autor:

Paweł Biel 225949

Prowadzący:

Dr inż. Jacek Cichosz

# Wstęp

W ramach laboratorium „Cyfrowe przetwarzanie sygnałów i obrazów” przygotowałem trzy skrypty, w języku Python w wersji 3.6, o nazwach:

* counting.py (tutaj program wymaga 2 argumentów, aby uruchomić skrypt  
  -i <ścieżka pliku wejściowego> -o <ścieżka pliku wyjściowego>)
* lab.py
* hist.py

# Skrypt pierwszy – counting.py

Skrypt ma za zadanie policzyć ziarenka ryżu ze zdjęcia ryze.png

Kod skryptu:

import numpy as np  
import argparse  
import imutils  
import cv2  
  
# zdeklarowanie wymaganych argumentów dla danego skryptu  
ap = argparse.ArgumentParser()  
ap.add\_argument("-i", "--image", required=True,  
 help="path to the input image")  
ap.add\_argument("-o", "--output", required=True,  
 help="path to the output image")  
args = vars(ap.parse\_args())  
  
# licznik znalezionych elementów  
counter = 0  
  
# wczytanie zdjęcia do programu (sciezka podana w argumencie skryptu)  
image\_orig = cv2.imread(args["image"])  
height\_orig, width\_orig = image\_orig.shape[:2]  
  
# wyjsciowe zdjecie z konturami  
image\_contours = image\_orig.copy()  
  
# zdeklarowanie ilosci kolorów  
colors = ['1', '2']  
for color in colors:  
  
 # zrobienie kopi oryginalnego zdjęcia  
 image\_to\_process = image\_orig.copy()  
 image\_to\_process2 = image\_orig.copy()  
  
 # zdefiniowanie tablic NumPy granic kolorów (wektory GBR)  
 if color == '1':  
 lower = np.array([60, 100, 20])  
 upper = np.array([240, 240, 210])  
  
 elif color == '2':  
 # odwrócenie kolorów obrazu  
 image\_to\_process = (255 - image\_to\_process)  
 lower = np.array([50, 50, 40])  
 upper = np.array([100, 120, 80])  
  
 # poszukiwanie kolorów w określonych granicach  
 image\_mask = cv2.inRange(image\_to\_process, lower, upper)  
 # zastosowanie maski  
 image\_res = cv2.bitwise\_and(image\_to\_process, image\_to\_process, mask=image\_mask)  
  
 # wczytanie obrazu, przekonwertowanie go na skalę szarości i rozmycie  
 image\_gray = cv2.cvtColor(image\_res, cv2.COLOR\_BGR2GRAY)  
 image\_gray = cv2.GaussianBlur(image\_gray, (5, 5), 0)  
  
 # wykonanie wykrywania krawędzi, wykonanie dylatacji + erozja, aby zamknąć szczeliny między krawędziami obiektu  
 image\_edged = cv2.Canny(image\_gray, 50, 100)  
 image\_edged = cv2.dilate(image\_edged, None, iterations=1)  
 image\_edged = cv2.erode(image\_edged, None, iterations=1)  
  
 # znalezienie kontur w mapie krawędzi  
 cnts = cv2.findContours(image\_edged.copy(), cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)  
 cnts = cnts[0] if imutils.is\_cv2() else cnts[1]  
  
 for c in cnts:  
 # jeśli kontur nie jest wystarczająco duży, zignoruj go (<5)  
 if cv2.contourArea(c) < 5:  
 continue  
  
 # wyliczenie otoczki wypukłej skończonego zbioru punktów płaszczyzny dla kontur  
 # funkcja korzysta algorytmu Grahama dla otoczki wypukłej  
 hull = cv2.convexHull(c)  
  
 # wyrysowanie konturów na kopi obrazu  
 cv2.drawContours(image\_contours, [hull], 0, (0, 255, 0), 1)  
 counter += 1  
  
# wypisanie ilosci obiektó  
print("{} obiektów".format(counter))  
# zapisannie obrazu z konturami  
cv2.imwrite(args["output"], image\_contours)

Lista kroków programu:

* Zdefiniowanie tablic kolorów
* Poszukiwanie kolorów w określonych granicach

Wczytanie obrazu, przekonwertowanie go na skalę szarości i rozmycie

* Wykonanie wykrywania krawędzi, wykonanie dylatacji + erozja, aby zamknąć szczeliny między krawędziami obiektu
* Znalezienie kontur w mapie krawędzi
* Dla każdej znalezionej kontury sprawdzenie ilości krawędzi i wykorzystanie algorytmu Grahama dla otoczki wypukłej
* Wypisanie wyników i zapisanie zmodyfikowanego obrazu

## O algorytmie Grahama

**Algorytm Grahama**  to efektywny [algorytm](https://pl.wikipedia.org/wiki/Algorytm) wyszukiwania [otoczki wypukłej](https://pl.wikipedia.org/wiki/Otoczka_wypuk%C5%82a) skończonego zbioru punktów płaszczyzny; nie istnieją warianty dla przestrzeni o wyższych wymiarach. Pomysłodawcą algorytmu jest Ronald Graham.

Czasowa [złożoność obliczeniowa](https://pl.wikipedia.org/wiki/Z%C5%82o%C5%BCono%C5%9B%C4%87_obliczeniowa) wynosi {\displaystyle O(n\log n)}O(n log n).

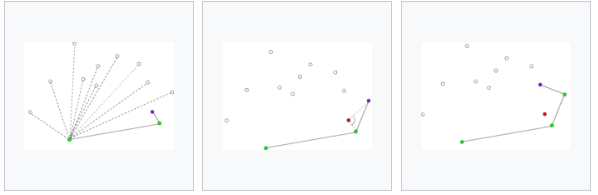
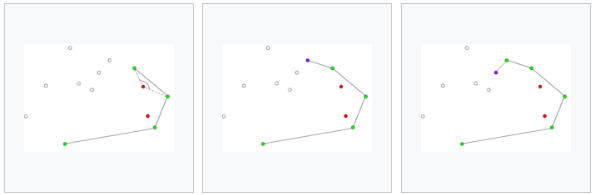
Algorytm przebiega następująco:

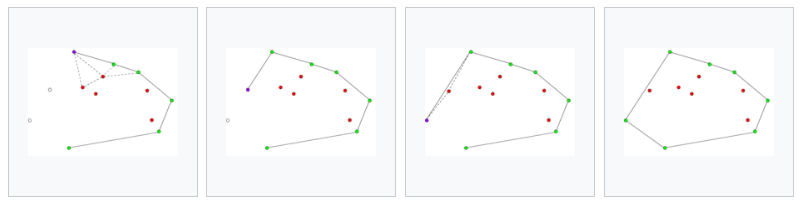
1. Wybierz punkt (ozn. O) o najniższej wartości współrzędnej *y*.
2. Przesuń wszystkie punkty tak, by punkt O pokrył się z początkiem układu współrzędnych.
3. Posortuj punkty [leksykograficznie](https://pl.wikipedia.org/wiki/Porz%C4%85dek_leksykograficzny) względem:
   * kąta pomiędzy wektorem {\displaystyle OP\_{i}}OPi a dodatnią osią układu współrzędnych,
   * odległości punktu {\displaystyle P\_{i}}Pi od początku układu współrzędnych.
4. Wybierz punkt (ozn. S) o najmniejszej współrzędnej y; jeśli kilka punktów ma tę samą współrzędną y, wybierz spośród nich ten  
   o najmniejszej współrzędnej x.
5. Przeglądaj listę posortowanych punktów poczynając od punktu S:
   * Od bieżącej pozycji weź trzy kolejne punkty (ozn. A, B, C).
   * Jeśli punkt B leży na zewnątrz [trójkąta](https://pl.wikipedia.org/wiki/Tr%C3%B3jk%C4%85t) AOC, to może należeć do otoczki wypukłej. Przejdź do następnego punktu na liście.
   * Jeśli punkt B leży wewnątrz trójkąta AOC, to znaczy, że nie należy do otoczki. Usuń punkt B z listy i cofnij się o jedną pozycję (o ile bieżąca pozycja jest różna od początkowej).

Ze względu na wcześniejsze kroki algorytmu (sortowanie i sposób wybierania analizowanych trójek punktów) dwa z trzech warunków należenia punktu B do trójkąta AOC są spełnione, tj. B leży po „wewnętrznej” względem trójkąta stronie prostych OA i OC. Zatem do stwierdzenia przynależności do trójkąta wystarczy zbadać, po której stronie odcinka AC znajduje się punkt B, w tym celu wykorzystuje się znak iloczynu wektorowego {\displaystyle |C-A|\times |B-A|}|C – A| x |B – A|.

W praktyce można również utożsamić krok 4. z 1., tzn. przyjąć, że {\displaystyle O=S} O=S.

Przykładowy przebieg algorytmu:



# Skrypt drugi – lab.py

Skrypt ma za zadanie wypisać podstawowe parametry zdjęcia, wyświetlić obraz, wyświetlić obraz z zakresu maski oraz wyświetlić histogram. Aby uzyskać pożądany wynik użyłem odpowiednio funkcji wbudowanych w bibliotekę OpenCV.

Wyniki, które zwraca skrypt:

Wysokość: 512,

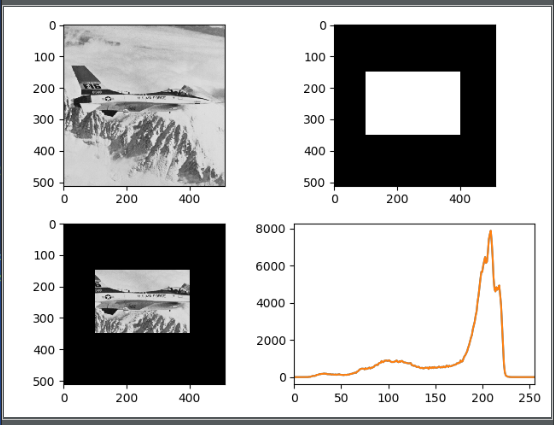
Szerokość: 512,

Ilosc pikseli: 786432,

Ilosc kanałów: 3,

Typ danych obrazu: uint8,

Typ pliku: tiff



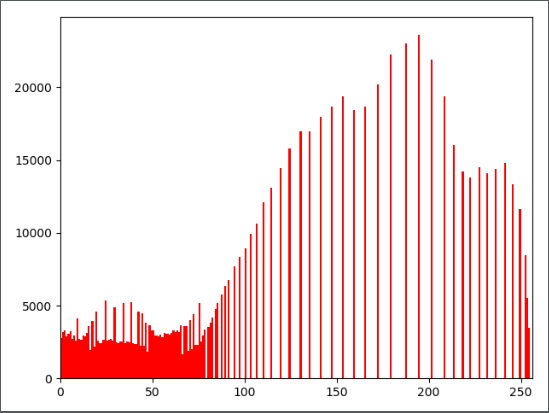
Kod programu:

import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
import imghdr  
  
# wczytanie obrazu  
path = "../standard\_test\_images/jetplane.tif"  
img = cv2.imread(path)  
  
cv2.imshow("tytul okna", img) # stworzenie okna z obrazkiem  
  
cv2.waitKey() # ważna funkcja!!!  
cv2.destroyAllWindows() # zamknięcie wszystkich okien  
# Funkcja waitKey() odpowiada za obsługę okienek (ich rysowanie/odświeżanie/interakcję z użytkownikiem).  
# Funkcja zwraca kod klawisza który został naciśnięty.  
  
  
height, width, channels = img.shape # zczytanie parametrów wysokosc, szerokosc ilosc kanałów  
print("Wysokość: %s,\nSzerokość: %s,\nIlosc pikseli: %s,\nIlosc kanałów: %s,\nTyp danych obrazu: %s,\nTyp pliku: %s"  
 % (height, width, img.size, channels, img.dtype, imghdr.what(path)))  
  
# stworzenie maski  
mask = np.zeros(img.shape[:2], np.uint8)  
mask[150:350, 100:400] = 255  
masked\_img = cv2.bitwise\_and(img, img, mask=mask)  
  
# Wyliczenie histogramu bez użytej maski  
hist\_full = cv2.calcHist([img], [0], None, [256], [0, 256])  
hist\_mask = cv2.calcHist([img], [0], mask, [256], [0, 256])  
hist, bins = np.histogram(img.flatten(), 256, [0, 256])  
  
# wyświetlenie paru obrazków na jednym "plot'cie"  
plt.subplot(221), plt.imshow(img, 'gray')  
plt.subplot(222), plt.imshow(mask, 'gray')  
plt.subplot(223), plt.imshow(masked\_img, 'gray')  
plt.subplot(224), plt.plot(hist\_full), plt.plot(hist\_full)  
plt.xlim([0, 256])  
  
plt.show()

# Skrypt trzeci – hist.py

Skrypt maskuje tablice do danej wartości, warto zwrócić uwagę na to, że histogram we wcześniejszym skrypcie leży w jaśniejszym regionie. Dlatego też ten skrypt wykorzystuje pełne spectrum. Do tego potrzeba funkcji transformacji, która mapuje piksele wejściowe w jaśniejszym regionie, aby wyprowadzać piksele w pełnym obszarze. Następnie znajdujemy minimalną wartość histogramu (z wyłączeniem 0) i stosujemy równanie wyrównania histogramu.

Wynik programu:



Kod programu:

import cv2  
import numpy as np  
from matplotlib import pyplot as plt  
  
path = "../standard\_test\_images/jetplane.tif"  
img = cv2.imread(path)  
  
hist, bins = np.histogram(img.flatten(), 256, [0, 256])  
cdf = hist.cumsum()  
  
# maskowanie tablicy do danej wartosci  
# Możesz zobaczyć histogram leży w jaśniejszym regionie.  
# Potrzebujemy pełnego spektrum. Do tego potrzebujemy funkcji transformacji,  
# która mapuje piksele wejściowe w jaśniejszym regionie, aby wyprowadzać piksele w pełnym obszarze  
cdf\_m = np.ma.masked\_equal(cdf, 0)  
# Teraz znajdujemy minimalną wartość histogramu (z wyłączeniem 0) i stosujemy równanie wyrównania histogramu  
cdf\_m = (cdf\_m - cdf\_m.min()) \* 255 / (cdf\_m.max() - cdf\_m.min())  
cdf = np.ma.filled(cdf\_m, 0).astype('uint8')  
cdf\_normalized = cdf \* hist.max() / cdf[-1]  
  
img = cdf[img]  
plt.hist(img.flatten(), 256, [0, 256], color="r")  
plt.xlim([0, 256])  
plt.show()