Instytut Informatyki UP Kraków

Przetwarzanie obrazów cyfrowych :: projekt zaliczeniowy

SPRAWOZDANIE

Autor: Rafał Kita

18.01.2019

1 CEL I ZAKRES PROJEKTU

Celem projektu było napisanie programu, który umożliwia zliczanie ilości monet na obrazie wejściowym oraz policzenie sumy ich nominałów. Dla każdej monety dodatkowo należało wyliczyć odpowiednie współczynniki.

2 OPROGRAMOWANIE

W programie użyłem następujących bibliotek:

- numpy
- warnings
- matplotlib
- skimage
- cv2
- scipy
- time
- math

Funkcje wykorzystane w projekcie z bibliotek:

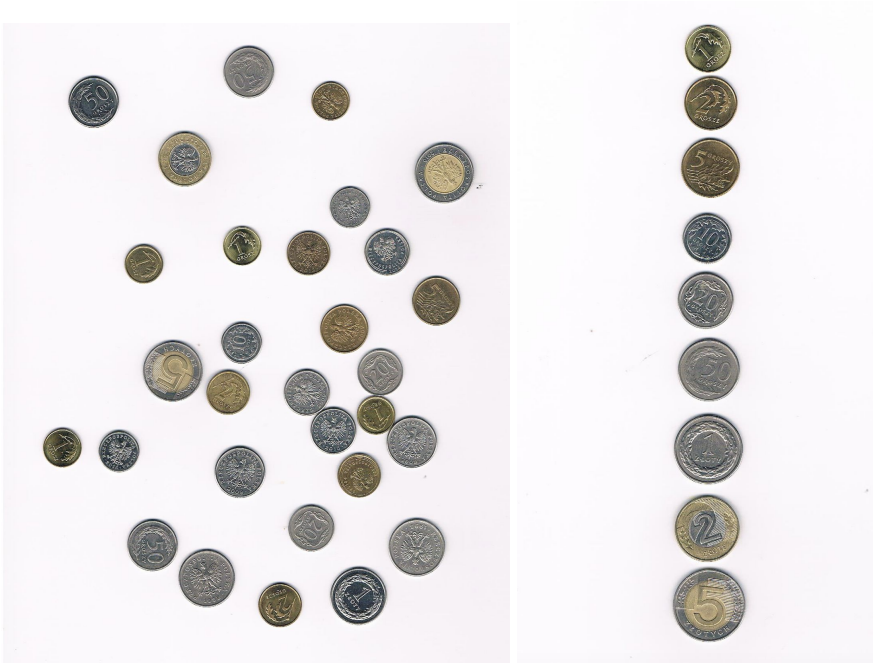
- plt.figure()
- plt.imshow()
- plt.axis()
- plt.show()
- io.imread()
- color.rgb2gray()
- cv2.blur()
- cv2.threshold()
- np.ones()
- morphology.remove_small_holes()
- ndi.binary_fill_holes()
- cv2.erode()
- cv2.dilate()
- distance.euclidean()
- math.sqrt()

Funkcje wykorzystane w projekcie stworzone przez autora:

- showImage(src img) - wyświetla obraz.
- import_image(src img) - importuje obraz wejściowy i podaje go obróbce przed segmentacją.
- split(src img_bin, tablica index, wysokość, szerokość, początek x, poczatek y, aktualny index) - poddaje obraz binarny segmentacji przez rekurencje.
- split_first() - funkcja po segmentacji porządkuje indeksy .
- countiong() - funkcja zlicza ile dane obiekty na obrazie posiadają pikseli.
- show_countion() - Wyświetla spis obiektów razem z sumą pikseli.
- my_coins() - funkcja wczytuje dziewięć monet i poddaje je obróbce, by potem poddać je porównaniu z elementali z obrazu.
- count_valus() - za pomocą funkcji segmentacji (split_first()) i my_coins() możemy teraz policzyć nominały funkcji.
- procent_object_image() - funkcja zlicza jaki procent powierzchni obrazu stanowią obiekty.
- getFigure() - funkcja zlicza nam ilość pikseli w danym obiekcie.
- cog2() - funkcja zlicza środek ciężkości obiektu.
- computeBB() - funkcja zlicza współczynnik Blair-Bliss.
- computeFeret() - funkcja zlicza współczynnik Ferenta.

3 DANE WEJŚCIOWE

Danymi wejściowymi są skany monet np.:



Monety zostały ułożone w dowolnym stylu. Skany monet otrzymały jasne tło, dzięki czemu łatwiej jest oddzielić elementy od siebie.

4 OPIS ALGORYTMU

Na początku programu musiałem ustawić zdjęcie, które chcę poddać badaniom mojego programu. w drugim wierszu wywołałem funkcję import_image(img), która poddaje obraz binaryzacji, dylatacji i erozji, a potem zwraca obraz wchodzący i wychodzący. W trzecim wierszu znajduje się funkcja showImage(image), która wyświetla wchodzące zdjęcie.

```
img = 'k01.jpg'
image, im = import_image(img)
showImage(image)
```

Funkcja import_image(img), w pierwszym wierszu jest przypisywana ścieżka obrazu do zmiennej, a w kolejnym wierszy za pomocą funkcji bibliotecznej io.imread(path + img), łączymy ścieżkę z nazwą pliku. W kolejnym wierszu za pomocą funkcji bibliotecznej color.rbg2gray(image), przekształcamy obraz wchodzący na obraz szary. Następnie obraz poddajemy przekształceniu wartości na int, za pomocą funkcji wbudowanej o nazwie img_as_ubyte(image_gray). Następnie nasz obraz poddajemy rozmyciu z maską 3x3 za pomocą funkcji bibliotecznej cv2.blur(image_byte,(3,3)), a wierszu niżej obraz jest poddany binaryzacji z progami 177 i 155 za pomocą funkcji cv2.threshold(). W kolejnych wierszach z obrazu są usuwane małe obiekty za pomocą funkcji morphology.remove_small_holes() z maską 5x5. w następnych wierszach obiekty są wypełniane za pomocą funkcji ndi.binary_fill_holes(), a następnie obraz jest konwertowany przez funkcje już nam znaną image_as_ubyte(), by następnie poddać je erozji i dylatacji za pomocą funkcji cv2.erode() i cv2.dilate() z maską 5x5, a na końcu naszej funkcji jest zwracany obraz wejściowy i obraz po przekształceniach.

```
def import_image(img):
    path = '../images/'
    image = io.imread(path + img)
    image_gray = color.rgb2gray(image)

    image_byte = img_as_ubyte(image_gray)
    image_blur = cv2.blur(image_byte, (3,3))
    th, image_byte = cv2.threshold(image_blur, 170, 255, cv2.THRESH_BINARY_INV)

    kernel = np.ones((5,5),np.uint8)
    coins_cleand_holes = morphology.remove_small_holes(image_byte, 255)

    fill_coins = ndi.binary_fill_holes(coins_cleand_holes)
    fill_coins = img_as_ubyte(fill_coins)

    kernel = np.ones((5,5),np.uint8)
    a = 0
    jump = fill_coins
    while a <= 1:
        jump = cv2.erode(jump,kernel,iterations = 10)
        jump = cv2.dilate(jump,kernel,iterations = 6)
        a+=1
    dilation = cv2.dilate(jump,kernel,iterations = 2)
    # showImage(dilation)
    return image, dilation
```

Funkcja showImage() wyświetla obraz na ekranie komputera za pomocą funkcji bibliotecznych plt.figure, plt.imshow(), plt.axis() i plt.show().

```
def showImage(image):
    plt.figure(figsize=(100,100))
    plt.imshow(image, cmap="gray")
    plt.axis('off')
    plt.show()
```

Następnie w bloku startowym jest tworzona tablica dla przyszłych indeksów z segmentacji.

```
len_y = len(im)
len_x = len(im[0])
index = [[0 for y in range(len_y)] for x in range(len_y)]
```

Poniżej jest wywoływana funkcja split_first(im, index, len(im), len(im[0])), która zwraca tab - czyli tablice indeksów po segmentacji obrazu wejściowego, oraz ilość obiektów na obrazie.

```
tab, place = split_first(im, index, len(im[1]), len(im))

# print(place)
```

Funkcja split_first() zajmuje się uporządkowaniem indeksów w tablicy indeksów po segmentacji, zatem funkcja je uzupełnia do postaci uporządkowanej i zmienia indeksy na mniejsze, gdy w ciągu indeksów wykryje indeks bez zastosowania.

```
def split_first(im, index, len_x, len_y):
    index, place = split(im, index, len_x, len_y)
    all_index = []

    for y in range(len_y):
        for x in range(len_x):
            all_index.append(index[y][x])

    all_index = list(set(all_index))
    len_index = len(all_index)
    for i in range(len_index):
        for y in range(len_y):
            for x in range(len_x):
                if index[y][x] == all_index[i]:
                    index[y][x] = i

    return index, len_index
```

Kolejnym etapem jest funkcja split(), która sprawdza, czy w podanym przedziale wszystkie piksele są takie same, jeśli nie to jest dzielona na cztery części i każda część jest wykonywana przez rekurencję.

```
def split(im, index, size_x, size_y, start_x=0, start_y=0, place=1):
    status = 1
    tab_index = []

    for y in range(start_y, size_y):
        for x in range(start_x, size_x):
            if im[start_y][start_x] != im[y][x]:
                status = 0

            if im[y][x] == im[y][x-1] and x > 0 and x == start_x:
                if index[y][x-1] > 0:
                    tab_index.append(index[y][x-1])

            if im[y][x] == im[y-1][x] and y > 0 and y == start_y:
                if index[y-1][x] > 0:
                    tab_index.append(index[y-1][x])

    if status == 1:

        tab_index = list(set(tab_index))
        tab_index.sort()
        now = place

        if len(tab_index) > 0:
            now = tab_index[0]
        for y in range(start_y, size_y):
            for x in range(start_x, size_x):
                index[y][x] = now

        if len(tab_index) > 1:
            for i in range(len(tab_index)):
                if i > 0:
                    for y in range(len(im)):
                        for x in range(len(im[0])):
                            if index[y][x] == tab_index[i]:
                                index[y][x] = tab_index[0]

        place += 1

    else:

        new_x = int(((size_x - start_x)/2) + start_x)
        new_y = int(((size_y - start_y)/2) + start_y)

        tab1, place = split(im, index, new_x, new_y, start_x, start_y, place)
        tab2, place = split(im, index, size_x, new_y, new_x, start_y, place)
        tab3, place = split(im, index, new_x, size_y, start_x, new_y, place)
        tab4, place = split(im, index, size_x, size_y, new_x, new_y, place)

        table = tab1 + tab2 + tab3 + tab4

    return index, place
```

Następna funkcja countiong(), zwraca ilość pikseli ile dany obiekt posiada.

```
def countiong(tab, place, image):
    count_tab = []
    for i in range(place):
        count = 0
        for y in range(len(image)):
            for x in range(len(image[1])):
                if tab[y][x] == i:
                    count += 1
                    if i > 0:
                        image[y][x] = [10*i,5*i,6*i]

        count_tab.append(count)
    return count_tab, image
```

Następna funkcja to count_value(), która zlicza nominały monet i zwraca sumę monet.

```
def count_values(coins_from_image, coins):
    suma = 0
    for a in range(0, len(coins_from_image)):
        if coins_from_image[a] < 44000:
            find_min = []
            for b in range(len(coins)):
                find_min.append(abs(coins_from_image[a] - coins[b]))
        #         print("ok")

            tmp = min(find_min)
            for c in range(len(coins)):
                if abs(coins_from_image[a] - coins[c]) == tmp:
                    moneta = c
                    break

            if moneta == 0:
                suma += 1
            elif moneta == 1:
                suma += 2
            elif moneta == 2:
                suma += 5
            elif moneta == 3:
                suma += 10
            elif moneta == 4:
                suma += 20
            elif moneta == 5:
                suma += 50
            elif moneta == 6:
                suma += 100
            elif moneta == 7:
                suma += 200
            elif moneta == 8:
                suma += 500

    return str(suma/100) + " zł"
```

Na koniec są liczone współczynniki, a po wyliczeniu są wyświetlane.

```
Obraz nr:  k01.jpg
Ilosc obiektow na obrazie: 46
Obiekty zajmuja powierzchnie obrazu w: 8%
Wartosc monet na obrazie wynosi: 4.7 zł
Obiect  1
--> Liczba punktow:  6961
--> Srodek ciezkosci:  [227.94742134750754, 646.0613417612412]
--> Blair-Bliss:  0.9988409687852937
--> Ferent:  1.0454545454545454
Obiect  2
--> Liczba punktow:  7104
--> Srodek ciezkosci:  [410.4166666666667, 916.9669200450451]
--> Blair-Bliss:  0.999241737398679
--> Ferent:  1.0333333333333334
Obiect  3
--> Liczba punktow:  7016
--> Srodek ciezkosci:  [555.1875712656785, 1202.6141676168756]
--> Blair-Bliss:  0.9990628767007067
--> Ferent:  1.0337078651685394
```

5 WYNIKI EKSPERYMENTALNE

L.P	nazwa obrazu	ilość wykrytych monet	skuteczność (%)	Ilość poprawnych wykrytych nominałów	skuteczność (%)
1	k01.jpg	46/46	100%	46/46	100%
2	k02.jpg	34/34	100%	34/34	100%
3	k03.jpg	7/7	100%	7/7	100%
4	k04.jpg	7/7	100%	7/7	100%
5	k05.jpg	7/7	100%	7/7	100%
6	k06.jpg	7/7	100%	7/7	100%
7	k07.jpg	6/6	100%	6/6	100%
8	k08.jpg	5/5	100%	5/5	100%
9	k09.jpg	20/20	100%	20/20	100%
10	k10.jpg	36/36	100%	36/36	100%
11	k11.jpg	26/26	100%	26/26	100%
12	k12.jpg	30/30	100%	30/30	100%
13	k13.jpg	9/9	100%	9/9	100%
14	k14.jpg	2/2	100%	2/2	100%

15	k15.jpg	1/1	100%	1/1	100%

6 WNIOSKI

W przypadku 15 zdjęć, które użyłem w w/w programie, program zliczył ilość monet bezbłędnie i tak samo sytuacja się tyczy w rozpoznawaniu nominałów. Program bezbłędnie działa na w/w obrazach.