Instytut Informatyki UP Kraków

Przetwarzanie obrazów cyfrowych :: projekt zaliczeniowy

SPRAWOZDANIE

Autor: **Wojciech Fornagiel**

21 stycznia 2019

1 CEL I ZAKRES PROJEKTU

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

2 OPROGRAMOWANIE

W programie używam następujących bibliotek:

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

Funkcje wykorzystane w projekcie:

- warnings.filterwarnings()
- color.rgb2gray()
- np.invert()
- cv2.blur()
- np.zeros_like()
- cv2.threshold()
- cv2.distanceTransform()
- np.uint8()
- cv2.dilate()
- cv2.morphologyEx()
- plt.figure()
- plt.imshow()
- plt.suptitle()
- plt.show()
- np.bincount()

- distance.euclidean()
- math.sqrt()
- cv2.findContours()

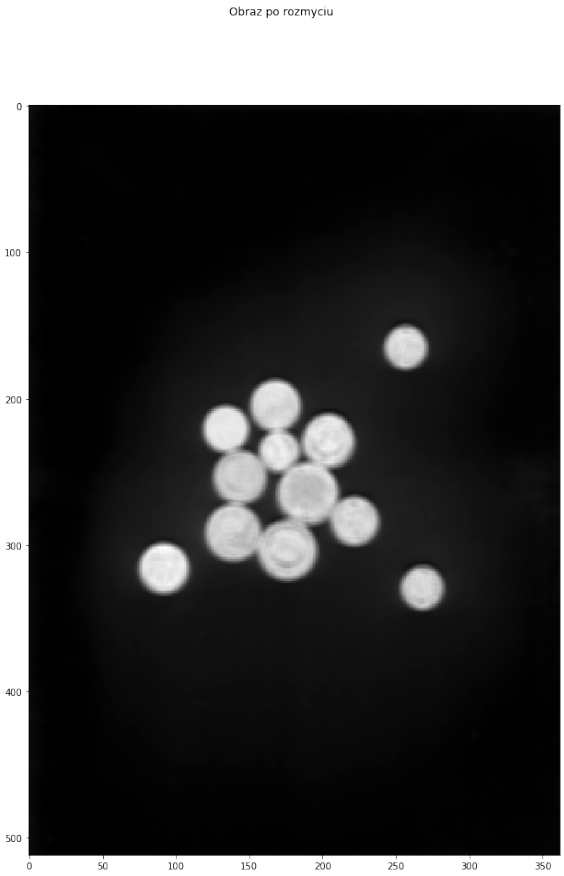
3 DANE WEJŚCIOWE

Danymi wejściowymi są skany monet np.:

Użyłem skanów ze względu na stałą wielkość monet na wszystkich obrazach co pozwala na dokładniejsze zliczanie nominałów. Dodatkowo monety są na białym tle co pozwala na łatwiejsze oddzielenie ich od otoczenia.

4 OPIS ALGORYTMU

Pierwszą rzeczą, którą robię po załadowaniu obrazu jest jego zmniejszenie, ponieważ skany są w bardzo dużej rozdzielczości i zdecydowanie wydłużało to czas wykonywania programu. Następnie Zamieniam obraz na odcienie szarości i wykonuję na nim operację rozmycia w celu wyeliminowania nierówności na poszczególnych monetach. Tak wygląda obraz po wstępnym rozmyciu:

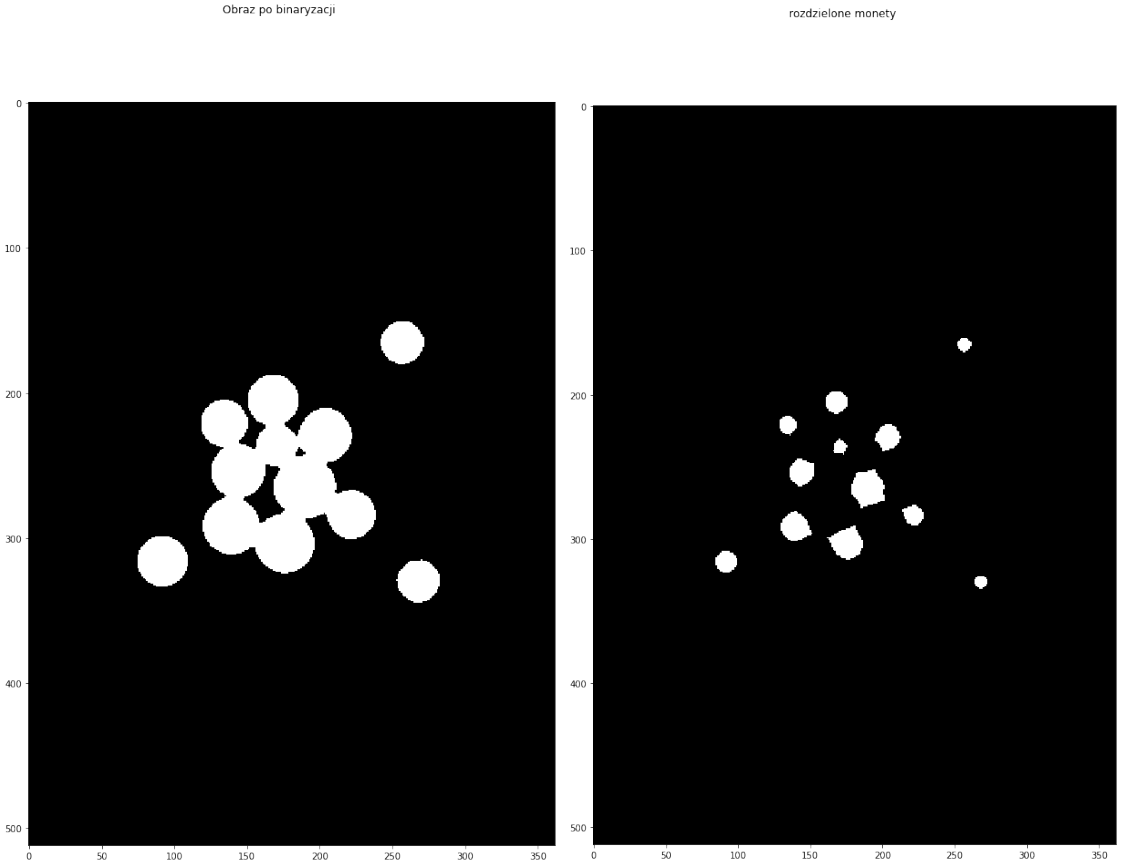



```
url ='zdjecia/8.jpg'
im = io.imread(url)
im = cv2.resize(im,(362,512))

#showimg(im,title="Obraz oryginalny")

im2 = color.rgb2gray(im)
im2 = img_as_ubyte(im2)
im2 = np.invert(im2)
im2 = cv2.blur(im2, (5,5))
#showimg(im2,title="Obraz po rozmyciu")
```

Następnym krokiem jest binaryzacja obrazu za pomocą funkcji cv2.threshold() i zastosowanie funkcji cv2.distanceTransform() na wyniku binaryzacji. Następnie stosuję ponownie binaryzację w celu rozdzielenia obiektów. Wynik binaryzacji i rozdzielenia obiektów:



```
th = 150
th,bim = cv2.threshold(im2, thresh=th, maxval=255, type=cv2.THRESH_OTSU)

#showimg(bim,title="Obraz po binaryzacji")

element = np.ones((3,3),np.uint8)

dt = cv2.distanceTransform(bim, distanceType=cv2.DIST_L2, maskSize=3)
ret, fg = cv2.threshold(dt,0.5*dt.max(),255,0)

#showimg(dt,"Odległość od środka")

#wyznaczanie części będącej na pewno monetą
fg = np.uint8(fg)

#showimg(fg,"rozdzielone monety")
```

Następnie stosuję na obrazie dylatację oraz operację otwarcia w celu wygładzenia krawędzi i łatwiejszego procesu segmentacji. Tworzę też kopię tabeli obrazu, która jest wypełniona zerami, aby móc w niej zapisać odpowiednie numery obiektów.

Na przetworzonym obrazie stosuję napisaną przeze mnie segmentację przez rozrost z dodatkową funkcją, która usuwa obiekty mniejsze niż 50, które są wynikiem nierówności obiektów.

```
def del_empty_obj(tab_obj,last_obj):
    x=0
    global coin_count
    while x <= last_obj:
        tab2.append(x)
        x+=1
    for i in range(tab_obj.shape[0]):
        for j in range(tab_obj.shape[1]):
            if(tab_obj[i][j] != 0):
                tab2[tab_obj[i][j]] += 1
    for y in range(len(tab2)):
        if(tab2[y]<50):
            tab2[y] = 0
        if(tab2[y]!=0):
            coin_count += 1

def region_growing(im,tab_obj):
    obj=1
    last_obj=obj
    for i in range(im.shape[0]):
        for j in range(im.shape[1]):
            if(im[i][j]!=0):
                if(im[i][j]==im[i][j-1] or im[i][j]==im[i-1][j] or im[i][j]==im[i][j+1] or im[i][j]==im[i+1][j] or im[i][j]==im[

                    #sprawdzanie czy pixele obok są w jakimś obiekcie
                    if(im[i][j]==im[i][j-1]):
                        obj = tab_obj[i][j-1]
                    if(im[i][j]==im[i-1][j]):
                        obj = tab_obj[i-1][j]
                    if(im[i][j]==im[i-1][j-1]):
                        obj = tab_obj[i][j-1]
                    if(im[i][j]==im[i-1][j+1] and tab_obj[i-1][j+1]>0):
                        obj = tab_obj[i-1][j+1]

                    if(obj>last_obj):
                        last_obj = obj

                    if(tab_obj[i][j] == 0 and im[i][j] != 0):
                        a = i
                        b = j
                        while im[a][b] != 0:
                            if(im[a][b]==im[a-1][b]):
                                obj = tab_obj[a-1][b]
                                break
                            if(im[a][b]==im[a-1][b+1] and tab_obj[a-1][b+1]>0):
                                obj = tab_obj[a-1][b+1]
                                break
                            b=b+1


                    tab_obj[i][j] = obj


                    if(im[i][j+1]==0):
                        obj = last_obj+1

    del_empty_obj(tab_obj,last_obj)
```

Segmentacja sprawdza czy którykolwiek z pikseli obok jest tego samego koloru i nie jest czarny. Jeśli tak to sprawdza czy którykolwiek z pikseli na górze lub po lewej ma już przydzielony jakiś numer obiektu. Jeżeli tak to dany piksel przyjmuje tą wartość, a jeżeli nie to w pętli, która idzie do momentu napotkania tła sprawdzam czy sąsiedzi kolejnych pikseli mają jakiś numer obiektu. Jeżeli tak to nasz pierwszy piksel, który omawialiśmy dostaje ten numer obiektu. Po dojściu do tła numer obiektu jest zwiększany.

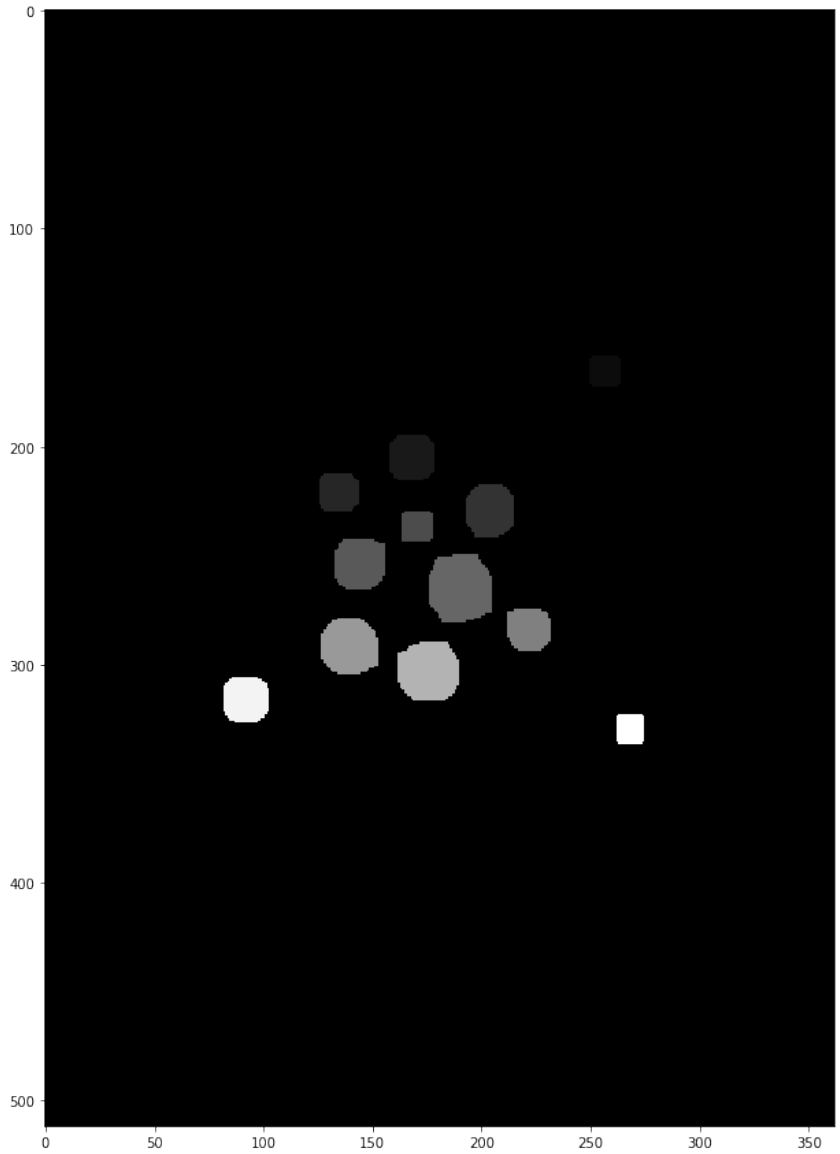
Następnie jest wywoływania druga funkcja usuwająca nieznaczące obiekty. Dodatkowo w niej następuje zliczenie ilości obiektów.

```
#wyrównanie obiektów
seg = cv2.dilate(fg,kernel=element,iterations=3)
seg = cv2.morphologyEx(seg, op=cv2.MORPH_OPEN, kernel=element, iterations=5)

#segemntacja
region_growing(seg,tab_obj)

print(f"Wykryto {coin_count} monet")
```

Wynik segmentacji



Po segmentacji wypisywana jest informacja z ilością monet:

```
Wykryto 12 monet
```

Aby policzyć wartości monet używam skanów pojedynczych monet z każdej wartości i przeprowadzam na nich te same operacje co na obrazie wejściowym. Daje mi to przybliżony rozmiar każdej wartości który następnie poprawiam ręcznie w celu zwiększenia efektywności.

Następnie za pomocą funkcji np.bincount() i np.ravel() jestem w stanie uporządkować wielkości obiektów do prostej tablicy jednowymiarowej o długości liczby obiektów na obrazie.

Potem porównuję wielkości obiektów do wielkości poszczególnych nominałów i zliczam ich wartość:

```
while i < len(sz):
    if sz[i]>=385:
        value+=500
        total[0] += 1
#         print("5 zl")
    elif sz[i]>=330 and sz[i]<385:
        value+=100
        total[2] += 1
#         print("1 zl")
    elif sz[i]>=280 and sz[i]<330:
        value+=200
        total[1] += 1
#         print("2 zl")
    elif sz[i]>=215 and sz[i]<280:
        value+=50
        total[3] += 1
#         print("50 gr")
    elif sz[i]>=170 and sz[i]<215:
        value+=5
        total[6] += 1
#         print("5 gr")
    elif sz[i]>=125 and sz[i]<170:
        value+=20
        total[4] += 1
#         print("20 gr")
    elif sz[i]>=110 and sz[i]<125:
        value+=2
        total[7] += 1
#         print("2 gr")
    elif sz[i]>70 and sz[i]<110:
        value+=10
        total[5] += 1
#         print("10 gr")
    elif sz[i]<=70 and sz[i]>20:
        value+=1
        total[8] += 1
#         print("1 gr")
#     else:
#         print("Skip")
    i+=1
```

Wartość monet dzielę przez 100 i dostaję wynik w złotówkach:

```
Ilosc monet ulozona wartosciami od 5zl w dol:
[1, 1, 1, 2, 1, 1, 2, 1, 2]

Suma : 9.44 zl
```

Aby obliczyć jaki % obrazu stanowią monety sprawdzam ilokrotnie średnio są zmniejszane obiekty. Dzięki temu mogę pomnożyć wielkość obiektu razy tą wartość i podzielić przez ogólną wielkość obrazu. Po podzieleniu przez 100 daje to przybliżony % jaki oryginalna moneta zajmuje na obrazie.

```
def area_percent(tab):
    i = 1
    T = []
    while i < len(tab):
        if tab[i] != 0:
            T.append(((tab[i]*3)/fg.size)*100)
        i += 1
    return T
```

Aby wyliczyć środek geometryczny obrazu biorę współrzędne wszystkich pikseli danego obiektu i wyliczam z nich średnią.

Dla współczynnika Fereta wyliczam maksymalną szerokość i wysokość obiektu i następnie dzielę je przez siebie.

```
def center(im,obj_nr,obj,tab):
    x = 0
    y = 0
    licznik = 0
    for i in range(im.shape[0]):
        for j in range(im.shape[1]):
            if obj_nr[i][j] == obj:
                x += i
                y += j
                licznik +=1
    x = x/licznik
    y = y/licznik
    tab.append(x)
    tab.append(y)
    return tab
```

```
def Feret(im,obj_nr,obj):
    max_x = 0
    min_x = 362
    x = 0
    max_y = 0
    min_y = 512
    y = 0
    for i in range(im.shape[0]):
        for j in range(im.shape[1]):
            if obj_nr[i][j] == obj:
                if i < min_y:
                    min_y = i
                if i > max_y:
                    max_y = i
                if j < min_x:
                    min_x = j
                if j > max_x:
                    max_x = j
    x = max_x - min_x
    y = max_y - min_y

    return x/y
```

Do wyliczenia współczynnika Blaira-Blissa potrzebuję wcześniej tablicę ze współrzędnymi wszystkich pikseli w danym obiekcie. Po zrobieniu tego wyliczam współczynnik na podstawie wzoru:

$$R_B = \frac{F}{\sqrt{2\pi * \sum_i r_i^2}}$$

r_i= odległość piksela obiektu od środka ciężkości obiektu
i- numer piksela obiektu

```
def BlairBliss(points,tab):

    s = len(points)
    mx = tab[1]
    my = tab[0]

    r = 0
    for point in points:
        r = r + distance.euclidean(point,(my,mx))**2

    return s/(math.sqrt(2*math.pi*r))
```

Aby wyznaczyć współczynnik Haralicka wyznaczam kontury obiektów za pomocą funkcji cv2.findContours(). Drugim argumentem jest już wyznaczony wcześniej środek obiektu.

Stosuję się tutaj do wzoru:

$$R_H = \sqrt{\frac{\left(\sum_i d_i\right)^2}{n * \sum_i d_i^2 - 1}}$$

d_i- odległość pikseli konturu obiektu od jego środka ciężkości
n- liczba pikseli konturu
i- numer piksela obiektu

```python
def Haralick(center,cont):

    n = len(cont)
    mx, my = center

    a1 = 0
    a2 = 0
    for i in range(n):
        a1 += distance.euclidean((cont[i][0][1], cont[i][0][0]),(my,mx))
        a2 += (distance.euclidean((cont[i][0][1], cont[i][0][0]),(my,mx))**2 - 1)
    return math.sqrt((a1**2)/(n*a2))
```

Na sam koniec wypisuję wszystkie informacje o każdym obiekcie w pętli co daje końcowy wynik programu:

```
Wykryto 12 monet

Ilosc monet ulozona wartosciami od 5zl w dol:
[1, 1, 1, 2, 1, 1, 2, 1, 2]

Suma : 9.44 zl

Obiekt 1:
   Zajmuje: 0.31% zdjecia
   Srodek geometryczny: (165,256)
   Feret: 1.0
   Blair-Bliss: 0.9994303613566252
   Haralick: 0.9997419132718147

Obiekt 2:
   Zajmuje: 0.65% zdjecia
   Srodek geometryczny: (205,167)
   Feret: 0.9333333333333333
   Blair-Bliss: 0.9992299419219861
   Haralick: 0.9953304906631729

Obiekt 3:
   Zajmuje: 0.47% zdjecia
   Srodek geometryczny: (221,134)
   Feret: 0.8461538461538461
   Blair-Bliss: 0.996174647634484
   Haralick: 0.999485518540771

Obiekt 4:
   Zajmuje: 0.76% zdjecia
   Srodek geometryczny: (229,203)
   Feret: 0.9444444444444444
   Blair-Bliss: 0.9912595272346052
   Haralick: 0.9878134643665503
```

Z informacją na końcu, ile łącznie zajmują monety:

```
Obiekty zajmuja lacznie okolo 8.17% miejsca na zdjeciu
```

5 WYNIKI EKSPERYMENTALNE

NUMER OBRAZU	ILOŚĆ WYKRYTYCH MONET/ILOŚĆ MONET	SKUTECZNOŚĆ W %	ILOŚĆ POPRAWNIE WYKRYTYCH NOMINAŁÓW/ILOŚĆ MONET	SKUTECZNOŚĆ W %
1	8/8	100%	8/8	100%
2	9/9	100%	9/9	100%
3	12/12	100%	11/12	92%
4	15/15	100%	13/15	87%
5	17/17	100%	15/17	88%
6	20/20	100%	18/20	90%
7	21/21	100%	17/21	81%
8	12/12	100%	11/12	92%
9	21/21	100%	19/21	90%
10	17/17	100%	15/17	88%
11	21/21	100%	16/21	76%
12	21/21	100%	17/21	81%
13	21/21	100%	19/21	90%
14	25/25	100%	20/25	80%
15	16/16	100%	15/16	94%

6 WNIOSKI

W przypadku 15 zdjęć, które używałem program zliczył ilość monet bezbłędnie na każdym z nich.

Skuteczność przy rozpoznawaniu nominałów niestety nie była aż tak dobra – średnio wynosiła 88,6%. Powodem takiego wyniku jest bardzo zmienny rozmiar obiektów po rozdzieleniu w zależności od

tego czy stykały się z innymi obiektami czy nie. Najczęstszą pomyłką było rozpoznawanie 10 gr jako 1 gr i na odwrót. Jeżeli moneta 10gr była otoczona z każdej strony to po rozdzieleniu zazwyczaj zmniejszała się do wielkości monety 1gr co powodowało błędy. Parę razy ręcznie poprawiałem przedziały wielkości monet i 88,6% to najwięcej, ile udało mi się osiągnąć.