Politechnika Śląska
Wydział Automatyki Elektroniki i Informatyki
Kierunek: Automatyka i Robotyka sem. 7

Gliwice
Rok akademicki
2020/2021
Semestr zimowy

Laboratorium
Przetwarzania Informacji Wizyjnej
Rozpoznawanie płytek (kamieni) domina

Choiński Kamil
Stabla Oskar
Gr. I

Liczba oczek

1. Wstęp

Celem projektu w ramach przedmiotu o nazwie Przetwarzanie Informacji Wizyjnej było stworzenie programu będącego w stanie rozpoznać na podanym przez użytkownika obrazie kostkę domina, a następnie zliczyć jej liczbę oczek wraz z podziałem na segment górny i dolny. Środowisko programistyczne nie było z góry narzucone, jednak postanowiliśmy, iż całość zostanie wykonana w programie Matlab z uwagi na jego doskonałe możliwości przetwarzania informacji w obrazie. W celu usprawnienia prac nad projektem wykorzystane zostało środowisko Git, które umożliwiło asynchroniczną pracę nad jednym kodem źródłowym.

2. Podobne rozwiązania

- Projekt Domino-Detection

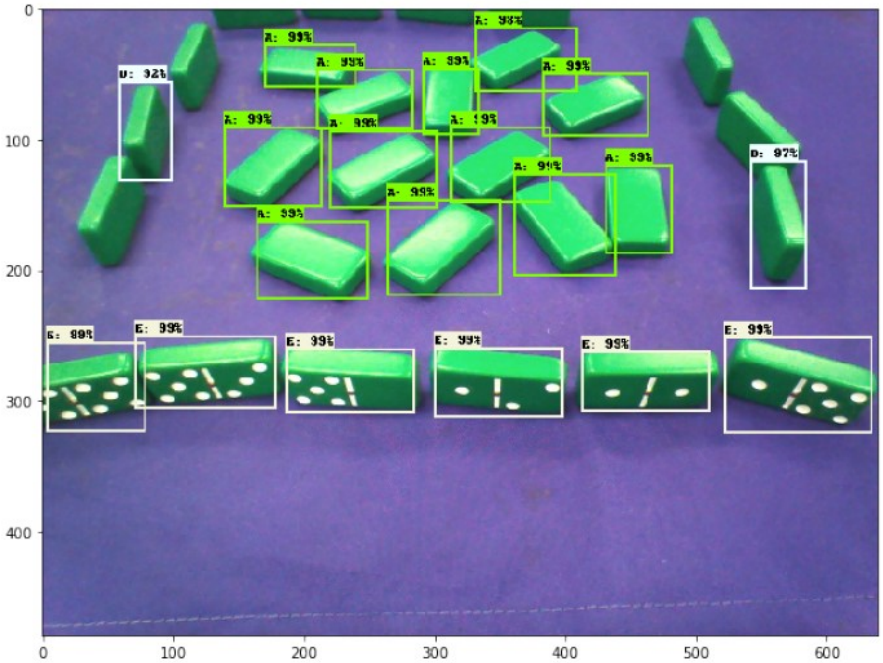
https://github.com/karthikVenkataramana/Domino-Detection

Projekt używa języka Python i Google Object detection API

Dzięki użyciu Google Object detection API program jest w stanie rozpoznawać w jakim położeniu znajduje się domino. Po wykryciu domino program determinuje kierunek skierowania oczek domino według oznaczeń poniżej.

Oczka do dołu - A
Oczka do góry - B
Stoi i patrzy w lewo - C
Stoi i patrzy w prawo - D
Stoi i patrzy w kamerę - E
Stoi i nie patrzy w kamerę - F

Nie jest to efekt jaki chcieliśmy osiągnąć w naszym projekcie, jednak jest to ciekawa implementacja wykrywania domino z użyciem Google Object detection API.

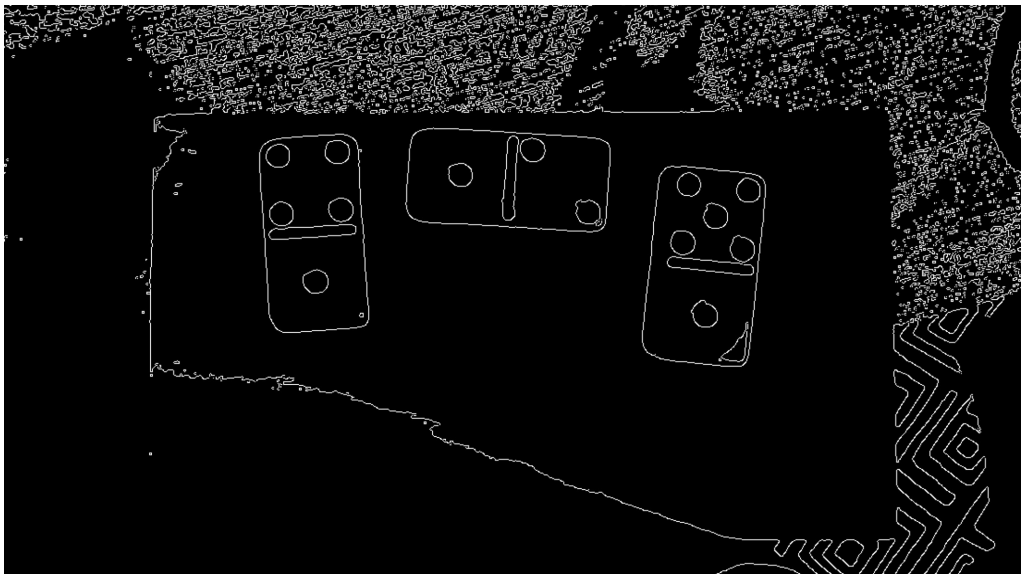


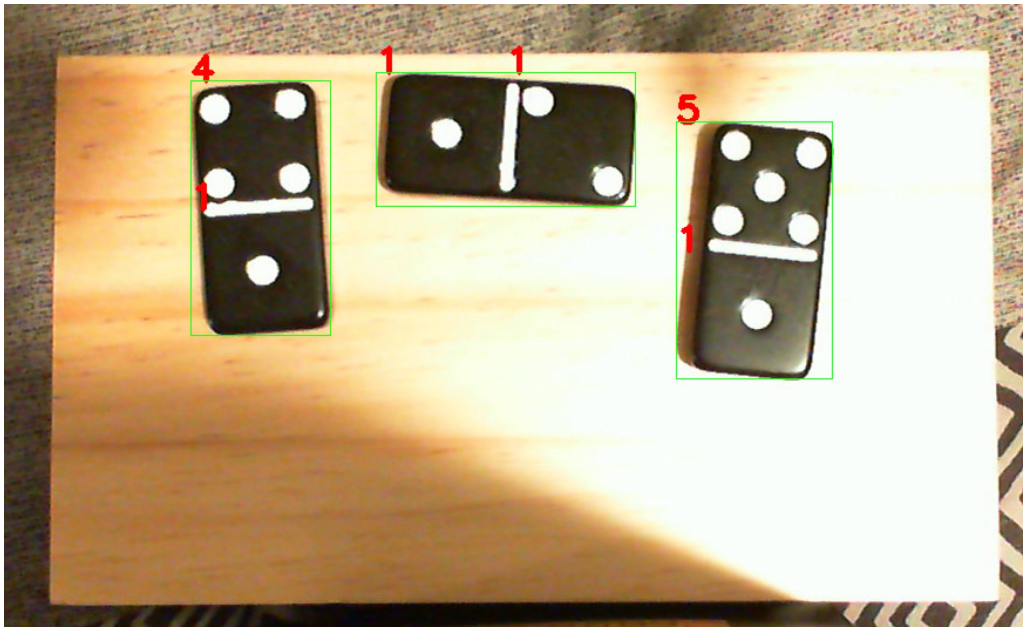
- Projekt DominoesDetection

https://github.com/HookJordan/DominoesDetection
Projekt ten używa języka C++ wraz z biblioteką OpenCV 3.0
Jego celem jest wykrywanie kostek domino na załadowanym obrazie
i wyświetlenie liczby oczek przy wykrytym domino.

Dzięki użyciu biblioteki OpenCV 3.0 możliwe jest zlokalizowanie kostki domino i jego wyodrębnienie z reszty obrazu. Pozwala to na dalszą analizę osobnego domino dzięki użyciu osobnej funkcji która dzieli domino na 2 części i osobno wypisuje liczbę oczek dla każdej części.

Projekt nie działa w każdych warunkach, jednak wyświetla poprawne wartości przy dobrym oświetleniu jak pokazane poniżej. Program ma też problem przy ułożonym domino w innej orientacji niż poziomo (przy pochylonym zdjęciu)





3. Implementacja projektu w Matlabie

Wykrywanie kostek domina i zliczanie oczek podzielone jest na parę etapów:

1. Wczytanie obrazu poprzez okno dialogowe, w którym użytkownik wybiera pożądaną próbkę, którą chce poddać analizie

2. Pozbycie się kolorystyki. W tym celu wykorzystano funkcję rgb2gray(), która zwraca obraz w skali szarości

3. Obliczenie progu binaryzacji, który stanowi wyznacznik, powyżej której wartości piksel ma wartość 1, a poniżej której 0. Wykorzystano do tego metodę OTSU, czyli metodę progowania globalnego, opartego na histogramie obrazu szarego. W Matlabie metodę tę wykonuje funkcja graythresh(), przyjmująca obraz szary i zwracająca próg binaryzacji.

4. Binaryzacja obrazu. Posiadając obraz szary jak i próg binaryzacji możemy uzyskać obraz czarno-biały poprzez wywołanie funkcji im2bw()

5. Wyłuskanie obszarów, należących do zbinaryzowanego obrazu. W idealnym przypadku będą to tylko i wyłącznie koła/elipsy, linie oraz cała kostka. W tym celu służy funkcja bwboundaries()

6. Analiza parametrów obszarów w celu ich dalszej klasyfikacji – Funkcja regionprops(), która zwraca strukturę, zawierającą 30 parametrów dla każdego obszaru, m.in. kąt nachylenia, położenie środka, stopień wypełnienia, kołowość i wiele innych

STATS

9x1 struct with 30 fields

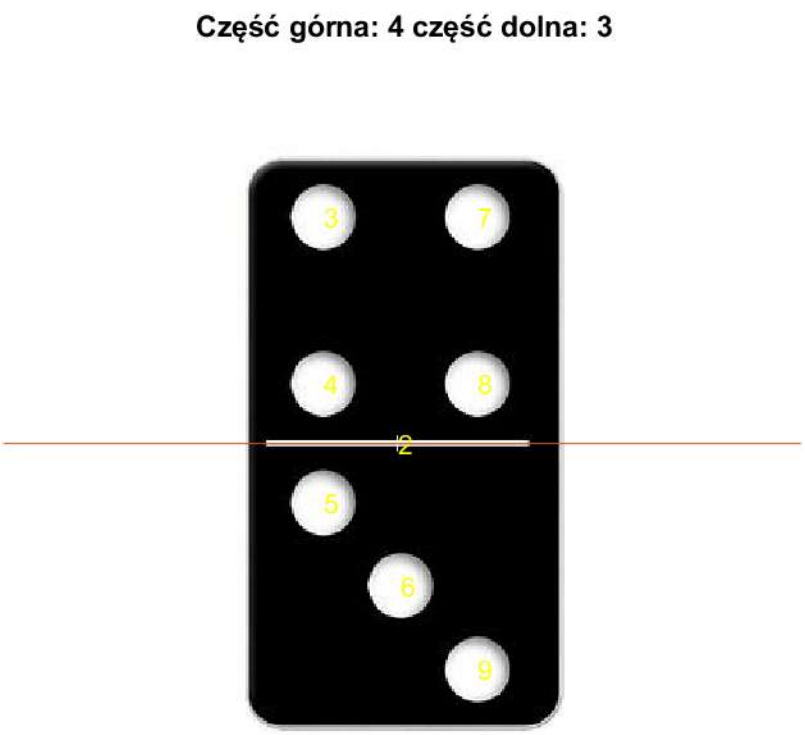
Fields	Area	Centroid	BoundingBox	Subarrayldx	MajorAxisLength	MinorAxisLength	Eccentricity	Orientation	ConvexHull	ConvexImage	ConvexArea	Circularity	Image
1	5595468	[1.2661e+03...	[0.5000,0.5000,25...	1x2 cell	3.0162e+03	2.9490e+03	0.2100	-13.6335	10001x2 double	2500x2500 logical	6250000	0.7328	2500x2500 l...
2	6112	[605.7349,1....	[539.5000,1.4895...	1x2 cell	131.4444	59.2293	0.8927	3.9902	132x2 double	60x132 logical	6204	0.8001	60x132 logical
3	2	[757.5000,1...	[756.5000,1.5345...	1x2 cell	2.3094	1.1547	0.8660	0	7x2 double	[1, 1]	2	6.5423	[1, 1]
4	6723	[841.1838,1....	[773.5000,1.5965...	1x2 cell	134.5044	63.6618	0.8809	2.1053	127x2 double	65x135 logical	6829	0.8197	65x135 logical
5	6417	[871.2364,1....	[804.5000,1.4775...	1x2 cell	133.2235	61.3454	0.8877	1.7961	149x2 double	61x133 logical	6504	0.8090	61x133 logical
6	5547	[899.2991,1....	[835.5000,1.3665...	1x2 cell	127.4187	55.4520	0.9003	1.5242	114x2 double	56x128 logical	5649	0.7698	56x128 logical
7	15394	[1.0909e+03...	[885.5000,1.5115...	1x2 cell	507.4593	40.2683	0.9968	23.9935	59x2 double	200x417 logical	16069	0.2218	200x417 logi...
8	5986	[1.1357e+03...	[1.0705e+03,1.46...	1x2 cell	129.5762	58.8366	0.8910	0.9102	136x2 double	59x130 logical	6076	0.8070	59x130 logical
9	7527	[1.3355e+03...	[1.2655e+03,1.68...	1x2 cell	140.9081	68.0371	0.8757	0.6759	131x2 double	68x140 logical	7642	0.8277	68x140 logical

7. Wykrycie linii – jest wykrywana w dwóch przypadkach, albo szerokość głównej osi elipsy zbudowanej na podstawie wykrytego obszaru jest co najmniej 4 razy większa od jej krótszej osi, albo gdy stopień wypełnienia obszaru w stosunku do „pudełka”, w którym jest definiowany przez regionprops() jest większy niż 0.9, a szerokośc i wysokość są od siebie różne (warunek na prostokąt, którym jest linia)

8. Utworzenie równania prostej – w celu dalszej analizy położenia oczek konstruujemy prostą przechodzącą przez punkt środka linii domina, o kącie nachylenia równym kątowi nachylenia głównej osi elipsy zbudowanej na tej linii.

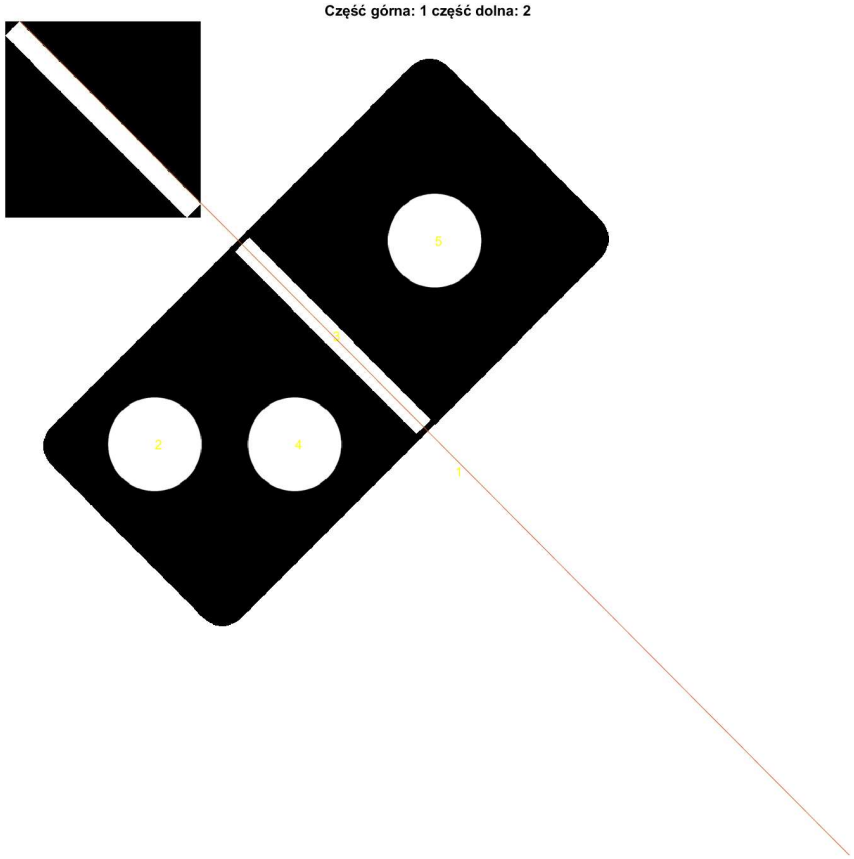
9. Wykrycie oczek i przyporządkowanie ich do określonego segmentu, górnego lub dolnego – W tym celu sprawdzamy, czy parametr Circularity (kołowość) danego obszaru zawiera się w przedziale od 0.76 do 1.1. Pozwala to przyporządkować zarówno koła jak i elipsy, które powstają przy przechylonej pozycji kostki domina. Następnie sprawdzamy, czy linia jest pionowa. Jeśli tak, to dochodzi do porównania współrzędnych iksowych środków kół/elips ze środkiem linii. W przypadku nachylenia innego niż poziome rozwiązywana jest nierówność y>ax+b, gdzie y i x to współrzędne środka koła/elipsy, a parametry „a" i „b" wyznaczane są z równania prostej. Jeśli y jest większe, to koło przypisywane jest do dolnego segmentu. Wynika to z tego, iż Matlab rozpoczyna indeksowanie drugiej współrzędnej od lewego górnego rogu ekranu.

10. Wyświetlenie otrzymanego rezultatu na obrazku



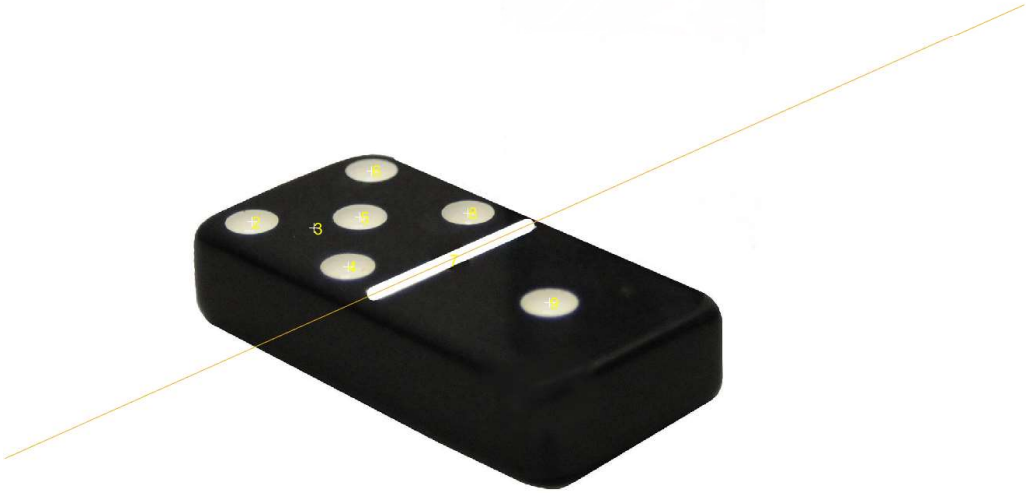
Przykład nr 1

Część górna: 1 część dolna: 2



Przykład nr 2

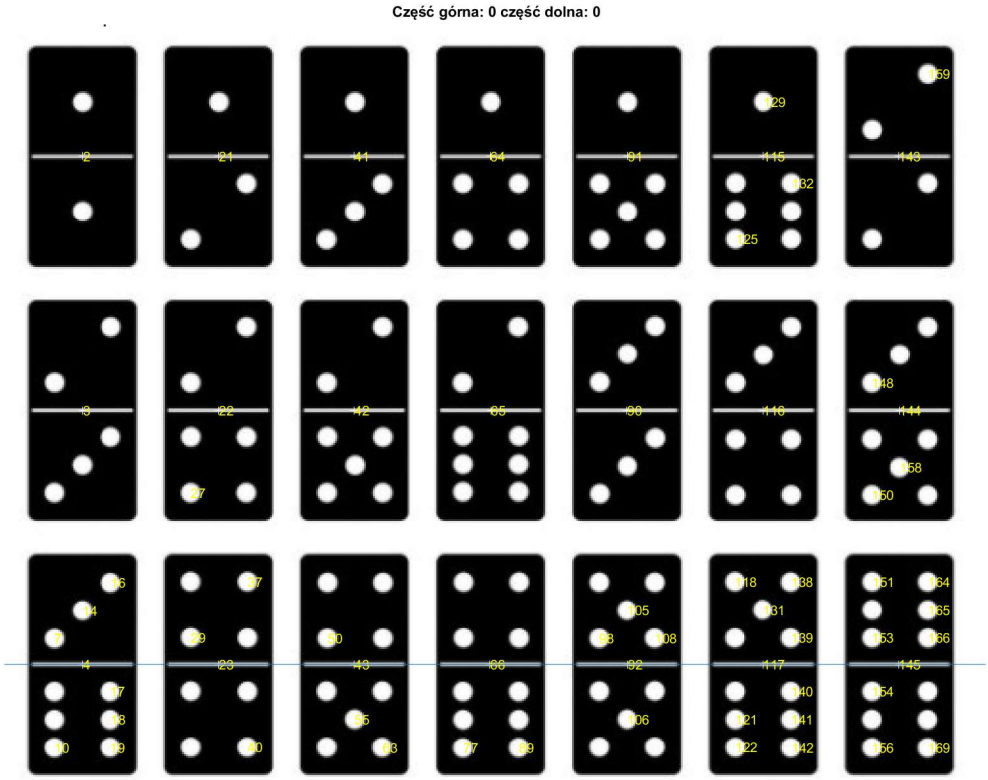
Część górna: 5 część dolna: 1



Przykład nr 3

4. Ocena efektywności programu

Na poniższym przykładzie nr 4 widać, że dla obrazu w niższej jakości nie wszystkie oczka są wykrywane. Program działa tylko dla jednej kostki domina na obrazie, przez co nie zostaną one zliczone poprawnie dla przykładu z wieloma kostkami, jednak wciąż jest w stanie wykryć linie i oczka w miarę swoich możliwości.



Przykład nr 4

Dla kostki z niedoskonałościami w postaci na przykład refleksów świetlnych zauważyliśmy wykrywanie w nich niepożądanych kształtów, co wpływa na liczbę wykrywanych oczek. W dodatku niektóre kostki domina posiadają dodatkowe oczko na środku linii, co również ma niekorzystny wpływ na działanie programu. Skutki zobrazowane na przykładzie nr 5.

Przykład nr 5

5. Wnioski

Biorąc pod uwagę narzędzia, którymi dysponowaliśmy jesteśmy zadowoleni z działania programu. Wykrywa on większość oczek i linii dzielących połówki kostki domina, a w przypadku braku niekorzystnych niedoskonałości zdjęcia zapewnia satysfakcjonującą dokładność zliczania. Niestety działa on jedynie dla pojedynczej kostki domina na jednym obrazie wejściowym. Taki sposób determinizmu liczby oczek obarczony jest wadami metod takich jak binaryzacja oparta na histogramie, który może zostać łatwo zaburzony przez artefakty powstałe przy zmniejszaniu jakości obrazu. Sądzimy jednak, że przy większym nakładzie prac i wykorzystaniu dodatkowych bibliotek możliwym byłoby wyłuskanie pojedynczej kostki z grupy, a następnie przeprowadzenie tych samych operacji dla tego obiektu.

6. Kod programu

```
clear all
clc
close all
    %RGB = imread('domino_4.jpg');
    [fn,pn]=uigetfile({'*.png'}, 'Wybierz obraz');
    RGB = imread([pn,fn]);
    figure,
    imshow(RGB),
    title('Original Image');

    GRAY = rgb2gray(RGB);

    threshold = graythresh(GRAY);


    BW = im2bw(GRAY, threshold);


    [B,L] = bwboundaries(BW, 'noholes');

    STATS = regionprops(L, 'all'); % we need 'BoundingBox' and
'Extent'

    figure,
    imshow(RGB),
    title('Results');
    hold on

    circles_1 = 0;
    circles_2 = 0;
    random_1 = 0;
    is_line = 0;
    lineSTAT=0;
    %bounding box defined for each shapes as
[x_cordinate,y_cordinate,x_width,y_width]
    for i = 1 : length(STATS)
        %for rectangles, we have x_width != y_width,extent =1
        if(((STATS(i).BoundingBox(3)~=STATS(i).BoundingBox(4)) &&
(STATS(i).Extent>=0.9)) ||
(STATS(i).MajorAxisLength>4*STATS(i).MinorAxisLength))
            is_line = is_line + 1;
            lineSTAT = STATS(i);
            imshow(STATS(i).ConvexImage)
            centroid = STATS(i).Centroid;
            plot(centroid(1),centroid(2),'w+');
            text(centroid(1),centroid(2),num2str(i),'Color','y');
        end
    end

    vertical=0;
    a = tand(lineSTAT.Orientation);
```

```
54.      if (lineSTAT.Orientation>=88)
55.          vertical=1
56.      end
57.      a=a*(-1);
58.      b = lineSTAT.Centroid(2) - (a*lineSTAT.Centroid(1));
59.
60.      x=0:1:length(BW)-1;
61.
62.      y=a*x+b;
63.      plot(x,y)
64.
65.      if (isstruct(lineSTAT))
66.          for i = 1 : length(STATS)
67.              centroid = STATS(i).Centroid;
68.             if(STATS(i).Circularity>=0.76 && STATS(i).Circularity<=
    1.1 )
69.                  if(vertical)
70.                      if(STATS(i).Centroid(1)<(lineSTAT.Centroid(1)))
71.                          circles_1= circles_1 +1;
72.                      else
73.                          circles_2 = circles_2 +1;
74.                      end
75.                  else
76.
    if(STATS(i).Centroid(2)<(a*STATS(i).Centroid(1)+b))
77.                          circles_1= circles_1 +1;
78.                      else
79.                          circles_2 = circles_2 +1;
80.                      end
81.                  end
82.                  plot(centroid(1),centroid(2),'w+');
83.
    text(centroid(1),centroid(2),num2str(i),'Color','y');
84.
85.
    elseif((STATS(i).BoundingBox(3)==STATS(i).BoundingBox(4)) &&
    (STATS(i).Extent > 0.76 && STATS(i).Extent < .795))
86.
87.                  plot(centroid(1),centroid(2),'wO');
88.
    text(centroid(1),centroid(2),num2str(i),'Color','y');
89.
90.
91.
    elseif((STATS(i).BoundingBox(3)~=STATS(i).BoundingBox(4)) &&
    (STATS(i).Extent > 0.76 && STATS(i).Extent < .795))
92.
93.                  plot(centroid(1),centroid(2),'w*');
94.
    text(centroid(1),centroid(2),num2str(i),'Color','y');
95.              end
96.
97.          end
98.      end
```

```
99.
100.
101.     title("Część górna: " + circles_1 + " część dolna: " +
   circles_2);
```