|  |  |
| --- | --- |
| **Politechnika Śląska** | **Gliwice** |
| **Wydział Automatyki Elektroniki i Informatyki**  **Kierunek: Automatyka i Robotyka sem. 7** | **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.

1. 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

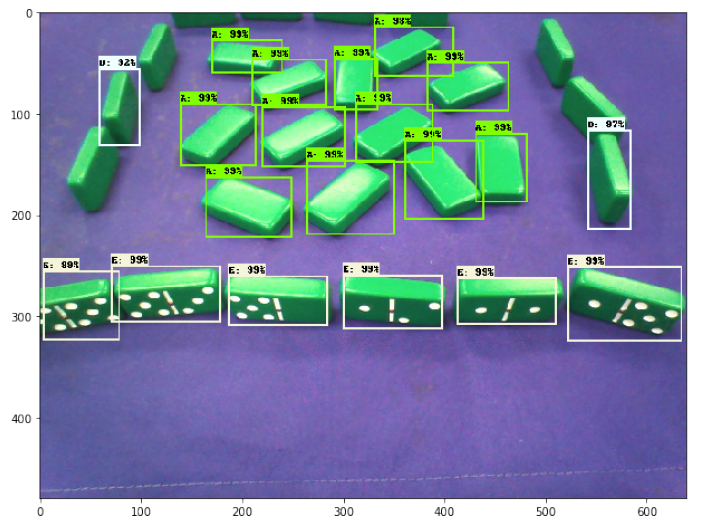
Stoi i patrzy w lewo - C

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>

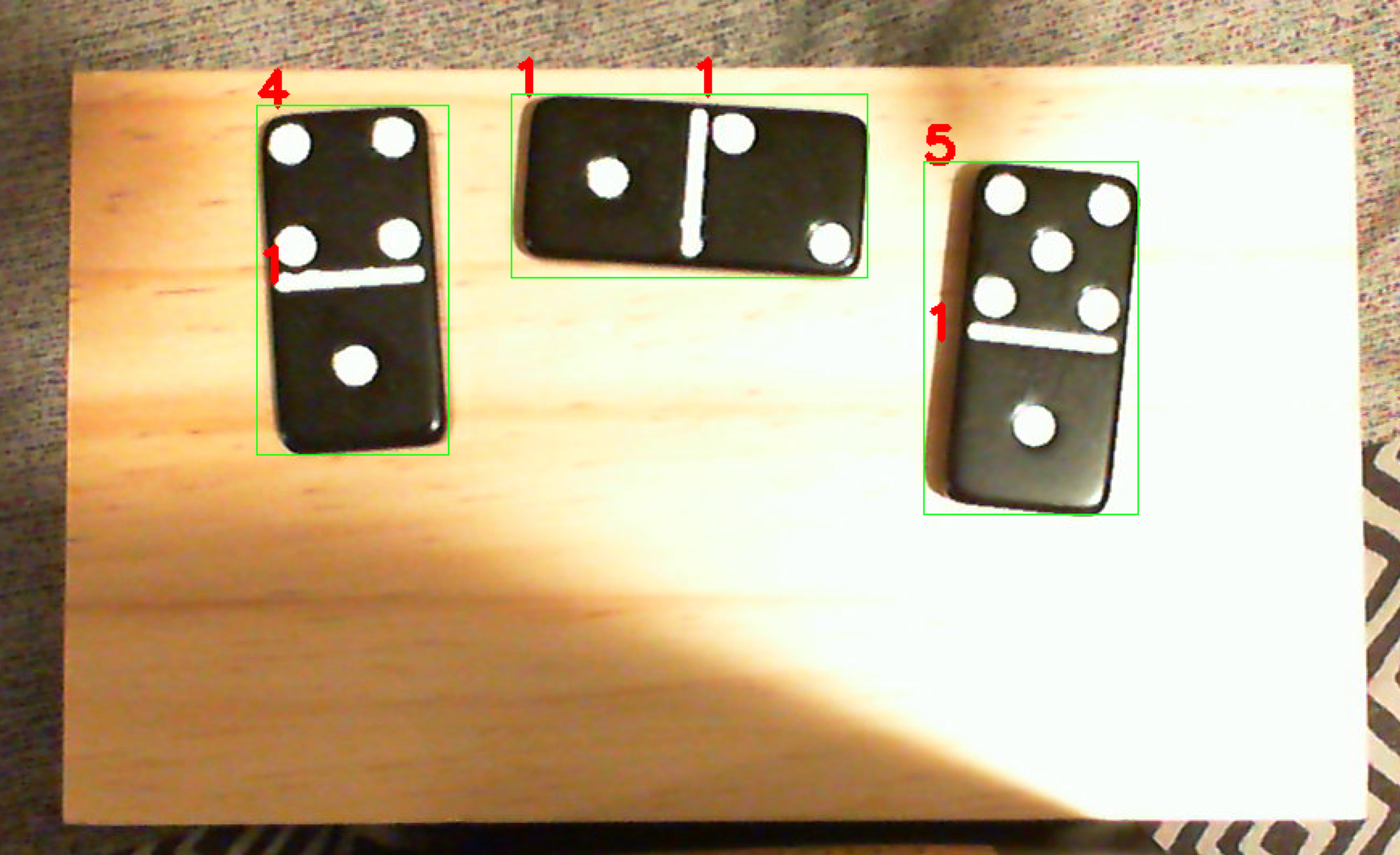
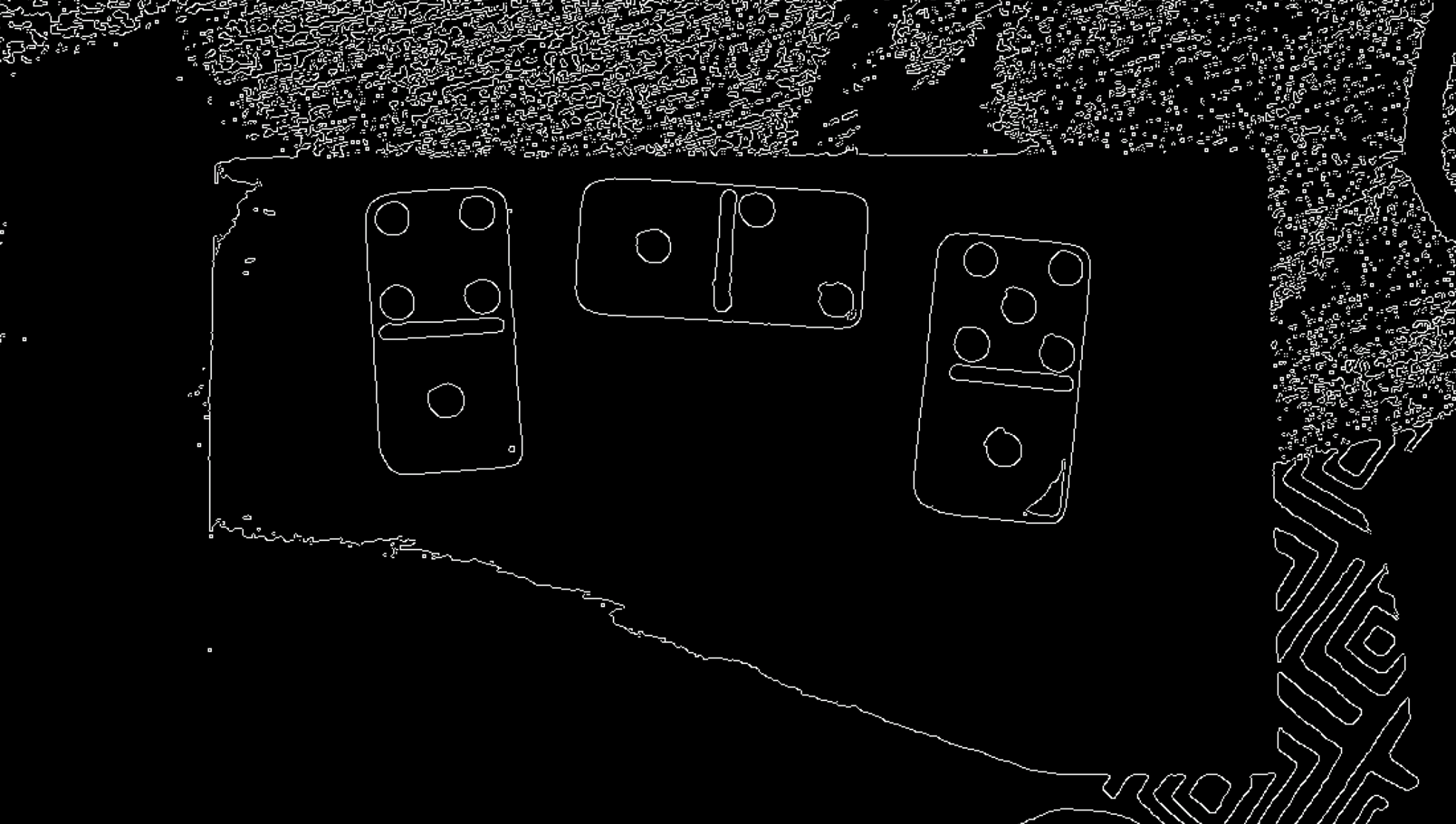
Projekt ten używa języka C++ wraz z biblioteką OpenCV 3.0

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.

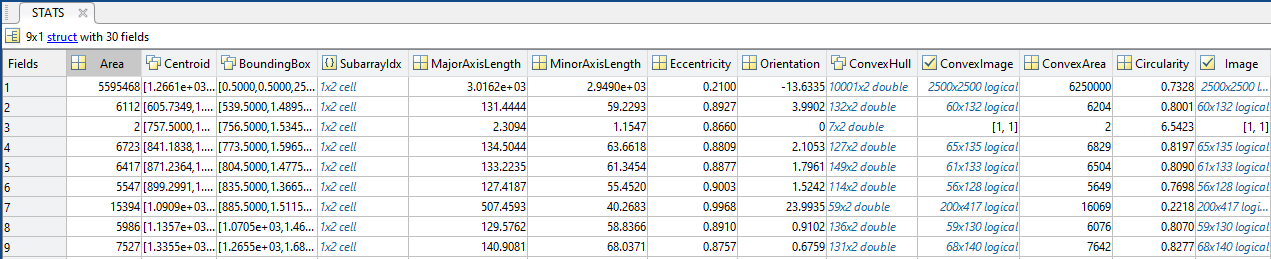
Projekt nie działa w każdych warunkach, jednak wyświetla poprawne wartości przy dobrym oświetleniu jak pokazane poniżej. Program ma też problem przy ułożonym domino w innej orientacji niż poziomo (przy pochylonym zdjęciu)



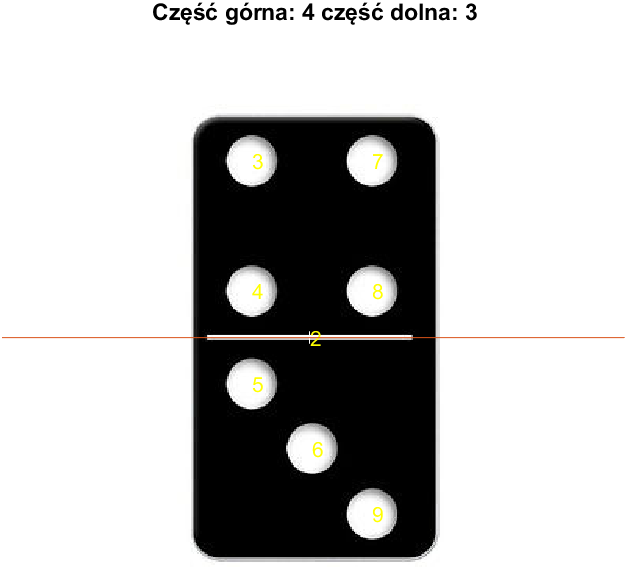
1. Implementacja projektu w Matlabie

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

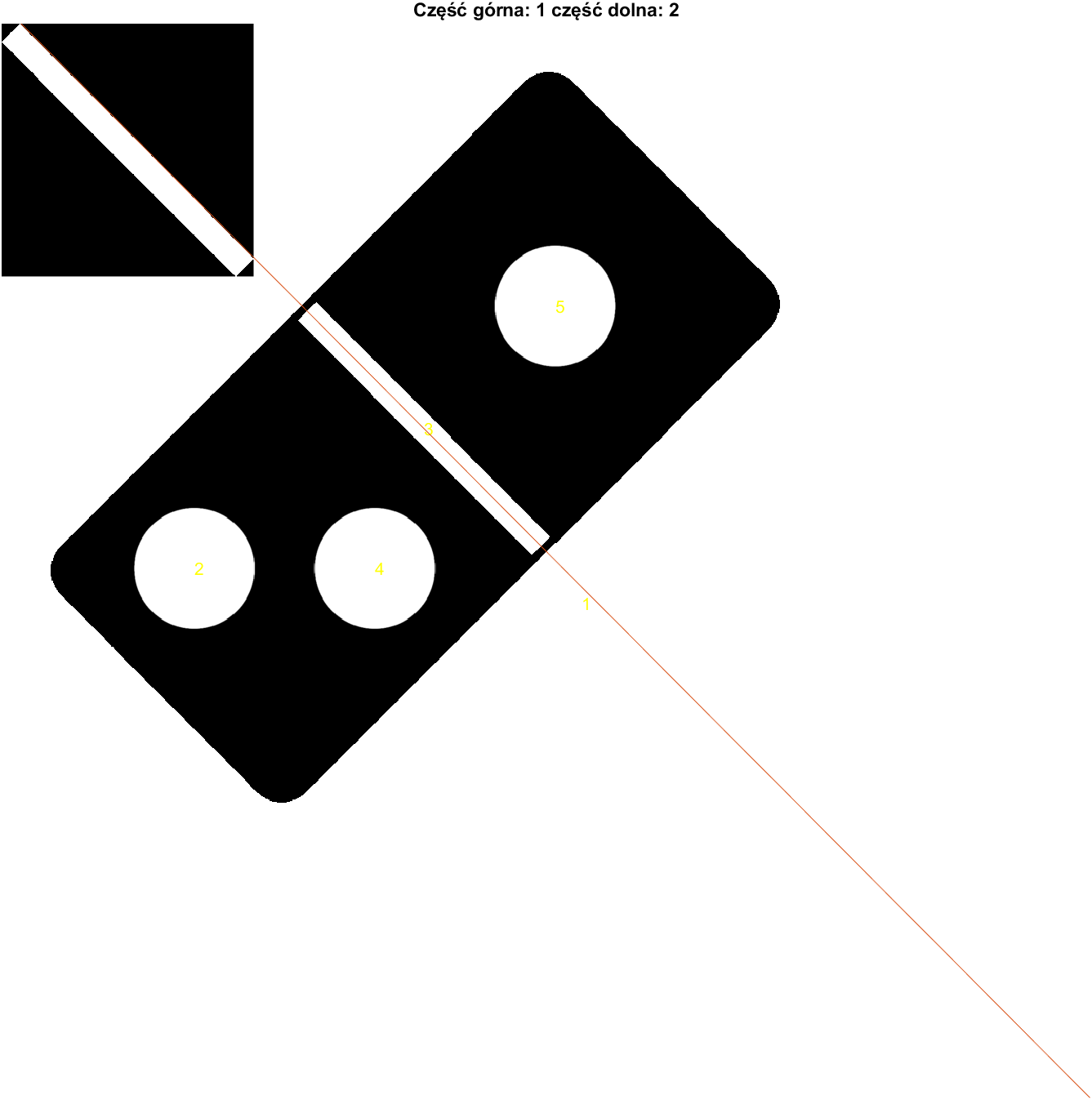
1. Wczytanie obrazu poprzez okno dialogowe, w którym użytkownik wybiera pożądaną próbkę, którą chce poddać analizie
2. Pozbycie się kolorystyki. W tym celu wykorzystano funkcję rgb2gray(), która zwraca obraz w skali szarości
3. Obliczenie progu binaryzacji, który stanowi wyznacznik, powyżej której wartości 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



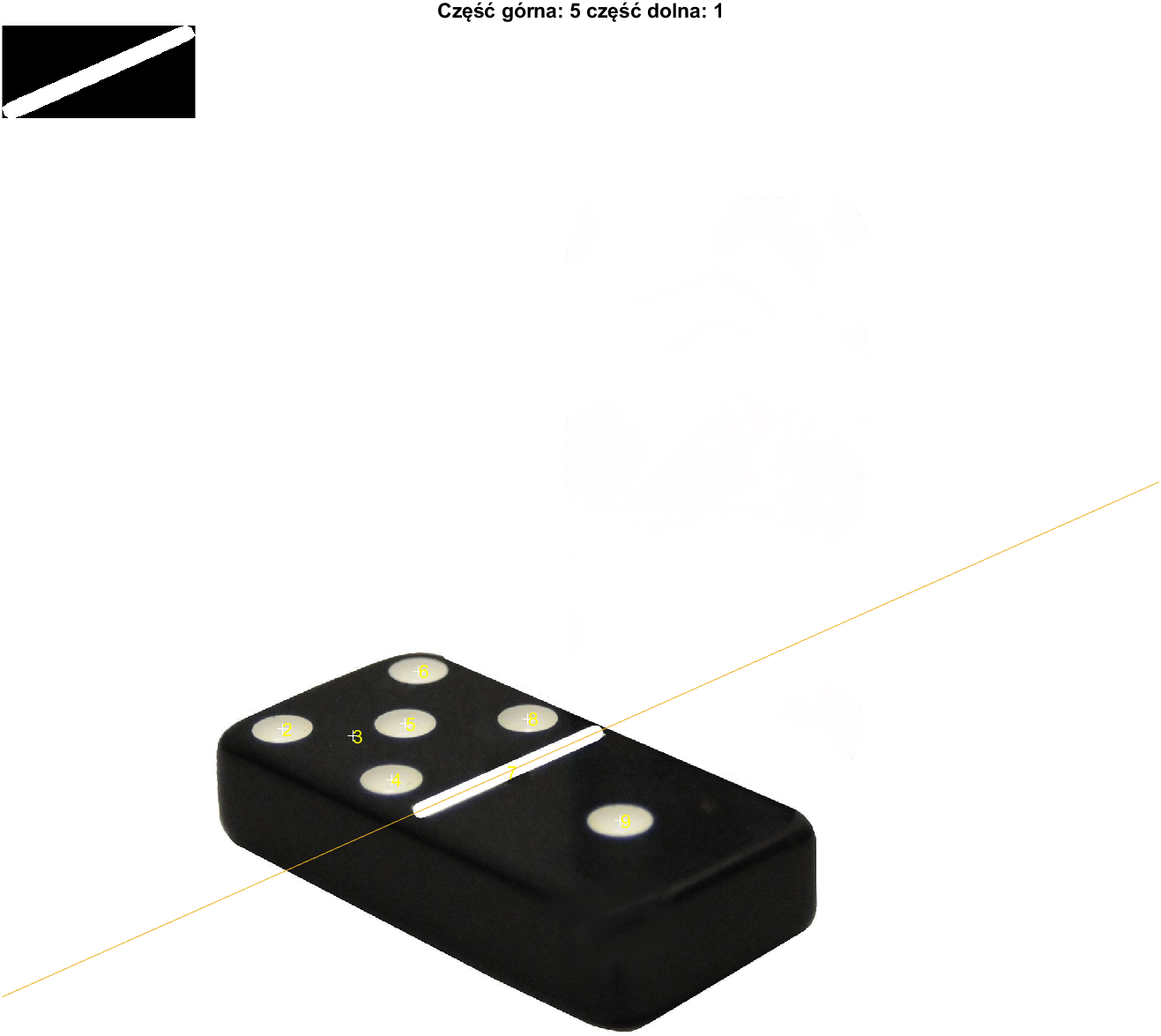
1. Wykrycie linii – jest wykrywana w dwóch przypadkach, albo szerokość głównej osi elipsy zbudowanej na podstawie wykrytego obszaru jest co najmniej 4 razy większa od jej krótszej osi, albo gdy stopień wypełnienia obszaru w stosunku do „pudełka”, w którym jest definiowany przez regionprops() jest większy niż 0.9, a szerokośc i wysokość są od siebie różne (warunek na prostokąt, którym jest linia)
2. 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.
3. 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.
4. Wyświetlenie otrzymanego rezultatu na obrazku



Przykład nr 1



Przykład nr 2



Przykład nr 3

1. Kod programu
2. clear all
3. clc
4. close all
5. %RGB = imread('domino\_4.jpg');
6. [fn,pn]=uigetfile({'\*.png'}, 'Wybierz obraz');
7. RGB = imread([pn,fn]);
8. figure,
9. imshow(RGB),
10. title('Original Image');
12. GRAY = rgb2gray(RGB);
14. threshold = graythresh(GRAY);

17. BW = im2bw(GRAY, threshold);

20. [B,L] = bwboundaries(BW, 'noholes');
22. STATS = regionprops(L, 'all'); % we need 'BoundingBox' and 'Extent'
24. figure,
25. imshow(RGB),
26. title('Results');
27. hold on
29. circles\_1 = 0;
30. circles\_2 = 0;
31. random\_1 = 0;
32. is\_line = 0;
33. lineSTAT=0;
34. %bounding box defined for each shapes as [x\_cordinate,y\_cordinate,x\_width,y\_width]
35. for i = 1 : length(STATS)
36. %for rectangles, we have x\_width != y\_width,extent =1
37. if(((STATS(i).BoundingBox(3)~=STATS(i).BoundingBox(4)) && (STATS(i).Extent>=0.9)) || (STATS(i).MajorAxisLength>4\*STATS(i).MinorAxisLength))
38. is\_line = is\_line + 1;
39. lineSTAT = STATS(i);
40. imshow(STATS(i).ConvexImage)
41. centroid = STATS(i).Centroid;
42. plot(centroid(1),centroid(2),'w+');
43. text(centroid(1),centroid(2),num2str(i),'Color','y');
44. end
45. end
47. vertical=0;
48. a = tand(lineSTAT.Orientation);
49. if (lineSTAT.Orientation>=88)
50. vertical=1
51. end
52. a=a\*(-1);
53. b = lineSTAT.Centroid(2) - (a\*lineSTAT.Centroid(1));
55. x=0:1:length(BW)-1;
57. y=a\*x+b;
58. plot(x,y)
60. if (isstruct(lineSTAT))
61. for i = 1 : length(STATS)
62. centroid = STATS(i).Centroid;
63. if(STATS(i).Circularity>=0.76 && STATS(i).Circularity<= 1.1 )
64. if(vertical)
65. if(STATS(i).Centroid(1)<(lineSTAT.Centroid(1)))
66. circles\_1= circles\_1 +1;
67. else
68. circles\_2 = circles\_2 +1;
69. end
70. else
71. if(STATS(i).Centroid(2)<(a\*STATS(i).Centroid(1)+b))
72. circles\_1= circles\_1 +1;
73. else
74. circles\_2 = circles\_2 +1;
75. end
76. end
77. plot(centroid(1),centroid(2),'w+');
78. text(centroid(1),centroid(2),num2str(i),'Color','y');
80. elseif((STATS(i).BoundingBox(3)==STATS(i).BoundingBox(4)) && (STATS(i).Extent > 0.76 && STATS(i).Extent < .795))
82. plot(centroid(1),centroid(2),'wO');
83. text(centroid(1),centroid(2),num2str(i),'Color','y');

86. elseif((STATS(i).BoundingBox(3)~=STATS(i).BoundingBox(4)) && (STATS(i).Extent > 0.76 && STATS(i).Extent < .795))
88. plot(centroid(1),centroid(2),'w\*');
89. text(centroid(1),centroid(2),num2str(i),'Color','y');
90. end
92. end
93. end

96. title("Część górna: " + circles\_1 + " część dolna: " + circles\_2);