

RAPORT Z PROJEKTU

Przetwarzanie Informacji Wizyjnej

Szybkie zliczanie/etykietowanie obiektów w obrazie binarnym

Grupa dziekańska **TI-3** Rok akademicki **2021/2022**. Semestr **VII**

Data wykonania raportu **18.10.2021**. Nr sekcji: **44** Nr tematu **26**.

Skład sekcji:

Albert Pintera

Jakub Kaniowski

Cel projektu

Celem projektu jest zaproponowanie oraz implementacja algorytmu zliczającego lub kolorującego białe obiekty na obrazie binarnym. W angielskiej literaturze określa się to zagadnienie terminem: *binary image connected components algorithm* lub *labeling algorithm* [1]. Implementacja może zostać zrealizowana w dowolnym środowisku programistycznym.

Wykonany program

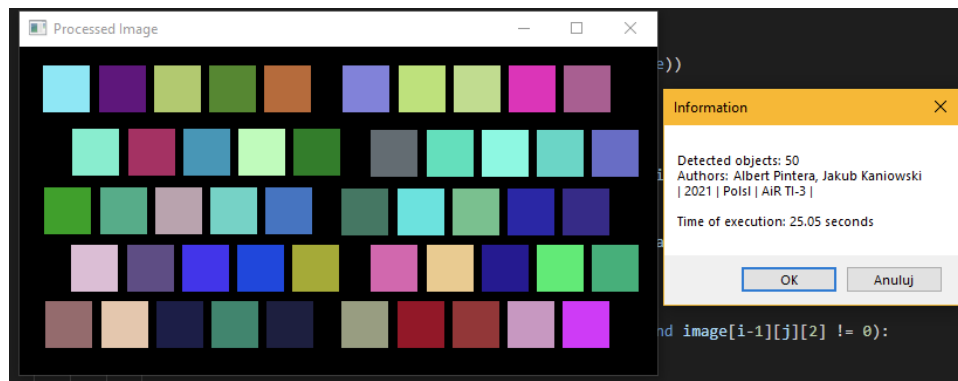
Do stworzenia programu realizującego zadanie projektowe zostało wykorzystane środowisko Python [2] wraz z biblioteką OpenCV [3]. Plikiem wejściowym jest obraz binarny przedstawiający białe obiekty na czarnym tle. Zaimplementowany algorytm [4] opiera się na analizie i przetworzeniu kolejnych pixeli.

Działanie zastosowanego algorytmu można przedstawić w kilku następujących po sobie krokach:

1. Wczytanie obrazu
2. Sprawdzenie wymiarów obrazu
3. Kolorowanie białych obszarów wiersz po wierszu (generowanie nowych kolorów)
4. Zmiana koloru danego pixela na podstawie analizy jego otoczenia.
5. Ponowna analiza otoczenia danego pixela (od prawej do lewej obrazu) w celu połączenia sąsiadujących ze sobą kolorów w konkretnym obiekcie.
6. Ponowna analiza otoczenia danego pixela (od dołu do góry obrazu) w celu połączenia sąsiadujących ze sobą kolorów w konkretnym obiekcie.
7. Zliczenie obiektów w oparciu o liczbę wszystkich występujących kolorów w obrazie.
8. Prezentacja wyników na ekranie.

Prezentacja działania

Interfejs programu prezentuje się następująco:



Okno "Processed Image" wyświetla pokolorowany obraz (detekcja obiektów białych). Okno "Information" zawiera wynik działania programu – wykryte obiekty oraz czas wykonywania się programu.

Dodatkowo, zdublowana informacja wyświetla się w konsoli systemowej.

```
PS M:\PIW> & C:/Users/Kuba/AppData/Local/Programs/Python/Python38/python.exe m:/PIW/main.py

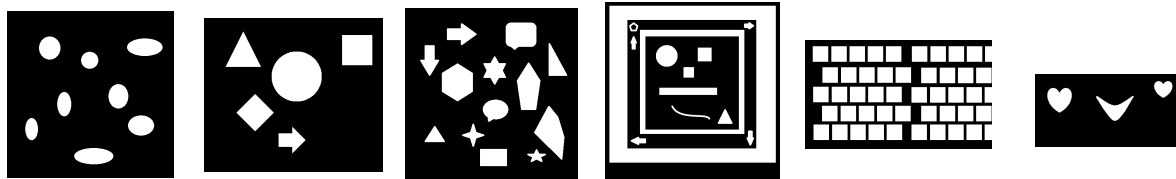
Program info:

Detected objects: 50
Authors: Albert Pintera, Jakub Kaniowski
| 2021 | Polsl | AiR TI-3 |

Time of execution: 25.02 seconds
```

Wygenerowano kilka testowych obrazów:

Shapes.png Shapes1.png Shapes2.png Shapes3.png Shapes4.png Shapes5.png



Wyniki działania programu:

Platforma testowa:

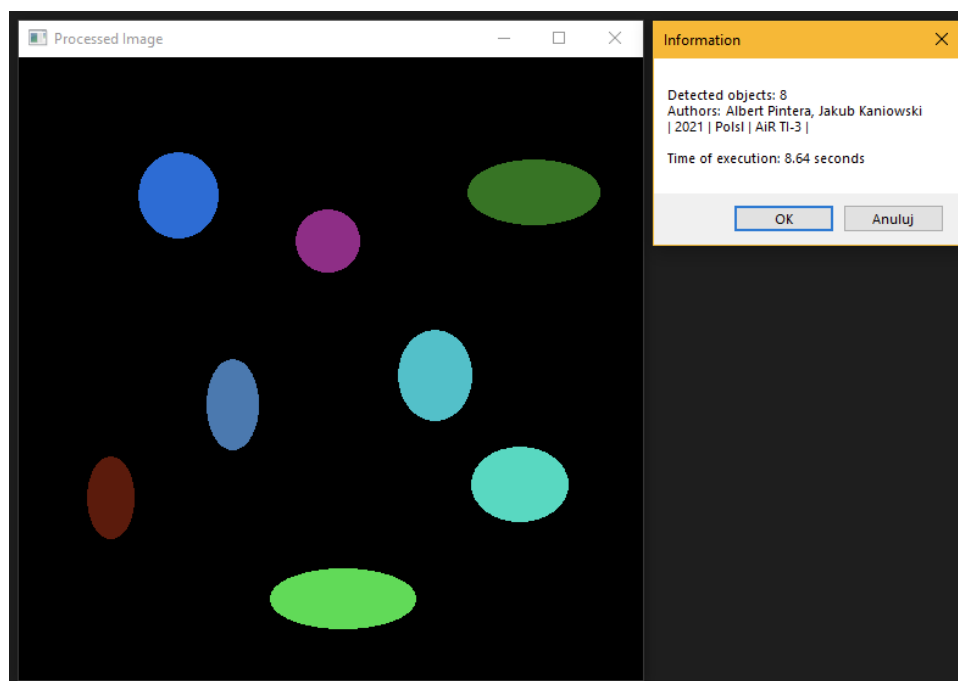
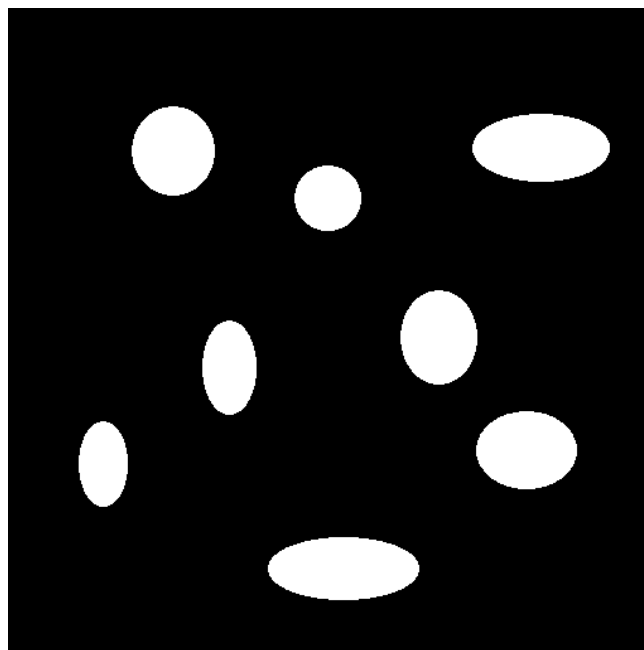
Intel Core i5-6200U (2.30GHz 2 rdzenie, 4 wątki)

Pamięć RAM: 8GB (DDR4, 2133MHz)

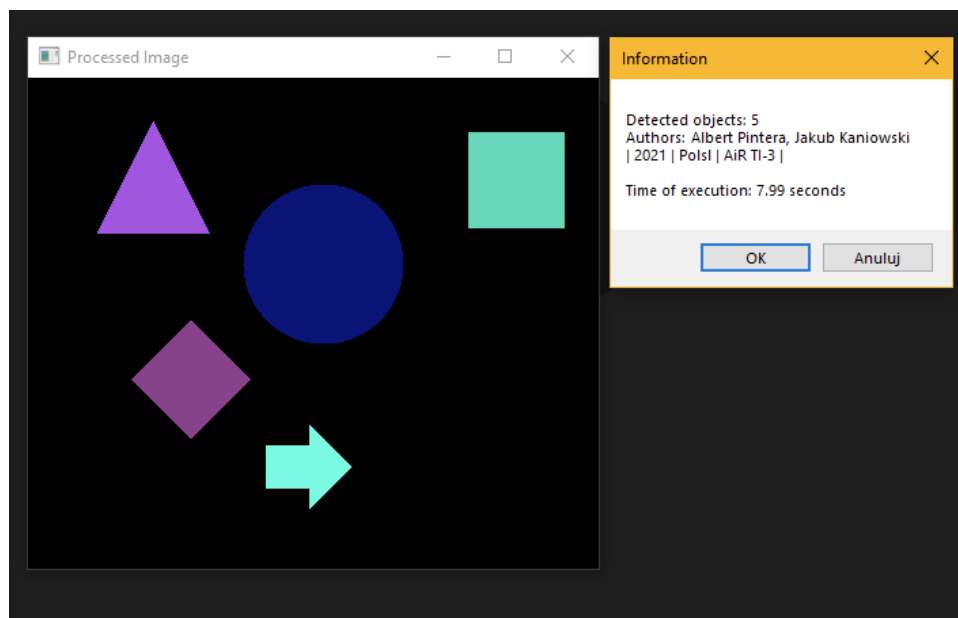
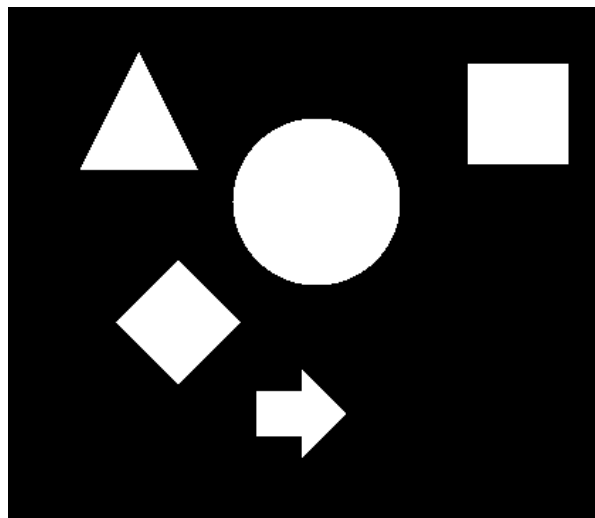
Karta graficzna (wykorzystywana): Zintegrowana Intel HD Graphics 520 (VRAM: 1GB)

System operacyjny: Windows 10 (1809)

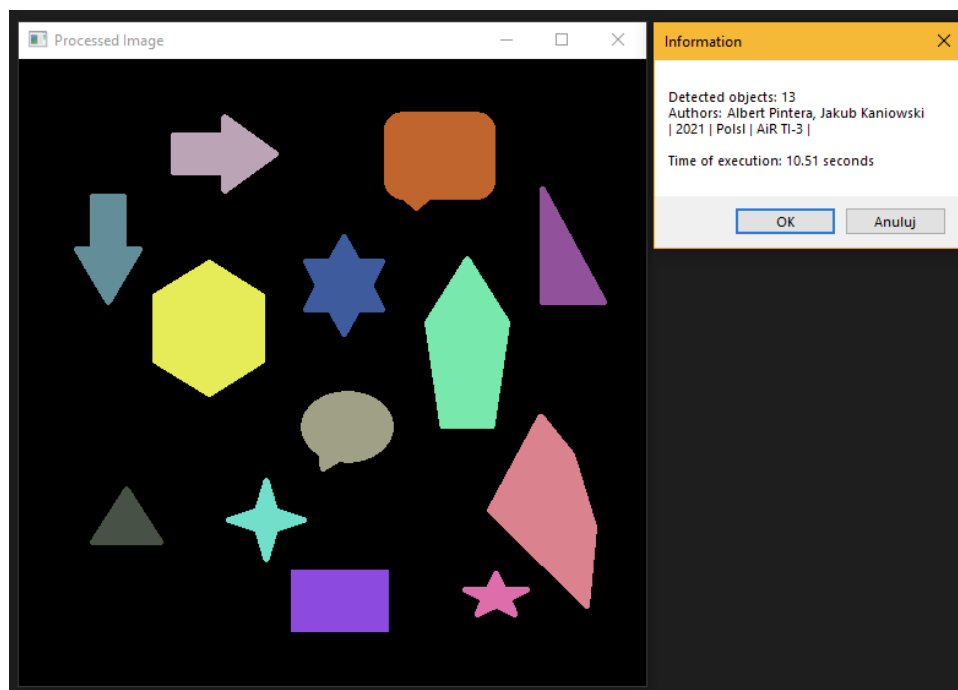
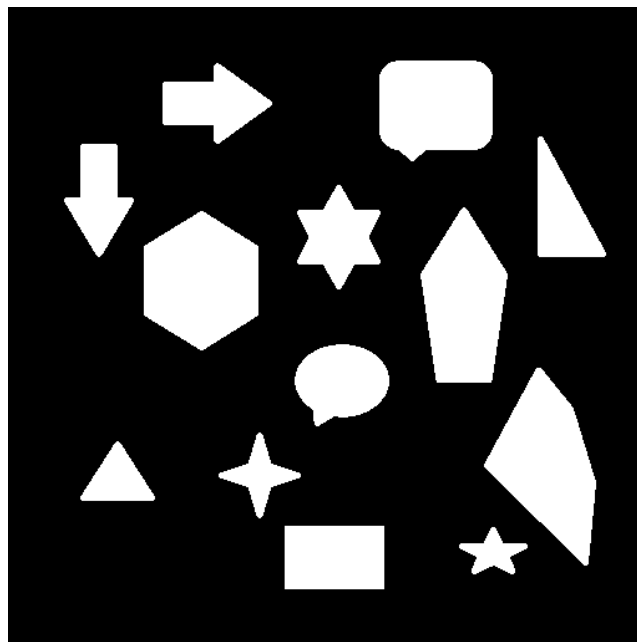
Obraz Shapes.png. Obrazów rzeczywistych: 8. Wynik działania programu: 8 obiektów



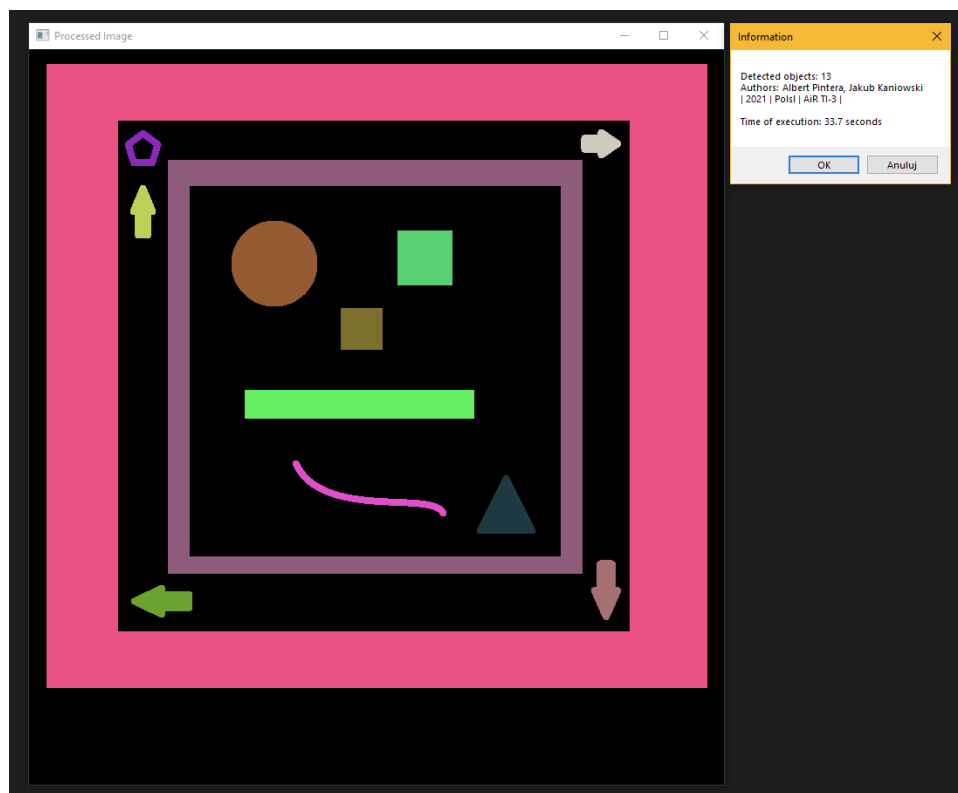
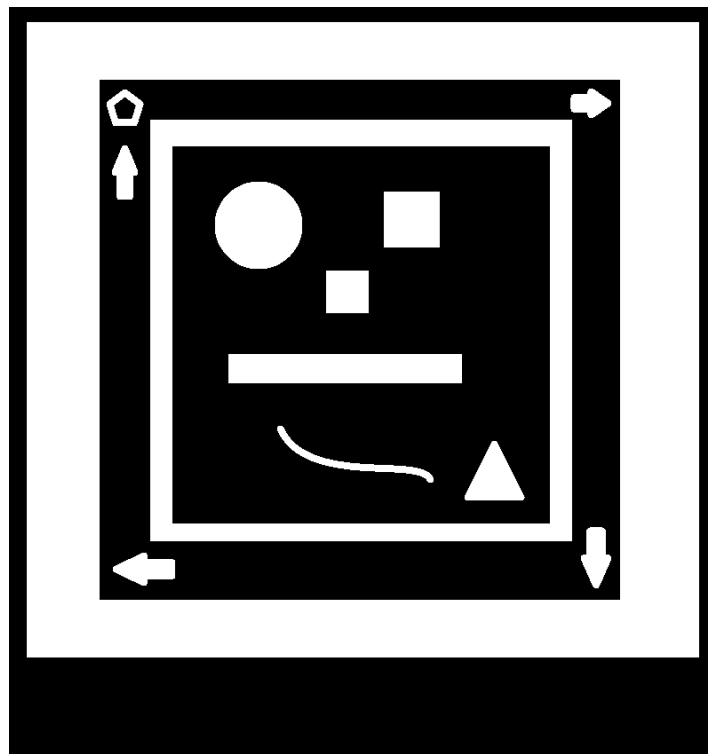
Obraz Shapes1.png. Obrazów rzeczywistych: 5. Wynik działania programu: 5 obiektów



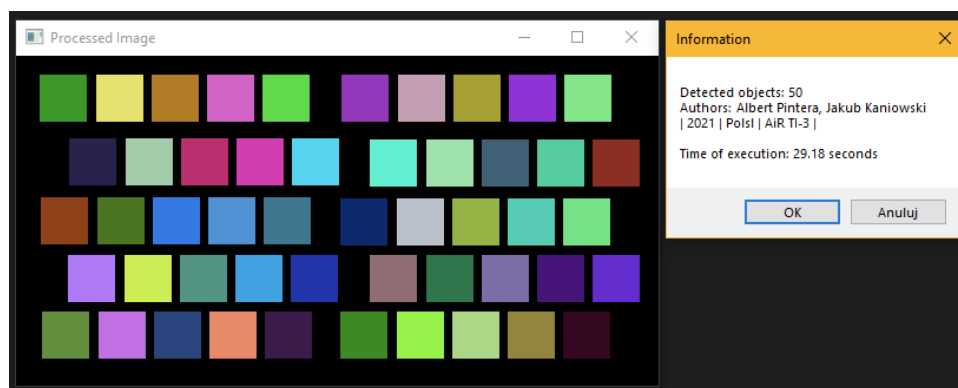
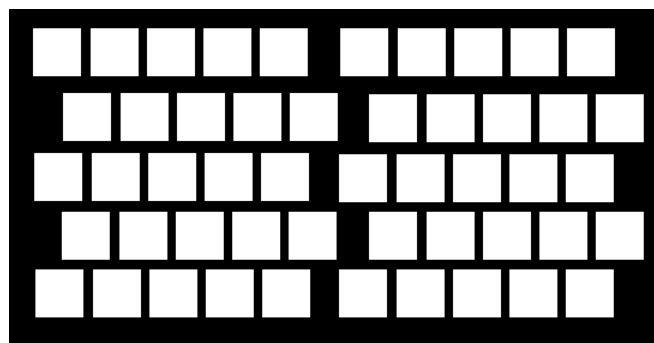
Obraz Shapes2.png. Obrazów rzeczywistych: 13. Wynik działania programu: 13 obiektów



Obraz Shapes3.png. Obrazów rzeczywistych: 13. Wynik działania programu: 13 obiektów

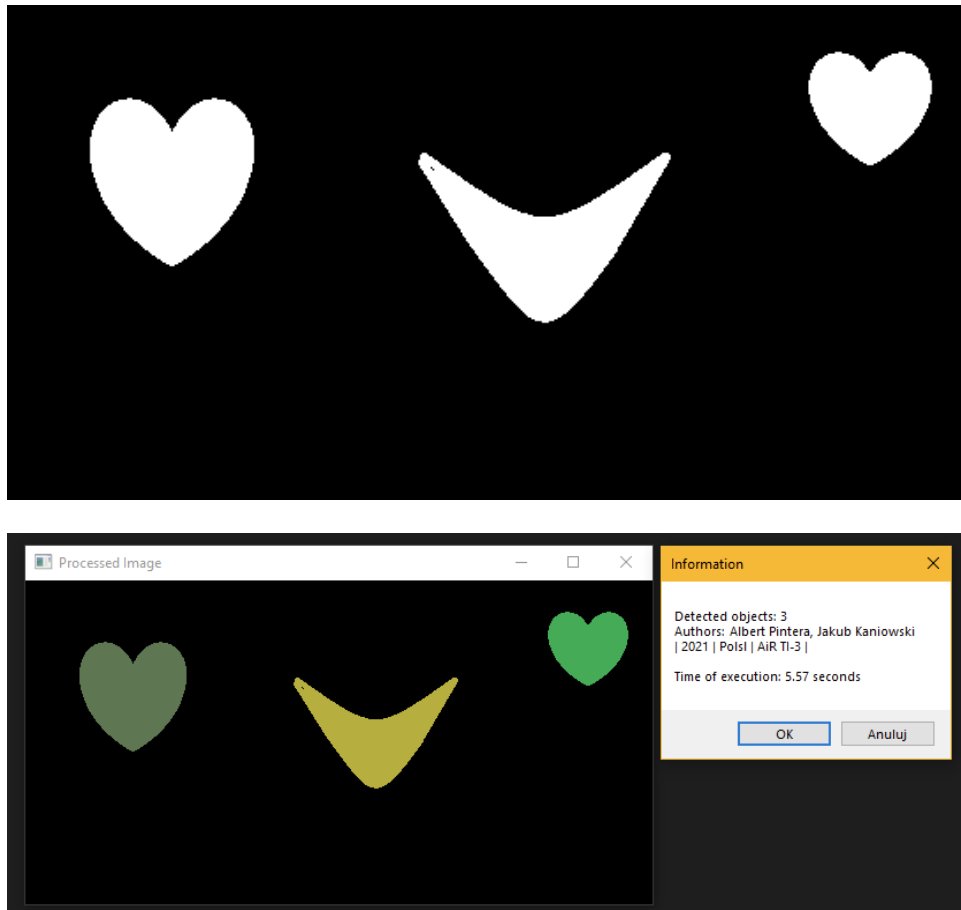


Obraz Shapes4.png. Obrazów rzeczywistych: 50. Wynik działania programu: 50 obiektów



Obraz Shapes5.png. Obrazów rzeczywistych 3. Wynik działania programu: 3.

Co bardzo istotne, w pierwotnym programie, wykryto błąd - program napotykał problem z kolorowaniem obiektów wklęsłych i niektórych wypukłych np. serc. Rozwiązaniem tego problemu okazała się zmiana koncepcji przechodzenia algorytmu przez obraz. Dodano kolejną pętlę, tym razem przechodzącą od dołu do góry obrazu. Dotychczasowe pętle, które wstępnie kolorowały obraz przechodziły z góry na dół obrazu. Taka implementacja opisywanej metody, pozwoliła na wyeliminowanie poruszonego problemu.



Wnioski

Program wykonany w ramach projektu spełnia stawiane mu wymagania. Algorytm koloruje obiekty na obrazie, ale również zwraca ich liczbę. Zaimplementowane operacje wykonują się stosunkowo szybko, jednakże czas ten zależy w dużej mierze od rozmiaru oraz złożoności obrazu wejściowego (tj. Liczby białych i czarnych elementów). Działanie programu zweryfikowano wykorzystując metodę biblioteki OpenCV *cv2.connectedComponents()* [5], która stanowi funkcję realizującą algorytm etykietowania. Wyniki gotowej funkcji porównywane z napisanym programem były spójne, co oznacza poprawne działanie.

Bibliografia

- (1) https://en.wikipedia.org/wiki/Connected-component_labeling
(dostęp: 29.10.2021 r.)
- (2) <https://docs.python.org/3.9/>
(dostęp: 10.11.2021 r.)
- (3) https://docs.opencv.org/4.x/d2/d96/tutorial_py_table_of_contents_imgproc.html
(dostęp: 10.11.2021 r.)
- (4) <https://iq.opengenus.org/connected-component-labeling/>
(dostęp: 13.11.2021 r.)
- (5) https://docs.opencv.org/4.x/d3/dc0/group_imgproc_shape.html#gaedef8c7340499ca391d459122e51bef5
(dostęp: 10.11.2021 r.)

Dodatek A. Kod źródłowy programu

Wersja czytelna dostępna w załącznikach (main.py)

```
# Politechnika Slaska - Wydział Automatyki Elektroniki i Informatyki
# Autorzy: Albert Pintera, Jakub Kaniowski
# 2021

import cv2
import random
import numpy as np
import ctypes
import time

start_time = time.time()    #Rozpoczęcie zliczania czasu wykonania programu

counter_of_objects = 0     #Licznik obiektów
list_of_colors = []        #Lista kolorów

#Lista kolorów - pierwszy nowy kolor
new_color = list(np.random.choice(range(1, 255, 1), size=3, replace=False))

image = cv2.imread('shapes5.png', 0)    #Wczytanie obrazu (ścieżka)
image = cv2.cvtColor(image, cv2.COLOR_GRAY2RGB)    #Zamiana barw na obrazie

#Wymiary obrazu
rows = image.shape[0]
cols = image.shape[1]

cols1 = image.shape[1]-1

#Funkcja losująca kolejne kolory
def generateColor(rng, i):
    global counter_of_objects
    return (np.random.choice(range(1, (rng-1), 1), size=i, replace=False))

for i in range(rows):
    for j in range(cols):

        if image[i][j][0] == 255 and image[i][j][1] == 255 and image[i][j][2] == 255:
            # 255 - biały kolor
            image[i][j] = new_color

            if image[i][j+1][0] == 0 and image[i][j+1][1] == 0 and image[i][j+1][2] == 0:
                # 0 - czarny kolor
                new_color = generateColor(256,3)
            #Losuje nowy kolor jak znajdzie nowy biały kolor

            if(i>0):
                if (image[i-1][j][0] != 0 and image[i-1][j][1] != 0 and image[i-1][j][2] !=
0):
                    #Algorytm rozgląda się wyżej, na poprzedni wiersz od aktualnego miejsca
                    image[i][j] = image[i-1][j]

                elif(image[i][j-1][0] != 0 and image[i][j-1][1] != 0 and image[i][j-1][2] !=
0):
                    #Algorytm rozgląda się w lewo od aktualnego miejsca
                    image[i][j] = image[i][j-1]

                elif(image[i-1][j-1][0] != 0 and image[i-1][j-1][1] != 0 and image[i-1][j-
1][2] != 0):
                    #Algorytm rozgląda się w lewo w poprzednim wierszu od aktualnego miejsca (lewy
górny róg)
                    image[i][j] = image[i-1][j-1]

                elif(image[i-1][j+1][0] != 0 and image[i-1][j+1][1] != 0 and image[i-
1][j+1][2] != 0):
                    #Algorytm rozgląda się w prawo w poprzednim wierszu od aktualnego miejsca
(prawy górny róg)
                    image[i][j] = image[i-1][j+1]

            #Ponowny odczyt wiersza, z przeciwnej strony (od prawej do lewej - ujednolici kolorów)
            for k in range(cols1):
                if(k<cols1):
```

```

        if(image[i][cols1-k-1][0] == 0 and image[i][cols1-k-1][1] == 0 and image[i][cols1-
k-1][2] == 0):
            continue
        if((image[i][cols1-k][0] != 0 and image[i][cols1-k][1] != 0 and image[i][cols1-
k][2] != 0) and (image[i][cols1-k][0] != 255 and image[i][cols1-k][1] != 255 and
image[i][cols1-k][2] != 255)):
            image[i][(cols1-k)-1] = image[i][cols1-k]

#Eliminacja problemu "bananowego" - algorytm przechodzi przez obraz od dołu do góry.
for i in range(rows-1, 0, -1):
    for j in range(cols-1, 0, -1):

        if(image[i][j][0] == 0 and image[i][j][1] == 0 and image[i][j][2] == 0):
            continue
        if((image[i][j][0] != 0 and image[i][j][1] != 0 and image[i][j][2] != 0) and
(image[i][j][0] != 255 and image[i][j][1] != 255 and image[i][j][2] != 255) and (image[i][j-
1][0] != 0 and image[i][j-1][1] != 0 and image[i][j-1][2] != 0)):
            image[i][j-1] = image[i][j]

        if (image[i+1][j][0] != 0 and image[i+1][j][1] != 0 and image[i+1][j][2] !=
0):
            image[i][j] = image[i+1][j]
            elif(image[i+1][j+1][0] != 0 and image[i+1][j+1][1] != 0 and
image[i+1][j+1][2] != 0):
                image[i][j] = image[i+1][j+1]

#Ponowny odczyt całego obrazu w celu zliczenia występujących obiektów
for i in range(rows):
    for j in range(cols):
        if (image[i][j][0] != 255 and image[i][j][1] != 255 and image[i][j][2] != 255) and
(image[i][j][0] != 0 and image[i][j][1] != 0 and image[i][j][2] != 0):
            if any(np.array_equal([image[i][j][0],image[i][j][1],image[i][j][2]], x) for x in
list_of_colors):
                continue
            else:
                list_of_colors.append(np.array([image[i][j][0],image[i][j][1],image[i][j][2]]))

counter_of_objects = len(list_of_colors)    #Przypisanie liczby zliczonych kolorów do zmiennej
przechowującej liczbę obiektów

#Wyswietlenie uzyskanych wyników
cv2.imshow('Processed Image', image)

#Dla każdego systemu
print("\n\nProgram info: \n\nDetected objects: " + str(counter_of_objects) + "\nAuthors:
Albert Pintera, Jakub Kaniowski\n| 2021 | Polsl | AiR TI-3 |" + "\n\nTime of execution: " +
str(float("{0:.2f}".format(time.time() - start_time))) + " seconds")

#Dla windowsa (okomentowac w razie potrzeby)
ctypes.windll.user32.MessageBoxW(0, "Detected objects: " + str(counter_of_objects) +
"\nAuthors:\tAlbert Pintera, Jakub Kaniowski\n| 2021 | Polsl | AiR TI-3 |" + "\n\nTime of
execution: " + str(float("{0:.2f}".format(time.time() - start_time))) + " seconds",
"Information " , 1)

cv2.waitKey(0)
cv2.destroyAllWindows()

```