POLITECHNIKA WARSZAWSKA

CYFROWE METODY PRZETWARZANIA OBRAZU/ WIDZENIE MASZYNOWE

PROJEKT 2

Detekcja znaków drogowych

MARCIN SZYMCZAK 314910

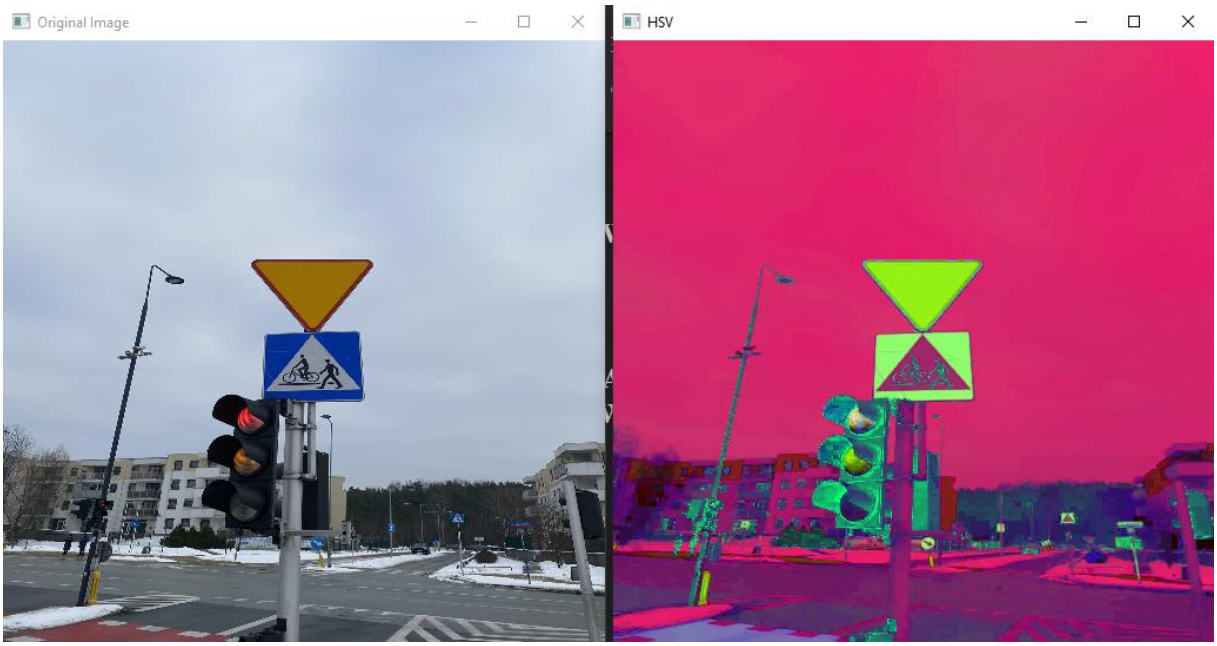
Detekcja znaku ustąp pierwszeństwa

Cały projekt zamieściłem w klasie Project_2, która w konstruktorze wymaga podania ścieżki do zdjęcia, a następnie zmienia wymiar zdjęcia na (1024, 1024).

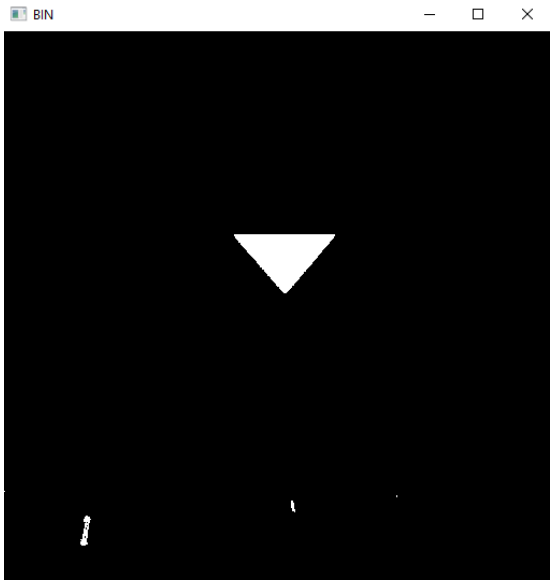
```
int main() {
    std::string path = "C:/Users/marcin/Desktop/zakaz3.jpg";
    Project_2 result(path);
    result.yellow_triangle();
    result.red_circle();

    return 0;
}
```

Funkcja do detekcji znaku – yellow_triangle()

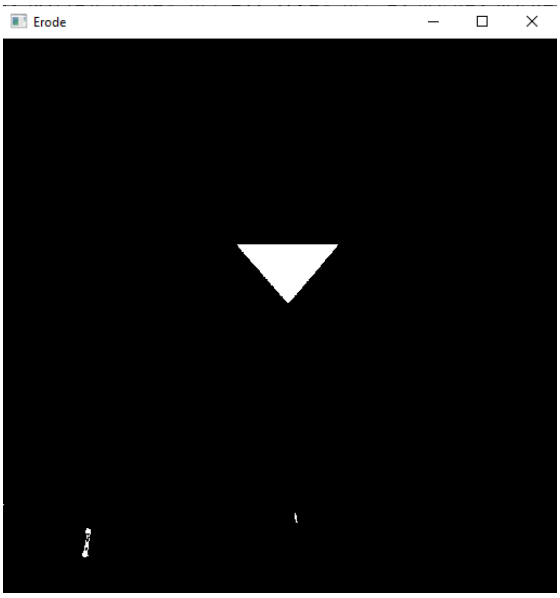
Detekcje znaku zaczynamy od zmiany z palety RGB (w OpenCV z BGR) na HSV.



Następnie tworzymy maskę dla koloru żółtego i na jej podstawię robimy binaryzację, czyli pixele koloru żółtego mają wartość 255, a reszta wartość 0.



Następnie wykonywana jest operacja erozji w celu otrzymania lepszego kształtu oraz pozbycia się szumu.



Następnie w masce znajdujemy kontury, ale tylko zewnętrzne (flaga cv::RETR_EXTERNAL).

```
//find all contours
std::vector<std::vector<cv::Point>> contours;
cv::findContours(erode_mat, contours, cv::RETR_EXTERNAL, cv::CHAIN_APPROX_SIMPLE);
```

Następnie po wszystkich znalezionych konturach wyszukuję tylko te, które po przybliżeniu funkcją cv::approxPollyDP mają 3 wierzchołki, czyli trójkąty.
Następnie wykorzystując fakt, że jest tylko jeden taki znak który nie posiada żadnego symbolu w sobie (jest tylko kolor żółty), szukam żółtego trójkąta który ma największą ilość żółtych pixeli na zdjęciu. Jak znajdziemy największą ilość żółtych pixeli w trójkącie to zapisujemy jego index z wektora konturów, a następnie rysujemy kontur tego trójkąta.

```
//find contour with the biggest amount of yellow pixels (pixel > 255)
int max_amount_of_white_pixels = 0;
int max_index = 0;


for (size_t i = 0; i < contours.size(); i++) {

    //additional check if the contour is a triangle
    std::vector<cv::Point> approx;
    float epsilon = 0.02 * cv::arcLength(contours[i], true);
    cv::approxPolyDP(contours[i], approx, epsilon, true);


    if (approx.size() == 3) {
        cv::Rect roi_rect = cv::boundingRect(contours[i]);
        cv::Mat roi = erode_mat(roi_rect);

        int white_pixels = cv::countNonZero(roi);

        if (white_pixels > max_amount_of_white_pixels) {
            max_index = i;
            max_amount_of_white_pixels = white_pixels;
        }
    }

}
cv::drawContours(img, contours, max_index, cv::Scalar(0, 255, 0), 2);
```

Kolejnym etapem jest znalezienie środka konturu.

```
cv::Moments moment = cv::moments(contours[max_index]);
int center_x = moment.m10 / moment.m00;
int center_y = moment.m01 / moment.m00;
```

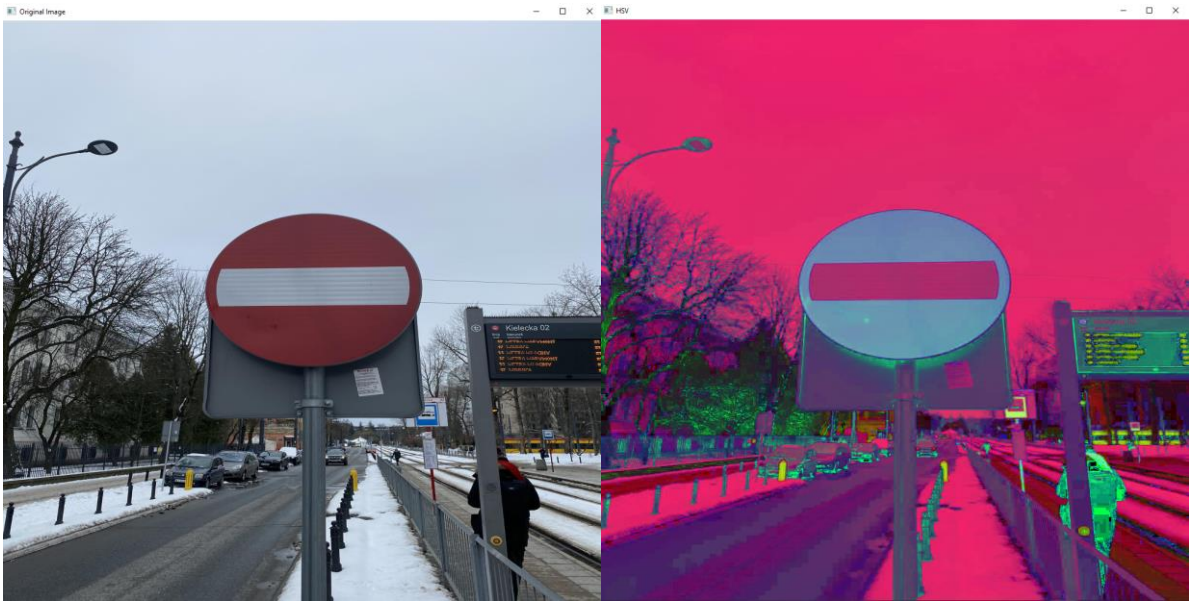
Następnie rysujemy środek konturu i piszemy jego współrzędne w lewym górnym rogu.

Detekcja znaku zakaz wjazdu

Detekcje tego znaku zaczynamy od wywołania funkcji red_circle().

```
int main() {
    std::string path = "C:/Users/marcin/Desktop/lmao.jpg";
    Project_2 result(path);
    //result.yellow_triangle();
    result.red_circle();

    return 0;
}
```

Detekcje zaczynamy od konwersji zdjęcia z palety barw RGB na HSV.



Następnie tworzymy maskę dla koloru czerwonego i binaryzujemy ją, czyli pixele mieszczące się w masce zamieniamy na wartość 255, resztę na 0.

```
//cv::Scalar lower_threshold1 = cv::Scalar(0, 100, 20);
//cv::Scalar upper_threshold1 = cv::Scalar(10, 255, 255);
cv::Scalar lower_threshold2 = cv::Scalar(150, 40, 0);
cv::Scalar upper_threshold2 = cv::Scalar(179, 255, 255);

//mask for red color
cv::Mat red_mask;
//cv::inRange(hsv, lower_threshold1, upper_threshold1, red_mask1);
cv::inRange(hsv, lower_threshold2, upper_threshold2, red_mask);
//cv::Mat red_mask = red_mask2; | red_mask2;

//red color = 255, rest = 0
cv::Mat binarization_red;
cv::threshold(red_mask, binarization_red, 0, 255, cv::THRESH_BINARY);
cv::imshow("BIN", binarization_red);
cv::waitKey(0);
```

Następnie dokonujemy operacji dylatacji aby pozbyć się dziur w znaku. (w szczególności dla zdjęcia nr 2 😊)

```
//dilate to rid off holes in sign
cv::Mat dilate_mat;
cv::Mat kernel = cv::getStructuringElement(cv::MORPH_ELLIPSE, cv::Size(5, 5));
cv::dilate(binarization_red, dilate_mat, kernel);
cv::imshow("Erode", dilate_mat);
cv::waitKey(0);
```

Kolejnym etapem jest znalezienie konturów ale w odpowiedniej hierarchii za pomocą flagi cv::RETR_TREE, która zwraca nam kontury i jej potomki, które są konturami wewnątrz tego konturu.
Znak zakaz wjazdu jest to okrąg, który posiada wewnątrz tylko 1 kontur, który jest prostokątem. Czyli detekcja polega na tym, że znajdujemy okrąg. Następnie sprawdzamy czy ma tylko 1 potomka (1 kontur w sobie), następnie sprawdzamy czy ten kontur jest

prostokątem (czy ma 4 wierzchołki). Jeżeli uda się nam taki znak znaleźć to jeszcze sprawdzamy pole prostokąta. Akurat dla tego znaku to pole musi być największe ponieważ inne znaki mają w sobie kontury o 4 krawędziach, ale nie są one tak duże jak nasz szukany prostokąt.

```
//find all contours
std::vector<std::vector<cv::Point>> contours;
std::vector<cv::Vec4i> hierarchy;
cv::findContours(dilate_mat, contours, hierarchy, cv::RETR_TREE, cv::CHAIN_APPROX_SIMPLE);

int max_area_of_children = 0;
int index_of_max_area_of_children = 0;
for (size_t i = 0; i < contours.size(); i++) {
    std::vector<cv::Point> approx;
    float epsilon = 0.02 * cv::arcLength(contours[i], true);
    cv::approxPolyDP(contours[i], approx, epsilon, true);
    std::cout << approx.size() << std::endl;
    int child = hierarchy[i][2];
    //find contours that are circles
    if (approx.size() >= 8 && cv::contourArea(contours[i]) > 200) {

        int number_of_children = 0;

        while (child != -1) {
            number_of_children++;
            child = hierarchy[child][0];
        }
        //find circles with only one contour that is the biggest rectangle
        if (number_of_children == 1) {
            child = hierarchy[i][2];
            std::vector<cv::Point> approxchild;
            float epsilonchild = 0.02 * cv::arcLength(contours[i], true);
            cv::approxPolyDP(contours[child], approxchild, epsilonchild, true);
            std::cout << approxchild.size() << std::endl;
            if (approxchild.size() == 4) {
                float area_children = cv::contourArea(contours[child]);
                if (area_children > max_area_of_children) {
                    max_area_of_children = area_children;
                    index_of_max_area_of_children = i;
                }

            }

        }

    }
```

Ostatnim punktem jest narysowanie konturu oraz wyliczenie środka konturu i wypisanie jego współrzędnych w lewym górnym rogu obrazka.

Zdjęcia użyte do programu:

1. Znak ustąp pierwszeństwa

- ustap1.jpg – normalne zdjęcie
- ustap2.jpg – inna perspektywa
- ustap3.jpg – zdjęcie z oddali
- znaki.jpg – tablica znaków

2. Znak zakaz wjazdu

- zakaz1.jpg – normalne zdjęcie
- zakaz2.jpg – inna perspektywa
- zakaz3.jpg – zdjęcie z oddali
- znaki-zakazu.png – tablica znaków

Wnioski

- Wykrywanie znaków bazujące na kolorze bardzo zależy od wykonanych zdjęć, przykład dla znaku zakazu 2 zdjęcie. Od górnej części znaku odbija się mocniej światło przez co dobranie zakresu HSV dla maski było strasznym zadaniem.
- Dla różnych warunków pogodowych te metody nie były by prawdopodobnie najlepszym rozwiązaniem.
- Zaproponowane metody nie są w stanie wykryć więcej niż jednego znaku na zdjęciu☹.