POLITECHNIKA WARSZAWSKA
WYDZIAŁ MECHATRONIKI

Cyfrowe Przetwarzanie Obrazy / Widzenie maszynowe

Sprawozdanie

Projekt PII

Rozpoznawanie kart do gry „Uno”

Prowadzący:

mgr inż. Damian Krawczyk

mgr inż. Filip Brzeski

Wykonała: Miłosława Wilkowiecka (297383)

Warszawa 2022

1. Opis teoretyczny

Do analizy wykorzystano

Do analizy otrzymano 4 sceny w 4 różnych wersjach (oryginał, blur, sól i pieprz, gradient) – łącznie 16 obrazów.

4 oryginalne sceny zostały wykonane przez nas indywidualnie podczas zajęć laboratoryjnych. Zdjęcia wykonywane były z góry, dzięki czemu nie musieliśmy przejmować się wypaczeniem perspektywistycznym powstałych zdjęć. Po wyborze 3 cyfr znajdujących się na kartach oraz dodając do tego karty specjalne *Reverse* oraz *Stop*, otrzymaliśmy zestaw łącznie 5 różnych symboli na karatach w 4 różnych kolorach, co sumarycznie daje nam 20 różnych kart. Zanieczyszczenia takie jak blur, sól i pieprz czy gradien najprawdopodobniej zostały dodane komputerowo.

Ze względu na naturę projektu korzystanie z Template Matchingu jak i analizy zdjęć na podstawie ich nazwy zostało zabronione. Komentarze dotyczące kodu umieszczone są w pliku *.cpp przesłanym wraz ze sprawozdaniem oraz w punkcie czwartym sprawozdania.

2. Schemat blokowy

Schemat blokowy przedstawiony jest na ostatniej stronie sprawozdania, przed oświadczeniem o samodzielności.

3. Opis problemów i ich rozwiązanie

Spośród 16 otrzymanych zdjęć, 12 z nich zostało poddane dodatkowej obróbce (blur, sól i pieprz, gradient). Dodatkowo, wszystkie zdjęcia miały delikatny, zielonkawy odcień, co może być błędem kamery. Każde ze zdjęć posiada wymiary 2590x1942 pixele. Aby odpowiednio móc analizować i interpretować wyniki otrzymane z analizy kart, należało je najpierw obrócić, i przenieść do wybranego przez nas punktu. Znając położenie czterech rogów każdej z kart, byłoby to zadanie banalnie proste, jednak należało użyć do tego funkcji, a nie wpisywania „z palca”. Aby osiągnąć odpowiedni efekt należało znaleźć kontury poszczególnych kart. Przy kartach bez dodatkowych zanieczyszczeń znalezienie konturów, a co za tym idzie punktów granicznych kart było proste. Jednak należy wziąć pod uwagę zaistniałe zanieczyszczenia obrazu.

- Blur – rozmycie obrazu. Do zniwelowania tego efektu posłużyłam się filtrem unsharpMask przy którym wykorzystujemy GaussianBlur() oraz funkcję addWeighted(). Ze względu na znaczne rozmazanie obrazu zdecydowałam się na zastosowanie tego filtru trzykronie.
- Sól i pieprz – jeden z rodzajów szumu polegający na zamianie nasycenia losowych pixeli w obrazie na 0 (czarny pixel) lub 255 (biały pixel). Aby dokładnie pozbyć się tego efektu zastosowałam filtr medianowy medianBlur() o rozmiarze kernela 7. Ze względu na wielokanałowość obrazu, minimalny rozmiar kernela wynosił 5, jednak to wielkość 7 przyniosła najlepsze efety.
- Gradient – jedna część obrazu oświetlona słabiej niż pozostałe. Najlepszym wyjściem byłoby użycie normalizacji histogramu lub algorytmu CLAHE (Contrast Limited Adaptative Histogram Equalization). Rozwiązaniem problemu okazało się obliczenie niego nasycenia pixeli obrazu w skali szarości, który po użyciu funkcji threshold() miał nam służyć do analizy i interpretacji symboli. Jako dolną granicę funkcji threshold() ustawiłam średnie nasycenie zwiększone o 40. Byłą to wystarczająco dużą różnica, aby bez problemu wykonać funkcję threshold() nawet na żółtej karcie, która była najjaśniejsza.

4. Dobór algorytmów

Funkcja blurDetector() wykorzystująca Laplacian pozwala nam stwierdzić, czy obraz jest rozmyty, czy ostry. Ważnym jest, aby obraz poddawany takiej analizie był jednokanałowy – najlepiej w skali szarości, gdyż znacznie przyspiesza to analizę.

```
bool blurDetector(Mat& imgGray, Mat& lapl) //zwraca wartośc true jeśli obraz jest rozmazany, false jeśli nie jest
{
    Laplacian(imgGray, lapl, CV_32FC1);
    Scalar mean, stddev;
    meanStdDev(lapl, mean, stddev, imgGray);
    double variance = stddev.val[0] * stddev.val[0];
    double threshold = 10;
    if (variance <= threshold) return true;
    else return false;
}
```

Funkcja PreProcessing() ma na celu przygotowanie całej sceny do znalezienia konturów, a także stworzenie obrazy Canny() który zlokalizuje nam krawędzie kart. W przypadku, gdy funkcja blurDetector() zwróci *true*, trzykrotnie wykonany zostanie filtr *unsharpMask.* Taka ilość operacji zapewni nam optymalne rozwiązanie.

```
void PreProcessing(Mat img, Mat& imgCanny, Mat& imgGray)
{
    Mat tmp,lapl;   //Macierze tymczasowe, do wykonywania operacji w funckji
    cvtColor(img, imgGray, COLOR_BGR2GRAY); //Konwertowanie na skalę szarości
    blurDetector(imgGray, lapl);    //wykrywanie rozmazania
    double alpha = 4;   //parametr alpha
    int threshUp = 200; //górna wartość thresholdu
    int threshDown = 100;   //dolna wartość threshldu

    if (blurDetector(imgGray, lapl))
    {
        for (int i = 0; i < 3; i++)
        {
        //zastosowanie filtru unsharpmask dla najlepszych rezultatów - trzykrotne
        GaussianBlur(imgGray, tmp, Size(301, 301), 2.0, 2.0);
        addWeighted(imgGray, 1 + alpha, tmp, -alpha, 0, imgGray);
        }
        threshDown = 140; // nowa dolna wartość thresholu - jeśli obraz rozmazany
        threshUp = 255; //nowa górna wartosc thresholdu - jeśli obraz rozmazany
        medianBlur(imgGray, imgGray, 5);    //usunięcie możliwego, nowopowstałego szumu
    }
    else
    {
        medianBlur(imgGray, imgGray, 7); //usunięcie szumu
    }

    Canny(imgGray, imgCanny, threshDown, threshUp); //stworzenie obrazu imgCanny

}
```

Funkcja ImgCountours() pozwala nam na znalezienie krawędzi na obrazie. Operacja dylatacji wykonana jest dwukrotnie, aby zapewnić zamkniecie wszystkich szczelin nie narażając obrazu naw na powiększenie szumów.

```
void ImgCountours(Mat& img, Mat& imgGray, Mat& imgCanny, vector<Mat>& imgWarp, vector<RotatedRect>& MinRect)
{
    vector<vector<Point>> contours; //utworzenie stuktury konturów
    vector<Vec4i> hierarchy;    //utworzenie struktury hierarchii
    Mat kernel = getStructuringElement(MORPH_RECT, Size(3, 3)); //stworzenie elementu strukturalnego do operacji dylatacji
    dilate(imgCanny, imgCanny, kernel); //operacja dylatacji
    dilate(imgCanny, imgCanny, kernel); //powtórzenie operacji dyatacji

    findContours(imgCanny, contours, hierarchy, RETR_EXTERNAL, CHAIN_APPROX_SIMPLE); //znajdowanie konturów
    vector<RotatedRect> minRect(contours.size());
    vector<vector<Point>> conPoly(contours.size());
    vector<Rect> boundRect(contours.size());    //stworzenie elementów potrzebnych do dalszych operacji
    int area;   //wartość obszaru ogarniczonego konturami
    Point2f src[4] = { {0.0f,0.0f}, {0.0f,0.0f},{0.0f,0.0f},{0.0f,0.0f} };  //współrzędne źródła
```

Do zaimplementowanej zmiennej area zostają po kolei zapisane wszystkie rozmiary obszarów otoczonych konturami. Podczas gdy niewielkie zabrudzenia lub mniejsze elementy kart mają wielkość powierzchni do ok. 20 000, same karty mają wielkość powierzchni ok. 350 000. Dlatego właśnie zdecydowałam się na granicę 300 000 jako granicę wejścia do dalszej obróbki i analizy. Dzięki temu wszystko wewnątrz, wykonuje się 4 razy (1 raz dla 1 karty). Pętla ze zmienną *j* pozwala nam na narysowanie konturów, poprzez połączenie linią kolejnych punktów granicznych wcześniej znalezionego konturu. Kontury te postanowiłam zaznaczyć kolorem pomarańczowym. Na wynikach końcowych doskonale widać linię konturową.

```
for (int i = 0; i < contours.size(); i++)   //przejście przez wszystkie kontury
{
    area = contourArea(contours[i]);    //przypisanie wartości obszaru ograniczonemu konturami

    if (area >= 300000) //warunek rozmiaru obszaru
    {
        minRect[i] = minAreaRect(contours[i]);
        Point2f crop_points[4];
        minRect[i].points(crop_points);
        for (int j = 0; j < 4; j++)
        {
            line(img, crop_points[j], crop_points[(j + 1) % 4], Scalar(0, 125, 255), 15);   //rysowanie obwodu łącząc punkty
        }
```

Następuje sprawdzenie czy karta domyślnie odwrócona jest pionowo, czy poziomo (z naszego punktu widzenia), a także w przypadku orientacji poziomej, zmienia ją na pionową obracając kartę.

```
//sprawdzenie i poprawa orientacji znalezionej karty (z poziomej na pionową
if (norm(crop_points[0] - crop_points[1]) < norm(crop_points[0] - crop_points[3]))
{
    for (int j = 0; j < 4; j++)
    {
        src[j].x = crop_points[j].x;
        src[j].y = crop_points[j].y;
    }
}
else
{
    for (int j = 0; j < 4; j++)
    {
        if (j == 0)
        {
            src[j].x = crop_points[j + 3].x;
            src[j].y = crop_points[j + 3].y;
        }
        else
        {
            src[j].x = crop_points[j - 1].x;
            src[j].y = crop_points[j - 1].y;
        }
    }
}
```

Następuje zapisanie punktów granicznych znalezionej karty do wektora. Przy niedomkniętych konturach może pojawić się błąd *vector subscript out of range*. Podwójna dylatacja zapobiega temu błędowi. Jest on najbardziej możliwy przy obrazach z nałożonym efektem *blur*.

```
            Mat matrix, Img_warp;
            matrix = getPerspectiveTransform(src, dst);
            warpPerspective(img, Img_warp, matrix, Point(w, h));    //przesunięcie karty do początku układu współrzędnych
            imgWarp.push_back(Img_warp);    //push_back zapobiega błędowi braku miejsca w wektorze
        }
    }

    MinRect = minRect;

}
```

Utworzenie maski karty, w kształcie okręgu pozwala nam na jak najdokładniejsze wyeliminowanie z obszaru przetwarzanego, białego koloru znajdującego się przy krawędziach karty. Mógłby on znacznie wpłynąć na późniejszą analizę kształtów symboli.

```
void maska_karty(Mat& karta)    //maska wycinająca nam symbol z karty do dalszej operacji
{

    Point mid = { 70,65 };  //środek okręgu
    Mat mask, after;
    mask = Mat::zeros(h, w, CV_8UC3);   //stworzenie maski

    circle(mask, mid, 45, { 255,255,255 }, -1); //stworzenie okręgu
    bitwise_and(karta, mask, karta);    //zastosowanie maski

}
```

Funkcja PostCropping() analizuje kartę na wielu płaszczyznach. Najpierw, jeżeli wykryte zotsanie rozmycie, zostaje ono zminimalizowane filtem *unsharpMask*. Dzieje się tak dlatego, że pracujemy na wycinku oryginalnego obrazu na który dodatkowo nałożona została maska. Aby jednoznacznie zdefiniować kolor analizowanej karty, najpierw inicjalizujemy wartości *ch1,ch2,ch3* wszystkie równe zero, aby dla każdej kolejnej karty wartość ta liczona była od początku. Po wykonaniu operacji threshold() otrzymujemy obraz trójkanałowy, z każdym kanałem oddzielnie zbinaryzowanym kanałem, dzięki czemu kolor niebieski dla wszystkich trzech kanałów ma wartości (255,0,0); kolor zielony ma wartości (0,255,0); kolor czerwony (0,0,255); a kolor żółty (0,255,255). Biorąc pod uwagę wystąpienie białych pixeli których wartości to (255,255,255) oraz czarnych pixeli o wartośćiach (0,0,0), zdacydowałam się na ustalenie minimalnej wartości intensywności koloru na 150 – wartość *range*. Następnie przechodząc przez cały wycinek, sumowana jest intensywność na każdym z kanałów, a następnie dzielona przez liczbę pixeli w wycinku.

```
void PostCropping(Mat& img, Mat& pre, string& color, string& symbol) //odnajdywanie koloru karty
{
    double alpha = 4;   //współczynnik alfa
    double ch1 = 0;     //wartość intensywności pierwszego kanału (B)
    double ch2 = 0;     //wartość intensywności drugiego kanału (G)
    double ch3 = 0;     //wartość intensywności trzeciego kanału (R)
    Mat tmp;            //macierz tymczasowa do dalszych operacji
    int range = 150;    //wartość którą musi przekroścyć średnia intensywnośc kanału, aby uznać, że dany kolor występuje
    if (blurDetector(img, tmp))     //wykrywanie rozmycia obrazu
    {
        for (int p = 0; p < 3; p++) //wyostrzanie wycietego fragmentu
        {
            GaussianBlur(pre, tmp, Size(3, 3), 1, 1);
            addWeighted(pre, 1 + alpha, tmp, -alpha, 0, pre);
        }
    }
    threshold(pre, tmp, 90, 255, THRESH_BINARY);    //binaryzacja (wszystkie kanały jeden po drugim)
    for (int i = 0; i < tmp.rows; i++)
    {
        for (int j = 0; j < tmp.cols; j++)  //przejście po całym wycinku i zliczenie sumarycznej wartośći intensywności dla poszczególnych kanałów
        {
            ch1 = ch1 + tmp.at<Vec3b>(i, j)[0];
            ch2 = ch2 + tmp.at<Vec3b>(i, j)[1];
            ch3 = ch3 + tmp.at<Vec3b>(i, j)[2];
        }
    }
    alpha = tmp.rows * tmp.cols;    //ponowne użycie współczynnika alfa aby nie tworzyć niepotrzebnych zmiennych
    ch1 = ch1 / alpha;              //wartość sredniej intensywności pierwszego kanału
    ch2 = ch2 / alpha;              //wartość sredniej intensywnośći drugiego kanału
    ch3 = ch3 / alpha;              //wartość średniej intensywności trzeciego kanału
```

Jeżeli średnia wartość nasycenia w niebieskim kanale jest większa niż 150, jest to karta niebieska. Podobnie robimy z pozostałymi kanałami.

```
    //warunkowe przypisanie koloru w zależności od średniej intensywności kanałów
    if (ch1 >= range)
    {
        color = "Blue";
    }
    else
    {
        if (ch2 >= range)
        {
            if (ch3 >= range)
            {
                color = "Yellow";
            }
            else
            {
                color = "Green";
            }
        }
        else
        {
            color = "Red";
        }
    }
    momenty(pre, symbol);   //wyliczenie HuMoments dla tych samych wycinków
}
```

Funkcja momenty() służy nam za przygotowanie wycinka, o znanym już kolorze do wyliczenia Momentów Hu. Momenty Hu można liczyć na biało-czarnym obrazie, stąd potrzeba zmiany obrazu na obraz w skali szarosci. Aby zapewnić optymalne odcięcie wcześniej kolorowych pixeli od koloru białego, dolną granicą funckji threshold jest średnie nasycenie pixeli obrazu w odcieniach szarości zwiększone o 40. Zbinaryzowany w ten sposób wycinek zostaje ponownie poddany filtracji medianowej medianBlur(), aby usunąć białe szumy utrudniające analizę obrazu, a także domknąć pojedyncze pixele. Do analizy kształtów będziemy brali pod uwagę Moment Hu nr 2 oraz Moment Hu numer 3, które ze względu na numerowanie od zera, mają iterację odpowiednio huMomenty[1] i huMomenty[2]. Odpowiednio przeliczone momenty przepisujemy do zmiennych *mom2* i *mom3*.

```
string momenty(Mat& pre, string& symbol)    //znajdywanie HuMoments dla wycinka z maską
{
    float mom2, mom3;   //zmienne przekazujące dalej HuMoment[1] i HuMoment[2]
    Mat post;   //obraz tymczasowy, do konwersji na skale szarości
    int med = 0;    //implementacja średniej wartości nasycenia - w skali szarości
    //ponieważ będzie ona uzyta do funkcj threshold (niweluje błąd związany z gradientem)
    //dlatego jest liczbąc całkowitą
    cvtColor(pre, post, COLOR_BGR2GRAY);    //transformacja kolorystyczna do sklai szarości - obraz pre do post
    for (int i = 0; i < post.rows; i++)
    {
        for (int j = 0; j < post.cols; j++)
        {
            med = med + post.at<uchar>(i, j);   //zliczanie sumarycznej wartości wszystkich pixeli w wycinku - w skali szarości
        }
    }
    med = med / (post.rows * post.cols);    //wyliczenie średniej
    med = med + 40; //zwiększenie średniej o 40
    medianBlur(post, post, 7);  //filt medianowy medianowe  - niwelacja błedu szumu sól z pieprzem
    threshold(post, post, med, 255, THRESH_BINARY); //binaryzacja obrazu w zależności od średniego nasycenia pixeli
    medianBlur(post, post, 5);  //filt medianowy, do usunięcia pozostałych szumów i uzupełnienia ubytków
    Moments momenty = moments(post, false); //tworzenie momentów
    double huMomenty[7];
    HuMoments(momenty, huMomenty);  //wyliczanie momentów
    for (int i = 0; i < 7; i++)
    {
        huMomenty[i] = -1 * copysign(1.0, huMomenty[i]) * log10(abs(huMomenty[i])); //zapisywanie momentów
    }
    mom2 = huMomenty[1];//przypisanie wartości drugiego momentu do zmiennej mom2 przekazywanej dalej
    mom3 = huMomenty[2];//przypisanie wartości trzeciego momentu do zmiennej mom3 przekazywanej dalej
    Symbol(mom2, mom3, symbol); //odnajdywanie symboli
    return symbol;  //zwrot sybolu
}
```

Funkcja Symbol() porównuje wcześniej otrzymane *mom2* i *mom3* ze skrupulatnie przeanalizowanymi, wcześniej otrzymanymi momentami. Na podstawie tych dwóch Momentów Hu, możemy jednoznacznie określić symbol znajdujący się na karcie.

```
string Symbol(float& mom2, float& mom3, string& symbol) //przypisywanie symbolu do momentów hu
{
    if (((mom2 <= 5.1) || (mom2 >= 7.35 && mom2 <= 7.6)) && ((mom3 >= 7 && mom3 <= 7.1) || (mom3 > 11.5 && mom3 < 14.2)))
    {
        symbol = "0";
    }
    if (((mom2 > 5.65 && mom2 < 5.8)||(mom2>5.96&&mom2<6.66)) && ((mom3 > 8.3 && mom3 < 8.4) || (mom3 > 8.6 && mom3 < 9.35) || (mom3 >9.36 && mom3 <9.7 )))
    {
        symbol = "1";
    }
    if (((mom2>5.8 && mom2<5.9)||(mom2 > 7.25 && mom2 < 7.74)) && ((mom3 > 8.5 && mom3 < 8.7) || (mom3 > 10.5 && mom3 < 11.2)))
    {
        symbol = "8";
    }
    if ((mom2 > 5.9 && mom2 < 6.36) && ((mom3 > 8.4 && mom3 < 8.6) || (mom3 > 11.6 && mom3 < 13.1)))
    {
        symbol = "Reverse";
    }
    if (symbol == "symbol")
    {
        symbol = "Stop";
    }
    return symbol;  //zwrot znalezionego symbolu
}
```

W funkcji main() następują wszelkie deklaracje i odwołania do funkcji.

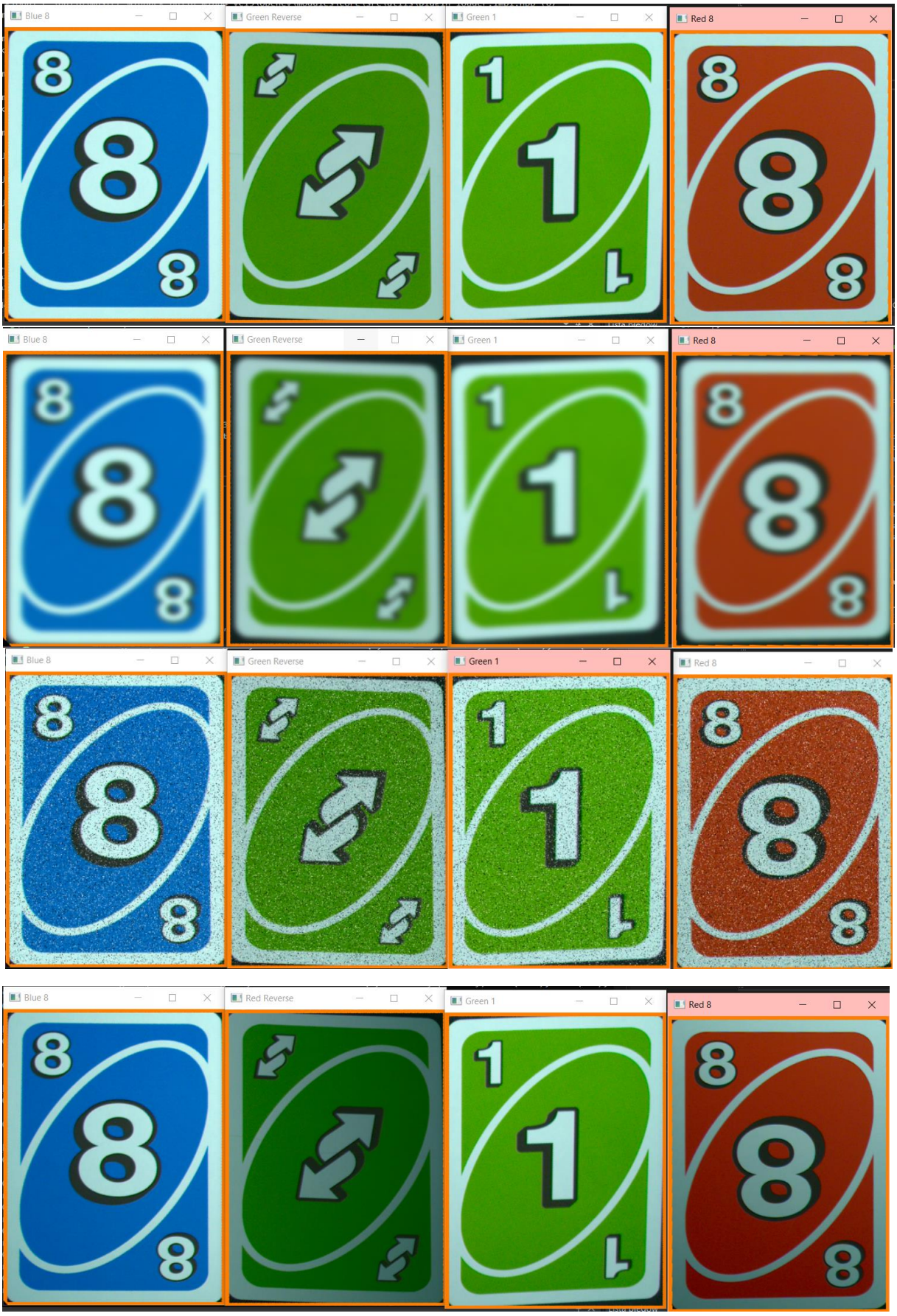
```
void main()
{

    string path = "images/4_gradient.png";      //wczytanie ścieżki obrazu
    Mat imgOriginal = imread(path);             //wczytanie obrazu
    vector<Mat> imgWarp;                        //stworzenie wektora do przekształcania kart
    Mat imgThre, imgCanny, imgPost;             //stworzenie macierzy na inne obrazy
    vector<RotatedRect> minRect;                //stworzenie wektora minRec
    string color1 = "color";                    //inicjalizacja kolorów dla 4 kart - domyślnie "color"
    string color2 = "color";
    string color3 = "color";
    string color4 = "color";
    string symbol1 = "symbol";                  //inicjalizacja symboli dla 4 kart - domyślnie "symbol"
    string symbol2 = "symbol";
    string symbol3 = "symbol";
    string symbol4 = "symbol";

    PreProcessing(imgOriginal, imgCanny, imgPost);  //PreProcessing, przygotowanie karty do znalezenia konturów
    ImgCountours(imgOriginal, imgPost, imgCanny, imgWarp, minRect); //znajdowanie konturów

    Mat karta1 = imgWarp[0];        //tworzenie i wpisywanie macierzy dla znalezionych kart
    Mat karta2 = imgWarp[1];
    Mat karta3 = imgWarp[2];
    Mat karta4 = imgWarp[3];

    Mat k1 = karta1.clone();        //tworzenie klonów znalezionych kart do późniejszego wyświetlenia
    Mat k2 = karta2.clone();
    Mat k3 = karta3.clone();
    Mat k4 = karta4.clone();
```

Po uzyskaniu informacji na temat symbolu i koloru karty, możemy wycięte wcześniej karty wyświetlić w oknach o nazwach składających się z koloru i symbolu danej karty.

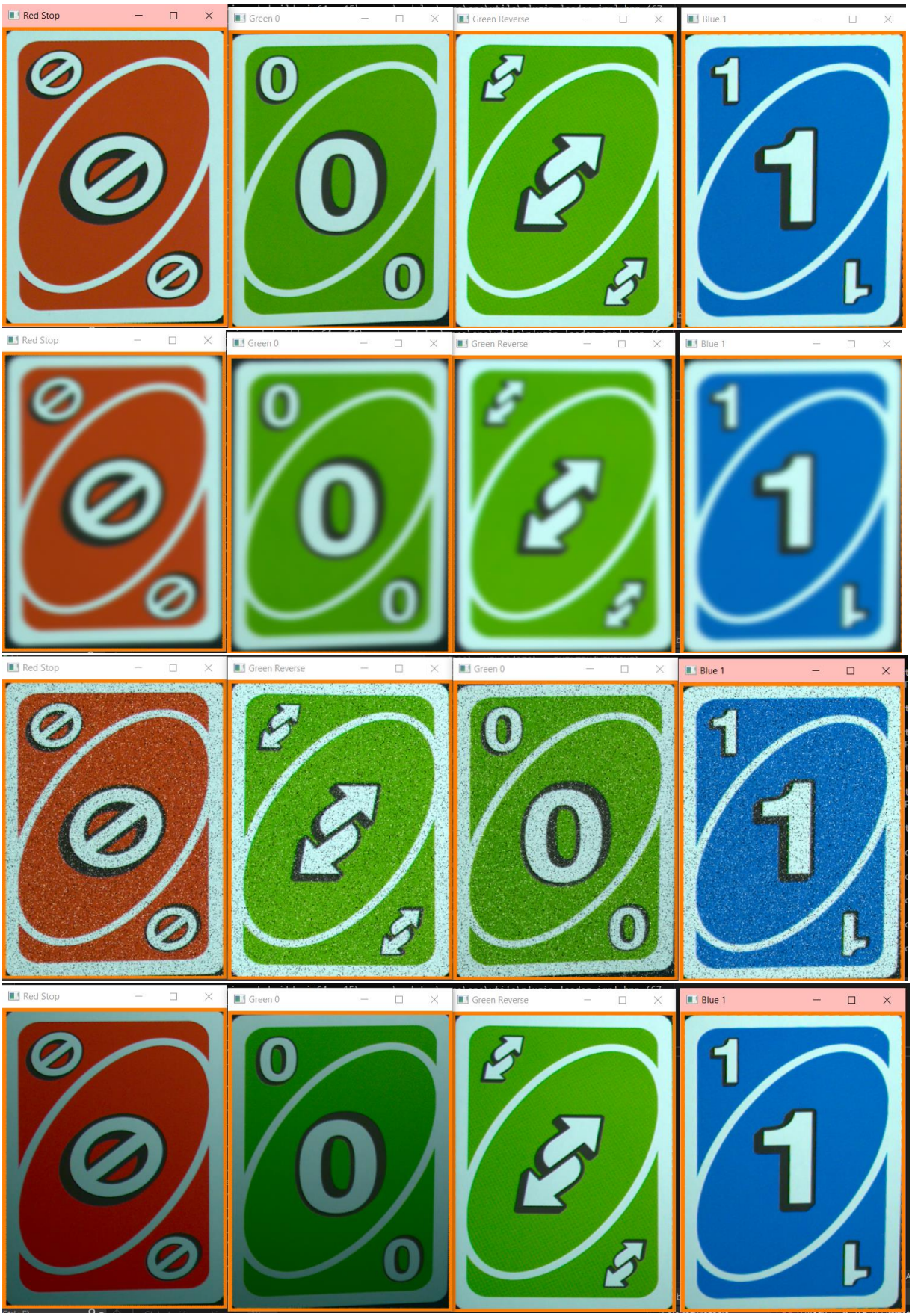
```
    maska_karty(karta1);            //tworzenie i nakładanie maski na wszytskie karty
    maska_karty(karta2);
    maska_karty(karta3);
    maska_karty(karta4);

    Rect roi(30, 30, 80, 80);       //Region Of Intrest - obszar w którym, w orientacji pionowej znajduje się symbol karty

    Mat imgCrop1 = imgWarp[0](roi); //tworzenie wycinków kart do rozpoznania kolorów i symboli
    Mat imgCrop2 = imgWarp[1](roi);
    Mat imgCrop3 = imgWarp[2](roi);
    Mat imgCrop4 = imgWarp[3](roi);

    cvtColor(imgOriginal, imgPost, COLOR_BGR2GRAY); //konwersja obrazu oryginalnego na skalę szarości

    PostCropping(imgPost, imgCrop1, color1, symbol1);       //przypisanie wycinkom kolorów i symboli
    PostCropping(imgPost, imgCrop2, color2, symbol2);
    PostCropping(imgPost, imgCrop3, color3, symbol3);
    PostCropping(imgPost, imgCrop4, color4, symbol4);

    imshow(color1 + " " + symbol1, k1);         //pokazanie rozpoznanych kart w oknach nazwanych zgodnie z rozpoznanym kolorem i symbolem
    imshow(color2 + " " + symbol2, k2);
    imshow(color3 + " " + symbol3, k3);
    imshow(color4 + " " + symbol4, k4);

    waitKey(0);

}
```

5. Ocena działania algorytmów, przedstawienie wyników.

W mojej ocenie zastosowane algorytmy poprawnie sprawiły swoją funkcję. Poniżej przedstawiam wyniki uzyskane dla wszystkich 4 wariancji wszystkich 4 scen. Obrazy końcowe wycięte są z oryginalnego zdjęcia z nałożonym pomarańczowym konturem.



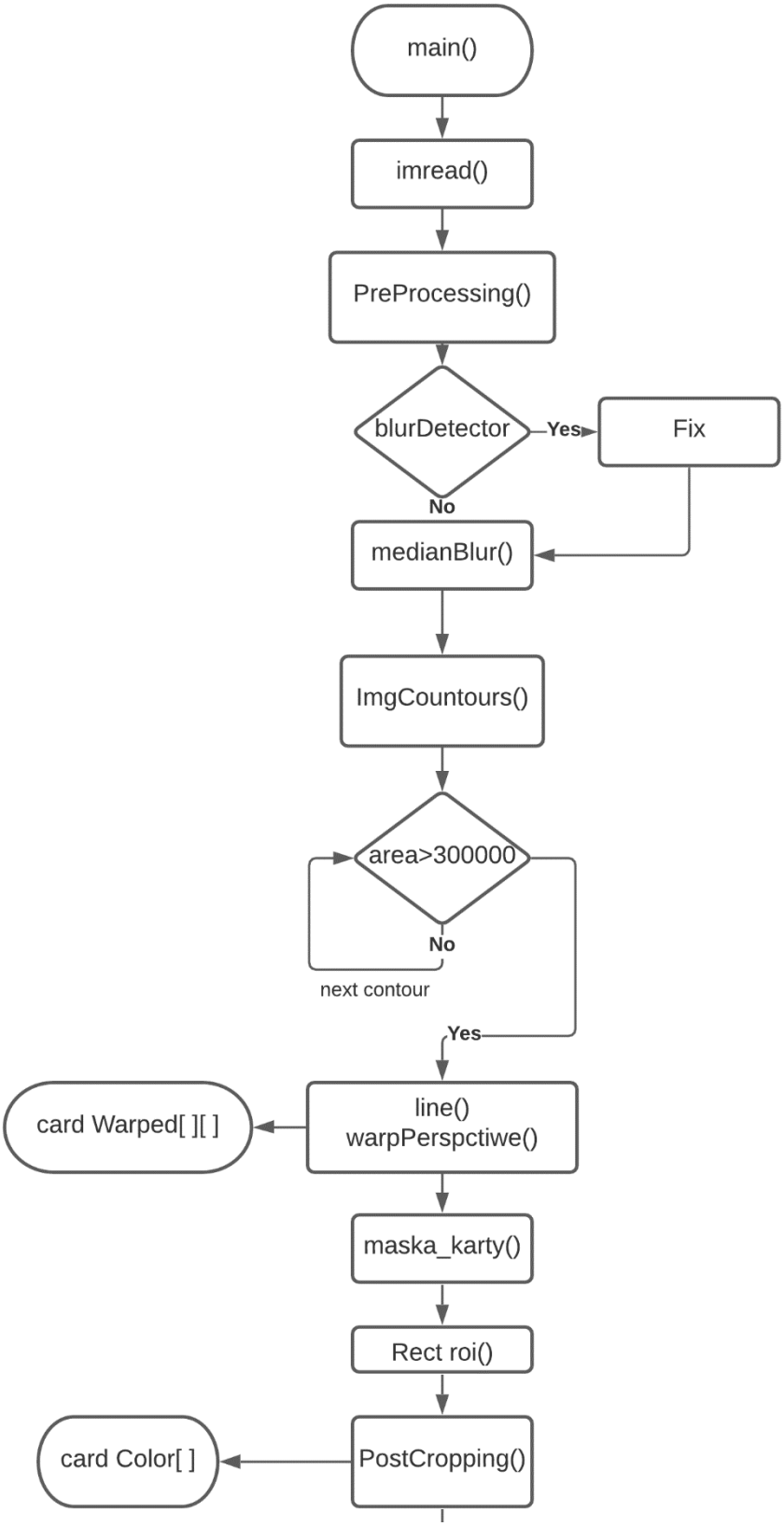
Warszawa 2022



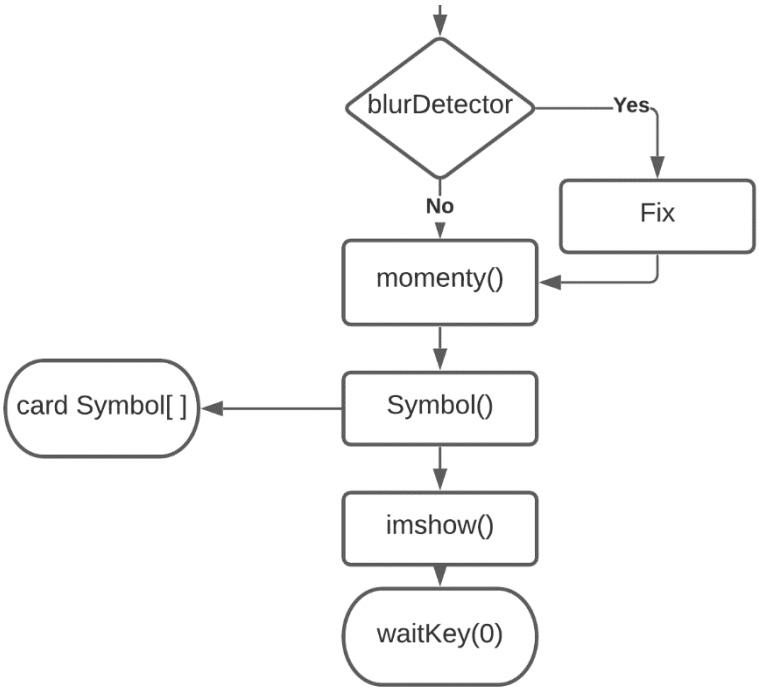
Warszawa 2022

Wszystkie obrazy wyników znajdują się w archiwum folderze wyniki.

Warszawa 2022



Warszawa 2022



Plik zawierający schemat znajduje się w folderze wyniki.

Oświadczenie

Oświadczam, że niniejsza praca stanowiąca podstawę do uznania osiągnięcia efektów uczenia się z przedmiotu Cyfrowe Przetwarzanie Obrazu została wykonana przeze mnie samodzielnie.

Imię i Nazwisko:	Miłosława Wilkowiecka
Nr. Indeksu:	297383
Data:	16.01.2022r

Warszawa 2022