Sprawozdanie z wykonania systemu biometrycznego w ramach przedmiotu Podstawy Biometrii

Filip Żmuda 263631

Grupa środa 13:15

Prowadzący: Dr inż. Jan Mazur

1. Cel projektu

Zadanie polega na zaprojektowaniu i implementacji w matlabie poszczególnych bloków systemu biometrycznego opartego o analizę tęczówki. Projektowany system ma zawierać:

- Blok akwizycji (i/lub blok czytania danych z dysku),
- Blok określania parametrów geometrycznych tęczówki,
- Blok wyznaczania kodu binarnego/tworzenia wzoru tęczówki,
- Blok porównywania dwóch wzorów tęczówek (wyznaczanie odległości pomiędzy wzorami),
- Blok podejmowania decyzji,
- Procedury niezbędne do oceny podstawowych parametrów systemu biometrycznego.

Bazą danych wykorzystaną do pracy przy projekcie jest dostarczona przez prowadzącego baza OFTA.

2. Blok akwizycji

2.1 Preprocessing

```
oko_org=imread(image_path);
oko_org=rgb2gray(oko_org);
dzielnik=2;
oko_org=imresize(oko_org,[600,860], 'bilinear');
oko_resized=imresize(oko_org,[600/dzielnik,860/dzielnik], 'bilinear');
oko_med = median_filter(oko_resized, 3);
oko_med = median_filter(oko_med, 3);
oko_blurred=custom_gaussian_filter(oko_med, 1.5);
```

Obraz jest wczytywany funkcją imread. Zdjęcia w bazie OFTA są trójwymiarowymi mapami bitowymi, dlatego dla uproszczenia przetwarzania obraz jest konwertowany do skali szarości funkcją *rgb2gray()*. Aby przyspieszyć przetwarzanie obrazu i ułatwić dostrajanie parametrów kodu dla różnych zdjęć dla całego programu ustawiany jest globalny dzielnik, przez którego wartość będą dzielone wymiary obrazu. Aby nie skomplikować zbytnio obliczeń na tak wstępnym etapie przetwarzania obrazu wykorzystywana jest metoda interpolacji dwuliniowej, polegającej na przeprowadzeniu dwóch interpolacji liniowych (w pionie i w poziomie). Sprowadza się to do obliczenia średniej wartości pikseli z określonego kwadratu ważonej odległościami obliczanego piksela od pikseli biorących udział w obliczeniach:

$$(1) \qquad r_0(a,b) = [d_1 + (d_2 - d_1)a] + [(d_3 - d_1) + (d_4 - d_3 - d_2 + d_1)a]b$$

Gdzie r_0 jest pikselem dla którego obliczana jest interpolowana wartość, a i b są odległościami w pionie i w poziomie tego piksela od wierszy i kolumn pikseli w rogach kwadratu, a d_1-d_4 są wartościami (intensywności dla obrazów w skali szarości, koloru dla obrazów kolorowych) tych pikseli.

W szczególnym przypadku, gdy obliczany piksel znajduje się dokładnie w środku obliczanego kwadratu, wynikiem interpolacji dwuliniowej jest średnia arytmetyczna wartości pikseli leżących w rogach obliczanego kwadratu.

$$(2) \qquad r_0(a,b) = \frac{(d_1 + d_2 + d_3 + d_4)}{4}$$

Tak zmniejszony obraz filtrowany jest dwukrotnie przy użyciu filtru medianowego 3x3, który pozwala rozmyć drobne krawędzie, takie jak rzęsy, które mogłyby wprowadzić dużą ilość szumu na dalszych etapach przetwarzania, a jednocześnie zachowuje on krawędzie większe takie jak granice między częściami oka, czy powieki. Ideą filtru jest przejście macierzą (w tym wypadku 3x3) po całym obrazie (z wyjątkiem ostatnich pikseli) i w każdej iteracji rozwinięcie jej do jednowymiarowego wektora i wybranie wartości mediany całej macierzy dla aktualnie rozpatrywanego piksela (znajdującego się w centrum macierzy). Implementacja tego filtru w matlabie:

```
margin = floor(mask_size / 2);
padded_image = padarray(image, [margin, margin], 'symmetric');
filtered_image = zeros(rows, cols);

for i = 1:rows
    for j = 1:cols
        window = padded_image(i:i+mask_size-1, j:j+mask_size-1);
        median_value = median(window(:));
        filtered_image(i, j) = median_value;
    end
end
```

Wstępnie przefiltrowany obraz filtrowany jest ponownie przy użyciu filtru Gaussa, zdefiniowanego wzorem:

$$(3) \qquad G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

W przełożeniu na kod w matlabie:

```
    filter_size = 2 * ceil(3 * sigma) + 1;
    [x, y] = meshgrid(-floor(filter_size / 2):floor(filter_size / 2), -
floor(filter_size / 2):floor(filter_size / 2));
    gaussian_filter = exp(-(x.^2 + y.^2) / (2 * sigma^2));
    gaussian_filter = gaussian_filter / sum(gaussian_filter(:));

    filtered_image = conv2(image, gaussian_filter, 'same');
```

Rozmiar maski jest ustawiany na $2 \cdot 3\sigma + 1$ w celu zapewnienia, że filtr będzie wystarczająco duży, aby uchwycić istotną część rozkładu Gaussa (ponad 99,7% wartości). Następnie tworzona jest siatka, która w obu wymiarach ma wielkość od -filter_size/2 do filter_size/2. Filtr Gaussa jest tworzony według wzoru (3) jednak czynnik $\frac{1}{2\pi\sigma^2}$ jest pomijany, ponieważ jest wyłącznie czynnikiem normalizującym. Normalizacja filtru następuje w kolejnej linii. Dzielenie współczynników filtru przez ich sumę sprawia, że suma wartości współczynników filtru wynosi 1. Tak przygotowany filtr jest splatany z obrazem z opcją ‘same’ zapewniającą, że wynikowy obraz będzie miał taki sam rozmiar jak wejściowy. Filtracja filtrem Gaussa pozwala zmniejszyć kontrast drobnych elementów obrazu, które mogłyby negatywnie wpłynąć na proces lokalizacji tęczówki, takich jak rzęsy, krawędzie powiek, czy plamki światła odbitego na rogówce oka.

Rys. 1 Wynik filtracji obrazu filtrami medianowymi i Gaussa

2.2 Wyznaczenie gradientu

W kolejnym kroku obliczany jest gradient obrazu przy użyciu operatora Sobela:

$$(4) \qquad s_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad s_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Gdzie s_x jest operatorem w kierunku poziomym, a s_y kierunku pionowym. W praktyce oznacza to nieintuicyjnie, że operator s_x znajduje krawędzie pionowe, ponieważ uwzględnia piksele po lewej i prawej stronie piksela centralnego (czyli właśnie w kierunku poziomym). Splot tych operatorów z obrazem daje wartości bliskie zeru tam, gdzie intensywność pikseli jest jednakowa na badanym obszarze (ponieważ operatory po obu stronach osi symetrii mają wartości przeciwne, zatem przemnożenie tych samych wartości z dodatnimi i ujemnymi wartościami operatora daje w sumie zero), a wartości o dużej wartości bezwzględnej tam, gdzie na obrazie znajduje się krawędź w danym kierunku (ponieważ suma różnych wartości intensywności pikseli przemnożona przez dodatnią i ujemną stronę operatora Sobela da przewagę jednej ze stron). Gradient całkowity oblicza się biorąc pierwiastek z sumy kwadratów obu gradientów:

$$(5) \qquad G = \sqrt{{G_x}^2 + {G_y}^2}$$

Iterując po całym obrazie i wykonując powyższe kroki otrzymujemy wyniki jak poniżej:

Rys. 2 Wyniki splotu obrazu z operatorami Sobela

Tak przeprowadzona detekcja krawędzi prowadzi do bardzo prostej binaryzacji obrazu, polegającej na wzięciu do obrazu zbinaryzowanego pikseli, które przekraczają określony próg.

Rys. 3 Obraz zawierający wyłącznie piksele o wartości większej od ustalonego progu (w tym wypadku 20)

3. Blok określania parametrów geometrycznych tęczówki

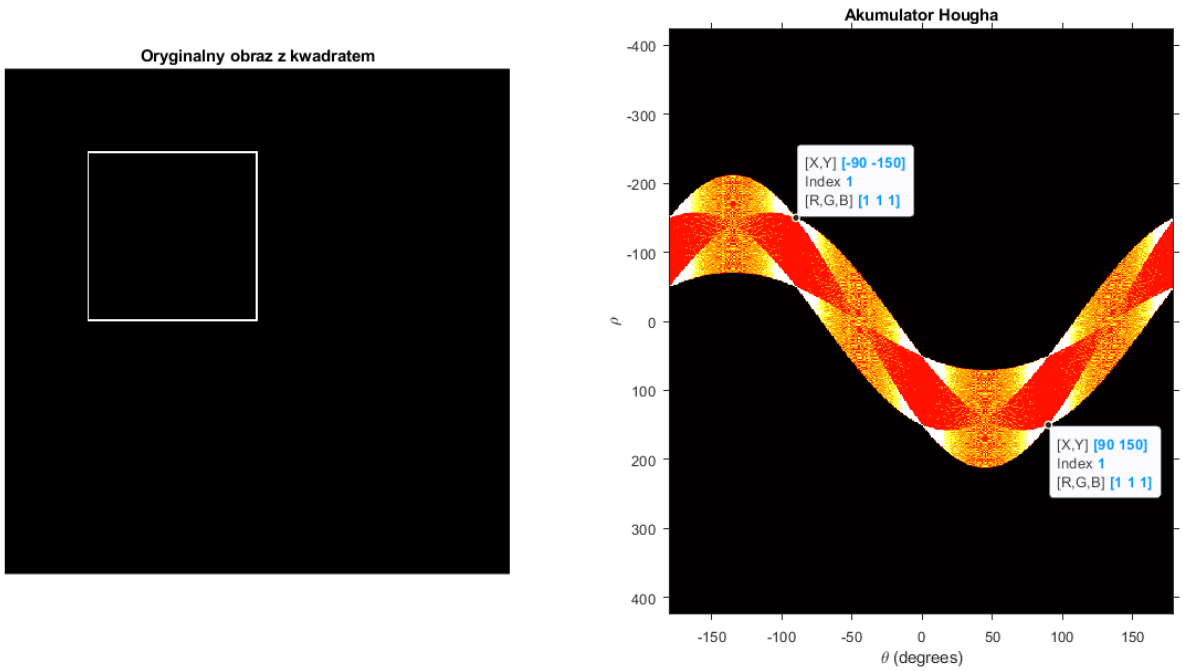
3.1 Transformata Hougha

Transformata Hougha jest zabiegiem stosowanym na wzorach matematycznych kształtów umieszczonych w układzie kartezjańskim pozwalającym odwzorować je w przestrzeni parametrycznej. Używając prostego przykładu: prosta w kartezjańskim układzie współrzędnych ma wzór $y = ax + b$, jeśli przekształcimy ten wzór do postaci $b = -ax + y$, to każdy punkt w dziedzinie przestrzennej (x,y) stanie się prostą w dziedzinie parametrycznej (a,b). Jeśli wiele punktów będzie leżało na jednej prostej w dziedzinie przestrzennej, w dziedzinie parametrycznej objawi się to wieloma liniami przecinającymi się w tym samym punkcie. Ponieważ transformata Hougha jest dualna, punkt przecięcia tych linii w dziedzinie parametrycznej da prostą w dziedzinie przestrzennej. W podejściu dyskretnym znalezienie punktu przecięcia odbywa się poprzez zdefiniowanie dwuwymiarowego akumulatora dziedziny parametrycznej i „rysowaniu" w nim linii (wpisywanie jedynek tam, gdzie przebiega linia). Przejście wielu linii przez jeden punkt spowoduje, że ten punkt będzie punktem o najwyższej wartości w całym akumulatorze. Problemem w podejściu przekształcającym punkty na linie jest fakt, że parametr a może przyjmować wartości z przedziału $(-\infty, \infty)$, co generuje problemy z wielkością akumulatora i pamięci. Z tego względu potrzebna jest lepsza parametryzacja. W tym celu można użyć równania parametrycznego prostej w postaci

$$x\,sin\theta - y\,cos\theta + \rho = 0 \qquad (6)$$

Gdzie θ to kąt nachylenia prostej względem osi x, a ρ to odległość prostej od początku układu współrzędnych. Ponieważ parametr θ jest ograniczony od 0 do π, a odległość ρ jest ograniczona wymiarami obrazu, rozwiązuje się problem z rozmiarem akumulatora. Teraz, gdy przeniesiemy punkt z dziedziny przestrzennej (x,y) do dziedziny parametrycznej (θ, ρ) zostanie on odwzorowany jako sinusoida. Punkty leżące na tej samej prostej w dziedzinie przestrzennej w dziedzinie parametrycznej wygenerują sinusoidy przecinające się w tych samych punktach.

Dualność transformaty w takim ujęciu pozostaje taka sama – prosta w dziedzinie przestrzennej daje punkt w dziedzinie parametrycznej, punkt w dziedzinie parametrycznej daje prostą w dziedzinie przestrzennej. Taki sam zostaje również proces znajdowania punktu z największą liczbą głosów w akumulatorze.



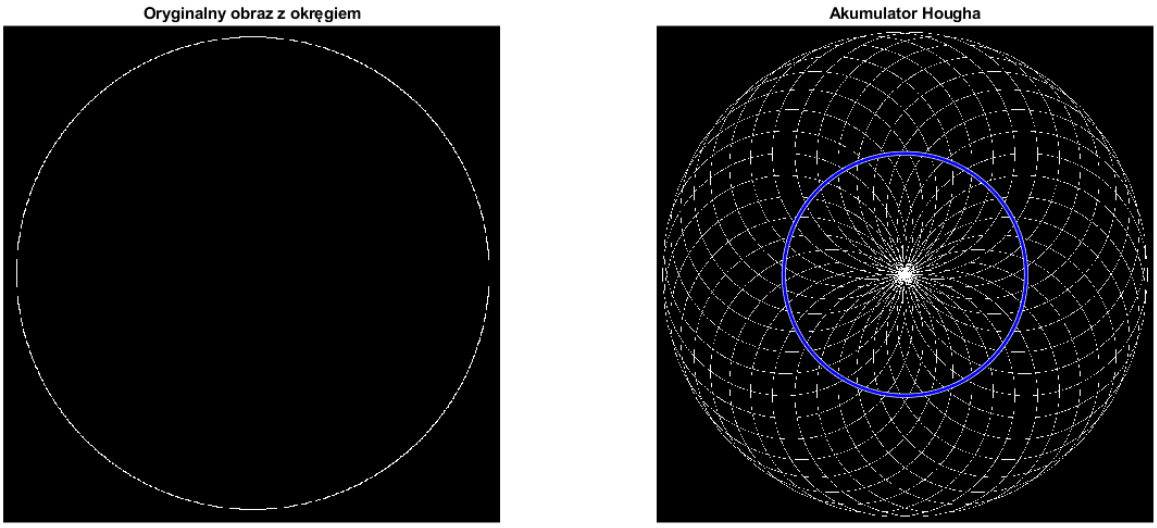
Rys. 4 Transformata Hougha przeprowadzona na kwadracie. Na wykresie akumulatora widać, że punkty przecięcia sinusoid powtarzają się co π.

Ponieważ kształt tęczówki jest bliższy okręgowi, niż linii prostej, do zlokalizowania jej używana jest Kołowa Transformata Hougha (Circular Hough Transform) wykorzystująca ten sam koncept, ale na równaniu okręgu. Jeśli równanie okręgu

$$(7) \qquad (x-a)^2 + (y-b)^2 = r^2$$

W którym a i b są współrzędnymi środka okręgu a r jego promieniem;
przeniesiemy do dziedziny parametrycznej (a,b), to każdy punkt na okręgu w dziedzinie przestrzennej zostanie odwzorowany jako okrąg w akumulatorze dziedziny parametrycznej. W przypadku, gdy promień okręgu jest znany, algorytm wygląda następująco:

Dla każdego punktu na okręgu rysujemy w akumulatorze okrąg o promieniu równym promieniowi rozpatrywanego okręgu. Ustawienie punktów w okręgu sprawia, że krawędzie kolejno rysowanych w akumulatorze okręgów będą przecinać się w tym samym punkcie. Ten punkt będzie środkiem szukanego okręgu.



Rys. 5 Wizualizacja Kołowej Transformaty Hougha w matlabie

Gdy promień nie jest znany (jak w przypadku próby zlokalizowania tęczówki na zdjęciu) należy zastosować algorytm wypełniający trójwymiarowy akumulator, czyli powtarzający ten sam schemat dla różnych wartości promienia z ustalonego zakresu:

```
[rows, cols] = size(grad);

for r = min_iris_radius:max_iris_radius
    for x = 1:cols
        for y = 1:rows
            if grad(y, x) == 1
                for theta = 0:pi/100:2*pi
                    a = round(x - r * cos(theta));
                    b = round(y - r * sin(theta));
                    if a > 0 && a <= cols && b > 0 && b <= rows
                        iris_accumulator(b, a, r) = iris_accumulator(b, a, r) + 1;
                    end
                end
            end
        end
    end
end

[max_iris_votes, max_iris_index] = max(iris_accumulator(:));
[iris_y, iris_x, iris_r] = ind2sub(size(iris_accumulator), max_iris_index);
```

Dla każdego zadanego promienia z określonego przedziału przeprowadzana jest iteracja po wszystkich pikselach zbinaryzowanego obrazu i dla każdego białego piksela (wartość równa jeden) obliczana jest wartość środka okręgu dla promienia branego pod uwagę w tym przejściu pętli. Równanie okręgu w postaci parametrycznej:

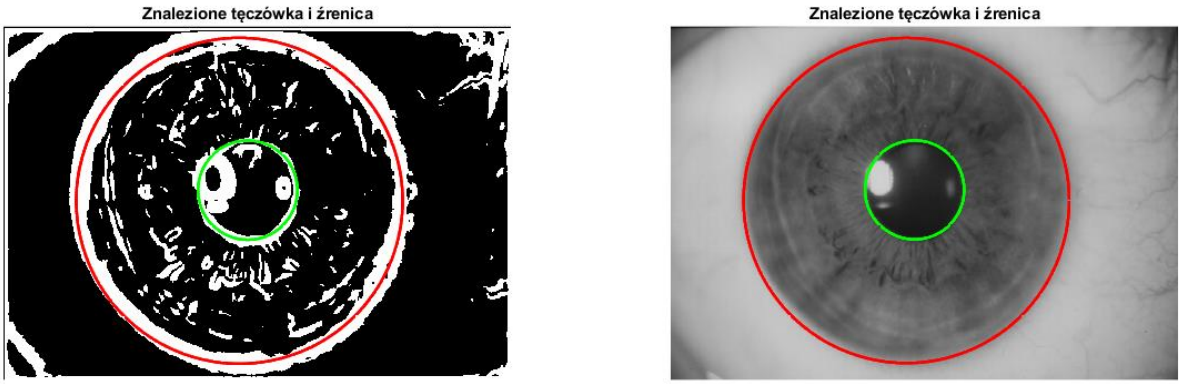
$$(8) \qquad x = a + r\cos(\theta)\,;\;\; y = b + r\,sin(\theta)$$

Przekształcono tak, aby można było obliczać za jego pomocą współrzędne środka okręgu:

$$(9) \qquad a = x - r\cos(\theta)\,;\;\; b = y - r\,sin(\theta)$$

Tak obliczone współrzędne środka okręgu wraz z aktualnie rozpatrywanym promieniem dostają jeden „głos” w swoim miejscu w akumulatorze. Jeśli dla innych punktów przy tym

samym promieniu współrzędne środka okręgu również będą w tym miejscu, liczba głosów takiego punktu w trójwymiarowym akumulatorze będzie wzrastać. Ostatecznie za środek okręgu tęczówki uznany zostanie piksel który otrzymał największą liczbę głosów. Ten sam proces powtarzany jest w ograniczonym obszarze dla źrenicy. Dobierając odpowiadające „najjaśniejszym" pikselom w akumulatorach promienie, można wyrysować znalezione okręgi:



Rys. 6 Wyrysowane na obrazie gradientowym i oryginalnym znalezione okręgi źrenicy i tęczówki

3.2 Operator całkowo-różniczkowy Daugmana (Daugman integro-differential operator)

Alternatywą dla transformaty Hougha jest algorytm lokalizacji tęczówki autorstwa Johna Daugmana, opierający się o operator całkowo-różniczkowy.

Idea polega na całkowaniu wartości pikseli wzdłuż obwodu okręgu oraz różniczkowaniu tej całki względem promienia okręgu, aby znaleźć maksymalną zmianę wartości, która odpowiada granicy między źrenicą a tęczówką lub tęczówką a twardówką. Zatem algorytm daugmana maksymalizuje poniższe wyrażenie:

$$(10) \qquad \max_{(x_0,y_0,r)} \left| G_\sigma(r) * \frac{\partial}{\partial r} \int_{(x_0,y_0,r)} \frac{I(x,y)}{2\pi r} ds \right|$$

Gdzie x_0, y_0 są współrzędnymi środka okręgu, r jest jego promieniem, I(x,y) to obraz, a G_σ to filtr Gaussa.

W praktyce, na uprzednio przefiltrowanym obrazie z którego wyznaczono gradient, operator działa w następujący sposób: dla każdej potencjalnej pozycji środka okręgu (x, y) i dla każdego promienia r od rmin do rmax, iteruje po kątach theta od 0 do 2*pi z niewielkim krokiem (w przykładzie $\frac{\pi}{36}$). Dla każdej pary (x, y) i r, oblicza wartości pikseli na obwodzie okręgu i sumuje je. Powtarza ten proces dla różnych wartości (x, y) i r, aby znaleźć najlepsze dopasowanie, które maksymalizuje wartość operatora. W kodzie implementacja wygląda następująco:

```
iris_accumulator = zeros(rows, cols, irisrmax - irisrmin + 1);
for x = 1:cols
    for y = 1:rows
        for r = irisrmin:irisrmax
            for theta = 0:pi/36:2*pi
                a = round(x - r * cos(theta));
                b = round(y - r * sin(theta));
                if a > 0 && a <= cols && b > 0 && b <= rows
                    iris_accumulator(y, x, r - irisrmin + 1) =
                          iris_accumulator(y, x, r - irisrmin + 1) + grad(b, a);
                end
            end
        end
    end
end

[max_pupil_votes, max_pupil_index] = max(pupil_accumulator(:));
[pupil_y, pupil_x, pupil_r] = ind2sub(size(pupil_accumulator), max_pupil_index);
pupil_r = pupil_r + rmin - 1;
```

najpierw inicjalizujemy akumulator dla tęczówki, następnie w pętli iterujemy przez wszystkie piksele obrazu, dla każdego piksela iterujemy przez wszystkie promienie w zadanym zakresie, a dla każdego promienia iterujemy przez kąty theta. Dla każdej wartości theta obliczamy współrzędne (a, b) na obwodzie okręgu, sprawdzamy, czy te współrzędne mieszczą się w granicach obrazu, jeśli tak, to dodajemy wartość gradientu w punkcie (b, a) do odpowiedniego akumulatora. Po zakończeniu pętli znajdujemy maksimum w akumulatorze, które odpowiada najlepszej detekcji źrenicy. Z uwagi na mniejszą stabilność działania algorytmu Daugmana w mojej implementacji i trudności z dostrojeniem parametrów do używanej bazy danych, zdecydowałem o używaniu w moim projekcie Transformaty Hougha jako narzędzia do lokalizacji obszaru tęczówki.

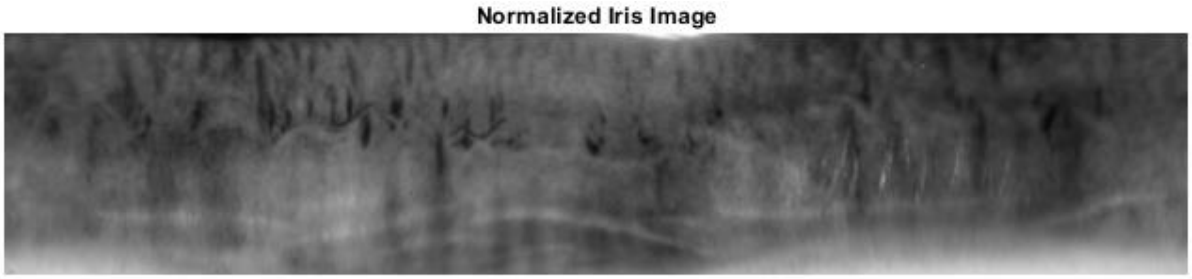
3.3 Normalizacja tęczówki

Proces normalizacji przekształca tęczówkę z pierścieniowego regionu tęczówki wokół źrenicy do prostokątnego obrazu. Aby przeprowadzić to przekształcenie definiowane są wektor kątów wokół środka źrenicy i wektor promieni od granicy źrenicy z tęczówką do granicy tęczówki z twardówką. Z połączenia tych dwóch definiowana jest siatka, na którą będą rzutowane kolejne punkty obrazu. Punkty obliczane są ze wzoru zamiany współrzędnych biegunowych na kartezjańskie:

$$x = r\,cos(\theta), \qquad y = r\sin(\theta) \tag{11}$$

Ponieważ współrzędne tak obliczonych punktów mogą nie być liczbami całkowitymi, używana jest funkcja interp2 do dwuwymiarowej interpolacji liniowej punktów rzutowanych na prostokątną siatkę. W przełożeniu na kod:

```
theta = 0:0.005:2*pi;
radii = linspace(pupil_r*dzielnik, iris_r*dzielnik, 128);
[thetaGrid, radiiGrid] = meshgrid(theta, radii);
x = pupil_x*dzielnik + radiiGrid .* cos(thetaGrid);
y = pupil_y*dzielnik + radiiGrid .* sin(thetaGrid);
normalizedIris = interp2(double(oko_org), x, y, 'linear', 0);
```



Rys. 7 Segment tęczówki po normalizacji

4. Blok wyznaczania kodu binarnego/tworzenia wzoru tęczówki

4.1 Filtr Gabora

Do wyekstrahowania cech obrazu znormalizowanej tęczówki używany jest filtr Gabora. Jego odpowiedź impulsową definiuje się jako falę sinusoidalną (falę płaską w przypadku filtrów Gabora 2D) pomnożoną przez funkcję Gaussa (tzw. koperta, ang. *envelope*). Ze względu na twierdzenie o splocie (splotowi dwóch sygnałów w dziedzinie czasu odpowiada mnożenie ich transformat Fouriera, mnożeniu dwóch sygnałów w dziedzinie czasu odpowiada splot ich transformat Fouriera), transformata Fouriera odpowiedzi impulsowej filtra Gabora jest splotem transformaty Fouriera funkcji sinusoidalnej i transformaty Fouriera funkcji Gaussa. Filtr ma składową rzeczywistą i urojoną reprezentującą kierunki ortogonalne. Z tych dwóch można utworzyć filtr złożony. Wtedy część rzeczywista zdefiniowana jest jako:

$$(12) \qquad G_{real}(x', y') = e^{-\frac{1}{2}(\frac{x'^2}{\sigma_x{}^2}+\frac{y'^2}{\sigma_y{}^2})} \cdot \cos\left(\frac{2\pi x'}{\lambda} + \varphi\right)$$

Gdzie:

$$(13) \qquad x' = x\cos(\theta) + y\sin(\theta), \quad y' = -xsin(\theta) + ycos(\theta)$$

Są współrzędnymi x i y obracanymi o kolejne kąty θ, kąt ten definiuje kierunek wykrywania cech w obrazie. λ jest długością fali sinusoidalnej, σ_x i j, σ_y to odchylenia standardowe w kierunkach poziomym i pionowym, określają rozmiar przestrzenny filtra (Gdy są takie same sprowadzają się do pojedynczego, nieindeksowanego σ). φ Jest przesunięciem fazowym funkcji sinusoidalnej.

Podobnie zdefiniowana jest część urojona filtru:

$$(14) \qquad G_{imag}(x', y') = e^{-\frac{1}{2}(\frac{x'^2}{\sigma_x{}^2}+\frac{y'^2}{\sigma_y{}^2})} \cdot \sin\left(\frac{2\pi x'}{\lambda} + \varphi\right)$$

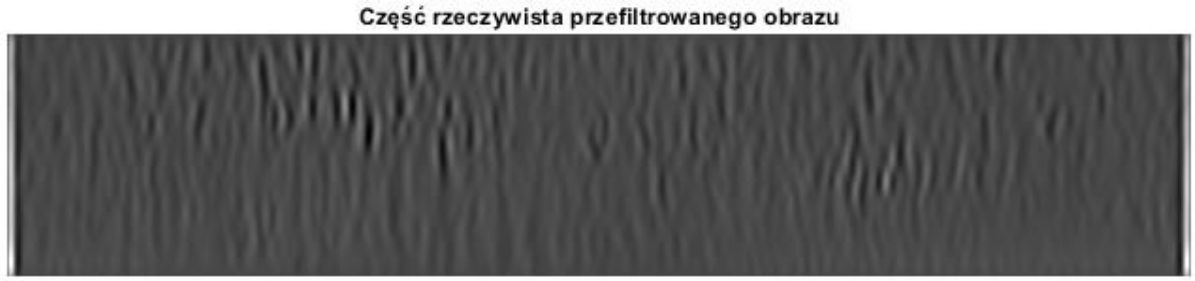
Ostatecznie:

$$(15) \qquad G(x', y') = G_{real}(x', y') + i \cdot G_{imag}(x', y')$$

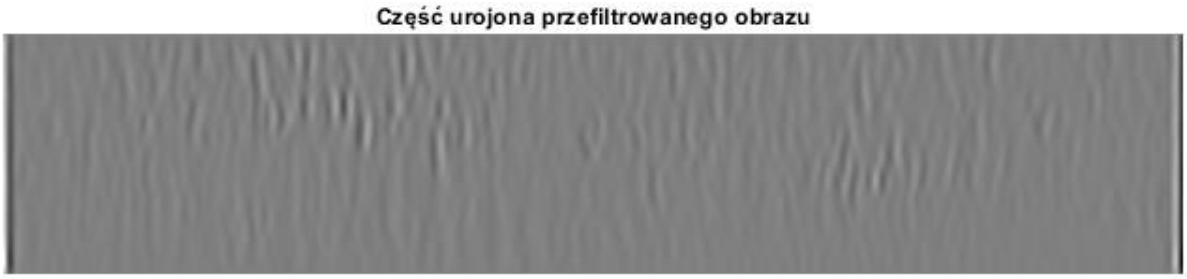
Tak zdefiniowany filtr w matlabie zaimplementowano w następujący sposób:

```
wavelength = 8;
theta = 0;
sigmaX = 4;
sigmaY = 4;
phaseOffset = 0;
[x, y] = meshgrid(-fix(sigmaX*3):fix(sigmaX*3), -fix(sigmaY*3):fix(sigmaY*3));
xPrime = x .* cos(theta) + y .* sin(theta);
yPrime = -x .* sin(theta) + y .* cos(theta);
realPart = exp(-0.5 * (xPrime.^2 / sigmaX^2 + yPrime.^2 / sigmaY^2)) .* ...
               cos(2 * pi * xPrime / wavelength + phaseOffset);
imagPart = exp(-0.5 * (xPrime.^2 / sigmaX^2 + yPrime.^2 / sigmaY^2)) .* ...
               sin(2 * pi * xPrime / wavelength + phaseOffset);

gaborFilter = realPart + 1i * imagPart;
realFilteredImg = filter2(real(gaborFilter), normalizedIris);
imagFilteredImg = filter2(imag(gaborFilter), normalizedIris);
filteredImgComplex = realFilteredImg + 1i * imagFilteredImg;
```



Rys. 8 Część rzeczywista wyniku filtracji



Rys. 9 Część urojona wyniku filtracji

4.2 Binaryzacja

Po tak przeprowadzonej filtracji obraz jest binaryzowany względem średniej wartości obrazu zespolonego. Jeśli dany piksel obrazu ma wartość większą niż średnia, to przyjmuje wartość true (1), a jeśli mniejszą – false (0). Tak powstały kod jest rozciągany do postaci jednoymiarowego wektora, a następnie dla wizualizacji tworzony jest obraz tego kodu o takich samych rozmiarach jak w poprzednich etapach analizy znormalizowanej tęczówki.

```
threshold = mean(filteredImgComplex(:));
binary_image = filteredImgComplex > threshold;
iris_code = reshape(binary_image, 1, []);
iris_code_image = reshape((iris_code.'), size(filteredImgComplex));
```

Rys. 10 Wygenerowany kod IrisCode przedstawiony jako obraz

5. Blok porównywania dwóch wzorów tęczówek (wyznaczanie odległości pomiędzy wzorami

Kiedy wyznaczymy kody dwóch tęczówek, możemy w łatwy sposób porównać jak bardzo są do siebie podobne wykorzystując odległość Hamminga. Jest to miara odmienności dwóch ciągów o takiej samej długości, wyrażająca liczbę pozycji, na których te dwa ciągi się różnią. W praktyce oznacza to, że należy obliczyć wartość funkcji Exclusive Or (XOR - Alternatywa rozłączna/wykluczająca), która zwraca 0, gdy bity na tej samej pozycji są takie same i 1 gdy są różne. Im większa ilość zwróconych jedynek, tym bardziej różnią się od siebie analizowane kody.

```
function distance = hammingDistance(code1, code2)
    if length(code1) ~= length(code2)
        error('Kody muszą być tej samej długości');
    end
    distance = sum(code1 ~= code2) / length(code1);
end
```

Wyznaczona odległość Hamminga jest normalizowana przez długość kodu, co znacznie ułatwia analizę wyników – Gdy funkcja zwróci 0, oznacza to, że przekazaliśmy jej dwa identyczne kody (co oznacza, że musieliśmy analizować wcześniej dwa razy dokładnie to samo zdjęcie). Gdy zwróci 1 oznacza to, że wszystkie bity w drugim kodzie są odwrotne względem pierwszego (co oznacza, że musieliśmy analizować dwa negatywne względem siebie obrazy). Zatem dwa kody są od siebie „najbardziej różne” gdy powyższa funkcja zwróci wartość 0.5.

6. Blok podejmowania decyzji

Aby zautomatyzować proces identyfikacji danego oka, wygenerowano kody IrisCode dla wszystkich zdjęć w bazie i zapisano je w plikach .mat w folderach ponumerowanych od 0 do 21, dla każdej osoby oddzielnie. Następnie z folderu każdej osoby usunięto kod pierwszego zdjęcia, aby nie było go w bazie i można było przeprowadzić porównanie kodu wygenerowanego dla tego zdjęcia z pozostałymi w bazie. Poniżej znajduje się kod funkcji, która generuje kod dla przekazanego zdjęcia i porównuje go z wszystkimi kodami w bazie obliczając odległość Hamminga do każdego z nich. Funkcja zwraca informację do oka której osoby w bazie zdjęcie przekazanego oka jest najbardziej podobne, najmniejszą wyznaczoną odległość Hamminga i informację o tym, czy system udziela właścicielowi zdjęcia przekazanego oka dostępu, czy też nie. Ta decyzja podejmowana jest na podstawie progu maksymalnej odległości, którego wartość ustalono wstępnie na **0.37**.

```
function [matched_person, minHammingDistance, access_granted] =
identify_iris(image_path, codes_path, threshold)
    new_iris_code = iris_recognition_ofta(image_path);
    minHammingDistance = inf;
    matched_person = -1;

    for i = 1:21
        person_folder = fullfile(codes_path, num2str(i));
        code_files = dir(fullfile(person_folder, '*.mat'));

        for j = 1:length(code_files)
            load(fullfile(person_folder, code_files(j).name), 'iris_code');
            hammingDist = hammingDistance(new_iris_code, iris_code);
            if hammingDist < minHammingDistance
                minHammingDistance = hammingDist;
                matched_person = i;
            end
        end
    end
    access_granted = minHammingDistance < threshold;
end
```

Wynik wywołania powyższej funkcji:

```
Dostęp przyznany. Najbardziej podobne oko należy do osoby nr 9.
Odległość = 0.293295.
Czas szukania: 32.8287 sekundy
```

7. Procedury niezbędne do oceny podstawowych parametrów systemu biometrycznego

Do oceny zaprojektowanego systemu biometrycznego posłużą następujące parametry i narzędzia:

- FAR – False Acceptance Rate – Współczynnik fałszywej (niewłaściwej) akceptacji. Jest to parametr określający jak często system przyznaje dostęp osobie nieuprawnionej

$$FAR = \frac{Liczba\ \ fałszywych\ akceptacji}{Liczba\ wszystkich\ prób\ autoryzacji\ nieuprawnionych} \tag{16}$$

- FRR – False Rejection Rate – Współczynik fałszywego (niewłaściwego) odrzucenia – określa jak często system odmawia dostępu osobie uprawnionej

$$FRR = \frac{Liczba\ \ fałszywych\ odrzuceń}{Liczba\ wszystkich\ prób\ autoryzacji\ uprawnionych} \tag{17}$$

- Macierz pomyłek to tabela przedstawiająca wyniki klasyfikacji modelu, która pokazuje liczbę poprawnych i niepoprawnych prognoz. Dla systemów biometrycznych macierz pomyłek pokazuje, jakie decyzje podejmuje system w odniesieniu do prawdziwych klas (osób).

$$M = \begin{bmatrix} TP & FN \\ FP & TN \end{bmatrix} \tag{18}$$

gdzie:

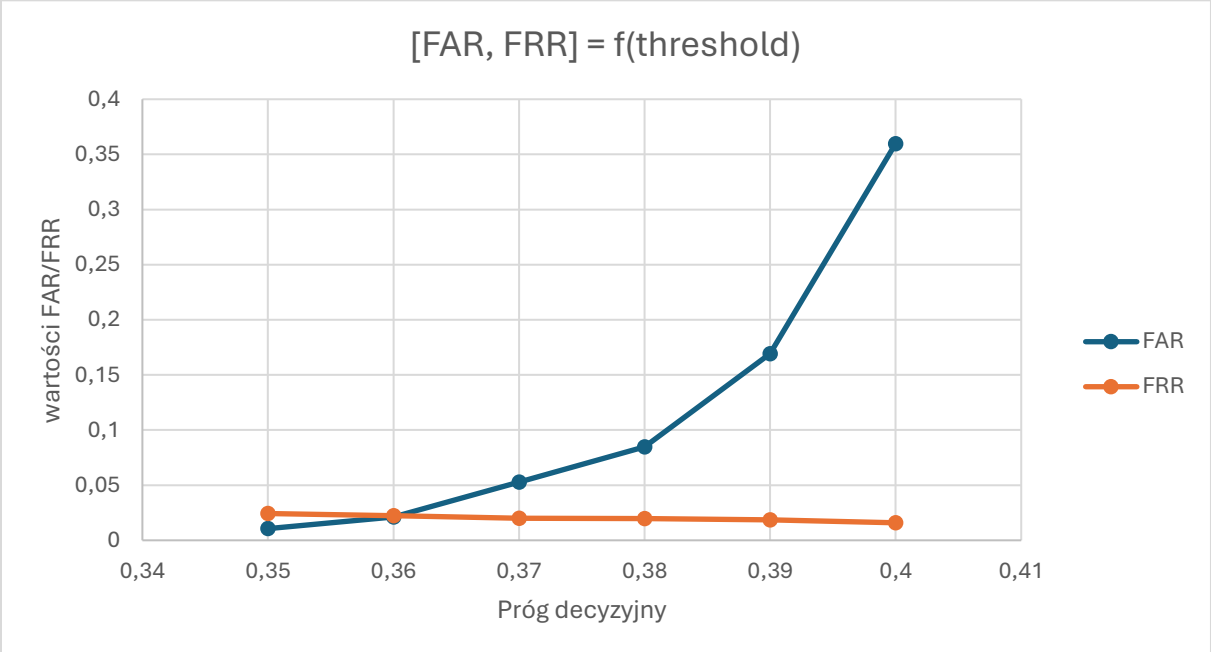
- TP (True Positive) to liczba przypadków, gdy system podjął poprawną decyzję o przydzieleniu dostępu osobie uprawnionej
- FN (False Negative) to liczba przypadków, gdy system podjął błędną decyzję nie przyznając dostępu osobie uprawnionej
- FP (False Positive) to liczba przypadków, gdy system podjął błędną decyzję przyznając dostęp osobie nieuprawnionej
- TN (True Negative) to liczba przypadków, gdy system podjął poprawną decyzję o nieprzydzieleniu dostępu osobie nieuprawnionej

Wyniki systemu dla kolejnych wartości progu akceptacji są następujące:

Tab. 1 Wartości FAR i FRR w zależności od wartości progu decyzyjnego

Threshold	FAR	FRR
0,35	0,0106	0,0243
0,36	0,0212	0,0222
0,37	0,0529	0,0201
0,38	0,0847	0,0196
0,39	0,1693	0,0185
0,4	0,3598	0,0159

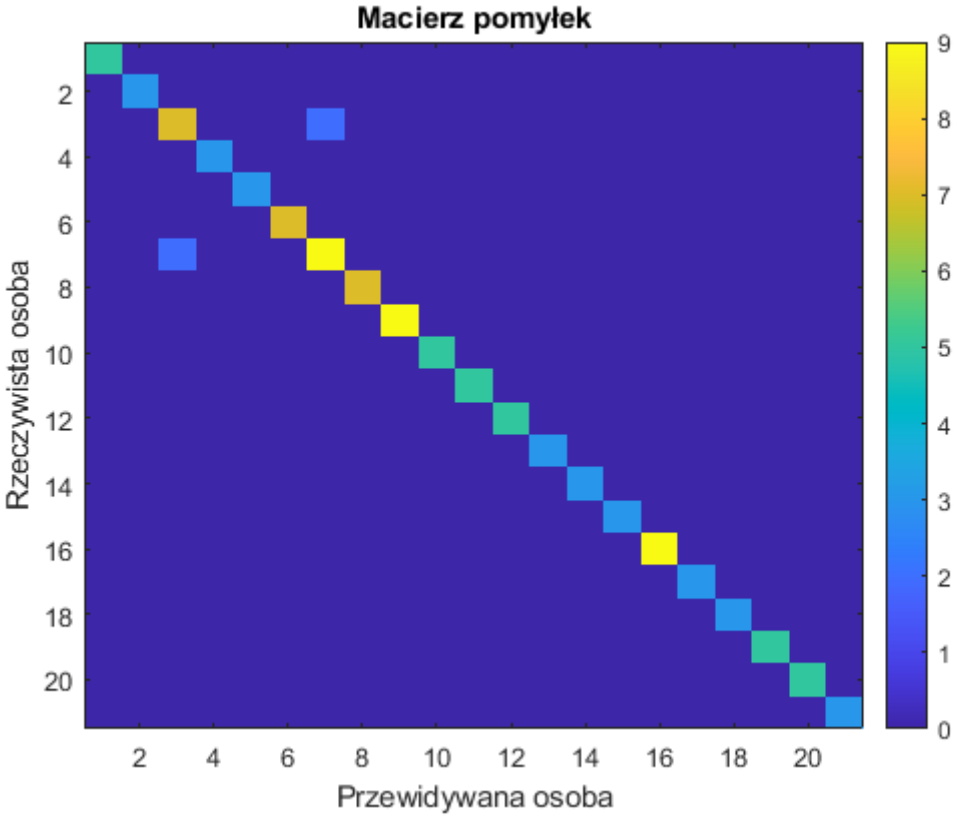
Dla wyznaczonych wartości wyrysowano wykres:



Rys. 11 Wykres FAR i FRR w funkcji wartości progu decyzyjnego

Wartości obu błędów są minimalizowane dla wartości progu **0.36**, zatem taką wartość należałoby przyjąć dla wyśrodkowanego systemu. Jednak ostateczna wartość progu zależy od potrzeb systemu. Jeśli system ma obsługiwać dostęp do czegoś, co nie musi być bardzo dokładnie zabezpieczone biometrycznie, bo na przykład ma jeszcze inne zabezpieczenia, to można podwyższyć próg dla wygody użytkowania. Jeśli jednak system musi być bardziej

bezwzględny, próg należałoby obniżyć. Dla progu 0.36 wyrysowano macierz pomyłek systemu:



Rys. 12 Macierz pomyłek systemu dla progu 0.36

8. Wnioski

Ponieważ wykorzystana w projekcie baza danych zawierała po trzy zdjęcia dwudziestu jeden osób, wyniki nie są bardzo precyzyjne i należy brać to pod uwagę. Tym bardziej, że część z tych zdjęć celowo, dla utrudnienia zadania algorytmowi lokalizacji tęczówki było robionych pod kątem, z opuszczoną powieką itd. Uważam jednak, że algorytm mimo tych utrudnień dobrze poradził sobie z postawionym zadaniem. W kolejnych fazach rozwoju projektu musiałby jednak zwiększyć elastyczność, na przykład ze względu na fakt, że wykorzystana w projekcie do porównywania kodów tęczówek odległość Hamminga jest wrażliwa na przesunięcia w kodzie. Aby poprawić działanie lokalizacji tęczówki można by było użyć bardziej skomplikowanych metod binaryzacji, na przykład algorytmu canny'ego, który zmniejszyłby rozmiar krawędzi na obrazie gradientowym (Rys. 3) do jednego piksela, co poprawiło by dokładność i zabezpieczyło przed błędami.

9. Bibliografia

https://en.wikipedia.org/wiki/Gabor_filter

https://twiki.fotogrametria.agh.edu.pl/pub/Publikacje/WebHome/V21_Marmol_Lenda.pdf

https://www.sciencedirect.com/topics/engineering/gabor-filter

https://youtu.be/XRBc_xkZREg?si=Ubutj3O7H3AIf4Id