Biometria

Projekt nr 2 - Iris Prediction

Mikołaj Rowicki | Filip Langiewicz
nr indeksu 327309 | nr indeksu 327293

Politechnika Warszawska
Wydział Matematyki i Nauk Informacyjnych
24 kwietnia 2025

Spis treści

1. Wstęp

Rozpoznawanie tęczówki oka jest jedną z najdokładniejszych metod biometrycznych, która osiąga wysoki poziom niezawodności i odporności na wszelkie próby oszustw. Niezbędne jest do tego jednak poprawnie wykonane, i o wysokiej rozdzielczości - zdjęcie oka. Na wstępie procesu konieczne jest dokładne wydzielenie obszaru tęczówki od pozostałych elementów oka, takich jak powieki, rzęsy czy odblaski.

W niniejszym sprawozdaniu opisujemy kolejno każdy etap przetwarzania obrazu tęczówki. Zaczynamy od segmentacji, do której zaliczamy odfiltrowywanie szumów i artefaktów przy pomocy sekwencji operacji morfologicznych, detekcję granic źrenicy i tęczówki w oparciu o analizę wykresów projekcji, a także rozwinięcie otrzymanej maski do stałego rozmiaru 64×360 pikseli. Druga część projektu wprowadza przetwarzanie rozwiniętego obrazu polegające na ekstrakcji jednowymiarowych sygnałów Gaborowskich i kodowaniu fazowym ich współczynników do postaci ciągu bitowego (IrisCode). Dzięki temu etapowi projekt zapewnia kompletny pipeline przeprowadzający użytkownika od wczytania obrazu oka do wygenerowania cyfrowego rekordu tęczówki gotowego do porównań w bazie danych.

2. Opis techniczny kodu

2.1. Technologie i biblioteki

W projekcie wykorzystaliśmy następujące biblioteki i narzędzia:

- `PIL (Pillow)` – wczytywanie obrazów w formatach BMP, PNG, JPG.
- `NumPy` – operacje macierzowe i obliczenia na tablicach pikseli.
- `OpenCV (cv2)` – przetwarzanie obrazu, operacje morfologiczne, detekcja komponentów, rysowanie okręgów.
- `matplotlib` – wizualizacja obrazów oraz wykresy projekcji i kodów (podmoduły `pyplot` i `patches`).
- `itertools` – generowanie kombinacji i permutacji, pomoc przy porównywaniu kodów tęczówki.
- `os` – operacje na plikach i ścieżkach.
- `io`, `contextlib` oraz tryb silent mode z backendem Agg (`matplotlib.use('Agg')`) – tłumienie wyjścia konsoli i wykresów przy automatycznych testach.

2.2. Struktura kodu i użytkowanie pipeline'u

Cały proces przetwarzania pojedynczego zdjęcia tęczówki został ujęty w funkcji `predictIris(path, f, save, output_width)`, której wywołanie uruchamia kolejno wszystkie etapy opisanego poniżej pipeline'u:

1. **Wczytanie i konwersja do skali szarości:**
   - `load_img(path)` – wczytuje obraz jako tablicę NumPy.
   - `convert_to_grayscale(image)` – konwertuje obraz do skali szarości za pomocą średniej z kanałów RGB.

2. **Globalna binaryzacja:**
   - `binarize_image(gray, X=4.5)` – próg dla źrenicy.
   - `binarize_image(gray, X=1.6)` – próg dla tęczówki.
3. **Segmentacja i detekcja:**
   - `clean_pupil(binary_image_pupil)` – oczyszcza wyodrębniony obraz źrenicy, zwraca `(x_pupil, y_pupil, r_pupil)` - współrzędne środka i promień źrenicy.
   - `clean_iris(binary_image_iris)` – oczyszcza wyodrębniony obraz tęczówki, zwraca promień tęczówki `r_iris` (zakładamy, że źrenica i tęczówka są współśrodkowe).
   - `draw_circle(...)` – rysuje na oryginalnym obrazie kontury źrenicy i tęczówki.
4. **Transformacja do współrzędnych biegunowych:**
   - `unwrap_iris(image, x_p, y_p, r_p, r_i)` – normalizuje obszar tęczówki do prostokąta o wymiarze domyślnie 64×360 pikseli.
5. **Wyświetlanie i zapisywanie wyników:**
   - `show_image(...)` – prezentuje w Notebooku aktualny obraz.
   - opcja `save=True` zapisuje oryginalny obraz z zaznaczonymi granicami źrenicy i tęczówki jako `ORIGINAL_NAME_found.bmp` w katalogu `found/`.
6. **Analiza rozwiniętej tęczówki:**
   - `visualize_band_boundaries(unwrapped_iris)` – nakłada granice ośmiu pasów.
   - `extract_central_bands(unwrapped_iris, output_width)` – zwraca listę 8 wycinków azymutalnych, przeskalowanych do szerokości `output_width`.
   - `visualize_bands(bands)` – wyświetla każdy pas osobno.
7. **Budowa IrisCode:**
   - `compute_full_iris_code(unwrapped_iris, f, num_bands, num_coeffs, output_width)` – wykonuje filtrację Gaborowską, koduje fazę do postaci bitowej.
   - `visualize_iris_code_by_band(iris_code, num_bands)` – rysuje wykresy bitów dla poszczególnych pasów.

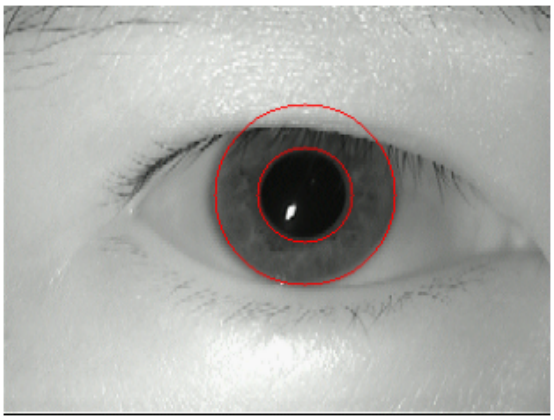
Przykład użycia:

```
# f - częstotliwość falek Gaborowskich, output_width - szerokość wycinków
# sigma - parametr funkcji okna Gaussowskiego
iris_code = predictIris(
    path="data/eye_p01_L1.bmp",
    f=np.pi,
    save=True,
    sigma=4.42,
    output_width=128
)
```

Po wykonaniu funkcji w bieżącym katalogu 'found/' ukaże się plik z naniesionymi okręgami źrenicy i tęczówki, w konsoli będą widoczne komunikaty z kolejnych etapów, a sama funkcja zwróci łańcuch 2048 bitów – w pełni zakodowany *IrisCode*.

3. Segmentacja tęczówki

Zagadnienie segmentacji tęczówki stanowi kluczowy etap w procesie automatycznego rozpoznawania osób na podstawie obrazu oka. Jego celem jest przygotowanie surowych danych wejściowych poprzez wczytanie obrazu w formacie RGB i jego konwersję do skali szarości, a także binaryzację i inne operacje morfologiczne. Na dalszych etapach, po uzyskaniu znormalizowanego obrazu tęczówki, stosuje się algorytmy ekstrakcji unikatowych cech oraz porównywania wygenerowanych kodów, co pozwala na niezawodne uwierzytelnianie. W niniejszym sprawozdaniu szczegółowo omówimy algorytm oparty na pracy Daugmana, która zakłada, że prawidłowa normalizacja pierścienia tęczówki w pseudobiegunowym układzie współrzędnych jest kluczowa dla uzyskania jednorodnej reprezentacji niezależnej od warunków akwizycji obrazu. Rysunek 3.1 przedstawia zdjęcie oka z wykrytymi przez nasz program granicami źrenicy oraz tęczówki.



Rysunek 3.1: Obraz oka z naniesionymi (na czerwono) granicami źrenicy i tęczówki wykrytymi algorytmem.

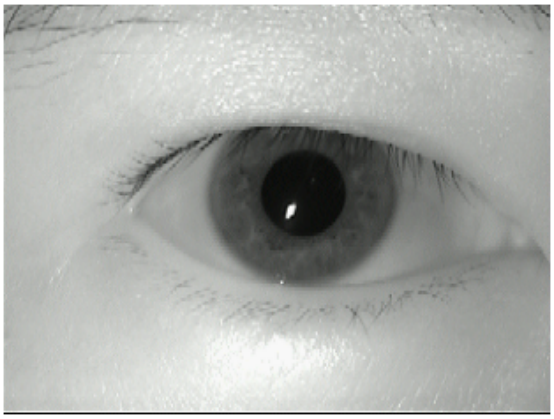
3.1. Przekształcenie do skali szarości

Na początku każda klatka lub zdjęcie RGB jest konwertowane do jednokanałowego obrazu w skali szarości. Proces ten realizowany jest przez uśrednienie wartości pikseli w trzech kanałach barwnych, co zapewnia równomierne odwzorowanie intensywności światła z każdej warstwy.

W implementacji korzystamy ze wzoru

$$g(i,j) = \left\lfloor \frac{R(i,j) + G(i,j) + B(i,j)}{3} \right\rfloor,$$

gdzie $g(i,j)$ oznacza wartość piksela w skali szarości, a R, G, B odpowiednio składowe czerwieni, zieleni i niebieskiego. Otrzymany obraz (przykładowo na rysunku 3.2) stanowi podstawę dla kolejnych operacji przetwarzania.



Rysunek 3.2: Przykładowy obraz oka po konwersji do skali szarości.

3.2. Globalna binaryzacja

Binaryzacja oparta jest na wyznaczeniu progu P, który stanowi średnią jasność całego obrazu w skali szarości. Dokładny wzór przyjmuje postać $P = \frac{1}{h \cdot w} \sum_{i=0}^{h-1} \sum_{j=0}^{w-1} A(i,j)$, gdzie h i w to wymiary obrazu (odpowiednio wysokość i szerokość), a $A(i,j)$ to wartość natężenia piksela. Do detekcji źrenicy i tęczówki wyznaczyliśmy dwa różne progi binaryzacji: $P_p = \frac{P}{X_P}$ dla źrenicy oraz $P_l = \frac{P}{X_l}$ dla tęczówki. Wartości współczynników X_P i X_l dobraliśmy eksperymentalnie tak, aby tęczówka i źrenica zostały poprawnie wykryte na jak największej liczbie zdjęć. Ostatecznie ustaliliśmy wartość $X_P = 4.5$ oraz wartość $X_l = 1.6$. Procedura binaryzacji sprowadza się do porównania każdego piksela ze wspomnianym progiem, gdzie piksele jaśniejsze od progu przyjmują wartość 255, a pozostałe 0.

3.3. Detekcja granic źrenicy

Detekcja granic źrenicy jest jednym z kluczowych etapów segmentacji tęczówki i wymaga precyzyjnego wyznaczenia optymalnego progu binaryzacji. Celem jest uzyskanie binarnej maski, w której źrenica będzie widoczna jako wyraźny, zwarty obiekt o jednolitej strukturze. Wybór

progu jest procesem eksperymentalnym, w którym kluczowym kryterium jest uzyskanie źrenicy jako jedynego lub przynajmniej największego zwartego obiektu w masce. Zbyt niski próg powoduje włączenie obszarów ciemnych poza źrenicą, np. rzęs czy powiek, podczas gdy zbyt wysoki – powoduje fragmentację źrenicy lub całkowite jej zanikanie. W niniejszej pracy ustaliliśmy doświadczalnie współczynnik binaryzacji $X_P = 4.5$.

Nawet po odpowiednio dobranej binaryzacji często w masce pojawiają się drobne artefakty, takie jak szum, dziury lub fragmenty odbić świetlnych. Do usunięcia tych elementów stosowane są operacje morfologiczne, które pozwalają poprawić jakość maski źrenicy. Najpierw wykonuje się operację zamknięcia morfologicznego, która wypełnia drobne dziury wewnątrz maski źrenicy, następnie stosuje się erozję, eliminując tym samym mniejsze struktury oraz fragmenty pozostałych artefaktów. Aby odzyskać pierwotny rozmiar źrenicy, realizowana jest dylatacja, a następnie filtr medianowy usuwa punktowy szum. Po tym etapie pozostają jeszcze czasem pojedyncze fragmenty większych artefaktów. Aby usunąć wszelkie pozostałe niepożądane obiekty, maska jest analizowana pod kątem komponentów spójnych, a ostatecznie wybierany jest tylko największy obiekt, którym powinna być źrenica. Zapewnia to funkcja 'keep_largest_object'. Końcowym krokiem jest zastosowanie filtru Gaussa, co pozwala na wygładzenie i stabilizację granicy źrenicy. Dokładny algorytm opisany jest poniżej.

Algorithm 1: Czyszczenie maski źrenicy (`clean_pupil`)

Data: `binary_image`

Result: `(x_pupil, y_pupil, r_pupil)`

```
kernel_5 ← ones(5×5)
kernel_3 ← ones(3×3)
img ← binary_image
img ←morphologyEx(img, CLOSE, kernel_5)
img ← erode(img, kernel_5, iters=2)
img ← dilate(img, kernel_3, iters=2)
img ← medianBlur(img, 5)
img ← keep_largest_object(img)
img ← GaussianBlur(img, 9×9)
img ← morphologyEx(img, OPEN, kernel_5)
(H,V) ← show_projections(img, binary_image)
return find_pupil(H,V)
```

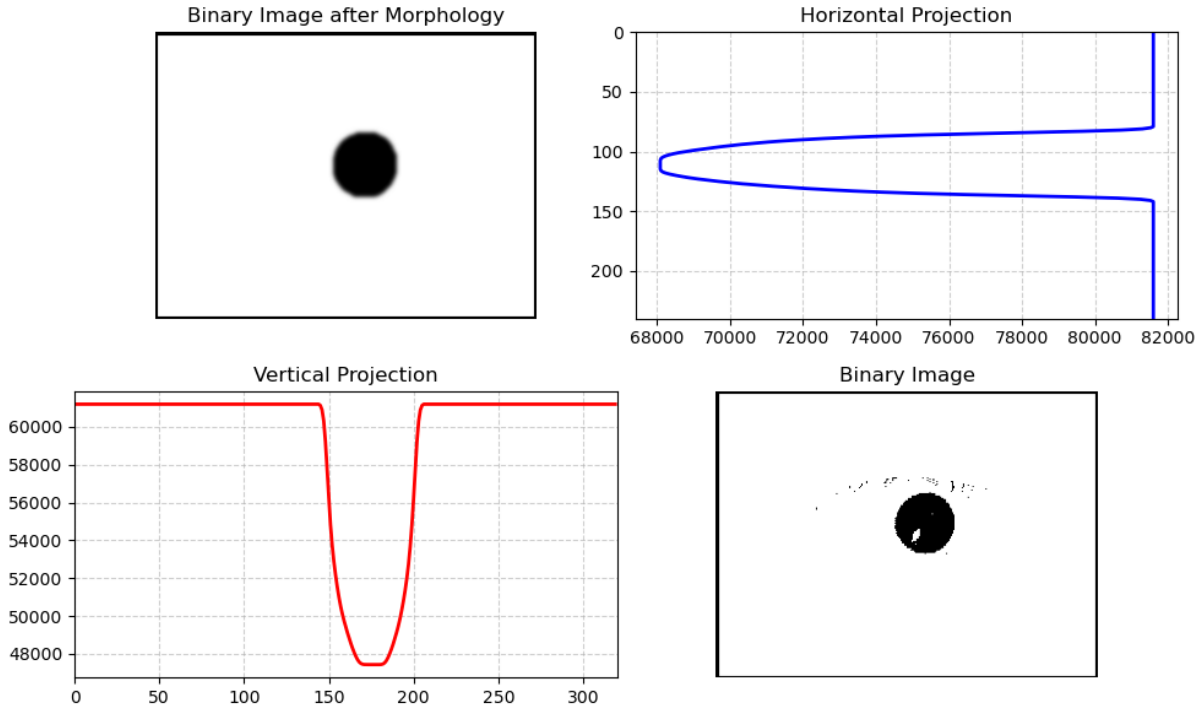
Przykład zdjęcia przetworzonego operacjami morfologicznymi, które według naszych testów

pozwalają poprawnie wykryć źrenicę w największej liczbie przypadków, przedstawia rys. 3.3.

Rysunek 3.3: Binarna maska źrenicy po zastosowaniu operacji morfologicznych.

Kolejnym etapem detekcji granicy źrenicy jest zastosowanie projekcji pionowej oraz poziomej na oczyszczonej masce. Projekcje te pozwalają na precyzyjne wyznaczenie środka źrenicy, jako punktu o najmniejszej wartości sumy jasności.

Na podstawie projekcji pionowej $V(x)$ i poziomej $H(y)$ (rysunek 3.4) określamy punkt (x_p, y_p) - środek źrenicy. W tym celu wybieramy punkt środkowy (medianę) ze wszystkich punktów, dla których minimum jest osiągane.



Rysunek 3.4: Projekcja pozioma (niebieska) i pionowa (czerwona) maski źrenicy. Minima wskazują środek źrenicy, a skoki wartości - jej krawędzie.

Następnie w celu określenia granic źrenicy analizowane są różnice między sąsiednimi warto-

ściami projekcji. Wektory różnicowe $\Delta H(y) = |H(y) - H(y-1)|, \quad \Delta V(x) = |V(x) - V(x-1)|$ wykazują znaczące zmiany w punktach krawędzi źrenicy. Zestawiając pierwsze i ostatnie istotne zmiany na obu osiach, wyznaczany jest promień źrenicy jako średnia tych odległości, co zapewnia dobrą odporność na ewentualne nieregularności. Opisaną logikę przedstawia poniższy algorytm.

Algorithm 2: `Detekcja źrenicy (find_pupil)`

Data: `projekcje H(y), V(x)`

Result: `(x_pupil, y_pupil, r_pupil)`

```
y_pupil ← median(argmin H)
x_pupil ← median(argmin V)
DeltaH ← |H(y) - H(y-1)|
DeltaV ← |V(x) - V(x-1)|
Edges_y ← { y : DeltaH(y) > 0.3 * max DeltaH }
Edges_x ← { x : DeltaV(x) > 0.3 * max DeltaV }
r_y ← 0.5 * (max Edges_y - min Edges_y)
r_x ← 0.5 * (max Edges_x - min Edges_x)
r_pupil ← 0.5 * (r_x + r_y)
return (x_pupil, y_pupil, r_pupil)
```

3.4. Detekcja granicy tęczówki

Proces detekcji granicy tęczówki przebiega podobnie jak detekcja źrenicy, lecz ze względu na większą złożoność strukturalną tęczówki stosuje się inne parametry binaryzacji i dodatkowe operacje morfologiczne. Eksperymentalnie ustalono próg binaryzacji $X_I = 1.6$, niższy niż w przypadku źrenicy, ze względu na większą jasność i teksturowość tęczówki. Początkowa binaryzacja często zawiera niepożądane elementy, jak powieki, rzęsy czy refleksy.

Aby je usunąć, stosuje się sekwencję operacji morfologicznych: dylatację (rozszerzenie białych obszarów), selekcję największego obiektu (zachowanie regionu z tęczówką), erozję (odsłonięcie konturów), ponowną dylatację (rekonstrukcja kształtu) oraz zamykanie (wypełnianie dziur i wygładzanie). Operacje te są powtarzane i uzupełniane intensywniejszą erozją w celu dokładnego wyodrębnienia czarnej tęczówki.

Na koniec stosuje się rozmycia (klasyczne i Gaussa) w celu redukcji szumu i wygładzenia krawędzi, z zachowaniem największego obszaru oraz operację otwierania, która usuwa drobne artefakty. Szczegóły algorytmu przedstawiono poniżej.

Algorithm 3: Czyszczenie maski tęczówki (clean_iris)

Data: binary_image

Result: iris

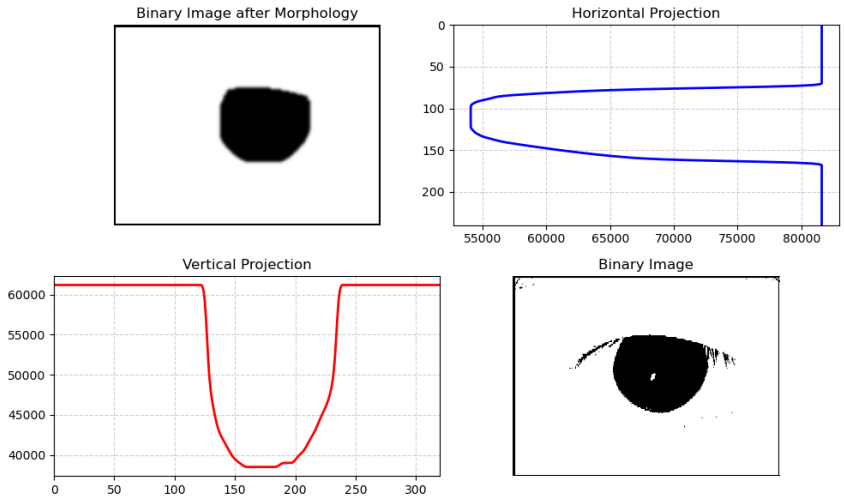
```
K_L ← ones(10 × 10)
K_M ← ones(5 × 5)
K_S ← ones(3 × 3)
img ← binary_image
img ← dilate(img, K_M, iters=1)
img ← keep_largest_object(img)
img ← erode(img, K_S, iters=5)
img ← dilate(img, K_M, iters=5)
img ← morphologyEx(img, CLOSE, K_L)
img ← dilate(img, K_L, iters=1)
img ← morphologyEx(img, CLOSE, K_L)
img ← erode(img, K_S, iters=10)
img ← blur(img, 5 × 5)
img ← keep_largest_object(img)
img ← morphologyEx(img, OPEN, K_L)
img ← GaussianBlur(img, 9 × 9)
(H, V) ← show_projections(img, binary_image)
return find_iris(H, V)
```

Obraz po tych wszystkich przekształceniach jest znacznie uproszczony i oczyszczony, a jego analiza pod kątem lokalizacji tęczówki staje się łatwiejsza.Przykład obrazu przetworzonego za pomocą tych operacji można zobaczyć na rys. 3.5.

Rysunek 3.5: Binarna maska tęczówki po obróbce morfologicznej.

Po operacjach morfologicznych następuje określenie granicy tęczówki. Ponieważ środek tę-

czówki jest identyczny ze środkiem źrenicy wyznaczonym wcześniej, do określenia promienia stosuje się analogicznie projekcje pionową i poziomą. Projekcje maski tęczówki $H(y)$ oraz $V(x)$ (rysunek 3.6) analizuje się podobnie jak w przypadku źrenicy, wykorzystując różnice między sąsiednimi wartościami.



Rysunek 3.6: Projekcje pozioma i pionowa dla maski tęczówki wraz z zaznaczonymi krawędziami.

Promień tęczówki jest jednak trudniejszy do precyzyjnego określenia, dlatego wprowadza się ważone uśrednienie promieni na podstawie krawędzi wykrytych w obu kierunkach:

$$r_i = \frac{3\,r_x + r_y}{4},$$

co pozwala uwzględnić fakt, że w pionie tęczówka zazwyczaj ma mniejszy wymiar z powodu zasłonięcia przez powiekę górną, co widać również na rysunku 3.5. Poniższy algortym przedstawia implementację naszej logiki w kontekście znajdowania promienia tęczówki.

Algorithm 4: `Detekcja granicy tęczówki (find_iris)`

Data: `projekcje H(y), V(x)`

Result: `r_iris`

```
DeltaH ← |H(y) - H(y-1)|, DeltaV ← |V(x) - V(x-1)|
Edges_y ← { y : DeltaH(y) > 0.4 * max DeltaH }
Edges_x ← { x : DeltaV(x) > 0.4 * max DeltaV }
r_y ← 0.5 * (max Edges_y - min Edges_y)
r_x ← 0.5 * (max Edges_x - min Edges_x)
r_iris ← 0.75 * r_x + 0.25 * r_y
return r_iris
```

3.5. Rozwinięcie tęczówki do prostokąta

Ostatnim kluczowym krokiem segmentacji jest transformacja wykrytej tęczówki z formy pierścieniowej na prostokątną reprezentację. Celem tego zabiegu jest normalizacja obrazu tęczówki, pozwalająca na późniejszą ekstrakcję cech niezależnych od zmiany skali, rotacji i oświetlenia. W tym celu wykorzystuje się przekształcenie współrzędnych biegunowych do kartezjańskich zgodnie z poniższym wzorem:

$$x = x_p + \big(r_p + r(r_i - r_p)\big)\cos\theta, \quad y = y_p + \big(r_p + r(r_i - r_p)\big)\sin\theta,$$

gdzie $r \in [0, 1]$ jest znormalizowanym promieniem od granicy źrenicy ($r = 0$) do tęczówki ($r = 1$), a parametr θ przyjmuje wartości od 0° do 360°, zapewniając pełne pokrycie obwodu. Wynikiem tej operacji jest obraz prostokątny o ustalonej rozdzielczości 64×360 pikseli, w którym kierunek poziomy odpowiada zmianom kąta, a kierunek pionowy promieniowi. Dzięki temu rozwiązaniu możliwe jest precyzyjne porównywanie wzorców tęczówki pomiędzy różnymi zdjęciami tej samej osoby lub różnych osób. Przykładowa tęczówka rozwinięta do formy prostokąta zaprezentowana jest na rysunku 3.7.

Rysunek 3.7: Rozwinięcie pierścienia tęczówki do reprezentacji prostokątnej.

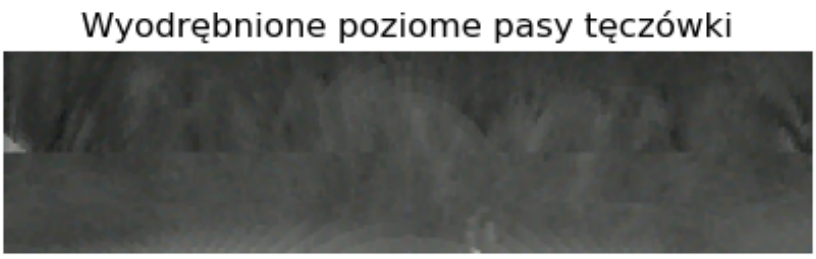
4. Rozpoznawanie na podstawie rozwiniętej tęczówki

Proces rozpoznawania rozpoczyna się po zakończonej segmentacji i rozwinięciu pierścienia tęczówki do prostokątnego obrazu o wymiarach 64×360 pikseli. Celem dalszych kroków jest utworzenie kompaktowego i odpornego na zakłócenia deskryptora, zwanego *kodem tęczówki*, którego porównanie umożliwia jednoznaczną identyfikację osoby.

4.1. Algorytm Daugmana – podział na osiem pasów

Rozwinięty prostokąt tęczówki jest dzielony na osiem współśrodkowych pasów radialnych o równej szerokości pionowej. Aby zminimalizować wpływ powiek i rzęs, każdy pas obejmuje jedynie część pełnego obwodu kąta. Cztery środkowe pasy opisują aż 330° pełnego koła, z pominięciem trzydziestostopniowego sektora, który w naszej implementacji znajduje się w górnej części tęczówki. Dwa kolejne pasy zawierają fragmenty o szerokości 226°, natomiast dwa zewnętrzne – po 180°. Tak przygotowany zestaw pasów, zaprezentowany na rysunku 4.1 (przed ucięciem) oraz rysunku 4.2 (po ucięciu), pozwala skupić się na najbardziej informatywnych fragmentach wzoru tęczówki, jednocześnie eliminując obszary najbardziej narażone na zasłonięcie.

Rysunek 4.1: Graficzna prezentacja granic centralnego regionu oraz wyciętych fragmentów pasów (linie czerwone); krawędzie pionowe oznaczono kolorem żółtym.



Rysunek 4.2: Wyodrębnione poziome pasy tęczówki wykorzystywane w dalszej analizie.

4.2. Transformata falkowa Gabora

Każdy z ośmiu pasów jest przetwarzany niezależnie. Najpierw tworzona jest jednowymiarowa reprezentacja sygnału, uzyskiwana przez uśrednianie intensywności kolumn z zastosowaniem okna Gaussowskiego ważonego wokół środka pasa. Dzięki temu sygnał zachowuje cechy tekstury tęczówki, a jednocześnie tłumiony jest szum z obszarów skrajnych. Implementację przedstawia poniższy algorytm.

Algorithm 5: `1-wymiarowa reprezentacja pasa (compute_1d_representation)`

Data: `obraz pasa band (h x w x 3)`

Result: `sygnał 1D I(x)`

```
band_gray ← średnia z kanałów RGB
center ← h / 2, sigma ← h / 8
weights[y] ← exp(-(y - center) ** 2 / (2 * sigma ** 2))
weights[y] ← weights[y] / sum(weights[i])
I(x) ← sum(band_gray[y,x] * weights[y])
return I(x)
```

Sygnał $I(x)$ podlega następnie dekompozycji przy użyciu falek Gabora

$$G(x_0,\ f,\ \sigma) = \exp\left(-\frac{(x-x_0)^2}{\sigma^2}\right) \exp\left(-i\,2\pi f(x-x_0)\right),$$

gdzie x_0 określa położenie centrum falki, a f jej częstotliwość. Zgodnie z oryginalną specyfikacją Daugmana przyjmuje się zależność

$$\sigma = \tfrac{1}{2}\,\pi f.$$

W każdym pasie rozmieszcza się równomiernie 128 centrów falek. Dla każdego centrum wyliczany jest współczynnik

$$c_k = \sum_x I(x)\, G(x_k,\ f,\ \sigma),$$

gdzie x_k to położenie k-tej falki. Amplituda $|c_k|$ zależy od lokalnego kontrastu, natomiast faza $\arg(c_k)$ od struktury wzoru. Sposób wyznaczenia współczynników rozwinięcia sygnałów $I(x)$ względem falek Gabora przedstawia poniższy algorytm.

Algorithm 6: `Obliczanie wsp. Gabora (compute_gabor_coefficients)`

Data: `sygnał I(x) długości L, częstotliwość f, liczba współczynników N`

Result: `wektor zespolony [c_1, ..., c_N]`

```
centers ← równomierny podział [0, L - 1] na N punktów
sigma ← 0.5 * pi * f
for k = 1 to N do
    window[i] ← exp(-((i - centers[k]) ** 2) / (sigma ** 2))
    mod[i] ← exp(-1j * 2 * pi * f * (i - centers[k]))
    c[k] ← sum(I[i] * window[i] * mod[i])
return c
```

4.3. Wyznaczenie kodu tęczówki

Uśredniony za pomocą okna Gaussowskiego sygnał $I(x)$ i wyznaczone na jego podstawie współczynniki rozwinięć względem falek Gabora wykorzystujemy do stworzenia kodu tęczówki. Ponieważ faza współczynników Gabora jest mniej wrażliwa na zmiany oświetlenia, współczynnik zostaje zakodowany dwubitowo wyłącznie na podstawie ćwiartki, w której leży:

$$\arg(c_k) \in \begin{cases} [0,\ \pi/2) & \to 00, \\ [\pi/2,\ \pi] & \to 01, \\ (-\pi, -\pi/2] & \to 11, \\ (-\pi/2, 0) & \to 10. \end{cases}$$

Otrzymujemy w ten sposób 256 bitów na pas, a łącznie dla całej tęczówki ciąg 2048 bitów uzyskiwany przez nas zgodnie z poniższym algorytmem.

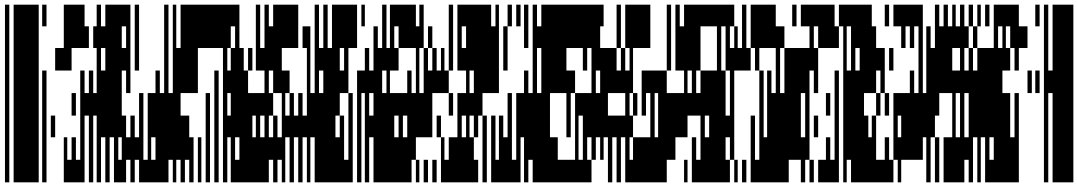
Algorithm 7: `Kodowanie fazowe współczynników (encode_coefficient)`

Data: `zesp. współczynniki {c_k}`

Result: `ciąg bitów code`

```
for każdy c_k do
    angle ← arg(c_k)
    koduj fazę zgodnie z podziałem ćwiartek → b_1 b_2
    code ← code | b_1 b_2
return code
```

Wynik przedstawiany jest przez nas jako prostokąt o wymiarach 8 na 256 punktów, gdzie czarny punkt to bit 0 a biały punkt to bit 1, ukazany na rysunku 4.3.



Rysunek 4.3: Przykładowy *IrisCode* (2048 bitów) wygenerowany z obrazu z rys. 3.2.

4.4. Porównywanie kodów

Miara podobieństwa między dwoma kodami tęczówki $C^{(i)}$ i $C^{(j)}$ definiowana jest jako odległość Hamminga obu wektorów bitowych, wyrażona wzorem

$$d(C^{(i)}, C^{(j)}) = \frac{1}{2048} \sum_{k=1}^{2048} \left| b_k^{(i)} - b_k^{(j)} \right|,$$

gdzie $b_k^{(i)}$ oznacza k-ty bit kodu $C^{(i)}$. Implemntację przedstawia poniższy algorytm.

Algorithm 8: `Hamming distance (hamming_distance_np)`

Data: `wektory bitów code1, code2`

Result: `odległość Hamminga`

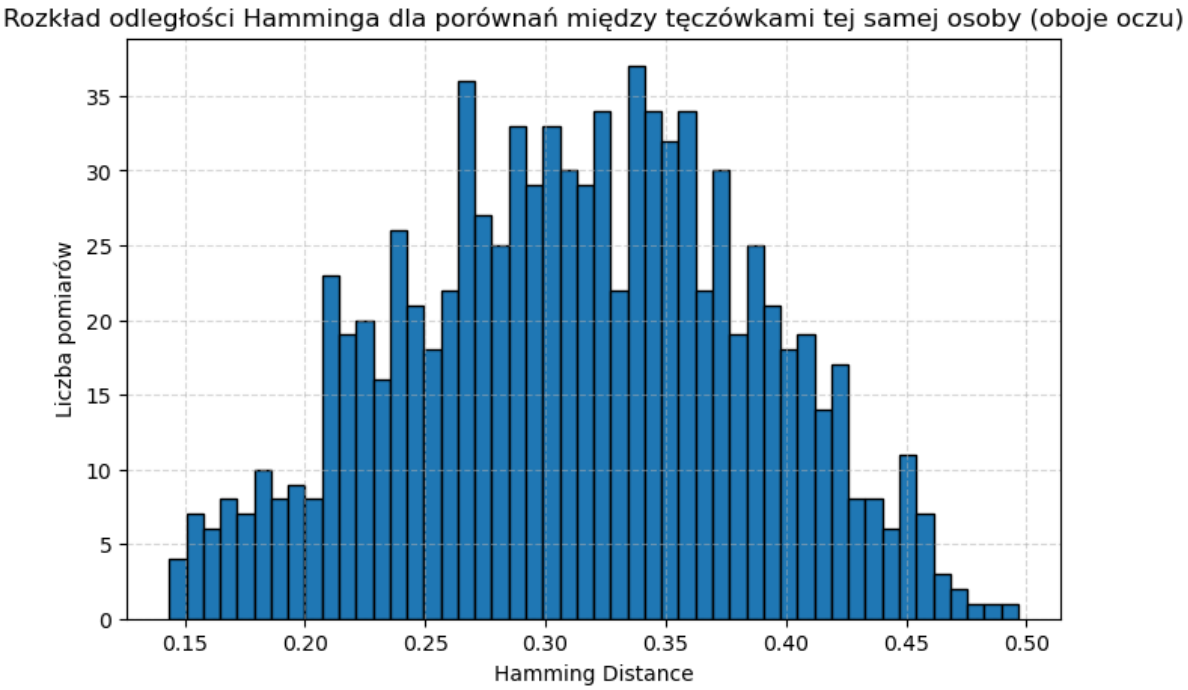
```
if len(code1) ≠ len(code2) then
    raise ValueError("Kody muszą mieć tę samą długość")
arr1 ← np.array([int(b) for b in code1])
arr2 ← np.array([int(b) for b in code2])
diff ← np.sum(arr1 != arr2)
return diff / len(code1)
```

W idealnym przypadku odległość Hamminga dla dwóch obrazów tej samej tęczówki powinna wynosić zero, jednak w praktyce rozbieżności w warunkach akwizycji powodują, że wartości te układają się w charakterystyczny rozkład.

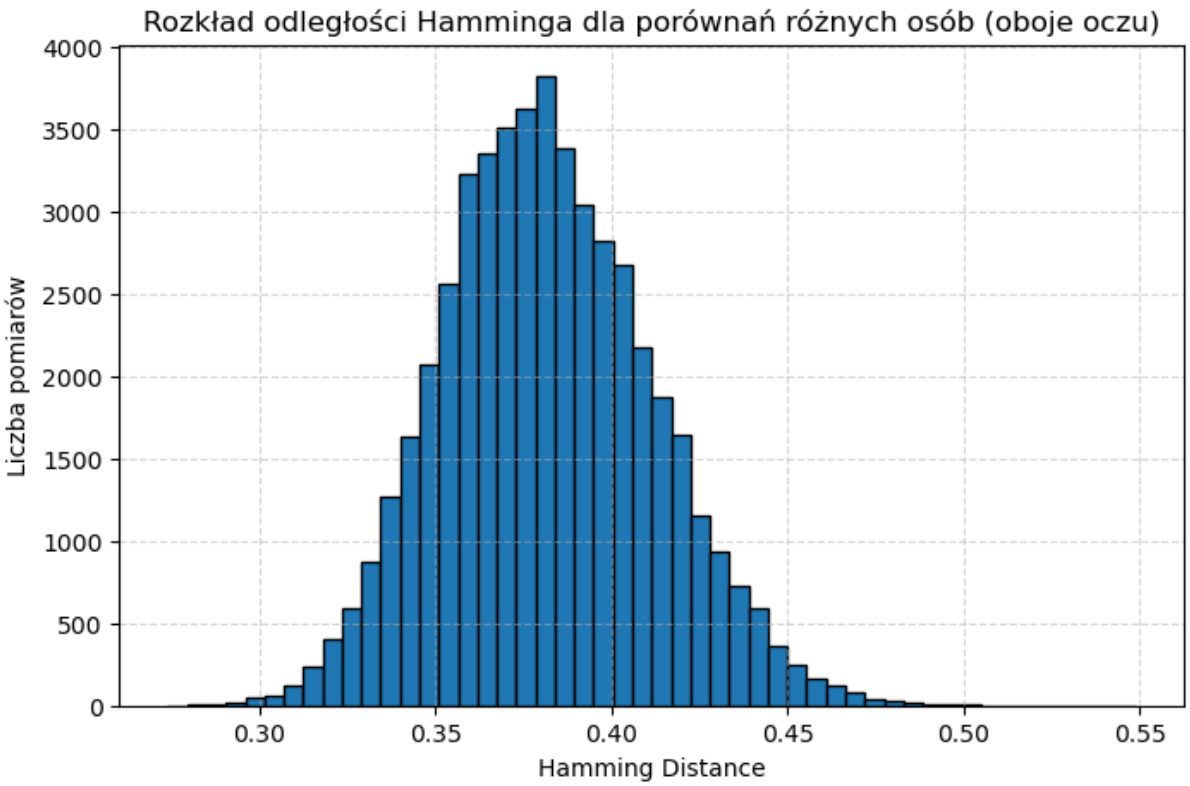
Wybór progu Hamminga determinuje poziom pewności rozpoznania: kody o odległości poniżej progu uznaje się za zgodne, pozostałe – za różne.

5. Wyniki

Poniżej na rysunkach 5.1 i 5.2 przedstawiono dwa histogramy rozkładów odległości Hamminga pomiędzy kodami tęczówek.



Rysunek 5.1: Histogram odległości Hamminga pomiędzy kodami tęczówki tej samej osoby (intra-person). Dla każdej osoby z zestawu porównano wszystkie pary jej pięciu obrazów.



Rysunek 5.2: Histogram odległości Hamminga pomiędzy *różnymi* osobami (cross-person). Każdy dystans obliczono dla par kodów pochodzących od dwóch różnych osób.

Na rysunku 5.1 widzimy, że odległości Hamminga dla tej samej osoby koncentrują się w niższym zakresie (około 0.27–0.35), co oznacza dobrą spójność kodu przy różnych ujęciach oka. Statystyki dla tych odległości to średnia = 0.3126, odchylenie standardowe = 0.07246 i mediana = 0.31519. Z kolei na rysunku 5.2 rozkład dystansów między różnymi osobami jest przesunięty w prawo (około 0.34–0.42) i ma nieco wyższe wartości średnie: średnia = 0.38147, odchylenie standardowe = 0.02976 i mediana = 0.37988.

Istotną cechą tych dwóch rozkładów jest ich częściowe pokrycie, ale również wyraźne przesunięcie średnich. Dzięki temu można dobrać próg Hamminga (np. około 0.34–0.35), poniżej którego kody uznajemy za należące do tej samej osoby, a powyżej – do różnych osób. Taki wybór progu minimalizuje ryzyko fałszywych akceptacji (gdy kody dwóch różnych tęczówek traktuje jako tożsame) oraz fałszywych odrzuceń (gdy dwa obrazy tej samej osoby uznajemy za różne). Z punktu widzenia bezpieczeństwa najważniejsze jest jednak, by unikać błędów pierwszego rodzaju, czyli zapobiegać sytuacjom, gdy tęczówki pochodzące od dwóch różnych osób zostaną uznane za 'matchujące'. To bowiem właśnie w takich sytuacjach nieupoważnione osoby mogą uzyskać dostęp do wrażliwych danych. Dlatego postanowiliśmy obniżyć próg akceptacji i ustalić go na 0.3219, czyli dwa odchylenia standardowe poniżej średniej w rozkładzie odległości Hamminga dla porównań

między tęczówkami różnych osób.

Aby usprawnić porównywanie kodów dwóch różnych tęczówek, stworzyliśmy funkcję `compare_two_images`, która przyjmuje jako argumenty identyfikatory dwóch obrazów tęczówki (tej samej lub różnych osób), parametr σ filtra Gaussowskiego oraz próg Hamminga $\tau = 0.3219$. W pierwszej kolejności wczytywane są surowe pliki `eye_pXX_eyeYZ.bmp`, następnie za pomocą funkcji `predictIris(..., sigma)` generowane są kody tęczówki, z których obliczana jest odległość Hamminga

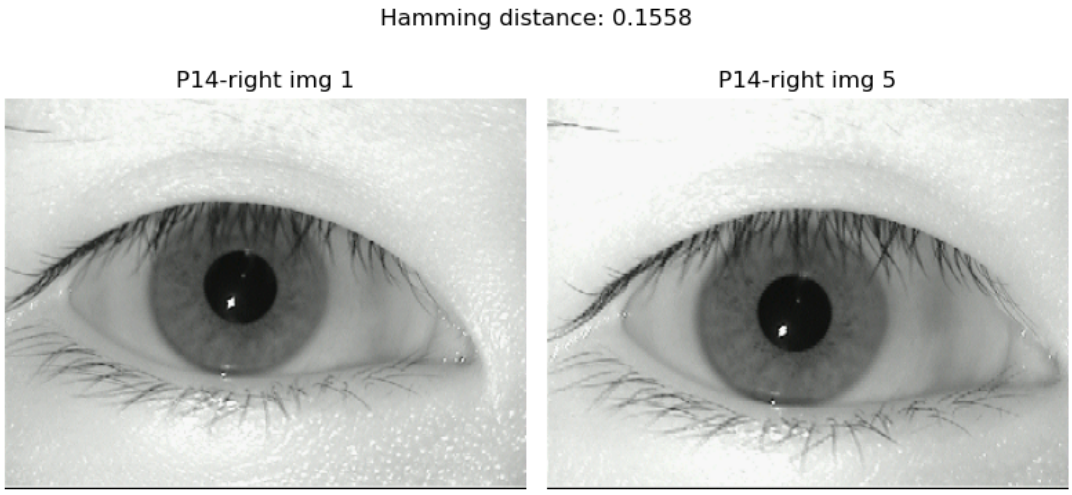
$$d = \text{Hamming}(\text{code}_1, \text{code}_2).$$

Oba obrazy wyświetlane są obok siebie z tytułem „Hamming distance: ", natomiast w konsoli drukowany jest komunikat „Zgodność tęczówek!" jeśli $d \leqslant \tau$ lub „Niezgodność tęczówek!" w przeciwnym razie. Funkcja zwraca wartość d, co pozwala na dalszą analizę wyników.

Przykład użycia tej funkcji w kodzie:

```
d = compare_two_images(
    person_id1=14,
    eye1="right",
    index1=1,
    index2=5,
    f_value=np.pi,
    sigma=4.42
)
```

Powyższe wywołanie ilustruje też rysunek 5.3.



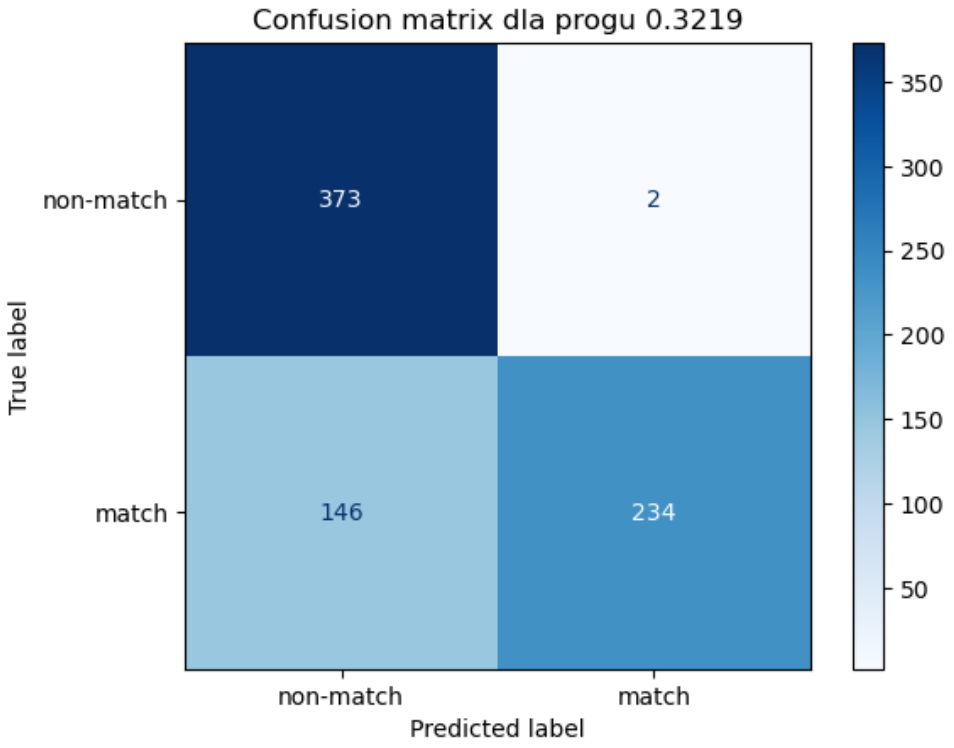
Rysunek 5.3: Przykład wywołania `compare_two_images(person_id1=14, eye1="right", index1=1, index2=5, f_value=np.pi, sigma=4.42)`

Na załączonym rysunku uzyskano:

$$d = 0.1558$$

oraz komunikat: `Zgodność tęczówek!`

W celu sprawdzenia jakości działania modelu i słuszności wybranego progu przeprowadziliśmy eksperyment na 755 parach zdjęć. Fotografie do eksperymentu zostały wybrane w sposób losowy, ale z zastrzeżeniem, aby w przybliżeniu w zbiorze testowym było tyle samo przypadków tęczówek 'matchujących' i tęczówek pochodzących od różnych osób. Wyniki przeprowadzonego eksperymentu można zobaczyć na rysunku 5.4, który przedstawia macierz pomyłek naszego modelu. Z kolei w tabeli 5.1 zostały przedstawione wartości metryk opisujących jakość naszych predykcji. Powodem do zadowolenia jest bardzo wysoka wartość metryki Precision (0.9915) - kluczowej przy minimalizacji błędu pierwszego rodzaju - przy jednoczesnym dobrym wyniku w metryce Gini (0.6105) świadczącej o poprawnym zachowaniu modelu we wszystkich przypadkach. Zbalansowany zbiór testowy pozwala nam wyciągać wnioski również z metryki Accuracy. Wartość 0.804 oznacza, że nasz model jest znacznie lepszy od modelu losowo dopasowującego etykiety do danych.



Rysunek 5.4: Macierz pomyłek dla progu Hamminga $\tau = 0.3219$. W wierszach znajdują się etykiety rzeczywiste (non-match/match), w kolumnach etykiety przewidywane.

Tabela 5.1: Metryki jakości rozpoznania przy progu Hamminga $\tau = 0.3219$

Metryka	Wartość
Precision	0.9915
Recall	0.6158
Accuracy	0.8040
F1-score	0.7597
AUC	0.8052
Gini	0.6105

6. Podsumowanie

6.1. Wnioski

Dzięki podejściu opartemu na metodzie Daugmana oraz zastosowaniu falki Gabora uzyskaliśmy wyraźne rozdzielenie rozkładu odległości Hamminga dla par obrazów tej samej osoby i par obrazów różnych osób. Średnia odległość intra-personalna (0.3126) jest znacznie mniejsza od wartości inter-personalnej (0.3815), co pozwala przyjąć próg $\tau = 0.3219$ minimalizujący błędy pierwszego rodzaju, a jednocześnie akceptowalnie kontrolujący liczbę fałszywych odrzuceń.

Uzyskane w eksperymencie metryki — w szczególności bardzo wysoka precyzja 0.9915 przy stosunkowo dobrej czułości 0.6158 — potwierdzają, że układ idealnie nadaje się do zastosowań wymagających najwyższego poziomu bezpieczeństwa (np. ochrona dostępu), gdzie kluczowe jest unikanie akceptacji nieautoryzowanych użytkowników. Jednocześnie wartość AUC = 0.8052 (Gini = 0.6105) wskazuje na solidną zdolność systemu do separacji klas w całym zakresie progów.

W praktycznych wdrożeniach można rozważyć następujące kierunki rozwoju:

- **Dynamiczne dostrajanie progu:** Automatyczne dostosowanie τ na podstawie profilu użytkownika lub bieżących warunków akwizycji (np. poziomu oświetlenia).
- **Optymalizacja parametrów falek:** Eksperymenty z liczbą centrów falki, wartością częstotliwości oraz szerokością okna Gaussowskiego (σ) mogą dodatkowo zwiększyć separację intra- i inter-osobową.
- **Rozszerzenie bazy uczącej:** Włączenie większej liczby osób i różnorodnych warunków (różne aparaty, kąty, poziomy oświetlenia) pozwoli jeszcze lepiej ocenić stabilność metody, zwłaszcza w tzw. „dzikich" scenariuszach.

Podsumowując, zaproponowany pipeline segmentacji i kodowania tęczówki zgodnie z metodą Daugmana daje rozdzielne rozkłady intra- i inter-osobowe, co umożliwia wiarygodne rozpoznawanie tożsamości na podstawie samego obrazu oka. Uzyskane wyniki potwierdzają jego przydatność w zastosowaniach, gdzie priorytetem jest wysoka pewność autoryzacji.

6.2. Napotkane problemy

Podczas realizacji eksperymentu napotkaliśmy na kilka trudności związanych z jakością zdjęć, które miały wpływ na skuteczność detekcji i rozpoznawania tęczówki. Największe problemy związane były z:

- **Słaba jakość zdjęć:** Niektóre zdjęcia były słabej jakości, co utrudniało wyodrębnienie szczegółów tęczówki. Niska rozdzielczość oraz zbyt duża kompresja zdjęć prowadziły do utraty szczegółów, które były kluczowe dla skutecznej analizy.
- **Obrazy w okularach:** Niektóre ze zdjęć zostały wykonane podczas noszenia okularów, które zakrywały część tęczówki i wprowadzały odblaski, utrudniając precyzyjną detekcję granic.
- **Gęste brwi:** Na niektórych zdjęciach brwi były gęste i umiejscowione w taki sposób, że zakrywały część tęczówki, przez co system błędnie identyfikował te obszary jako część tęczówki lub źrenicy.
- **Odbicia i refleksy:** Zdarzały się również sytuacje, w których na zdjęciach pojawiały się odbicia lub refleksy światła, co powodowało zakłócenia w obrazie i utrudniało dalszą analizę.

Podczas pracy nad projektem zauważyliśmy, że jakość danych wejściowych — czyli zdjęć oka — ma ogromny wpływ na skuteczność całego systemu. W szczególności, wiele problemów wynikało z niskiej jakości fotografii: rozmycia, złego oświetlenia, obecności okularów, a nawet przysłaniania tęczówki przez brwi. Zdjęcia wykonane w takich warunkach często prowadziły do błędnej segmentacji lub zniekształconego kodowania, co negatywnie wpływało na rozpoznawanie.

Z naszego doświadczenia wynika, że najlepsze rezultaty osiąga się przy zdjęciach wykonanych w dobrej rozdzielczości, z wyraźnie widoczną strukturą tęczówki. Warto unikać fotografowania w okularach, ponieważ mogą one zasłaniać część oka lub powodować odbicia światła.

Oświetlenie również ma kluczowe znaczenie. Zbyt ciemne lub zbyt jasne zdjęcia są trudne do dalszej obróbki.

Zdarzało się też, że brwi lub rzęsy nachodziły na oko, co utrudniało poprawną detekcję granic tęczówki — szczególnie przy gęstszych brwiach. Dlatego warto zwrócić uwagę na kadr i zadbać o to, żeby nic nie zasłaniało tęczówki. Równie ważna jest odległość i kąt fotografowania — zdjęcia robione z boku lub zbyt blisko oka często powodowały deformacje. Najlepiej fotografować z niewielkiej odległości, trzymając aparat naprzeciwko oka.

Dobrze jest również robić zdjęcia w miarę możliwości w podobnych warunkach — zachowanie spójności między zdjęciami poprawia ich jakość i ułatwia dalsze przetwarzanie.

Chociaż nie przeprowadzaliśmy osobnego badania nad jakością zdjęć, to powyższe wskazówki wynikają bezpośrednio z naszych obserwacji w trakcie pracy nad systemem i mogą znacznie poprawić jego działanie.

7. Bibliografia

- https://pages.mini.pw.edu.pl/~rafalkoj/www/?Dydaktyka:2024%2F2025:-_Biometria
- J. Daugman, "High confidence visual recognition of persons by a test of statistical independence", *IEEE Trans. Pattern Anal. Mach. Intell.*, 1993.
- Ślot, Krzysztof. *Wybrane zagadnienia biometrii.*