Biometria – Projekt 2

Rozpoznawanie człowieka na podstawie obrazu tęczówki

Jakub Półtorak
Michał Pytel

Kwiecień 2025

Spis treści

1 Wstęp

Celem drugiego projektu było opracowanie i zaimplementowanie kompletnego łańcucha przetwarzania pozwalającego na rozpoznawanie osób na podstawie zdjęć oczu. Implementacja miała zapewniać segmentację źrenicy, segmentację tęczówki, rozwinięcie wyodrębnionej tęczówki do postaci prostokątnej, ekstrakcję poszczególnych cech do kodu tęczówki za pomocą metody Daugmana, porównanie poszczególnych kodów za pomocą odległości Hamminga. Ocena działania metody miała zostać sprawdzona na ogólnodostępnym zbiorze danych *MMU Iris Dataset*

2 Opis aplikacji

2.1 Implementacja i środowisko

System został zaimplementowany w języku **Python**. W całości korzysta wyłącznie z bibliotek typu open source:

- **OpenCV–Python** – podstawowe operacje obrazowe (wczytywanie, zapisywanie, morfologia, filtrowanie),
- **NumPy** – szybkie obliczenia macierzowe,[1]
- **Matplotlib** – wizualizacje wyników i wszelakich wykresów
- **Seaborn** – wizualizacja heatmapy macierzy pomyłek
- **Itertools** – wygenerowanie wszystkich możliwych par różnych obrazów
- **Os** - operacje na plikach, generowanie nowych katalogów i nazw plików
- **Shutil** - używane do wyższego poziomu operacji na plikach i katalogach
- **Sys** - dostęp do zmiennych i funkcji środowiska w Pythonie
- **Typing** - typowanie statyczne, stosowane do krotek
- **Jupyter Lab** – środowisko eksperymentalne (notatnik `detekcja_teczowki.ipynb`).

2.2 Struktura projektu

Kod źródłowy składa się z notatnika Jupyter Notebook nazwanego `detekcja_teczowki.ipynb`, w którym zawarta jest cała logika biznesowa algorytmu i wszystkie niezbędne operacje do eksperymentów. Uruchomienie wszystkich komórek w dobrej kolejności pozwoli odtworzyć wszystkie uzyskane wyniki w raporcie. Dodatkowo wykorzystano zbiór danych *MMU-Iris-Database*, w którym zawarte są zdjęcia oczu używane w testowaniu metody. Niezbędne zdjęcia są umieszczone w katalogu *MMU-Iris-Database*, który w pewnym momencie przekształcany jest do katalogu *renamed_photos*. Ten ostatni katalog ostatecznie służy do analizy histogramowej metody Daugmana.

[1] Wszystkie operacje per–piksel zostały zwektorowane, co redukuje czas segmentacji dla obrazu

3 Metody przetwarzania

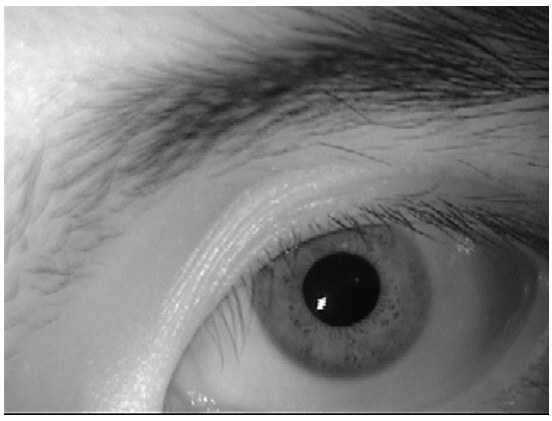
3.1 Segmentacja tęczówki

3.1.1 Przetwarzanie wstępne\konwersje

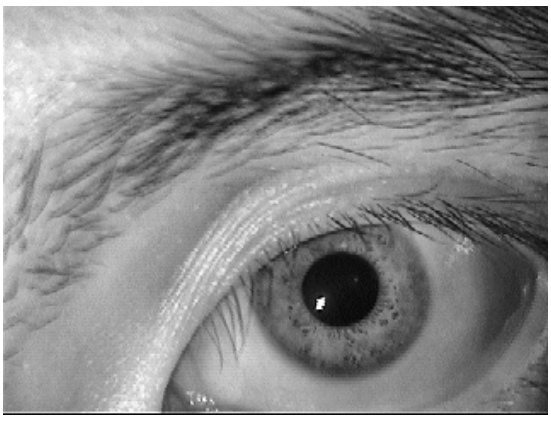
Przed rozpoczęciem procesu detekcji źrenicy zdjęcie poddawane jest wstępnej obróbce przez przekształcenie obrazu do skali szarości, jeśli nie jest już w takim formacie. Dodatkowo obiekt poddawany jest przekształceniu *CLAHE*, które ma na celu poprawę kontrastu zdjęcia. Z metody skorzystano poprzez użycie biblioteki `OpenCV`. Poprawa jest realizowana przede wszystkim lokalnie, zapobiega przeskalowaniu szumu, interpoluje także między kafelkami, aby unikać ostrych granic. Poniżej znajduje się pełna implementacja całego przetwarzania wstępnego:

```
_CLAHE = cv.createCLAHE(clipLimit=4.0, tileGridSize=(8, 8))

def to_gray(img_bgr: np.ndarray, equalize: bool = False) -> np.ndarray:
    gray = cv.cvtColor(img_bgr, cv.COLOR_BGR2GRAY)
    if equalize:
        gray = _CLAHE.apply(gray)
    return gray
```

Tak zaś prezentuje się porównanie zdjęcia przekształconego do skali szarości ze zdjęciem, na które dodatkowo naniesiono operację *CLAHE*. Oryginalnie zdjęcie jest już w skali szarości, więc wcześniej wprowadzone przekształcenie `to_gray(...)` nic nie zmienia. Rozważano osobę nr 40, lewe oko i pierwsze zdjęcie:



(a) Skala szarości



(b) Po zastosowaniu CLAHE

Rysunek 1: Porównanie obrazu w skali szarości oraz obrazu po wyrównaniu kontrastu metodą CLAHE.

3.1.2 Dodatkowe funkcje

W podejściu zastosowano także dodatkową funkcję `largest_component(...)`, która była wykorzystywana zarówno w detekcji źrenicy, jak i detekcji tęczówki. Służyła ona do wykrywania największego spójnego elementu maski. Pozostałe mniejsze fragmenty były pomijane w dalszych analizach. Poniżej znajduje się pełny kod tego przekształcenia:

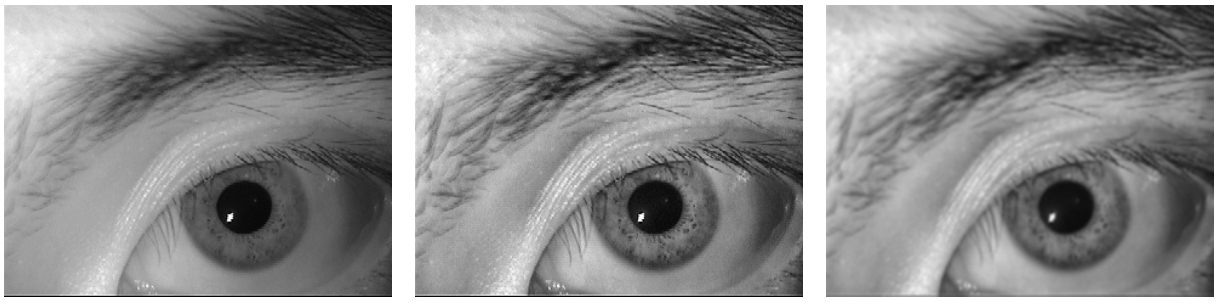
```python
def largest_component(mask_bin: np.ndarray) -> np.ndarray:
    num, labels, stats, _ = cv.connectedComponentsWithStats(mask_bin,
        connectivity=8)
    if num <= 1:  # tylko t o
        return np.zeros_like(mask_bin)
    # Indeks najwi kszego pola (1.. N 1 )
    areas = stats[1:, cv.CC_STAT_AREA]
    idx = 1 + int(np.argmax(areas))
    return np.where(labels == idx, 255, 0).astype(np.uint8)
```

3.1.3 Detekcja źrenicy

Proces skutecznej identyfikacji tożsamości na podstawie zdjęcia oka rozpoczęto od detekcji źrenicy. Cała logika wykrywania komponentu została opisana w funkcji `detect_pupil(...)`. Oto argumenty jakie funkcja przyjmowała i zwracała:

```python
def detect_pupil(
    img_path: str,
    gray: np.ndarray,
    X_P: float        = 4.5,
    kernel_size: int  = 5,
    morph_iter: int   = 1,
    debug: bool       = False,
    output_dir: str   = "debug_outputs"
) -> Tuple[Tuple[int, int], int, np.ndarray]:

    ...

    return (cy,cx),r+5,pupil
```

W porównaniach poszczególnych wykonywanych operacji brało udział pierwsze zdjęcie lewego oka osoby nr 40. Proces detekcji rozpoczęto od zastosowania operacji *CLAHE* na wcześniej przekształconym zdjęciu do skali szarości. Następnie zastosowano operację filtru Gaussowskiego(Gaussian Blur) korzystając z gotowej implementacji w bibliotece `OpenCV`. Jądro tego przekształcenia ustawiono na (5,5). Tak prezentuje się porównanie tych wstępnych przekształceń:



(a) Skala szarości (b) CLAHE (c) Filtr Gaussowski

Rysunek 2: Porównanie obrazu w skali szarości, po zastosowaniu CLAHE i filtru Gaussowskiego.

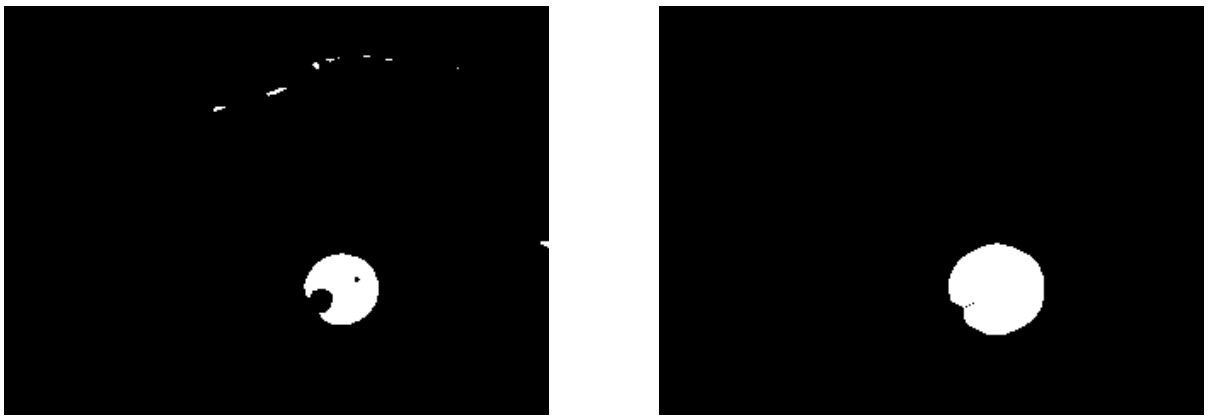
Tak przekształcone zdjęcie poddano następnie binaryzacji. Próg dla źrenicy wyznaczono przy pomocy następującego wzoru:

$$P = \sum_{i=0}^{h-1} \sum_{j=0}^{w-1} \frac{A(i,j)}{h\,w}, \tag{1}$$

Wartość progu P jest obliczana jako średnia jasność całego obrazu - h, w oznaczają wysokość i szerokość obrazu, a $A(i,j)$ – jasność piksela. Ostateczny próg P_P obliczany jest przez dodatkowe podzielenie przez stałą X_P wyznaczaną eksperymentalnie:

$$P_P = \frac{P}{X_P}, \tag{2}$$

Dla źrenicy stała wyniosła $X_P = 4.5$. Po binaryzacji zastosowano wcześniej wspomniane przekształcenie maski największej składowej. Następnie trzykortnie przeprowadzono dylatację. Jądro dylatacji nadal wynosiło 5×5. Poniżej przedstawiono wyniki tych operacji:



(a) Binaryzacja (b) MNS i dylatacja

Rysunek 3: Porównanie obrazu po binaryzacji, przekształceniu MNS(Maska Największej Składowej) i dylatacji

Następne operacje morfologiczne, które zastosowano to kolejno otwarcie i zamknięcie. Wykonano je po jednym razie. Jądro otwarcia było równe 9×9, a zamknięcia 5×5. Poniżej przedstawiono wyniki tych przekształceń:

(a) Otwarcie (b) Zamknięcie

Rysunek 4: Porównanie obrazu po otwarciu i zamknięciu

Ostatnie operacje przed projekcją, które zastosowano to *flood fill* i ponownie maska największej składowej. Operacja *flood fill* została krótko opisana poniżej:

Algorytm *flood fill* służy do wypełniania spójnego obszaru pikseli o tej samej wartości, zaczynając od zadanego punktu startowego $s = (x_0, y_0)$.

Niech $I : \Omega \to \{0, 1\}$ będzie binarnym obrazem na domenie $\Omega \subset \mathbb{Z}^2$, a $c = I(s)$ wartością koloru, który chcemy wypełnić. Definiujemy ciąg rosnących zbiorów:

$$\begin{aligned} R^{(0)} &= \{\, s\}, \\ R^{(n+1)} &= R^{(n)} \ \cup \ \big\{\, p \in \Omega \setminus R^{(n)} \mid \exists\, q \in R^{(n)} :\ p \in N(q) \ \wedge \ I(p) = c\big\}, \end{aligned}$$

gdzie $N(q)$ to sąsiedztwo (np. 4-lub 8-sąsiedztwo) punktu q. Ciąg ten stabilizuje się w kroku N, gdy

$$R = \bigcup_{n=0}^{N} R^{(n)},$$

i wtedy R jest maksymalnym spójnym obszarem o wartości c zawierającym punkt startowy.

W kontekście usuwania małych dziur:
1. Najpierw wykonujemy flood fill z punktu $(0, 0)$ na obrazie `closed`, co wyznacza obszar tła:

$$F = R \quad \text{dla } s = (0, 0),\ c = I(0, 0) = 0.$$

2. Odwracamy wynik, aby uzyskać maskę dziur:

$$H = \Omega \setminus F.$$

3. Łączymy oryginalny obraz z maską dziur:

$$\texttt{filled} = \texttt{closed} \ \cup \ H.$$

Tak prezentuje się maska po wykonaniu operacji *flood fill* oraz po masce największej składowej:

(a) Flood fill (b) Końcowa maska(po MNS)

Rysunek 5: Porównanie obrazu po *flood fill* i MNS

Ostatnie kroki w celu skutecznego wykrycia środka źrenicy i jej promienia to zastosowanie projekcji:

1. **Binaryzacja maski:** Niezerowe piksele maski źrenicy traktujemy jako jedynki, pozostałe jako zera. Otrzymujemy macierz

$$M(i,j) = \begin{cases} 1, & \text{jeśli piksel } (i,j) \text{ należy do źrenicy,} \\ 0, & \text{w przeciwnym razie.} \end{cases}$$

2. **Projekcja na osie:**

   - *Projekcja wierszowa:*

$$R(k) \;=\; \sum_{j} M(k,j), \quad k = 0,1,\ldots,h-1,$$

     czyli suma pikseli źrenicy w każdym wierszu.

   - *Projekcja kolumnowa:*

$$C(\ell) \;=\; \sum_{i} M(i,\ell), \quad \ell = 0,1,\ldots,w-1,$$

     czyli suma pikseli źrenicy w każdej kolumnie.

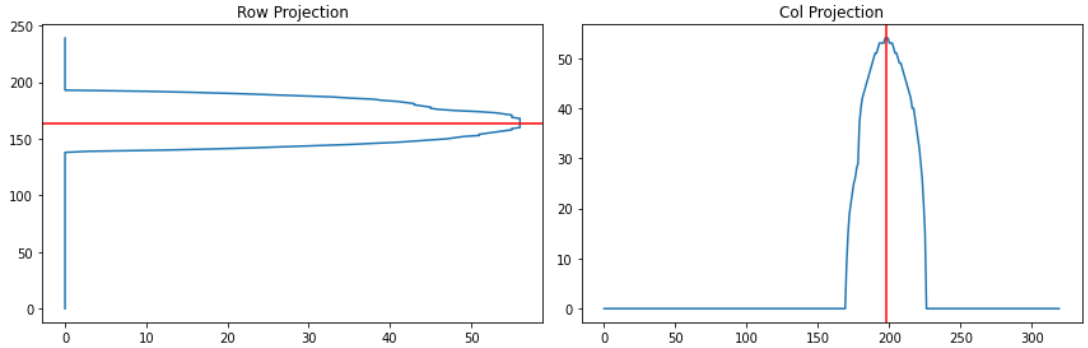
3. **Lokalizacja środka:** Wartości $R(k)$ i $C(\ell)$ osiągają maksimum w pobliżu środka okrągłej maski.

$$k^* = \arg\max_k R(k), \quad \ell^* = \arg\max_\ell C(\ell).$$

   Jeśli maksimum występuje wielokrotnie, za y-współrzędną środka przyjmuje się medianę indeksów:

$$y_c = \text{median}\big\{\, k : R(k) = \max_k R(k) \big\}, \quad x_c = \text{median}\big\{\, \ell : C(\ell) = \max_\ell C(\ell) \big\}.$$

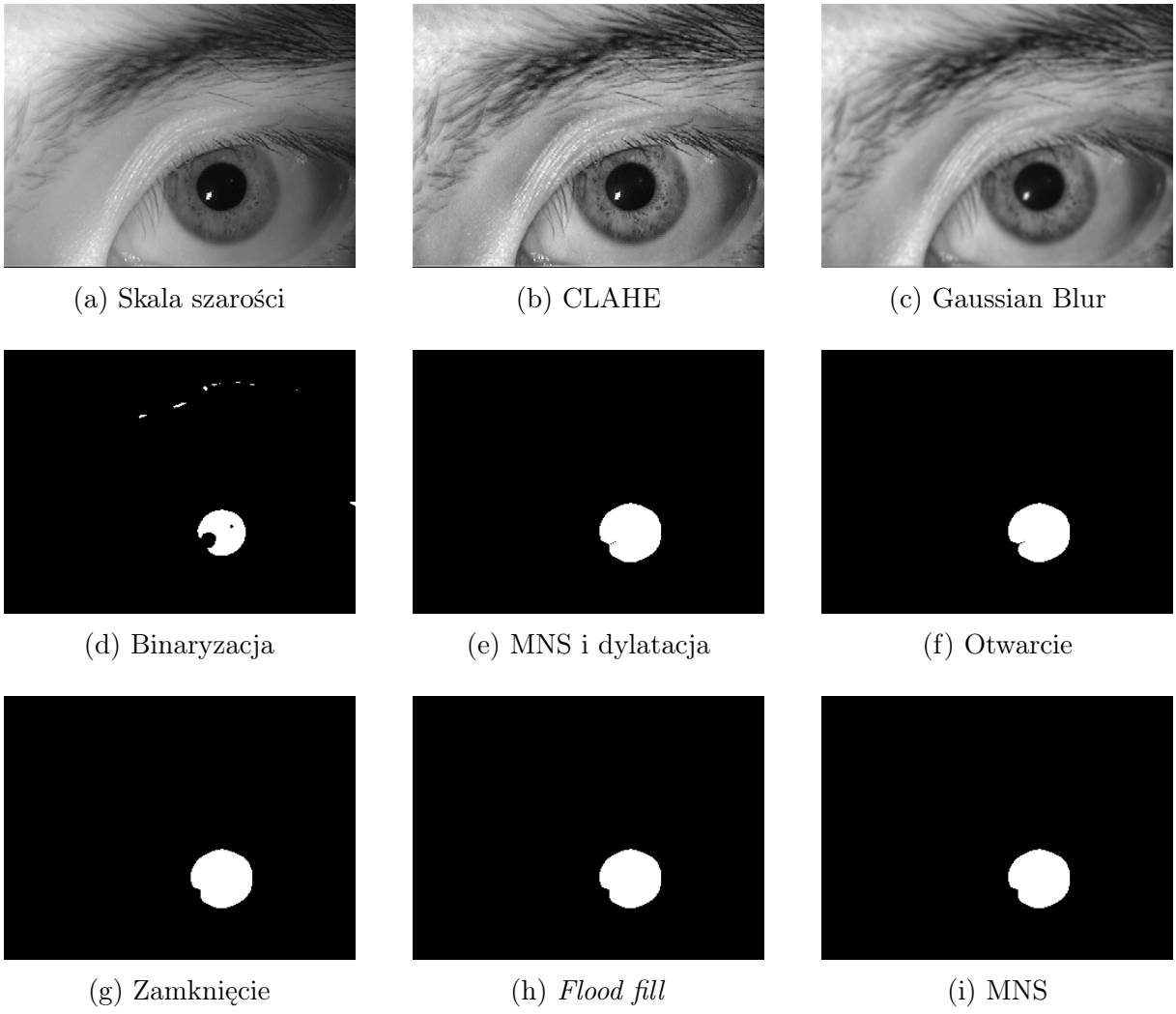
Tak prezentują się wykresy projekcji dla rozważanego przykładu:



Rysunek 6: Projekcje binarnej maski źrenicy: na lewym panelu widać projekcję wierszową (czerwona linia wyznacza y_c), na prawym panelu – projekcję kolumnową (czerwona linia wyznacza x_c).

Na końcu wyznaczany jest promień stosując najprostszą metodologię: Liczymy odległość euklidesową każdego piksela maski od wcześniej wyznaczonego środka. Z wszystkich wyników obliczamy średnią i to jest nasz wynik.

Po wielu eksperymentach i testach powyższe podejście jest jeszcze dodatkowo modyfikowane przez dodanie do końcowej długości promienia 5 pikseli. Wtedy wyniki są najlepsze. Poniżej znajduje się podsumowanie wszystkich operacji na obrazie w celu detekcji źrenicy:



(a) Skala szarości (b) CLAHE (c) Gaussian Blur

(d) Binaryzacja (e) MNS i dylatacja (f) Otwarcie

(g) Zamknięcie (h) *Flood fill* (i) MNS

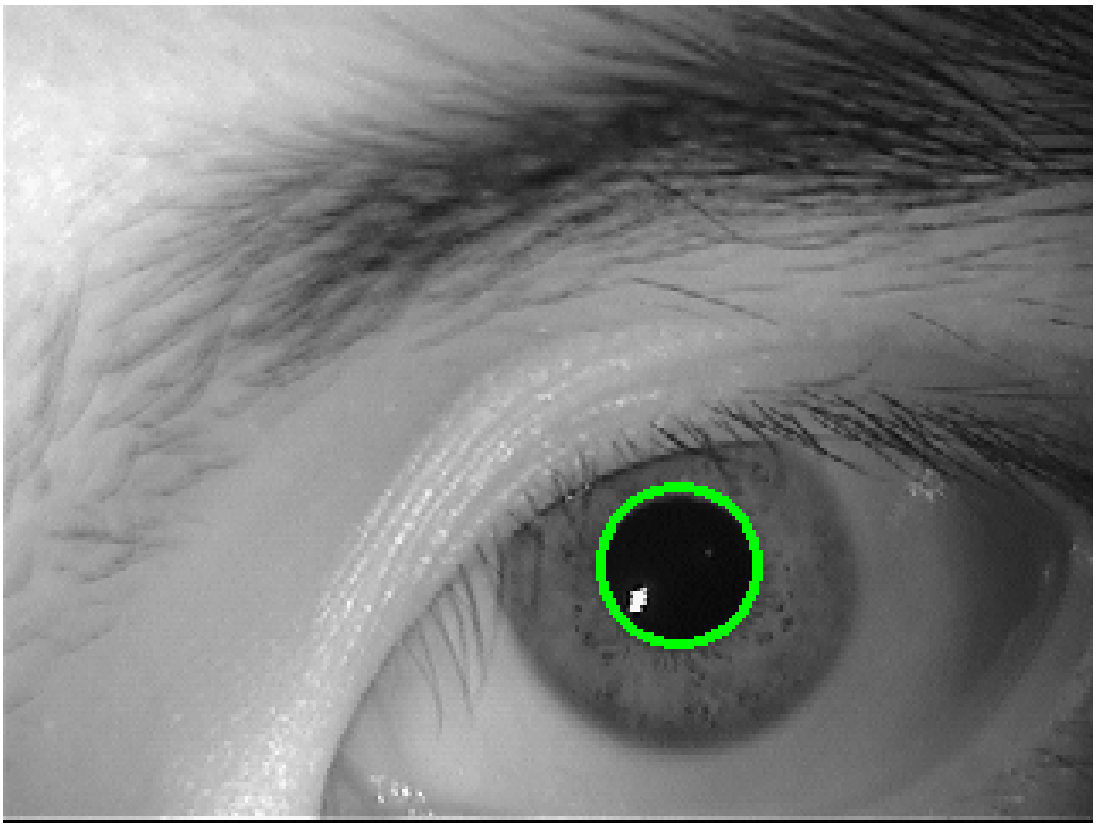
Rysunek 7: Kolejne etapy przetwarzania obrazu `tonghll1`: (a) konwersja do skali szarości, (b) lokalne wyrównanie kontrastu (CLAHE), (c) rozmycie Gaussowskie, (d) binaryzacja, (e) wybór największej składowej i dylatacja, (f) otwarcie, (g) zamknięcie, (h) flood fill, (i) ponowny wybór największej składowej (Final Mask).

Podsumowanie kroków przetwarzania obrazu:

1. **Konwersja do skali szarości.**
2. **CLAHE** – lokalne wyrównanie kontrastu (clip Limit=4.0, tileGridSize=(8,8)).
3. **Gaussian Blur** – rozmycie filtrem Gaussa (kernel 5×5).
4. **Binaryzacja** – próg globalny $P_P = P/X_P$, gdzie P to średnia jasność, $X_P = 4.5$.
5. **Maska Największej Składowej** - wybór największego spójnego obszaru.
6. **Dylatacja** – przywrócenie kształtu źrenicy (kernel 5×5, iteracje=3).

7. **Otwarcie** (kernel 9×9) – usunięcie drobnego szumu.
8. **Zamknięcie** (kernel 5×5) – wypełnienie małych dziur.
9. ***Flood fill* + MNS** – zalewowe wypełnienie tła, odwrócenie maski dziur i wybór największego spójnego obszaru jako źrenica.

Tak z kolei prezentuje się finalny wynik rozumowania przeprowadzonego w celu detekcji źrenicy:



Rysunek 8: Finalny wynik działania algorytmu detekcji źrenicy

3.1.4 Detekcja tęczówki

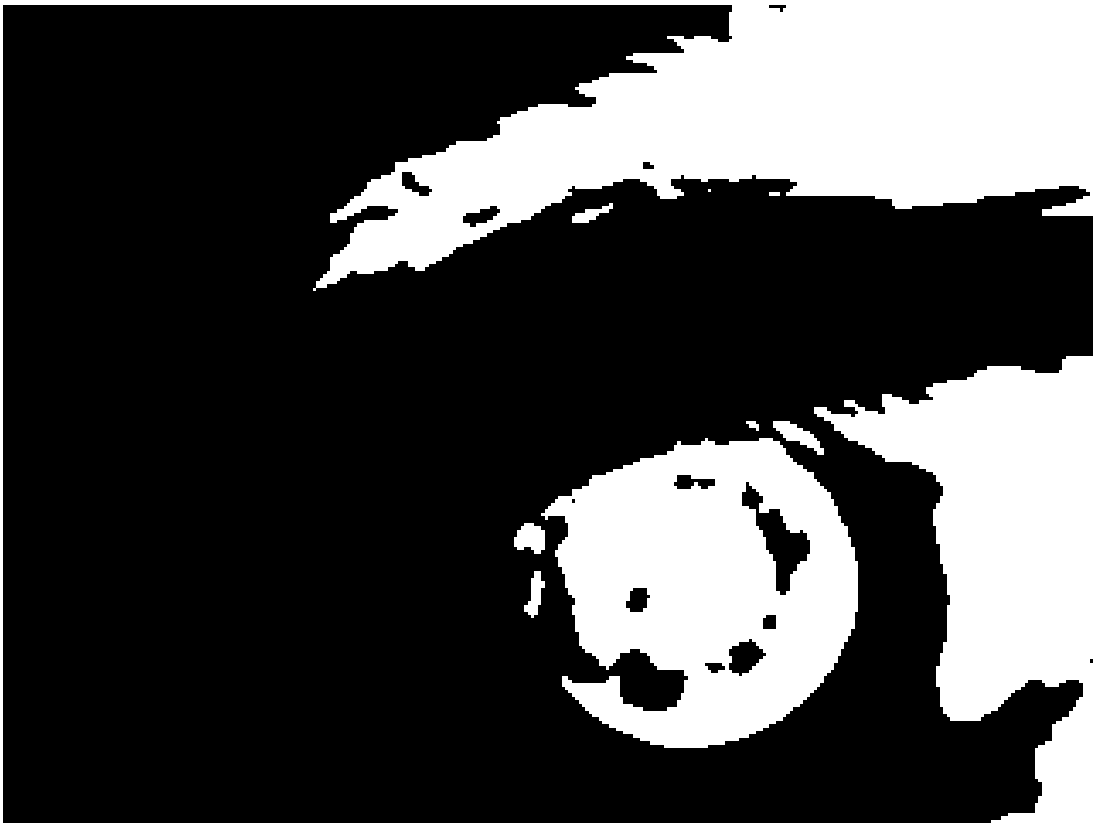
Kolejnym krokiem było wyznaczenie obszaru, w którym znajduje się tęczówka. Okręgi, które wyznaczają źrenicę i tęczówkę są współśrodkowe, więc detekcja tęczówki w dużej mierze zależy od skutecznej detekcji źrenicy. Cały proces detekcji jest opisany przez funkcję `detect_iris(...)`, która przyjmuje i zwraca następujące elementy:

```python
def detect_iris(
    img_path: str,
    gray: np.ndarray,
    center: Tuple[int,int],
    r_pupil: int,
    X_I: float       = 1.5,
    kernel_size: int = 7,
    morph_iter: int  = 2,
    alpha: float     = 0.9,
    debug: bool      = False,
    output_dir: str  = "iris_debug"
) -> Tuple[int, np.ndarray]:

...

return r_comb+2, iris_mask
```

Przetwarzanie rozpoczynamy od zastosowania filtra Gaussowskiego. W celach eksperymentalnych nie zastosowano metody *CLAHE*. Wyznaczanie maski rozpoczęto od zastosowania binaryzacji. Próg P binaryzacji wyznaczano analogicznie co w równaniu (1). Ostateczny próg P_I obliczany jest przez dodatkowe podzielenie przez stałą X_I wyznaczaną eksperymentalnie:

$$P_I = \frac{P}{X_I}, \tag{3}$$

Dla tęczówki stała wyniosła $X_I = 1.5$. Tak prezentuje się obraz po zastosowaniu binaryzacji:



Rysunek 9: Obraz po binaryzacji

Stosowanie operacji morfologicznych rozpoczęto od zastosowania zamknięcia o jądrze 7×7. Zrobiono to dwukrotnie. Następnie podobnie postąpiono z otwarciem. Parametry jądra i liczby iteracji pozostały takie same. Poniżej przedstawiono wyniki tych przekształceń:

(a) Zamknięcie (b) Otwarcie

Rysunek 10: Porównanie obrazu po zamknięciu i otwarciu

Kolejnymi operacjami morfologicznymi, które zastosowano były erozja i dylatacja. Wykonano je jednokrotnie, a ich jądra były rozmiaru 7×7. Wyniki tych operacji przedstawiono na następującym rysunku:

(a) Erozja (b) Dylatacja

Rysunek 11: Porównanie obrazu po erozji i dylatacji

Następnie, podobnie jak w przypadku źrenicy zastosowano maskę największej składowej. W dalszej części zastosowano erozję z jądrem przekształcenia 3×3 i liczbą iteracji równą 1, żeby wyznaczyć granicę maski. Chciano jedynie uzyskać białe piksele, które mają w sąsiedztwie piksele czarne - pozostał jedynie swego rodzaju pierścień graniczny. Dla tak przekształconej maski skorzystano z podejścia histogramu radialnego. Aby wyznaczyć promień tęczówki bazując na krawędziowej masce, zastosowano następujące kroki:

1. Zbieramy współrzędne wszystkich pikseli krawędzi:

$$\{(y_i, x_i)\} \;=\; \{(y, x) \mid \text{boundary}(y, x) > 0\}.$$

2. Obliczamy odległość euklidesową każdego takiego piksela od ustalonego środka źrenicy (c_y, c_x):

$$d_i = \sqrt{(y_i - c_y)^2 + (x_i - c_x)^2} \quad \text{(zaokrąglone do liczby całkowitej)}.$$

3. Definiujemy zakres możliwych promieni od $d_{\min} = 1.1 \cdot r_{\text{pupil}}$ do $d_{\max} = \lfloor \min(h, w)/2 \rfloor$ i budujemy histogram:

$$H(d) = \big|\{\, i : d_i = d\}\big|, \quad d = 0, 1, \ldots, d_{\max}.$$

4. Wyciągamy fragment histogramu $H_{\text{band}} = \big[H(d_{\min}), H(d_{\min}+1), \ldots, H(d_{\max}-1)\big]$ i znajdujemy wartość maksymalną: $\max H_{\text{band}}$.

5. Zbiór promieni odpowiadających maksimum to

$$R = \{\, d : H(d) = \max H_{\text{band}},\ d_{\min} \le d < d_{\max}\}.$$

6. Ostateczny promień tęczówki wybieramy jako medianę elementów zbioru R:

$$r_{\text{hist}} = \text{median}(R).$$

Takie podejście pozwala wybrać promień, dla którego w rozwarstwie krawędzi uformowana jest największa koncentracja pikseli, co odpowiada obwodowi tęczówki. Żeby nie bazować jedynie na tym rozwiązaniu wprowadzono drugą metodologię liczenia promienia. Była to swego rodzaju projekcja bazująca na wcześniej wyznaczonych krawędziach. Aby wyznaczyć promień na podstawie liniowych projekcji krawędzi:

1. **Projekcja pozioma:** Wyznaczamy wektor wartości krawędzi w wierszu przechodzącym przez środek (c_y, c_x):

$$\text{row} = \big[\text{boundary}(c_y, x)\big]_{x=0}^{w-1}.$$

   Następnie zbieramy wszystkie pozycje x występowania krawędzi: $X = \{\, x : \text{row}[x] > 0\}$. Jeżeli $X \neq \emptyset$, obliczamy odległości od środka:

$$\text{left} = c_x - \min X, \quad \text{right} = \max X - c_x.$$

   W przeciwnym wypadku przyjmujemy $\text{left} = \text{right} = r_{\text{hist}}$.

2. **Projekcja pionowa:** Analogicznie dla kolumny:

$$\text{col} = \big[\text{boundary}(y, c_x)\big]_{y=0}^{h-1}, \quad Y = \{\, y : \text{col}[y] > 0\},$$

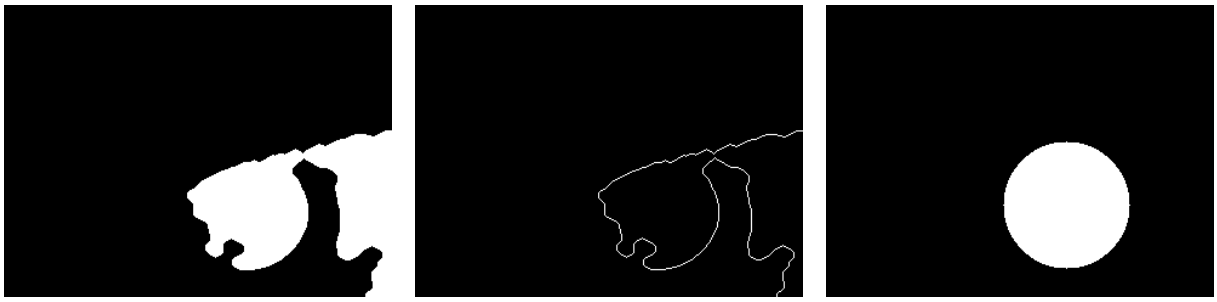
$$\text{up} = c_y - \min Y, \quad \text{down} = \max Y - c_y,$$

   lub $\text{up} = \text{down} = r_{\text{hist}}$ gdy $Y = \emptyset$.

3. **Średnia odległość:** Ostateczny promień projekcyjny przyjmujemy jako średnią z czterech odległości:

$$r_{\text{proj}} = \frac{\text{left} + \text{right} + \text{up} + \text{down}}{4}.$$

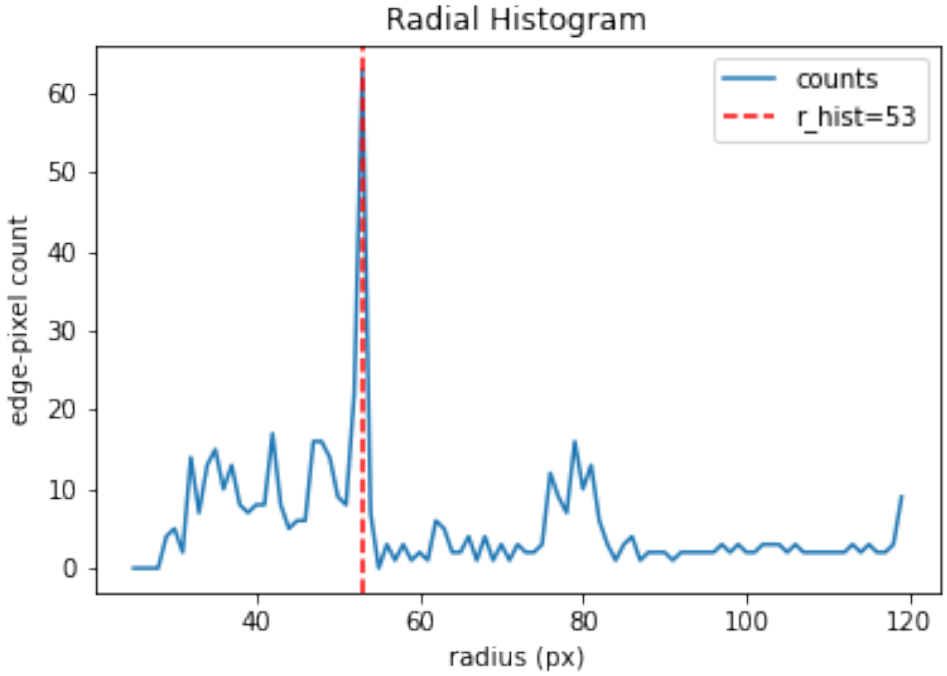
Ostatecznie promień wyznaczany jest jako średnia ważona obu podejść. Wynik z metogologii histogramowej jest brany z współczynnikiem $\frac{9}{10}$, a drugie podejście ma wagę na poziomie $\frac{1}{10}$. Poniżej przedstawiono wyniki wcześniej wspomnianych operacji:



(a) MNS (b) Boundary (c) Finalna maska

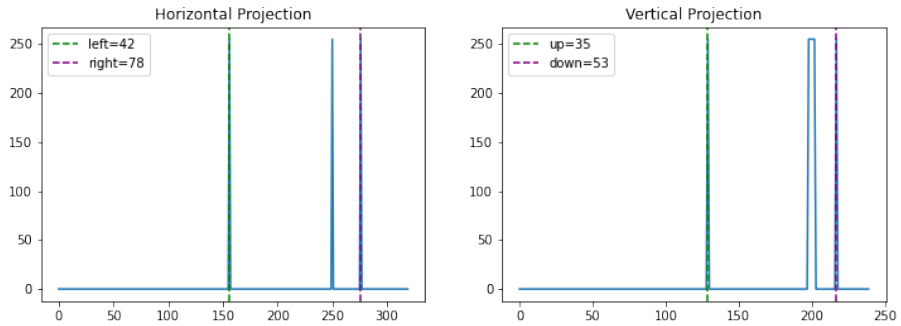
Rysunek 12: Porównanie obrazu po zastosowaniu MNS, wyznaczeniu granicy i odpowiedniego promienia.

Tak z kolei prezentuje się histogram radialny dla rozważanego przypadku:



Rysunek 13: Histogram radialny

Poniżej znajdują się wyniki drugiego podejścia:



Rysunek 14: Podejście oparte na projekcji

Promień z pierwszego podejścia ma 53 piksele, a z drugiej metodologii 52 piksele. Po eksperymentach, stwierdzono, że końcowy promień będzie jeszcze zwiększony o 2 piksele. Podsumowując tak prezentują się wszystkie przekształcenia, które wykonaliśmy na naszym zdjęciu:

(a) Gaussian Blur

(b) Binaryzacja

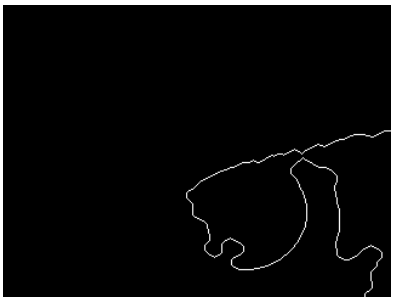
(c) Zamknięcie

(d) Otwarcie

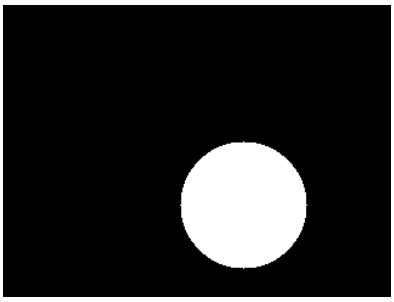
(e) Erozja

(f) Dylatacja

(g) Największy komponent



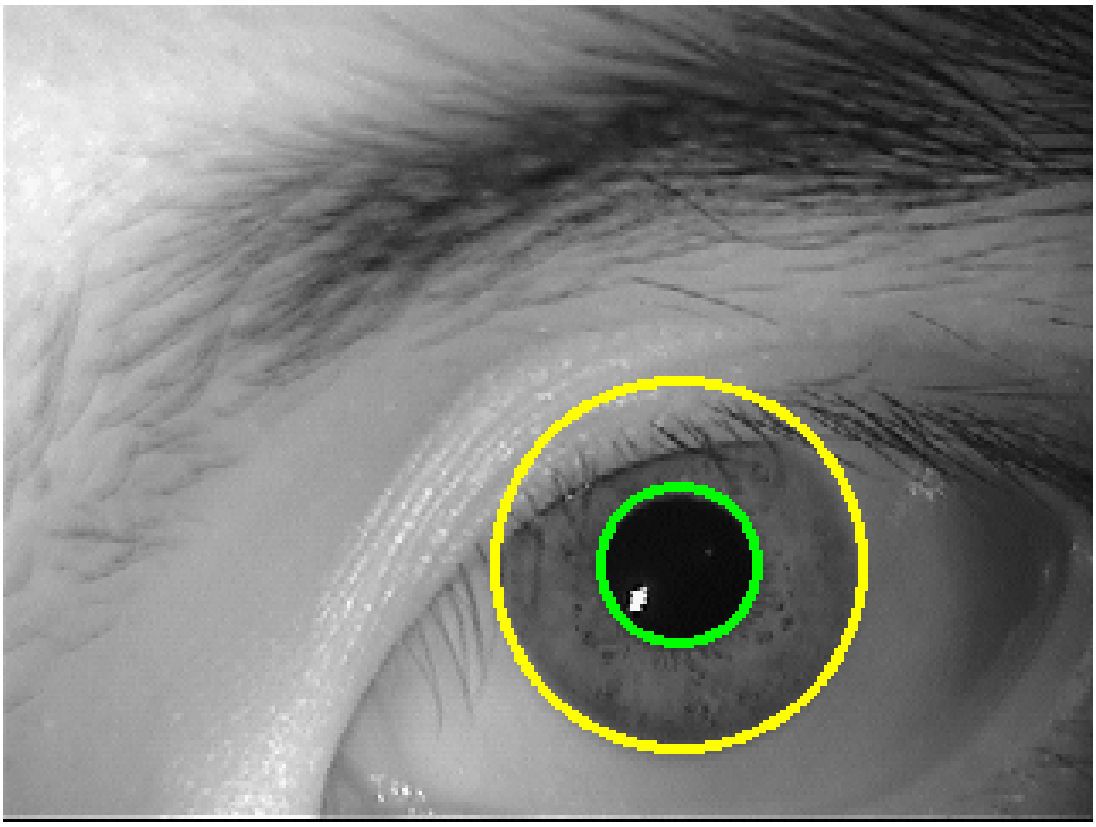
(h) Boundary



(i) Finalna maska tęczówki

Rysunek 15: Kolejne etapy detekcji tęczówki na rozważanym obrazie: (a) rozmycie Gaussa, (b) binaryzacja odwrócona, (c) zamknięcie, (d) otwarcie, (e) erozja, (f) dylatacja, (g) wybór największego komponentu, (h) różnica (gradient morf.), (i) ostateczna, wypełniona maska tęczówki.

Ostatecznie detekcja tęczówki wraz z wcześniej wyznaczoną źrenicą dla rozważanego przypadku prezentują się następująco:



Rysunek 16: Końcowa detekcja tęczówki i źrenicy

3.2 Rozwinięcie tęczówki do prostokąta

Po detekcji tęczówki przystąpiono do rozwinięcia jej w pasmo prostokątne. Logika algorytmiczna była zawarta w funkcji `unwrap_iris(...)`, która przyjmowała i zwracała następujące argumenty:

```
def unwrap_iris(
    img_gray: np.ndarray,
    center:    Tuple[int,int],
    r_pupil:   int,
    r_iris:    int,
    radial_res:  int = 64,
    angular_res: int = 360
) -> np.ndarray:


...

    return unwrapped
```

Rozwinięcie okrągłej strefy tęczówki do postaci prostokątnej realizowane jest według następującej metodologii:

- Definiujemy dwie siatki parametrów:

$$\theta_j = -\frac{\pi}{2} + \frac{2\pi}{N_\theta}\, j, \quad j = 0, 1, \ldots, N_\theta - 1,$$

$$r_i = r_{\text{pupil}} + \frac{i}{N_r - 1}\,(r_{\text{iris}} - r_{\text{pupil}}), \quad i = 0, 1, \ldots, N_r - 1,$$

gdzie $N_r = \texttt{radial\_res}$, $N_\theta = \texttt{angular\_res}$.

- Przekształcamy do współrzędnych kartezjańskich:

$$x_{i,j} = x_c + r_i \cos\theta_j, \qquad y_{i,j} = y_c + r_i \sin\theta_j,$$

gdzie (x_c, y_c) to środek źrenicy.

- Tworzymy obraz rozwinięty U o wymiarach $N_r \times N_\theta$ przez interpolację oryginalnego obrazu $I(x, y)$:

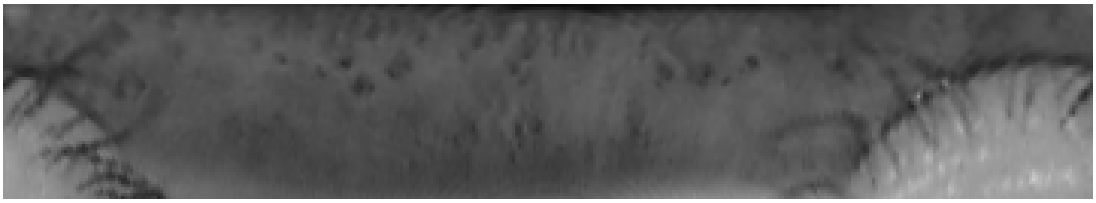
$$U(i, j) = I\big(x_{i,j},\, y_{i,j}\big),$$

zwykle z użyciem interpolacji liniowej.

- Piksele, których przemapowane współrzędne wychodzą poza oryginalny obraz, maskujemy na 0.

W efekcie wiersze i odpowiadają radialnym odległościom od źrenicy do granicy tęczówki, a kolumny j kątowi θ_j.

Poniżej przedstawiono wyniki działania algorytmu dla wcześniej rozważanego przykładu:



Rysunek 17: Rozwinięcie tęczówki do prostokąta

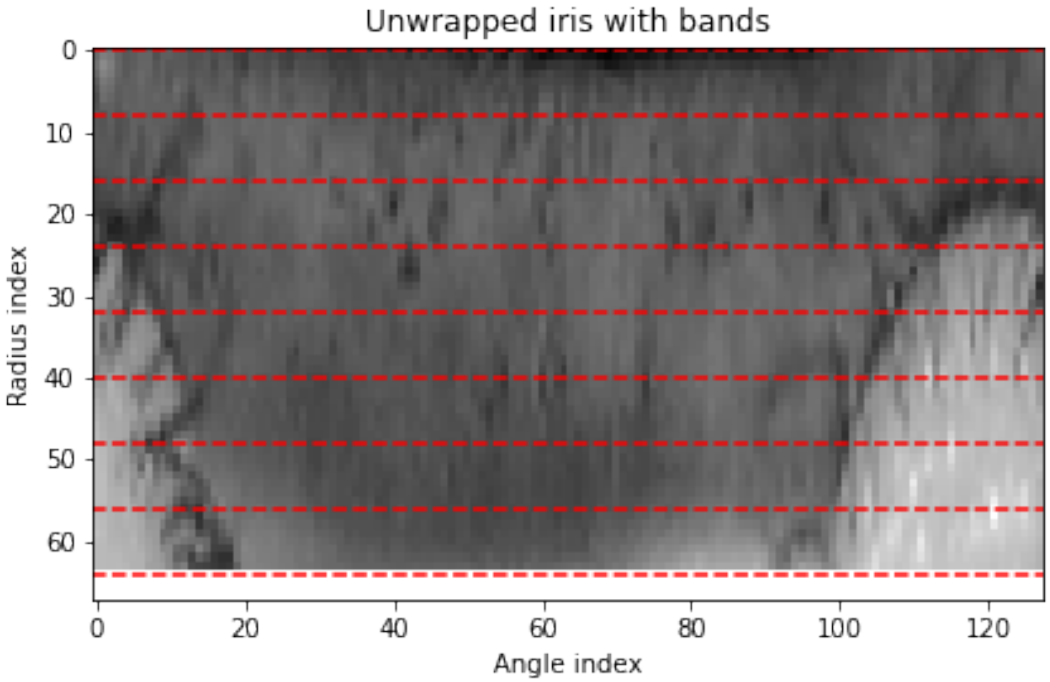
3.3 Kodowanie tęczówki

3.3.1 Algorytm kodowania

Kolejnym etapem w całym procesie było zakodowanie informacji zawartej w tęczówce. W tym celu skorzystano z algorytmu Daugmana. Cała logika biznesowa i wszystkie niezbędne transformacje zostały uwzględnione w funkcji `encode_iris(...)`, która przyjmuje i zwraca następujące argumenty:

```
def encode_iris(
    unwrapped:         np.ndarray,
    num_radial_bands: int   = 8,
    angular_points:   int   = 128,
    crop_top_frac:    float = 0.0,
    crop_bottom_frac: float = 0.0,
    gabor_freq:       float = 0.1,
    gabor_sigma:      float | None = None,
    radial_sigma:     float | None = None,
    debug:            bool  = False
) -> Tuple[np.ndarray, np.ndarray]:


...


    return code_raw
```

Przed rozpoczęciem kodowania rozwinięta tęczówka jest dzielona na 8 radialnych pasów, które będą odpowiadać poszczególnym fragmentom tęczówki. W pierwotnym podejściu próbowano uciąć fragmenty podstawowego obrazu, w celu usunięcia potencjalnych fragmentów powiek, rzęs lub źrenicy. Ostatecznie najlepsze wyniki metody były osiągane bez stosowania tej operacji. Na poniższym zdjęciu przedstawiono podział na wspomniane pasy radialne dla rozważanego wcześniej oka:



Rysunek 18: Podział tęczówki na pasy radialne

Następnym krokiem było ustalenie parametrów falki Gabora oraz okna Gaussa, aby osiągnąć najlepsze wyniki:

$$f_G = \pi, \qquad \sigma_G = \tfrac{1}{2}\,\pi\, f_G, \qquad \sigma_R = \tfrac{1}{2}\,\text{band\_height}, \qquad L = \text{angular\_points} = 128.$$

Dalszy algorytm przebiega następująco:

1. **Uśrednianie radialne:** Dla pasa b bierzemy fragment $\mathbf{B}_b \in \mathbb{R}^{\text{band\_height}\times N_\theta}$ i przekształcamy go na wektor $\mathbf{v} \in \mathbb{R}^{N_\theta}$ przez korelację z oknem Gaussa $\mathbf{g} \in \mathbb{R}^{\text{band\_height}}$:

$$\mathbf{v} = \frac{\mathbf{B}_b^T\,\mathbf{g}}{\sum_i g_i} \quad \longrightarrow \quad \mathbf{v} = \frac{\mathbf{v} - \mu(\mathbf{v})}{\sigma(\mathbf{v})}.$$

2. **Próbkowanie kątowe:** Z wektora $\mathbf{v}$ wybieramy L próbek $\mathbf{v}_{\text{samp}} = \big[v_{j_k}\big]_{k=0}^{L-1}$, gdzie $j_k = \lfloor k\, N_\theta / L \rfloor$.

3. **Zastosowanie falki Gabora 1-D:** Dla każdego $k = 0, \ldots, L-1$ definiujemy:

$$w_k(x) = \exp\!\Big(-\tfrac{(x-k)^2}{\sigma_G^2}\Big), \quad x = 0, \ldots, L-1,$$

i obliczamy odpowiedź zespoloną

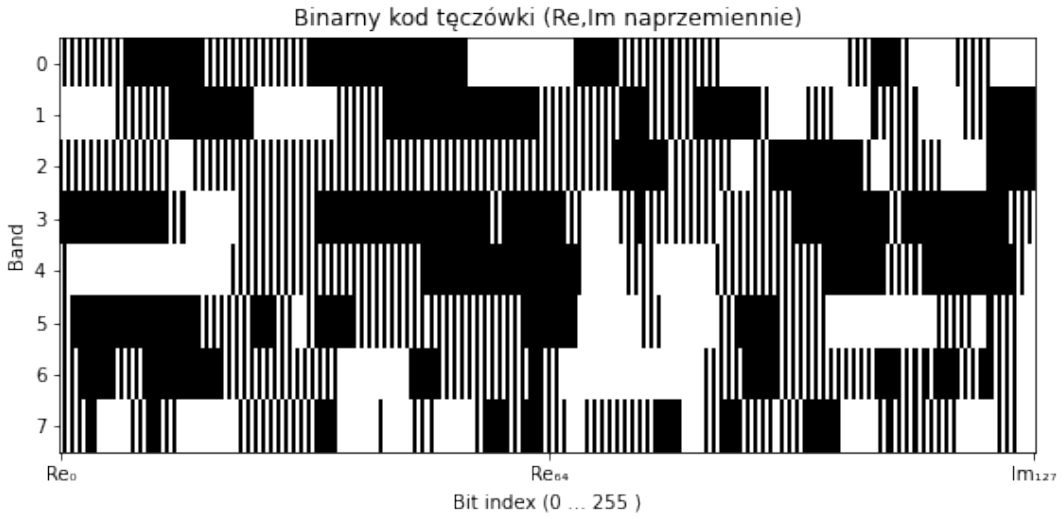
$$R_k = \sum_{x=0}^{L-1} \mathbf{v}_{\text{samp}}(x)\, w_k(x)\, \exp\bigl(-\mathrm{i}\, 2\pi\, f_G\, x\bigr).$$

4. **Binarne kodowanie:** Dla każdej odpowiedzi R_k kodujemy dwa bity:

$$\text{code}[b,k,0] = \begin{cases} 1 & \Re(R_k) \geq 0 \\ 0 & \Re(R_k) < 0 \end{cases}, \quad \text{code}[b,k,1] = \begin{cases} 1 & \Im(R_k) \geq 0 \\ 0 & \Im(R_k) < 0 \end{cases}.$$

Efektem jest macierz binarna o wymiarach $(B, L, 2)$.

Ostatecznie tak prezentuje się wizualizacja kodu tęczówki dla rozważanego przykładu:



Rysunek 19: Kod tęczówki

Poszczególne bity części rzeczywistej i zespolonej są zwizualizowane koło siebie. To znaczy, że informacja z jednego poprzedniego fragmentu jest przedstawiana w formie dwóch paseczków w kolorach czarnym lub białym w zależności od tego jaką wartość binarną przyjęły.

3.3.2 Odległość Hamminga

Jako miarę podobieństwa poszczególnych kodów przyjęto odległość Hamminga, którą zdefiniowano w funkcji `hamming_distance(...)`. Na wejściu dajemy dwa kody, które chcielibyśmy porównać o strukturze takiej, jak otrzymaliśmy z funkcji kodującej tęczówkę.

Niech $c_i^{\text{Re}}(b,\theta)$ oraz $c_i^{\text{Im}}(b,\theta)$ będą bitami rzeczywistymi i urojonymi kodu i (gdzie $b = 1,\ldots,B$ numeruje pasy radialne, a $\theta = 1,\ldots,L$ punkty kątowe). Definiujemy:

$$\Delta^{\text{Re}}(b,\theta) = \bigl|c_1^{\text{Re}}(b,\theta) - c_2^{\text{Re}}(b,\theta)\bigr|, \quad \Delta^{\text{Im}}(b,\theta) = \bigl|c_1^{\text{Im}}(b,\theta) - c_2^{\text{Im}}(b,\theta)\bigr|.$$

Odległość Hamminga to stosunek liczby różnych bitów w obu kanałach do łącznej liczby bitów:

$$d_H = \frac{\sum_{b=1}^{B}\sum_{\theta=1}^{L}\Bigl[\Delta^{\text{Re}}(b,\theta) + \Delta^{\text{Im}}(b,\theta)\Bigr]}{2\,B\,L}.$$

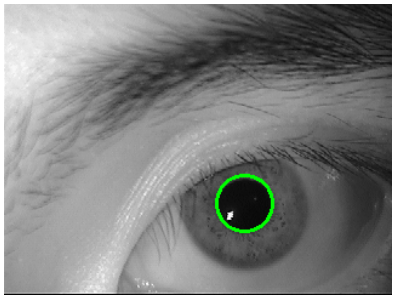
4 Eksperymenty i wyniki

4.1 Zbiór danych

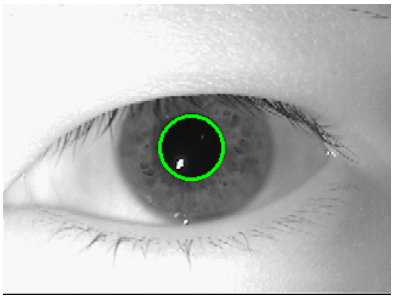
Do oceny wydajności i skuteczności metodologii zastosowano pakiet zdjęć **MMU-Iris-Dataset** zawierający 450 obrazów 90 oczu 45 osób w różnych warunkach oświetlenia.

4.2 Detekcja źrenicy

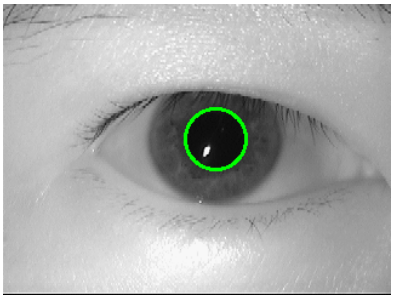
Poniżej przedstawiono kilka wyników, dla których starano się przeprowadzić detekcję źrenicy:



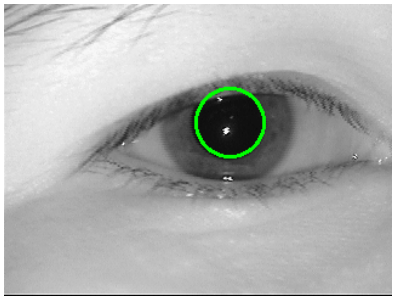
(a) Osoba 40 oko lewe zdjęcie nr 1



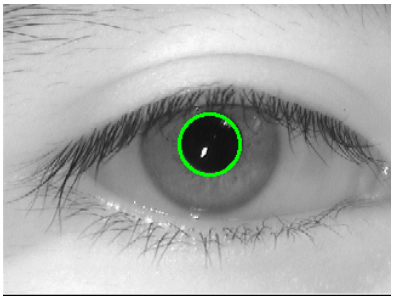
(b) Osoba 42 oko lewe zdjęcie nr 2



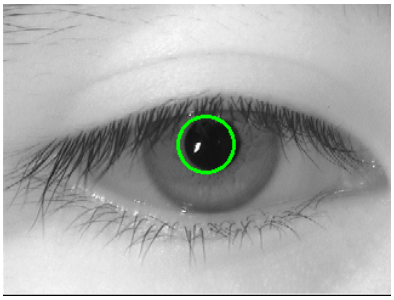
(c) Osoba 42 oko prawe zdjęcie nr 3



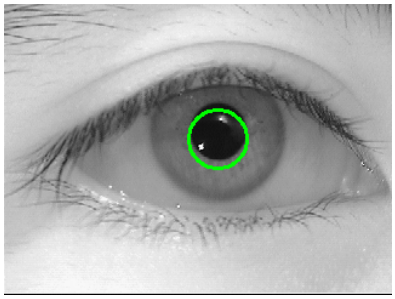
(d) Osoba 43 oko lewe zdjęcie nr 1



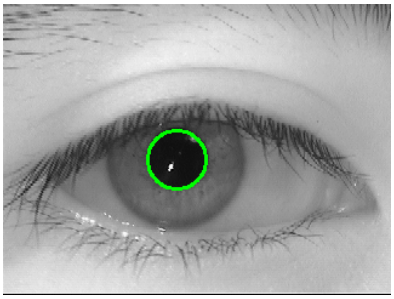
(e) Osoba 34 oko prawe zdjęcie nr 1



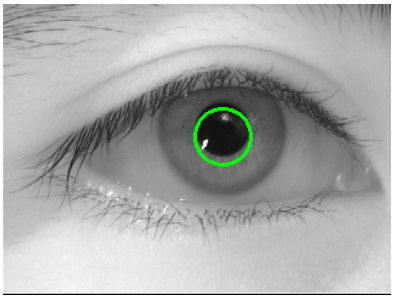
(f) Osoba 34 oko prawe zdjęcie nr 2



(g) Osoba 34 oko prawe zdjęcie nr 3



(h) Osoba 34 oko prawe zdjęcie nr 4

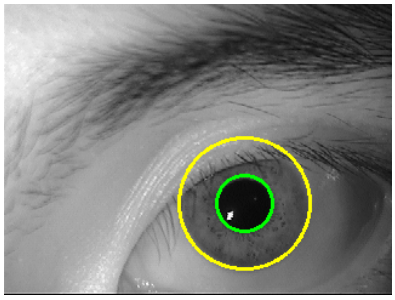


(i) Osoba 34 oko prawe zdjęcie nr 5

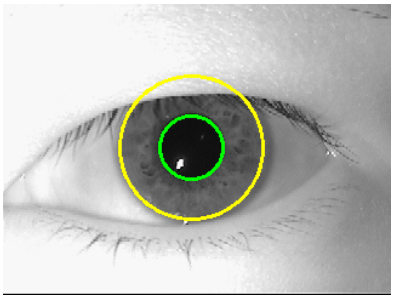
Rysunek 20: Przykładowe wyniki detekcji źrenicy

4.3 Detekcja tęczówki

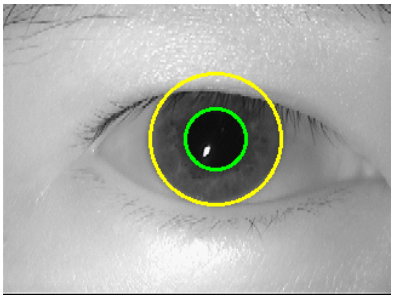
Analogiczne wyniki, dodając detekcję tęczówki, przedstawiono dla tych samych przykładów:



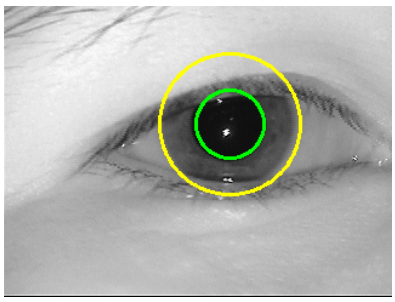
(a) Osoba 40 oko lewe zdjęcie nr 1



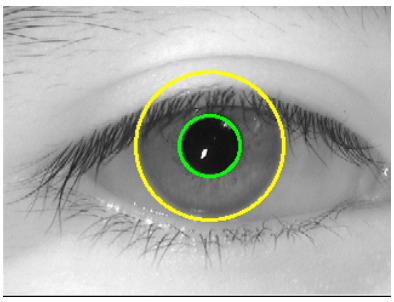
(b) Osoba 42 oko lewe zdjęcie nr 2



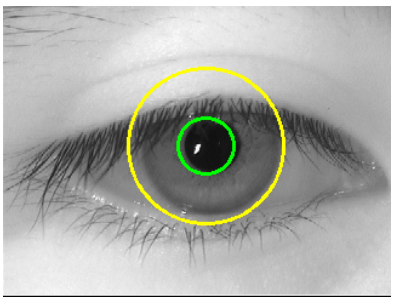
(c) Osoba 42 oko prawe zdjęcie nr 3



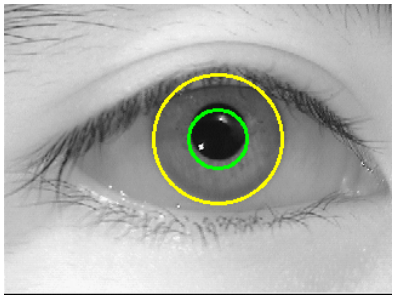
(d) Osoba 43 oko lewe zdjęcie nr 1



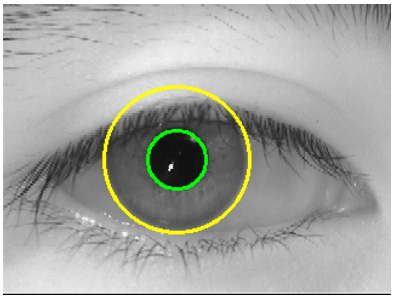
(e) Osoba 34 oko prawe zdjęcie nr 1



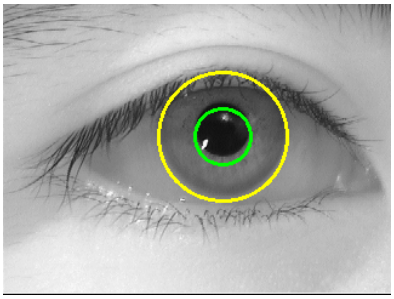
(f) Osoba 34 oko prawe zdjęcie nr 2



(g) Osoba 34 oko prawe zdjęcie nr 3



(h) Osoba 34 oko prawe zdjęcie nr 4



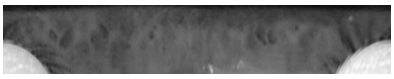
(i) Osoba 34 oko prawe zdjęcie nr 5

Rysunek 21: Przykładowe wyniki detekcji tęczówki

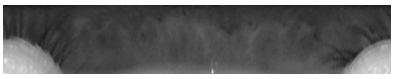
4.4 Rozwinięcie tęczówki do prostokąta

Dla tych samych przykładów przeprowadzono rozwinięcie tęczówki do prostokąta:

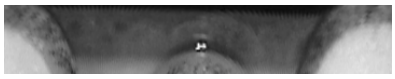
(a) Osoba 40 oko lewe zdjęcie nr 1



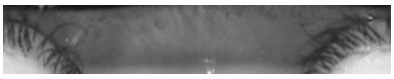
(b) Osoba 42 oko lewe zdjęcie nr 2



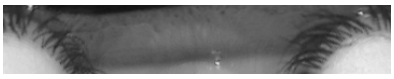
(c) Osoba 42 oko prawe zdjęcie nr 3



(d) Osoba 43 oko lewe zdjęcie nr 1

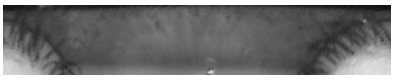


(e) Osoba 34 oko prawe zdjęcie nr 1



(f) Osoba 34 oko prawe zdjęcie nr 2

(g) Osoba 34 oko prawe zdjęcie nr 3



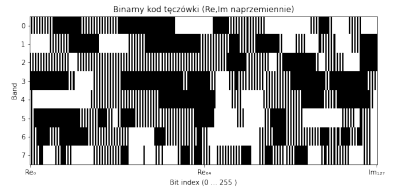
(h) Osoba 34 oko prawe zdjęcie nr 4

(i) Osoba 34 oko prawe zdjęcie nr 5

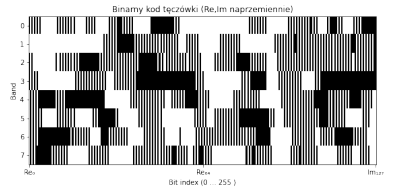
Rysunek 22: Przykładowe wyniki rozwinięcia tęczówki do prostokąta

4.5 Kod tęczówki

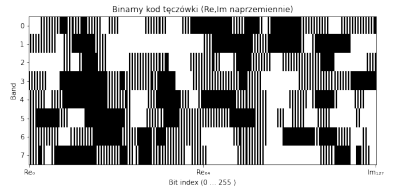
Ostatnie porównanie przeprowadzono dla kodów rozważanych przykładów tęczówek:



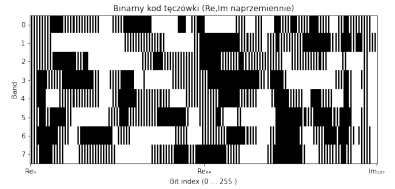
(a) Osoba 40 oko lewe zdjęcie nr 1



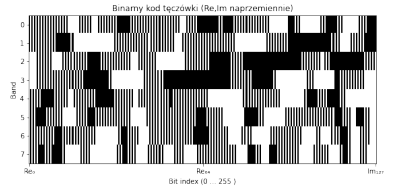
(b) Osoba 42 oko lewe zdjęcie nr 2



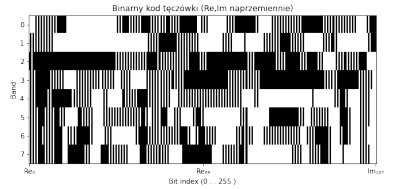
(c) Osoba 42 oko prawe zdjęcie nr 3



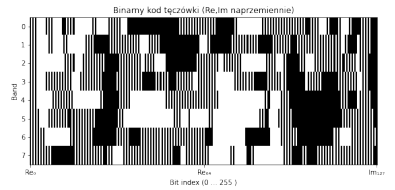
(d) Osoba 43 oko lewe zdjęcie nr 1



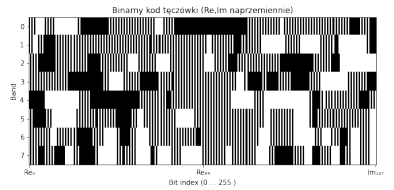
(e) Osoba 34 oko prawe zdjęcie nr 1



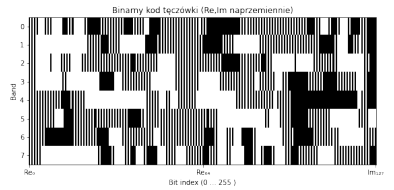
(f) Osoba 34 oko prawe zdjęcie nr 2



(g) Osoba 34 oko prawe zdjęcie nr 3



(h) Osoba 34 oko prawe zdjęcie nr 4

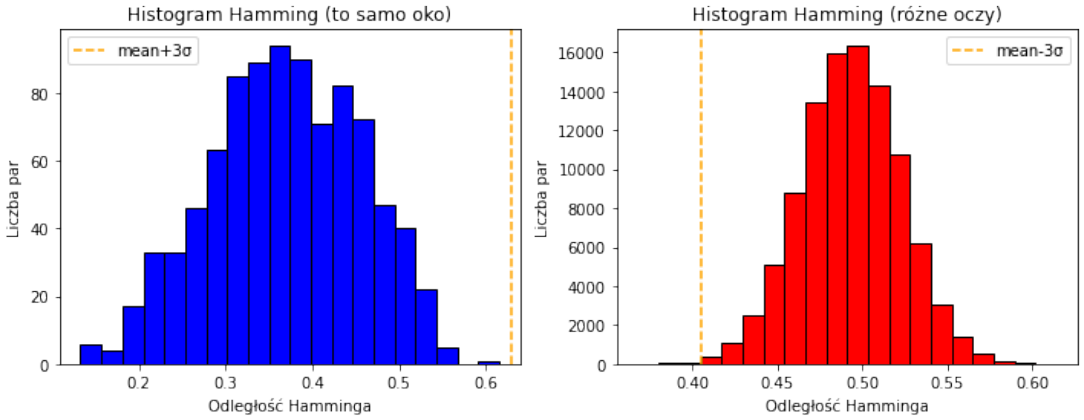


(i) Osoba 34 oko prawe zdjęcie nr 5

Rysunek 23: Przykładowe wizualizacje kodów tęczówek

4.6 Skuteczność działania metodologii

W celu zbadania skuteczności metodologii przeprowadzono eksperymenty. Dla każdej pary różnych kodów tęczówek z rozważanego zbioru policzono odległość Hamminga. Wszelkie parametry poszczególnych operacji pozostały takie same jak we wcześniej opisywanej sytuacji. Dla rozważanych przykładów wygenerowano histogramy, które podzielono wedle klucza: oko tej samej osoby, oko innej osoby. Poniżej przedstawiono te wykresy:



Rysunek 24: Histogramy odległości hamminga

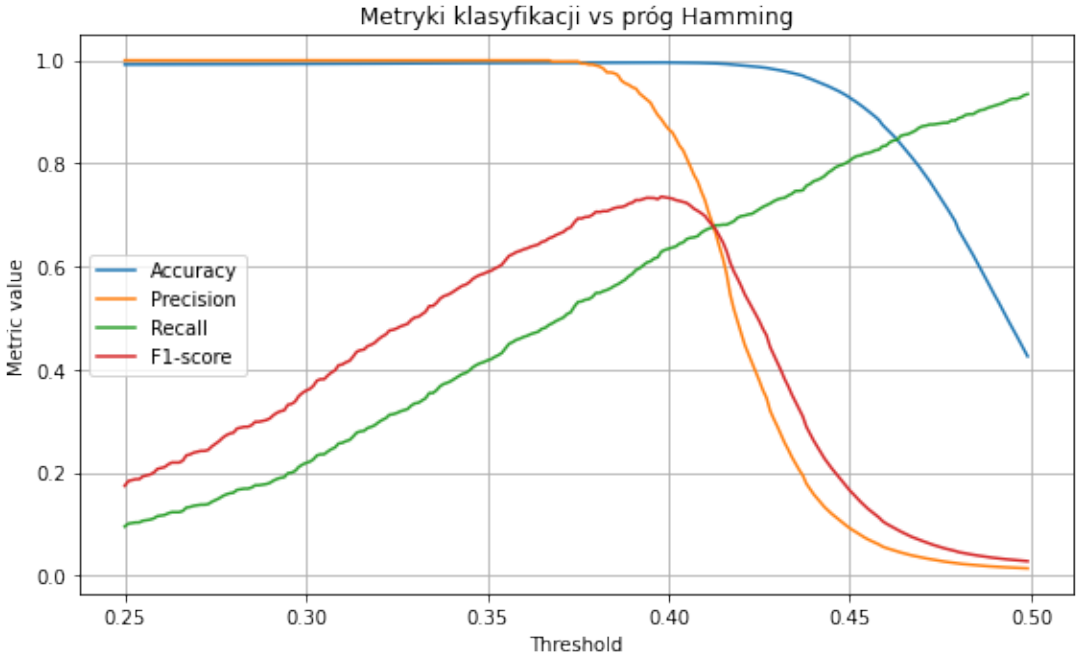
Dla tych wielkości policzono również podstawowe statystyki:

	Średnia	Odchylenie standardowe	Minimum	Maksimum	Liczność
`same_eye`	0,3687	0,0867	0,1323	0,6152	900
`diff_eye`	0,4929	0,0292	0,3677	0,6143	100125

Tabela 1: Statystyki odległości Hamminga dla par pochodzących z tego samego oka (`same_eye`) oraz z różnych oczu (`diff_eye`).

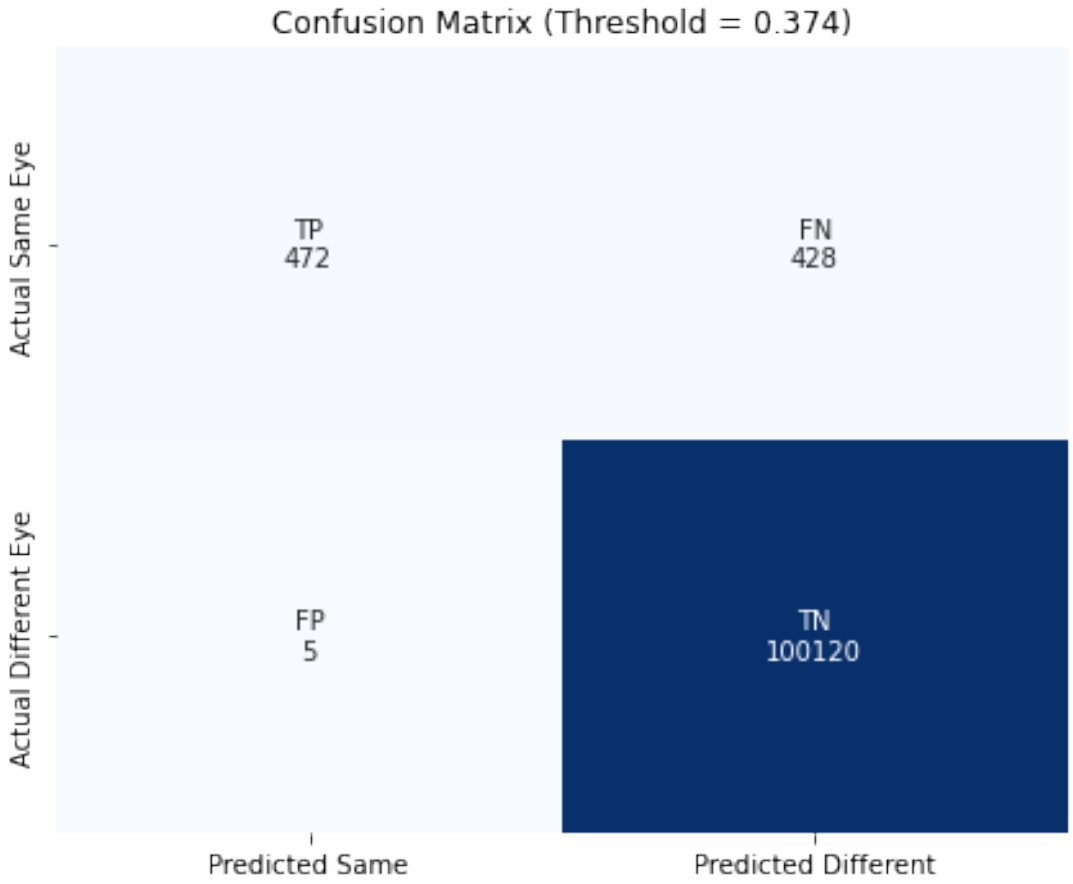
$$\text{Próg}_{\text{same\_eye}} = \mu + 3\sigma = 0{,}6287, \qquad \text{Próg}_{\text{diff\_eye}} = \mu - 3\sigma = 0{,}4052$$

Progi zostały wyznaczone orientacyjnie za pomocą reguły 3 sigm. Następnie narysowano wykres poszczególnych metryk klasyfikacji w zależności od zadanego thresholdu. Wartości thresholdu rozważano w przedziale $[0.25, 0.5]$.



Rysunek 25: Wykres wartości metryk klasyfikacji w zależności od thresholdu

Chcieliśmy zmaksymalizować F1-Score, przy uwzględnieniu bardzo wysokiej wartości metryki precision. Ostatecznie, po wielu eksperymentach, dla rozważanej metodologii, udało się uzyskać F1-score na poziomie trochę większym niż 0.70 przy jednoczesnym precision większym niż 0.99. Niestety poprzez niepotrzebne kombinowanie z kodem i parametrami, F1-score spadł do wartości w okolicach 0.685, przy jednoczesnym precision na poziomie mniej więcej 0.99. Poniżej przedstawiono macierz pomyłek dla thresholdu 0.374:



Rysunek 26: Macierz pomyłek dla thresholdu 0.374

Niżej znajduje się tabela z odpowiednimi metrykami dla rozważanej metodologii:

Metryka	**Wartość**
Accuracy	0.995714
Precision	0.989518
Recall	0.524444
F1-score	0.685548

Tabela 2: Metryki klasyfikacyjne dla rozważanej metodologii

5 Wnioski

Udało się osiągnąć dość satysfakcjonujące wyniki, jeśli chodzi o identyfikacje poszczególnych osób. Oczywiście przy większej ilości czasu, można byłoby bezproblemowo dostroić hiperparametry. Oczywiście zdjęcia mogłyby być robione w znacznie lepszych warunkach, dzięki czemu uzyskalibyśmy jescze lepsze wyniki. Mimo wszystko podstawowe zadanie odrzucania i detekcji niepożądanych osób jest realizowane w dość zadowalający sposób przy jednoczesnym w miarę satysfakcjonującym progu wpuszczania osób pożądanych. W kontekście systemów bezpieczeństwa, przyjęte podejście można uznać za udane, szczególnie jeśli priorytetem jest unikanie fałszywych dopasowań kosztem odrzucenia niektórych prawidłowych.