

Biometria - Projekt 2

Bartłomiej Wójcik, Tomasz Żywicki

22 kwietnia 2025

1 Opis aplikacji

Celem projektu była ekstrakcja cech z ludzkiej tęczówki, mająca na celu możliwość rozpoznawania człowieka za pomocą jego unikalnego kodu tęczówki. W zaimplementowanym przez nas rozwiązaniu, najpierw znajdywane są granice zarówno żrenicy jak i tęczówki, następnie dzięki tym granicom tęczówka jest "rozwijana" do prostokąta na podstawie czego tworzony jest unikalny kod tęczówki.

1. Streamlit - interfejs graficzny
2. PIL i Open CV - operacje na obrazach
3. Numpy - operacje na pikselach, wykorzystanie przy filtrze Gabora
4. Matplotlib - do wizualizacji rezultatów

2 Opis metod segmentacji i kodowania tęczówki

W ramach realizacji projektu stworzona została klasa `IrisSegmentation`, która realizuje kolejne etapy klasycznego algorytmu Daugmana dla rozpoznawania tęczówek. Poniżej przedstawione zostały metody tej klasy.

2.1 Konwersja do skali szarości

`to_grayscale()` przekształca obraz kolorowy (w formacie RGB lub BGR) do jednowymiarowego obrazu w skali szarości, wykorzystując współczynniki luminancji zgodne z percepcją ludzkiego wzroku:

$$\text{Gray} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

2.2 Progowanie binarne

`compute_threshold(X_I, X_P)` oblicza progi progowania na podstawie średniej jasności obrazu:

$$T_{\text{pupil}} = \frac{\mu}{X_P}, \quad T_{\text{iris}} = \frac{\mu}{X_I}$$

gdzie μ to średnia intensywność w skali szarości. Domyślne wartości $X_P = 2.5$, $X_I = 1.2$ ustalone empirycznie, natomiast dla każdego obrazu wybór najlepszych wartości parametrów wymaga odpowiedniego zmodyfikowania domyślnych wartości.

Metody `binarize_pupil()` i `binarize_iris()` wykonują progowanie binarne, tworząc maski o wartościach 0 lub 255. Operacje są realizowane jako:

$$\text{binary_mask} = (\text{gray} < T) \cdot 255$$

dzięki czemu powstaje obraz wejściowy do dalszej morfologii.

2.3 Wykrywanie źrenicy

`detect_pupil()` używa klasycznych operacji morfologicznych z jądrem kwadratowym 5×5 (closing, opening), co pomaga w usunięciu szumów oraz połączeniu fragmentów źrenicy:

$$\text{mask} = \text{open}(\text{close}(\text{binary_pupil}))$$

Po oczyszczeniu obrazu binarnego przy pomocy operacji morfologicznych (zamykania i otwierania) identyfikowane są wszystkie spójne obszary (kontury). Spośród nich wybrany jest obszar o największej powierzchni, który uznaje się za źrenicę. Następnie dla tego obszaru wyznaczany jest najmniejszy okrąg, który go otacza — określając w ten sposób przybliżony środek i promień źrenicy.

Jeśli konturów nie uda się znaleźć, wykorzystywana jest alternatywna metoda `_detect_pupil_with_projections()`, która bazuje na analizie projekcji pionowych i poziomych binarnej maski. Maksymalne gęstości pikseli binarnych wskazują na potencjalny środek źrenicy.

2.4 Wykrywanie granicy tęczówki

`detect_iris()` najpierw budowana jest maska pierścienia pomiędzy granicą źrenicy a potencjalną tęczówką, a następnie wykonywane jest wykrywanie krawędzi za pomocą detektora Canny'ego. Detektor Canny'ego działa w dwóch progach (niski i wysoki, domyślnie 100 i 200), co pozwala na identyfikację silnych i słabych krawędzi.

Wykrycie granicy tęczówki następuje poprzez funkcję `cv2.HoughCircles()`. Jeśli metoda Hougha zawiedzie, promień granicy tęczówki estymowany jest heurystycznie jako $r_{\text{iris}} = 2.8 \cdot r_{\text{pupil}}$.

2.5 Rozwinięcie tęczówki

`unwrap_iris()` stosuje transformację z układu biegunowego do kartezjańskiego. Dla zadanego liczbę promieni R (np. 64) oraz liczbę kierunków kątowych Θ (np. 360), obliczane są współrzędne:

$$x(\theta, r) = x_0 + r \cdot \cos(\theta), \quad y(\theta, r) = y_0 + r \cdot \sin(\theta)$$

Dla każdej kombinacji r i θ , wartość piksela pobierana jest przez bilinearną interpolację z oryginalnego obrazu.

2.6 Kodowanie tęczówki

`generate_iris_code()` dzieli obraz rozwiniętej tęczówki na pasma radialne (domyślnie 8). Dla każdego paska wykonywana jest analiza odpowiedzi filtra Gabora:

$$G(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right) \cdot \exp(2\pi ifx)$$

Filtr Gabora w formie zespolonej rozdzielany jest na część rzeczywistą i urojoną. Konwolucja realizowana jest przez:

$$s_r = I * \text{Re}(G), \quad s_i = I * \text{Im}(G)$$

Wyniki konwolucji są progowane względem średniej wartości w danym segmencie, co skutkuje binarnym kodem (dwa bity na każdy segment – jeden od części rzeczywistej, jeden od urojonej).

2.7 Odległość Hamminga

`hamming_distance()` oblicza proporcję bitów różniących się pomiędzy dwoma kodami:

$$HD = \frac{1}{N} \sum_{i=1}^N [a_i \oplus b_i]$$

Gdzie \oplus oznacza operację XOR, a N to liczba bitów.

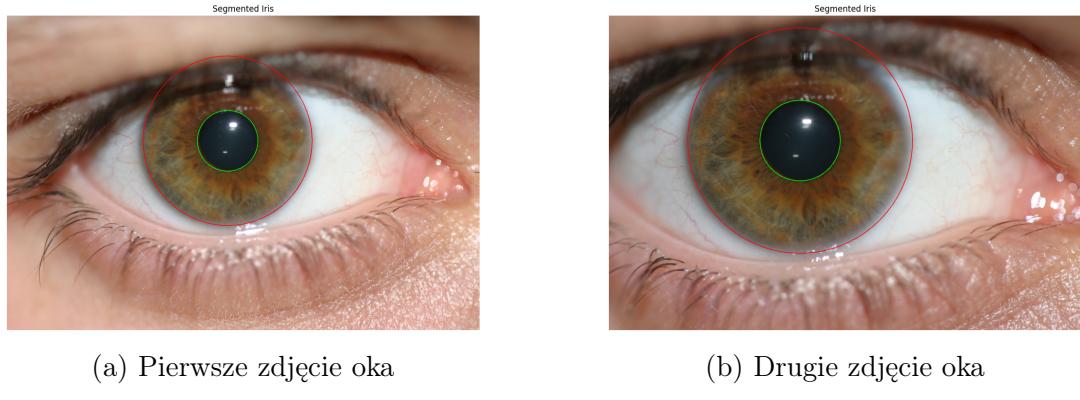
2.8 Wizualizacja

`visualize_segmentation()` rysuje wykryte okręgi tęczówki i żrenicy na oryginalnym obrazie za pomocą funkcji `cv2.circle()`, z podpisami współrzędnych środka i promienia.

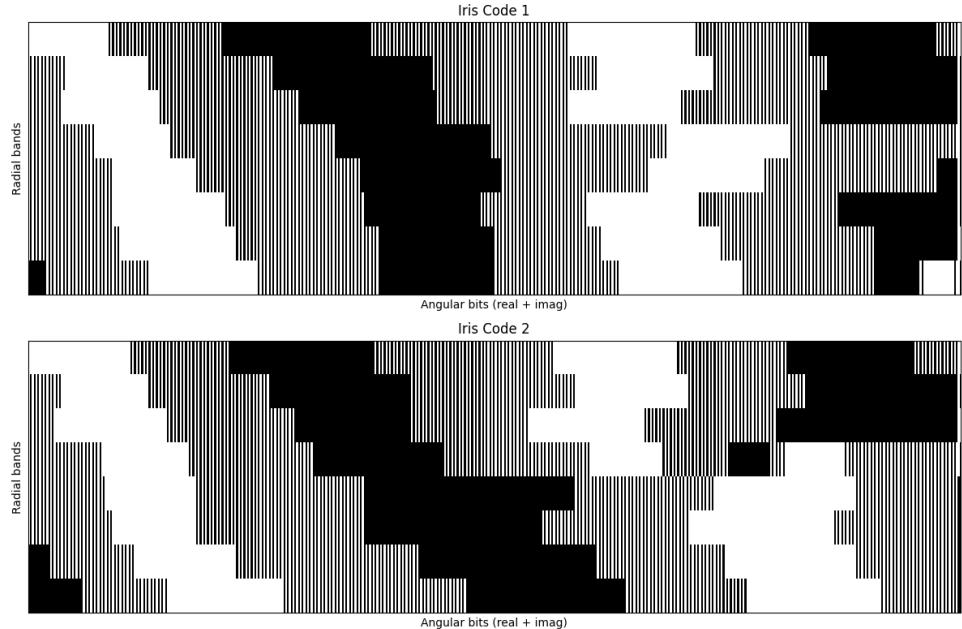
`visualize_iris_code()` prezentuje kod binarny tęczówki jako obraz w odcieniach szarości, gdzie 0 oznacza czarne pole, a 1 – białe.

3 Rezultaty

Dla różnych zdjęć tego samego oka osiągnęliśmy dystans Hamminga na poziomie 0.17, co jest poniżej progu 0.25, który przyjeliśmy za granice podobieństwa lub braku podobieństwa tęczówek. Natomiast jeśli chodzi o drugi zaprezentowany przykład, w którym porównywane były kody różnych oczu, dystans Hamminga wyniósł 0.3, czyli tęczówki nie zostały sklasyfikowane jako podobne.



Rysunek 1: Porównanie dwóch różnych zdjęć tego samego oka



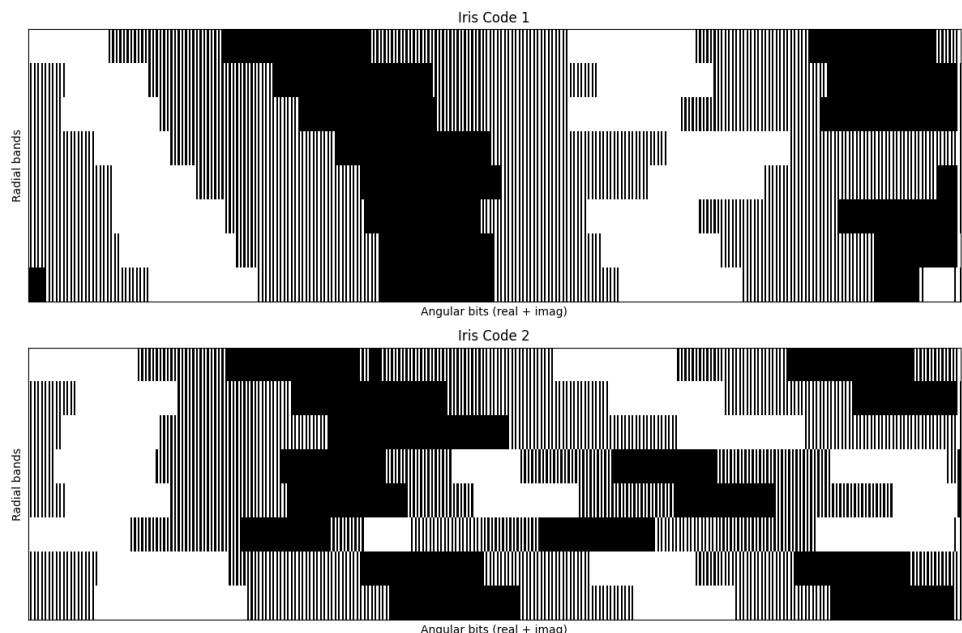
Rysunek 2: Porównanie kodu tęczówek dwóch różnych zdjęć tego samego oka



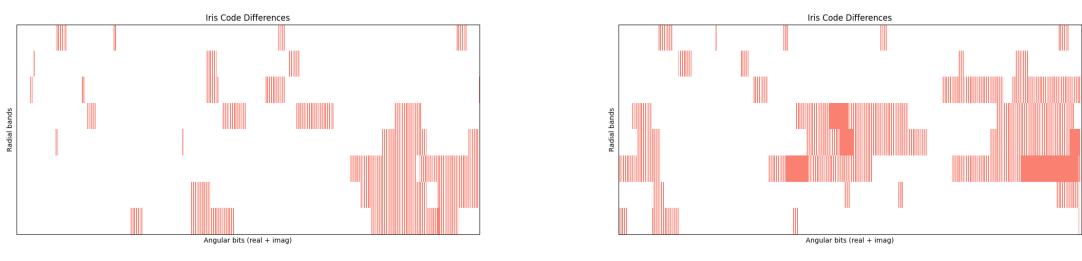
(a) Zdjęcie pierwszego oka

(b) Zdjęcie drugiego oka

Rysunek 3: Porównanie zdjęć dwóch różnych oczu



Rysunek 4: Porównanie kodu tęczówek dwóch różnych oczu



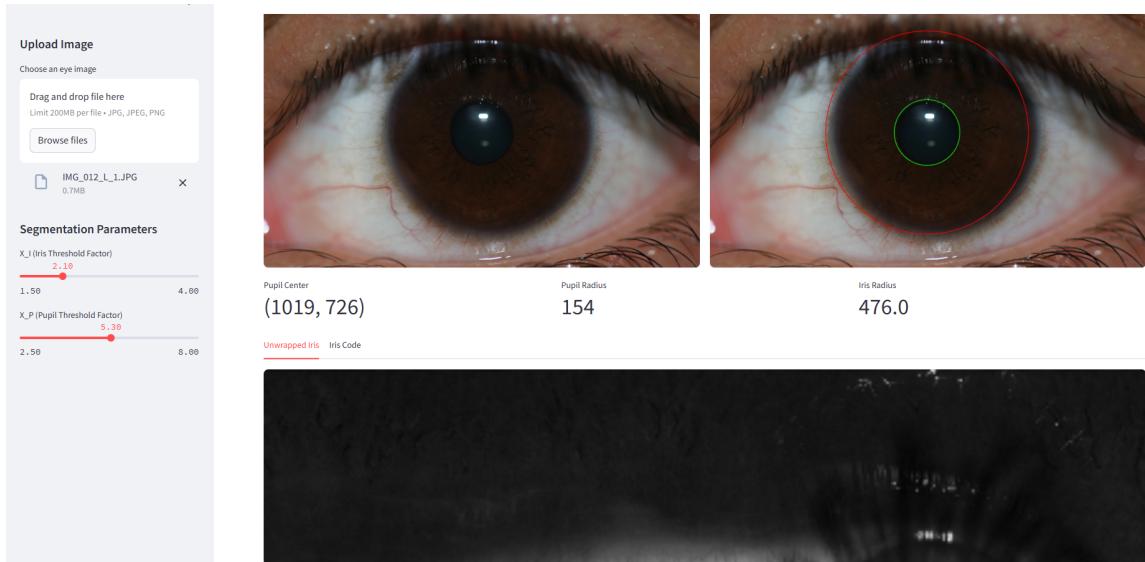
(a) Różne zdjęcia tego samego oka

(b) Zdjęcia różnych oczu

Rysunek 5: Różnice w kodach (kolor czerwony oznacza różnice w danym segmencie)

4 Aplikacja

Interfejs graficzny został stworzony przy użyciu framework'a Streamlit. W tej prostej aplikacji użytkownik, może wczytać zdjęcie oka, następnie przy odpowiednim dobraniu parametrów wyznaczyć odpowiednie granice źrenicy i tęczówki co pozwala na wygenerowanie rzeczywistego rozwinięcia tęczówki do prostokątu czy stworzenie kodu tęczówki, który może zostać wykorzystany do rozpoznawania tęczówki.



Rysunek 6: Przykładowy widok aplikacji