Aplikacja do rozpoznawania tęczówki oka

Biometria

Natalia Choszczyk | Mateusz Deptuch

Politechnika Warszawska

Wydział Matematyki i Nauk Informacyjnych

10 kwietnia 2025

Spis treści

1. Wprowadzenie

Głównym celem projektu było zaznajomienie się z zagadnieniem weryfikacji biometrycznej tęczówki oka ludzkiego. W ramach projektu zrealizowano kolejno przetwarzanie obrazu oka, wyodrębnienie charakterystycznych cech tęczówki oraz ich wykorzystanie do identyfikacji osoby.

Pierwszym etapem było przygotowanie obrazu poprzez konwersję do skali szarości oraz binaryzację z odpowiednimi progami do detekcji krawędzi źrenicy oraz tęczówki. Następnie wykryto granice źrenicy i tęczówki, korzystając z analizy projekcji oraz operacji morfologicznych, które pomogły oczyścić obraz z zakłóceń.

Po wyznaczeniu środka i promieni źrenicy oraz tęczówki, obraz tęczówki został przekształcony do formy prostokąta, co przygotowało dane do dalszej analizy. W kolejnych krokach, na rozwiniętym obrazie tęczówki, przeprowadzono ekstrakcję cech biometrycznych. W tym celu zastosowano algorytm Daugmana, oparty na transformacie falkowej Gabora, który umożliwia zakodowanie tekstury tęczówki w postaci binarnego wzorca (tzw. kodu tęczówki). Ostatecznie porównano kody przy użyciu odległości Hamminga, aby sprawdzić, jak dobrze system rozróżnia różne osoby.

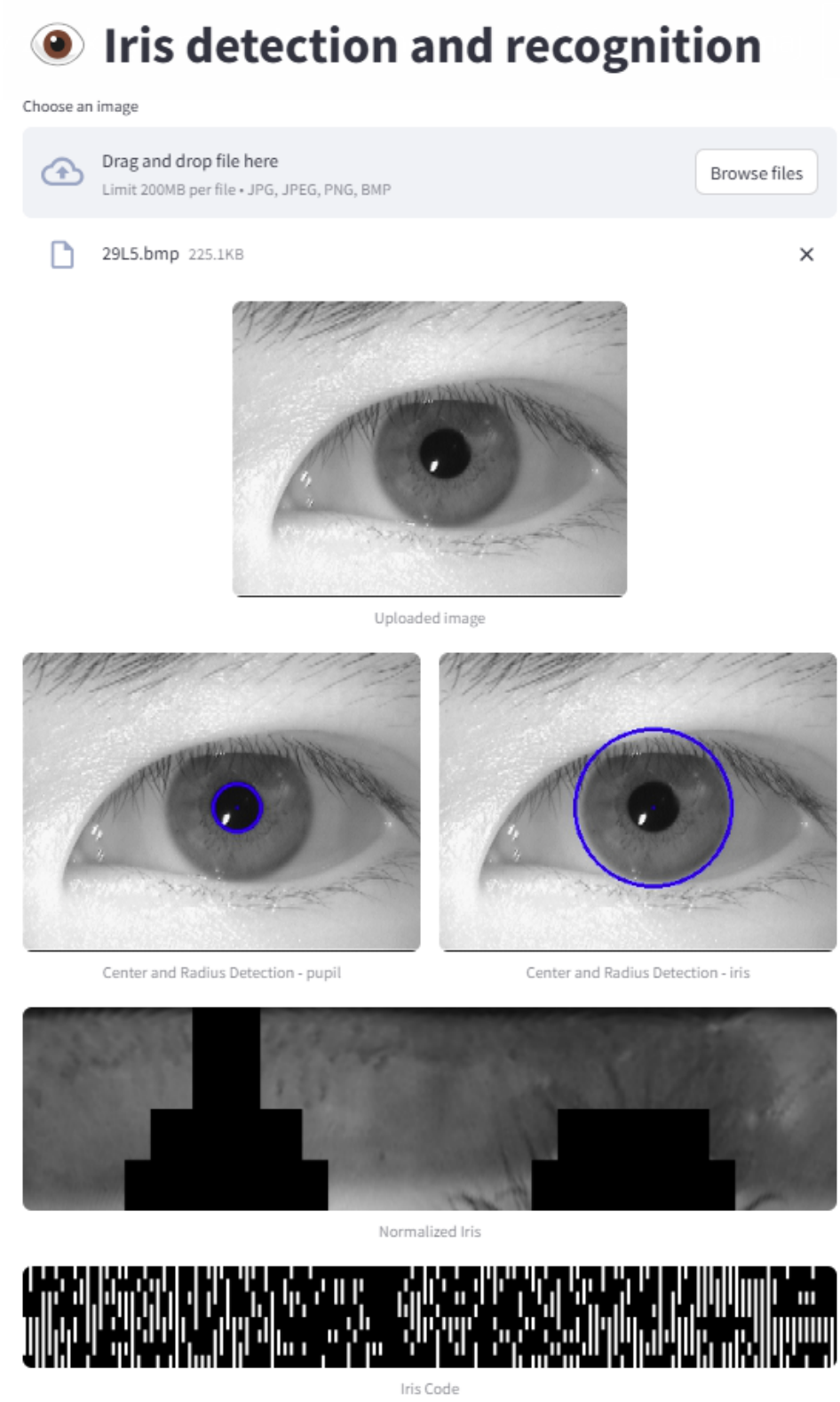
2. Interfejs użytkownika

Aplikacja działa w przeglądarce internetowej, wykorzystując bibliotekę `Streamlit`, która ma formę przejrzystego dashboardu.

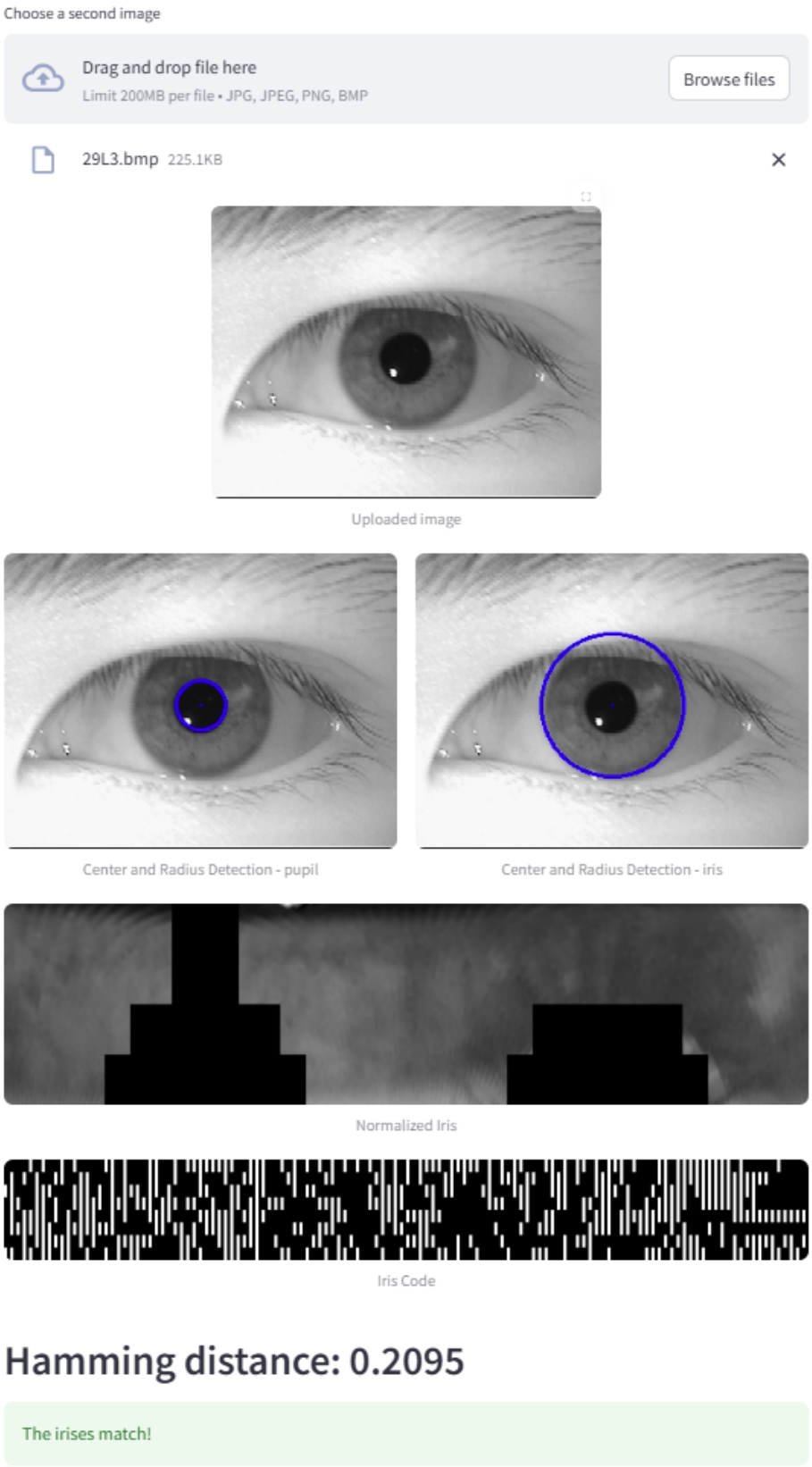
Użytkownik ma możliwość załadowania dwóch obrazów przedstawiających oko. Po wgraniu każdego zdjęcia, interfejs wyświetla kolejno:

- Oryginalne zdjęcie oka.
- Obraz z zaznaczoną wykrytą źrenicą.
- Obraz z zaznaczoną wykrytą tęczówką.
- Znormalizowaną tęczówkę, czyli przekształcony obraz oka w formie prostokątnego pasma z wyciętymi odpowiednimi fragmentami.
- Wygenerowany kod tęczówki w postaci binarnej.

Po przetworzeniu dwóch zdjęć, poniżej wyświetlana jest odległość Hamminga między ich kodami. Jeżeli jest ona wystarczająco niska, pojawia się komunikat: *„The irises match!”*, informujący o zgodności tęczówek. W przeciwnym przypadku dostajemy informację, że kod tęczówki nie jest zgodny.



Rysunek 2.1: Widok interfejsu użytkownika - załadowanie pierwszego zdjęcia



Rysunek 2.2: Widok interfejsu użytkownika - załadowanie drugiego zdjęcia i porównanie tęczówek

3. Struktura aplikacji

3.1. Wykorzystane biblioteki

- `numpy` - operacje numeryczne, m.in. tworzenie masek, obliczanie projekcji i interpolacji.
- `cv2` - przetwarzanie obrazów, na przykład operacje morfologiczne.
- `matplotlib` - wizualizacja wyników.
- `streamlit` - budowa interfejsu aplikacji.
- `scipy` - operacje macierzowe, filtry.

3.2. Zdefiniowane funkcje pomocnicze

W pliku `funcs.py` zostały zdefiniowane pomocnicze funkcje, z których korzystamy później w pliku aplikacji. Te funkcje to:

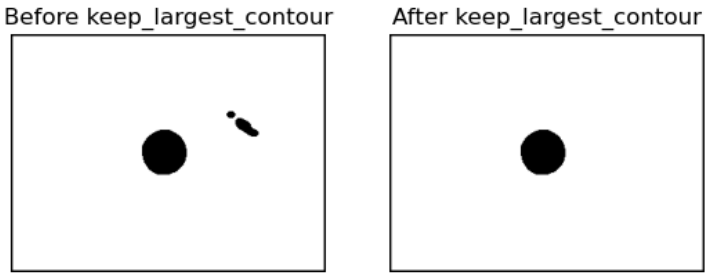
- `binarize`
- `keep_largest_contour`
- `get_pupil`
- `get_iris`
- `unwrap_iris_with_masks`
- `generate_iris_code`
- `hamming_distance`

`binarize`

Funkcja przyjmuje obraz w skali szarości i przekształca go do obrazu binarnego.

`keep_largest_contour`

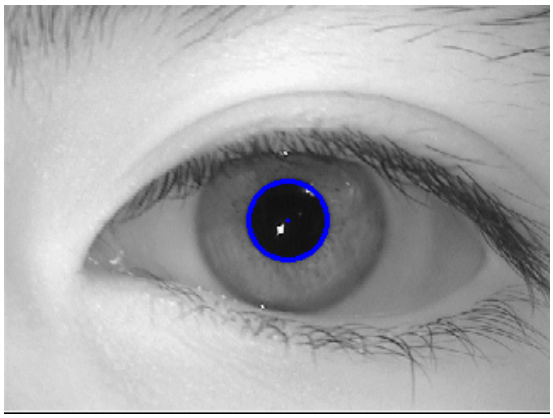
To funkcja wykorzystująca bibliotekę `OpenCV`, która przyjmuje obraz binarny i usuwa z niego wszystkie kontury poza tym największym pod względem pola powierzchni. Pozwala to na pozbycie się małych elementów, które zaburzają obraz. Funkcja ta jest wykorzystywana do wykrywania źrenicy oraz tęczówki.



Rysunek 3.1: Przykład użycia funkcji `keep_largest_contour`

`get_pupil`

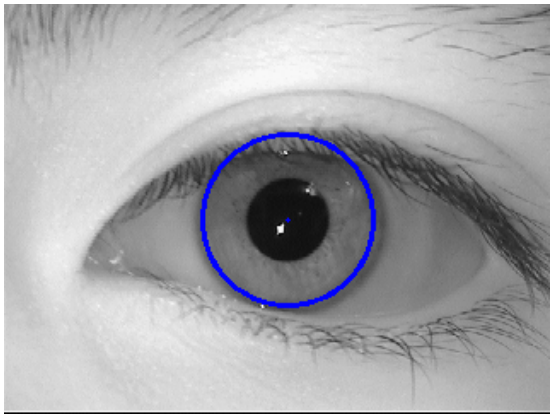
Funkcja, która na podstawie obrazu źrenicy w skali szarości lokalizuje jej środek oraz promień, a następnie zwraca obraz z zaznaczoną źrenicą. Na początku obraz jest binarizowany za pomocą funkcji `binarize`. Następnie wykonywana jest seria operacji morfologicznych: erozja, dylatacja, otwarcie i zamknięcie, mające na celu oczyszczenie obrazu z zakłóceń. Dodatkowo stosowany jest filtr medianowy w celu wygładzenia. Funkcja `keep_largest_contour` usuwa nieistotne wykryte kontury, pozostawiając tylko największy (najpewniej odpowiadający źrenicy). Po wstępnym przetworzeniu wykonywana jest projekcja pozioma i pionowa, na podstawie których wyznaczany jest środek źrenicy oraz jej promień. Na końcu funkcja zaznacza wykryty środek i promień na obrazie wejściowym.



Rysunek 3.2: Przykład działania funkcji `get_pupil`

`get_iris`

Funkcja służąca do wykrywania tęczówki na obrazie oka, bazując na wcześniej wyznaczonym środku i promieniu źrenicy. Na początku tworzona jest maska zakrywająca źrenicę oraz druga maska obejmująca tylko obszar poza nią, aby odizolować zewnętrzne części obrazu. Na podstawie tej drugiej maski obliczana jest średnia jasność otaczającego źrenicę obszaru, co pozwala ustalić dynamiczny próg binarizacji dostosowany do warunków oświetleniowych. Po binarizacji obraz jest czyszczony za pomocą operacji morfologicznych: dylatacji i erozji z wykorzystaniem różnych rozmiarów jąder. Następnie funkcja `keep_largest_contour` usuwa nieistotne kontury, pozostawiając tylko największy – zakładany jako obszar tęczówki. Kolejnym krokiem jest projekcja pozioma i pionowa, która pozwala obliczyć promień tęczówki. Na końcu funkcja zaznacza na obrazie środek (wcześniej wyznaczony na podstawie źrenicy) oraz promień wykrytej tęczówki.

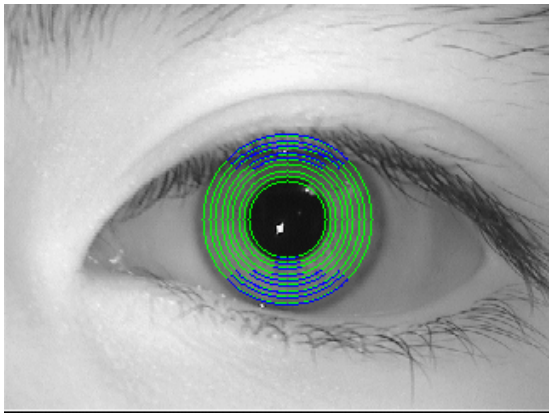


Rysunek 3.3: Przykład działania funkcji `get_iris`

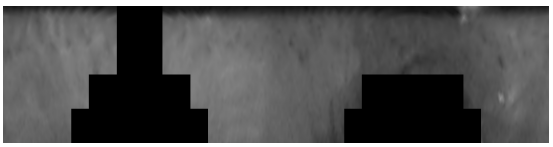
`unwrap_iris_with_masks`

Przekształca okrągły obszar tęczówki w prostokątny obraz, jednocześnie wykluczając wybrane fragmenty na podstawie ich położenia kątowego. Tęczówka jest podzielona na 8 pierścieni, gdzie z czterech najwęższych pasów usunięty został na dole wycinek kołowy o kącie 30°. W przypadku kolejnych dwóch pierścieni wycięty został wycinek kołowy o kącie 63° zarówno na górze jak i na dole. Dwa najszersze promienie również zostały wycięte z góry i z dołu wycinkiem kołowym kącie 90°. Rysunek 3.4 przedstawia wyznaczone okręgi z wyciętymi fragmentami.

Najpierw tworzona jest siatka w układzie biegunowym, gdzie każdemu punktowi przypisuje się pierścień i kąt. Na tej podstawie generowana jest maska, która odrzuca fragmenty znajdujące się w określonych zakresach kątów. Następnie obliczane są współrzędne punktów między źrenicą a granicą tęczówki, które służą do pobrania wartości pikseli z obrazu przy użyciu interpolacji. Na końcu maska kątowa jest nakładana na wynik, usuwając niechciane obszary. Zwracany jest prostokątny obraz rozwiniętej tęczówki.



Rysunek 3.4: Maska w układzie biegunowym z zaznaczonymi odrzuconymi fragmentami



Rysunek 3.5: Przykład rozwiniętej tęczówki

`generate_iris_code`

Funkcja generująca binarny kod tęczówki (tzw. iris code) na podstawie jej rozwiniętego obrazu, wykorzystując filtrację Gabora.

Na początku obliczany jest filtr Gabora za pomocą funkcji pomocniczej `build_gabor_kernel`. Następnie obraz rozwiniętej tęczówki dzielony jest na 8 pasów, a każdy pas dzielony jest na 128 równych pod względem liczby ważnych pikseli (niezerowych) kolumnowe segmenty. Dla każdego segmentu wykonywana jest filtracja rzeczywistą i urojoną częścią filtru Gabora, a następnie obliczany jest średni argument (faza) odpowiedzi filtru.

Na podstawie znaku i wartości bezwzględnej średniej fazy generowane są dwa bity reprezentujące dany fragment obrazu. Dla każdego z 8 (wierszy) × 128 (kolumn) segmentów uzyskujemy 2 bity, co daje finalnie macierz binarną o wymiarach 8 × 256.

Tak wygenerowany kod binarny wykorzystujemy do identyfikacji tęczówki.

Rysunek 3.6: Przykładowy kod tęczówki

`hamming_distance`

Funkcja oblicza odległość Hamminga pomiędzy dwoma kodami tęczówki. Weryfikuje zgodność ich wymiarów, a następnie zwraca stosunek liczby różnych bitów do całkowitej liczby bitów w kodzie. Jest to miara podobieństwa binarnych reprezentacji tęczówek — im mniejsza wartość, tym większe podobieństwo. Tęczówki uznajemy za takie same, gdy odległość Hamminga ma wartość poniżej 0.299.

4. Wnioski i podsumowanie

Podczas realizacji projektu napotkaliśmy kilka istotnych wyzwań, które miały wpływ na końcowe rezultaty.

Jednym z głównych problemów było wyznaczenie obszaru tęczówki. W wielu przypadkach jej jasność była bardzo zbliżona do otaczającej skóry, co znacznie utrudniało dokładne wykrycie granicy. Wymagało to zastosowania odpowiednich filtrów, metod morfologicznych oraz dopracowania masek, aby uzyskać możliwie precyzyjne wyniki.

Drugim wyzwaniem było poprawne wygenerowanie kodu tęczówki. Proces ten okazał się wrażliwy na różnego rodzaju zakłócenia i błędy segmentacji. Konieczne było odpowiednie dobranie parametrów, tak aby wygenerowany kod był stabilny i reprezentatywny dla danego obrazu oka. Dodatkowo bardzo istotne okazało się wycięcie odpowiednich fragmentów z obszaru tęczówki.

Dodatkową trudnością było ustalenie odpowiedniego progu dla odległości Hamminga. Zbyt niski próg powodował błędne odrzucenie poprawnych dopasowań, natomiast zbyt wysoki skutkował dopasowaniami nieprawidłowymi. Dobór tego parametru wymagał eksperymentowania oraz analizy wyników.

Pomimo tych trudności, projekt umożliwił zdobycie praktycznej wiedzy na temat przetwarzania obrazów biomedycznych, a także pozwolił zrozumieć, jak złożonym procesem jest rozpoznawanie tęczówki.

5. Bibliografia

- *Wybrane zagadnienia biometrii* - Krzysztof Ślot
- *How Iris Recognition Works* - John Daugman, 2004
- https://pages.mini.pw.edu.pl/∼rafalkoj/www/?Dydaktyka:2024
- Chat GPT