EKSTRAKCJA TĘCZÓWKI - DOKUMENTACJA

19 kwietnia 2025

Mateusz Karandys

Spis treści

1 Wstęp

Niniejszy dokument stanowi dokumentację aplikacji służącej do ekstrakcji tęczówki z obrazu oka przygotowanej w ramach przedmiotu Biometria na studiach Inżynierii i Analizy Danych na wydziale MiNI PW.

2 Opis aplikacji

Aplikacja umożliwia wydobycie tęczówki ze zdjęcia oka oraz rozwinięcie jej do prostokąta na podstawie którego tęczówki mogą być porównywane przy użyciu algorytmu Daugmana. Użytkownik może wczytać obraz oka w formacie bmp w skali szarości a następnie dokonać ekstrakcji tęczówki. W aplikacji wyświetlone zostaną wykryta źrenica, tęczówka oraz rozwinięcie tęczówki do postaci prostokątnej.

Aplikacja została stworzona w języku Python z wykorzystaniem bibliotek:

1. Tkinter - do budowy interfejsu graficznego
2. Numpy - do wygodnego operowania na macierzy obrazu
3. Pillow - do obsługi wyświetlania obrazu w okienku
4. OpenCV (cv2) - do stosowania operacji morfologicznych
5. Kagglehub - do wygodnego pobrania danych

3 Implementacja

Aplikacja została stworzona zgodnie z wzorcem projektowym Model-View-Controller. Poszczególne komponenty wzorca MVC znajdują się w folderze `src` plikach `model.py`, `ui.py`, `controller.py`. Są one komponowane w całą aplikację w klasie `App` z pliku `src/app.py`. Główna logika działania programu zaimplementowana została w pliku `src/processors/eye_processor.py`. Do uruchomienia programu służy plik `main.py`. Oprócz powyższych w folderze projektu znajdują się skrypt `script.py` do pobierania danych z kaggle, plik `README.md` oraz `requirements.txt` do łatwego uruchamiania aplikacji lokalnie.

3.1 Main

1. `main.py` - jest punktem wyjściowym aplikacji.
2. `script.py` - pobiera dane - zdjęcia oczu - z kaggla, z bazy danych `MMU-Iris-Database`.

3.2 Model-View-Controller

1. `app.py` - zawiera metadane potrzebne do stworzenia okienka aplikacji oraz przechowuje model, widok oraz kontroler.
2. `model.py` - przechowuje zmienne i obiekty definiujące aktualny stan aplikacji.
3. `ui.py` - definiuje układ podokienek, przycisków i etykiet w okienku aplikacji. Klasa `Ui` zawiera metody `display_image(image, cell, text)`, `remove_image(cell)` do wyświetlania zadanego obrazu `image` w komórce `cell` (od 1 do 4) z podpisem `text` oraz usuwania obrazu z wybranej komórki. Oprócz tego znajdują się tam dwie metody `show_info(info)` oraz `hide_info()` do wyświetlania i usuwania komunikatu `info` na ekranie. Służą one do informaowania użytkownika o błędach (np. o niewczytaniu zdjęcia).
4. `controller.py` - zawiera dwie główne metody `on_upload()` oraz `on_extract()` służące odpowiednio do obsługi wczytywania zdjęcia do programu oraz obsługi ekstracji tęczówki po naciśnięciu przycisku 'Extract' przez użytkownika. Znajduje się tu również pomocnicza metoda `display_image(image, cell=0, text=None)`, która przygotowuje zdjęcie do wyświetlenia i wywołuje odpowiednią metodę klasy `Ui`.

3.3 EyeProcessor

Klasa `EyeProcessor` zawiera główną logikę aplikacji. Odpowiada za wykrycie źrenicy wraz z jej środkiem oraz promieniem, za wykrycie tęczówki na podstawie źrenicy oraz za rozwinięcie tęczówki do prostokąta. Operacje te zostały zaimplementowane odpwiednio w metodach `process_pupil()`, `process_iris()` oraz `expand_to_rect()`. Omówimy zastosowane podejścia do powyższych problemów.

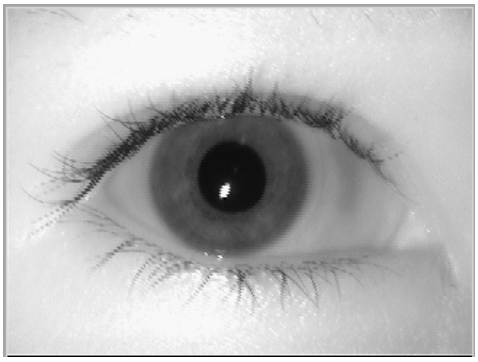
1. **Wykrycie źrenicy** - Obraz oka w skali szarości jest binaryzowany z progiem 40 (piksele ciemne, o wartościach szarości 0-40 stają się czarne, pozostałe 41-255 białe). Następnie stosowana jest operacja morfologiczna otwarcia z jądrem będacym macierzą 5 x 5 składającą się z jedynek. Po otwarciu stosowane jest zamknięcie z jądrem 9 x 9 również o elementach równych 1. Oba jądra zostały dobrane empirycznie. Później usuwane są z obrazu pozostałe, pojedyncze grupki czarnych pikseli, których nie udało się wyeliminować powyższymi operacjami morfologicznymi. Stosowana jest do tego analiza `connected components`. Na końcu wyznaczane są środek źrenicy oraz jej promień. Aplikajca wykorzystuje tutaj projekcję poziomą oraz pionową obrazu określając środki zakresów, w których występują piksele źrenicy. Za źrenicę przyjęte zostaje koło o parametrach wyznaczonych w poprzednim kroku.

2. **Wykrycie tęczówki** - Środek tęczówki dziedziczony jest jako środek źrenicy. Promień wyznaczany jest iteracyjnie. Metoda znajduje taki promień `r` z zakresu `[pupil_r, 3 * pupil_r]`, który maksymalizuje średnią szarość zdjęcia w pierścieniu kołowym o środku w środku źrenicy, promieniu zewnętrzym `r` oraz promieniu wewnętrznym `pupil_r`. Zastosowana takie podejście, gdyż tęczówka oka jest ciemniejsza od białka oka, więc wraz z przekroczeniem granic tęczówki średnia szarość pierścienia kołowego powinna się zmniejszać. Empirycznie metoda potwierdziła swoją skuteczność w znacznej większości przypadków (nie działa w około 5 na 460).

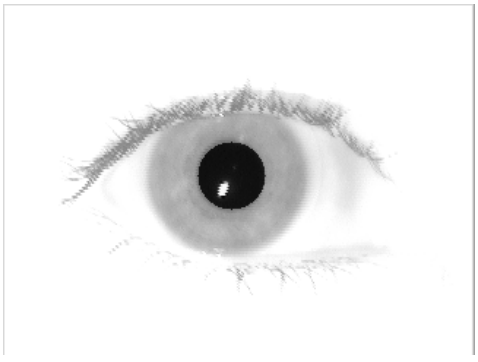
3. **Rozwinięcie tęczówki do prostokąta** - Pierścień o środku w środku źrenicy i promieniach wewnętrznym - źrenicy, zewnętrznym - tęczówki jest rozwijany do prostokąta stosując zmianę współrzędnych z biegunowych na prostokątne. W wyniku powstaje prostokąt o wymiarach `(iris_r - pupil_r)` x `(2π * iris_r)`. Puste piksele pozostałe po konwersji są uzupełniane wartością szarości 128.

4 Przykłady

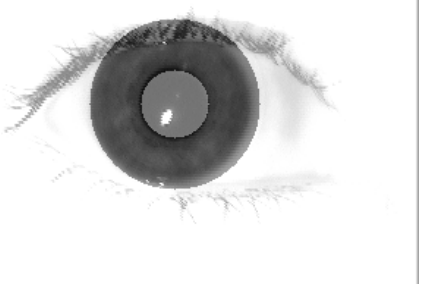
Na kolejnej stronie znajdują się zdjęcia trojga oczu wraz z wykrytymi przez aplikację źrenicami, tęczówkami oraz prostokątnymi rozwinięciami tęczówek.



Rysunek 1: Oryginał

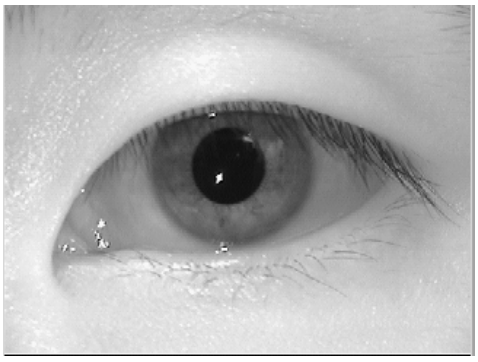


Rysunek 2: Źrenica

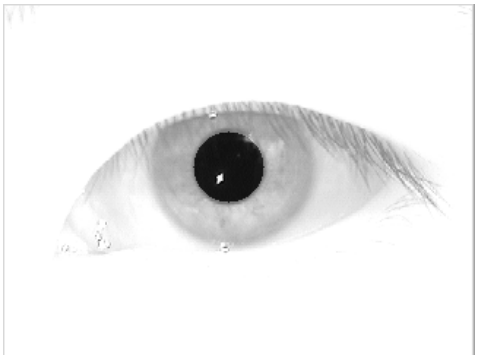


Rysunek 3: Tęczówka

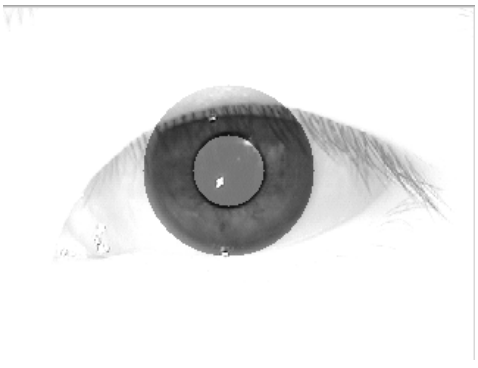
Rysunek 4: Tęczówka - prostokąt



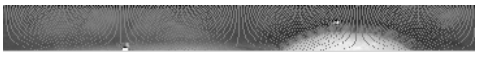
Rysunek 5: Oryginał



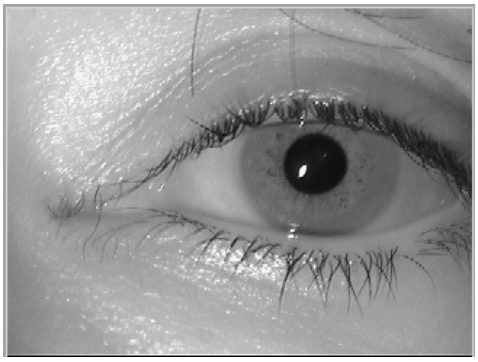
Rysunek 6: Źrenica



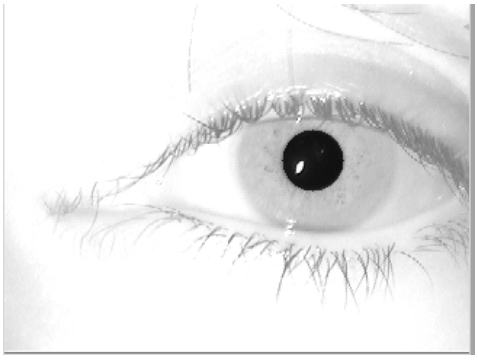
Rysunek 7: Tęczówka



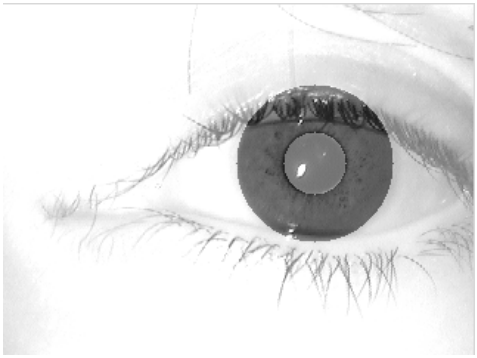
Rysunek 8: Tęczówka - prostokąt



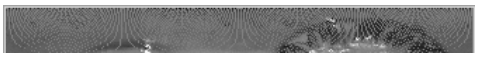
Rysunek 9: Oryginał



Rysunek 10: Źrenica



Rysunek 11: Tęczówka



Rysunek 12: Tęczówka - prostokąt

5 Podsumowanie

Stworzona aplikacja jest modularna i otwarta na rozszerzanie o nowe funkcjonalności, w szczególności na implementację algorytmu Daugmana do porównywania tęczówek. Główne trudności podczas tworzenia aplikacji obejmowały empiryczne dobranie progów binaryzacji, jąder operacji morfologicznych oraz stworzenie algorytmu wyznaczającego promień tęczówki. Struktura projektu i zastosowanie wzorca projektowego MVC znacznie ułatwiły organizację pracy i wprowadziły porządek w kodzie. Aplikacja wykrywa tęczówki z dużą dokładnością w przeważającej większości przypadków zdjęć pochodzących z bazy `MMU-Iris-Database`, co uznajemy za duży sukces. Słowem kończącym, uznajemy aplikację za ukończoną.