Uniwersytet Rzeszowski

Wydział Nauk Ścisłych i Technicznych

Instytut Informatyki

BIOMETRYCZNE SYSTEMY ZABEZPIECZEŃ

INSTRUKCJA DO ĆWICZEŃ LABORATORYJNYCH

TREŚCI KSZTAŁCENIA: IDENTYFIKACJA NA PODSTAWIE TĘCZÓWKI , ŚRODOWISKO MEGAMATCHER

Spis treści

Cele laboratorium

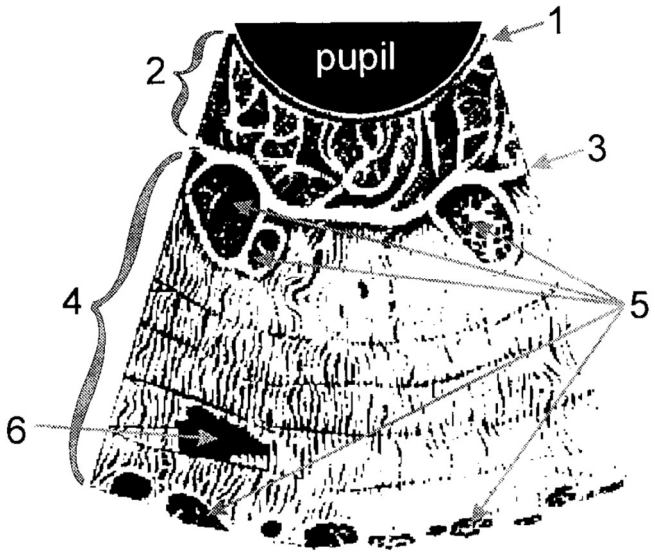
Celem laboratorium jest zapoznanie z technologiami identyfikacji daktyloskopijnej. W ramach zajęć omówiony będzie system MegaMatcher w kontekście identyfikacji osobników na podstawie cech tęczówki oka. Zadania będą zawierać część doświadczalną, a także praktyczną, polegającą na implementacji prostej aplikacji do identyfikacji biometrycznej z użyciem środowiska MegaMatcher SDK.

Wprowadzenie

Charakterystyka rozpoznawania tęczówki

Tęczówka jest wysuniętą do przodu częścią błony naczyniowej. Jest widoczna przez rogówkę, swą nazwę zawdzięcza wielkiej rozmaitości zabarwień. Barwa tęczówki może przyjmować różnorodne odcienie, od żółtej, zielonej i niebieskiej do fiołkowej oraz przez brązową do prawie czarnej. Barwa tęczówki zależy od ilości barwnik oraz budowy powierzchni przedniej.

Rysunek 1 przedstawia przednią powierzchnię ludzkiej tęczówki. „Pigment frill" jest to powierzchnia pofałdowana posiadająca pigment. „Pupillary area"– powierzchnia przejściowa między źrenicą a tęczówką. „Collarette"- jest to poszarpane koło w połowie tęczówki, rozdzielający ciemniejszy odcień tęczówki od jaśniejszego. „Ciliary area"- obszar rzęskowy. Krypty są to promieniste zagłębienia na powierzchni tęczówki. „Pigment spot" jest to plama na tęczówce zawierająca pigment.



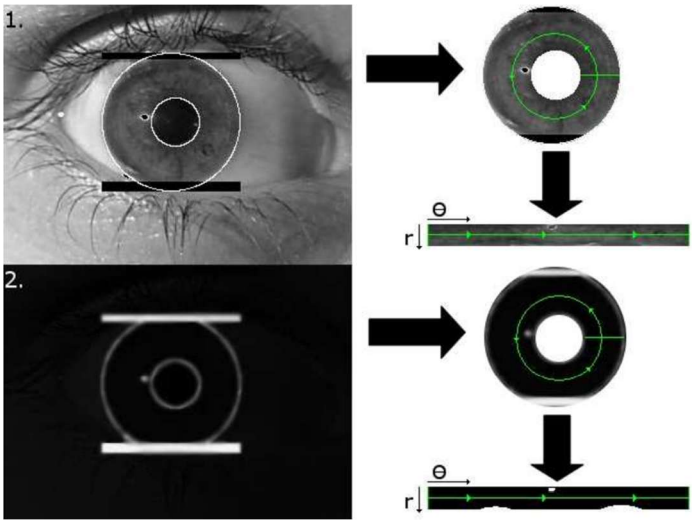
Rysunek 1: Przednia powierzchnia ludzkiej tęczówki: pupil- źrenica, 1–pigment frill, 2-pupillary area, 3-collarette, 4-ciliary area, 5-crypts- krypty, 6-pigment spot

Rozpoznawanie tęczówki to zaawansowany proces biometryczny, który wymaga kilku etapów przetwarzania obrazu. Cała procedura obejmuje normalizację kształtu tęczówki, ekstrakcję unikalnych cech z jej wzorca oraz zakodowanie ich w postaci binarnej. Dzięki temu możliwe jest porównywanie różnych obrazów tęczówek w sposób szybki i precyzyjny, niezależnie od warunków oświetleniowych, skali czy położenia oka na obrazie.

1. Normalizacja

W procesie normalizacji wykorzystuje się metodę **Rubber Sheet Model Daugmana**. Ponieważ źrenica nie zawsze znajduje się idealnie w centrum tęczówki, model przekształca obszar pierścienia tęczówki na obraz prostokątny (o wymiarach 240×20 pikseli), niezależnie od jej rozmiaru. Robi to poprzez

przeskalowanie punktów wzdłuż promieni, w zależności od kąta, z uwzględnieniem przesunięcia środka źrenicy względem tęczówki. W ten sposób promień i kąt stają się współrzędnymi kartezjańskimi.

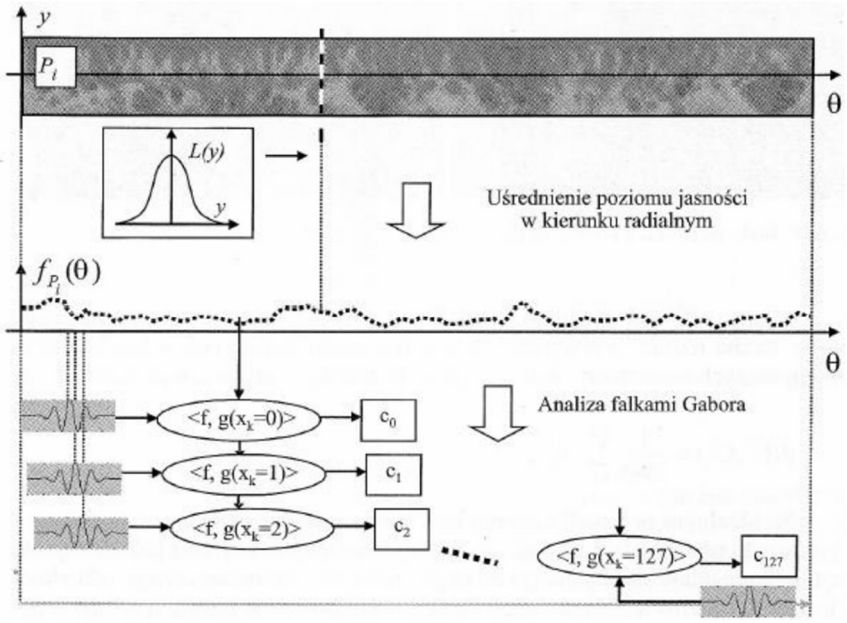


Rysunek 2: proces normalizacji tęczówki, 2 – proces normalizacji maski zakłóceń tęczówki

2. Wyodrębnianie cech

Znormalizowany obraz zostaje poddany analizie z użyciem **falek Gabora** – funkcji, które pozwalają wyodrębnić lokalne wzorce strukturalne tęczówki. Obraz reprezentowany jest jako zbiór współczynników (liczb zespolonych), opisujących podobieństwo fragmentów obrazu do tych falek. Ze względu na właściwości tęczówki, analiza ogranicza się do reprezentacji jednowymiarowej.

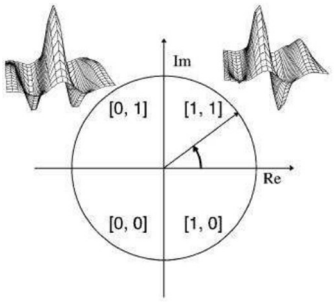
Analiza za pomocą falek Gabora jest wykonalna tylko w przypadku danych jednowymiarowych. W analizie obrazu tęczówki problem dwuwymiarowy można uprościć do jednowymiarowego, ponieważ punkty rozmieszczone promieniście są silnie skorelowane, a te po okręgu – słabo. Dlatego obraz wzdłuż promienia przekształca się w pas, w którym każdą kolumnę zastępuje się średnią jej wartości, uśrednianą z użyciem okna Gaussowskiego, faworyzującego środek.



Rysunek 3: Procedura wyznaczenia zbioru współczynników rozwinięcia obrazu pasa Pi tęczówki przy pomocy falek Gabor'a o stałej częstotliwości i stałym rozmyciu.

3. Ekstrakcja kodu tęczówki

Z każdego współczynnika falki Gabora wyodrębnia się **fazę**, która jest odporna na zmiany jasności i kontrastu. Faza ta jest kodowana na **dwa bity** – zależnie od ćwiartki płaszczyzny zespolonej, w której leży wynik.



Rysunek 4: Ekstrakcja fazy do kodu tęczówki.

4. Porównywanie

Ostatecznie porównuje się kody binarne dwóch tęczówek, licząc ile bitów się różni. To porównanie realizowane jest za pomocą **odległości Hamminga**, która mierzy różnice między dwoma ciągami binarnymi.

$$HD = \frac{1}{N}\sum_{j=1}^{N} x_j (XOR) y_j$$

W środowisku MegaMatcher w procesie porównywania uwzględniana jest także tzw. Maska zakłóceń, która ma na celu przypisanie wagi równej 0 tym bitom, które nie wchodzą w skład tęczówki (np. odpowiadają fragmentom powieki). Zmodyfikowana odległość Hamminga dla takiego podejścia przyjmuje następujący wzór:

$$HD = \frac{1}{N - \sum_{i=1}^{N} Xn_k (OR) Yn_k} \sum_{j=1}^{N} X_j (XOR) Y_j (AND) Xn'_j (AND) Yn'_j$$

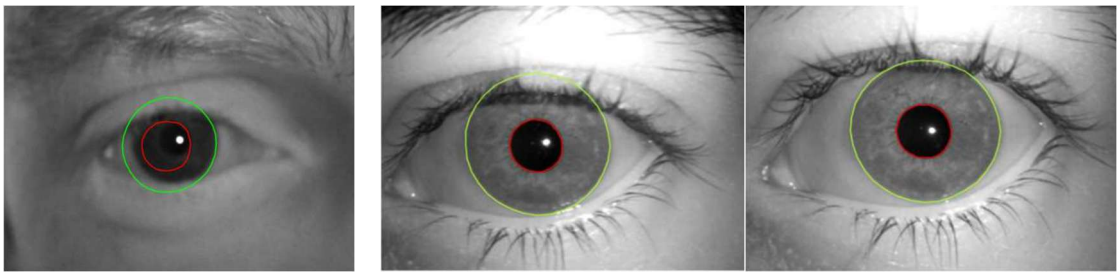
gdzie X_j i Y_j to kody tęczówek, Xn_j i Yn_j są to maski zakłóceń tych kodów, a N jest to liczba reprezentująca ilość bitów kodu tęczówki.

Skaner IriShield™ USB MK 2120U

Wyżej wymienione urządzenie to kompaktowy skaner tęczówki oka produkowany przez firmę IriTech. Cechuje się automatycznym wykrywaniem i przechwytywaniem obrazu tęczówki, który ponadto zabezpieczany jest unikalnym dla urządzenia 2048-bitowym kluczem RSA. Urządzenie jest kompatybilne z wieloma systemami operacyjnymi; Windows 32/64-bit, Linux 32-bit, WinCE, Embedded Linux oraz Android.

Rysunek 5: Skaner IriShield™ USB MK 2120U

Wykorzystując skaner IriShield pobranie obrazu tęczówki oka jest bardzo łatwe i wygodne dla użytkownika. Obraz ten jest uzyskiwany przy bliskim ułożeniu oka do skanera. Urządzenie działa z małym ryzykiem błędu, jedyny błąd, który może się pojawić to nieprawidłowa odległość oka, co może powodować złą widoczność obrazu tęczówki oka. Źle dobrana odległość może poskutkować niepoprawnym wyznaczeniem okręgów źrenicy i tęczówki, a co za tym idzie błędnym wektorem cech. Taka sytuacja uniemożliwia procesy segmentacji i porównywania tęczówek (Rysunek 6).



(a) (b) (c)

Rysunek 6: Przykłady wyznaczania okręgów tęczówki oka:: źle przetworzony obraz (a), prawidłowo przetworzony obraz (b) (c)

Wykonując pomiary należy zwrócić uwagę na kilka kwestii:

- Odległość od czujnika; powinna wynosić optymalnie około 5 cm (4.7 – 5.3 cm).
- Należy zminimalizować odbicia światła w oku. Również bezpośrednie działanie słońca w trakcie eksperymentu może pogorszyć jakość.
- Na czas przechwytywania należy zminimalizować ruch urządzenia.

Środowisko MegaMatcher

MegaMatcher to zaawansowany zestaw narzędzi programistycznych (SDK) przeznaczony do tworzenia systemów identyfikacji biometrycznej na dużą skalę. Obsługuje różne modalności biometryczne, takie jak odciski palców, twarz, tęczówka, głos oraz odciski dłoni.

Przeglądanie i tworzenie własnych gotowych rozwiązań w oparciu o SDK

Pakiet MegaMatcher dysponuje bogatą dokumentacją SDK, tutorialami, które prezentują działanie na krótkich przykładach oraz przykładowe aplikacje z pełnym kodem źródłowym.

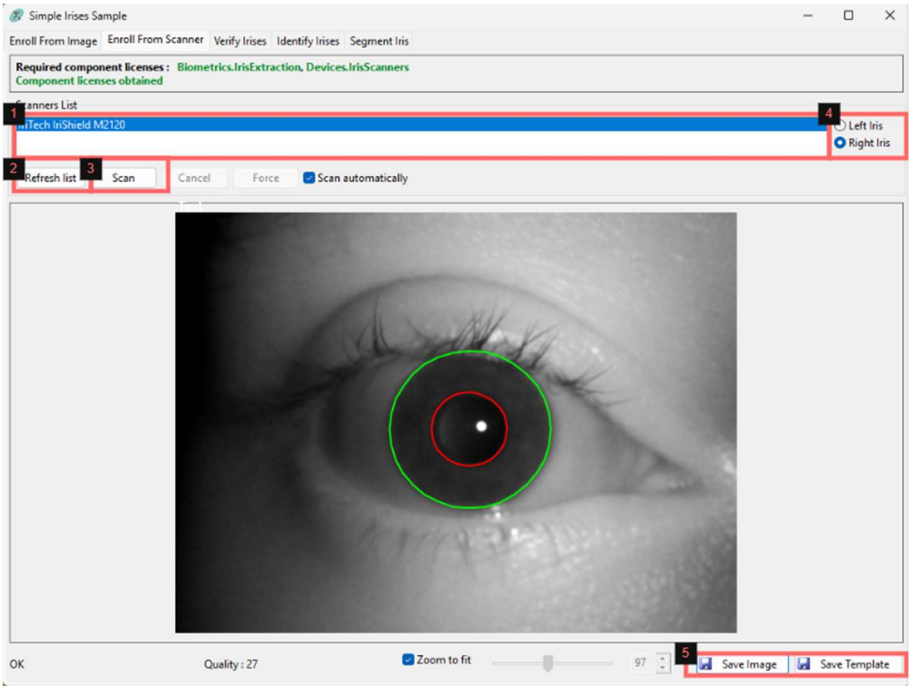
Aby móc uruchomić przykłady, należy tworzyć odpowiednią strukturę projektu. Bardzo ważna jest odpowiednia konfiguracji narzędzi budowania, integracją z natywnymi plikami dll zawierającymi niskopoziomiową logikę SDK, oraz systemem zarządzania licencjami. Dla uproszczenia można skorzystać z programu WorkspaceInitializer, który znajduje się na pulpicie stanowisk laboratoryjnych. Szczegółowa instrukcja korzystania z programu znajduje się w pliku *Inicjalizacja przestrzeni roboczej*.

Zadania do samodzielnego rozwiązania

Zadanie 1. Zbieranie danych tęczówki oka za pomocą Skanera IriShield-USB

a) Stworzyć indywidualny folder dla swojego zespołu – będzie to baza szablonów tęczówek oka.

b) Uruchomić program *SimpleIrisesSample*, znajdujący się na pulpicie (Rysunek 7). Wybrać zakładkę *Enroll From Scanner*.



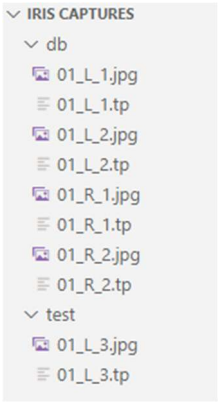
Rysunek 7: Program SimpleIrisesSample umożliwiający m. in. akwizycję obrazu tęczówki. (1) lista wykrywanych urządzeń, (2) przycisk odświeżania listy urządzeń, (3) przycisk rozpoczęcia skanowania, (4) przełącznik trybu lewego i prawego oka, (5) opcje zapisu.

c) Dla każdej osoby wykonać skany obojgu oczu. Po każdorazowym pomiarze zapisać zarówno **plik szablonu** w rozszerzeniu .**tp**, jak i **plik obrazu**, nadając im tę samą nazwę. Skorzystać z oznaczeń zaprezentowanych w Tabela 1. Wszystkie szablony i zdjęcia niech znajdują się w folderze **db**, dodać również nowe skany testowe do folderu **test** (Rysunek 8).

<nr osoby>_<oznaczenie oka>.tp, Np.: 01_L.tp, 01_R.tp itd.
<nr osoby>_<oznaczenie oka>.jpg, Np.: 01_L.tp, 01_R.tp itd.

	Oko Lewe	Oko Prawe
Oznaczenie	L	R

Tabela 1: Oznaczenia oczu

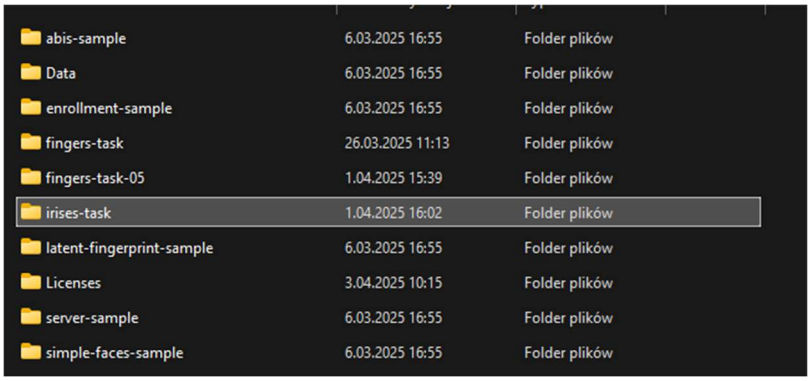


Rysunek 8: Fragment struktury bazy szablonów

Zadanie 2. Porównywanie szablonów w środowisku MegaMatcher

Aby rozpocząć realizację zadania należy zainicjować nową przestrzeń roboczą. W tym celu należy skorzystać z programu WorkspaceInitializer, znajdującego się na pulpicie. Szczegółowa instrukcja znajduje się w pliku ***Inicjalizacja przestrzeni roboczej****.*

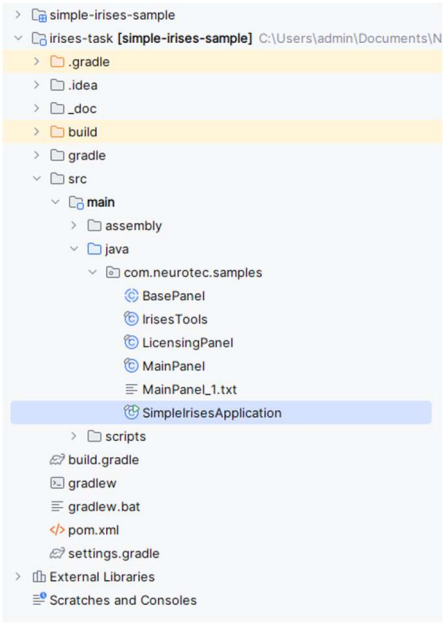
W przestrzeni roboczej umieścić folder z kodem początkowym, umieszczony w katalogu **„Lab 08”** na pulpicie.



Stworzyć prostą aplikację do porównywania szablonów tęczówki. W tym celu uzupełnić klasę MainPanel w projekcie irises-task. Do zrealizowania zadania może się przydać dokumentacja SDK, tutoriale oraz przykłady gotowych aplikacji.

Projekt został stworzony na bazie aplikacji demonstracyjnych pakietu Neurotec. W katalogu *com.neurotec.samples* znajduje się właściwy kod programu (Rysunek 9).

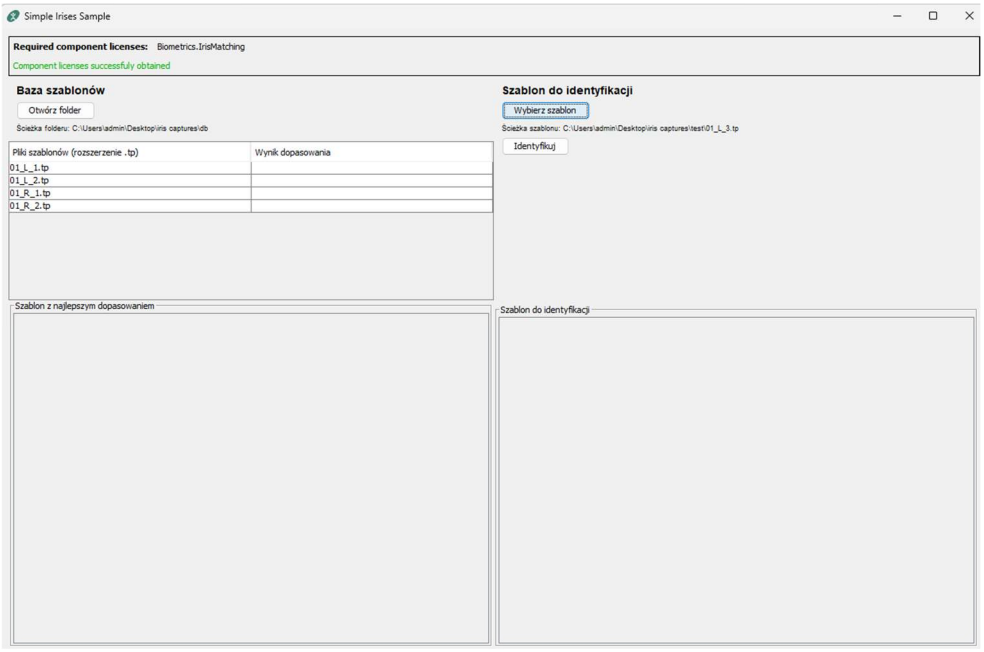
- Klasa **BasePanel –** klasa generyczna będąca modelem widoku. Po niej dziedziczy klasa MainPanel.
- Klasa **IrisesTools** – klasa odpowiedzialna za pobieranie dostępu do komponentów biometrycznych – w tym przykładzie do licencji `Biometrics.IrisMatching`.
- Klasa **LicensingPanel** – wyświetla na górze okna aplikacji informacje o uzyskanych licencjach.
- Klasa **MainPanel** – główny widok aplikacji, który będzie modyfikowany w ramach zadania.
- Klasa **SimpleIrisesApplication** – punkt wejściowy, inicjujący aplikację.



Rysunek 9: Struktura projektu

Wyjściowa postać aplikacji (Rysunek 10) pozwala na:

1. Wczytanie folderu bazy szablonów – są to pliki szablonów o rozszerzeniu .tp. o które będzie się opierał proces identyfikacji. Po wczytaniu wszystkie szablony pokazują się w tabeli.
2. Wczytanie szablonu do identyfikacji – jest to szablon, który będzie symulował nieznaną tęczówkę. Po wczytaniu wyświetli się ścieżka do szablonu.



Rysunek 10: Wyjściowa postać aplikacji

Należy uzupełnić kod następująco:

1. Po wybraniu szablonu do identyfikacji niech załadowany zostanie podgląd odpowiadającego jej obrazu. W tym celu należy stworzyć dwie funkcje:
   a) *getCorrespondingImagePath* - dla zadanego pliku szablonu zwraca ścieżkę do odpowiadającej mu ścieżki do obrazu:

```
private String getCorrespondingImagePath(File templateFile) {
    String parentPath = templateFile.getParent();
    String fileName = templateFile.getName().split("\\.")[0];

    // Próba wczytania obrazu z tej samej lokalizacji i o tej samej nazwie co
plik szablonu
    for (String fmt : ImageIO.getWriterFormatNames()) {
        String result = parentPath + File.separator + fileName + "." + fmt;
        if (new File(result).exists()) {
            return result;
        }
    }

    return null;
}
```

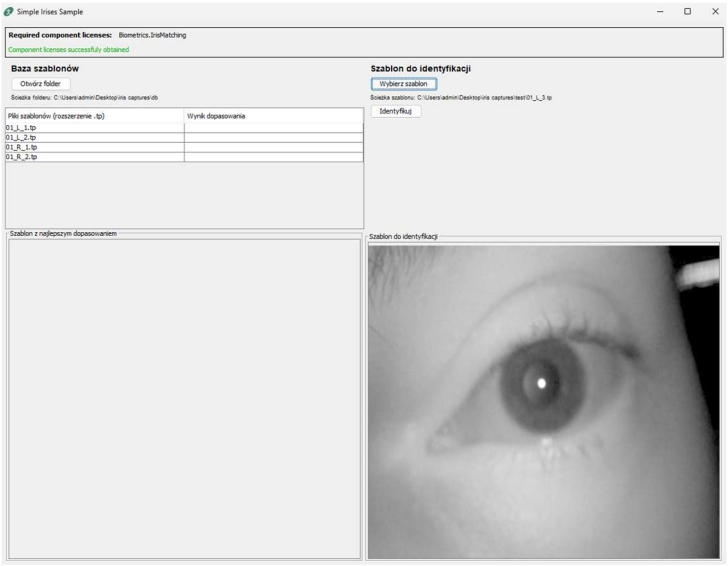
b) setIrisView – umieszcza obraz z podanej ścieżki wewnątrz określonego panelu.

```
private void setIrisView(String imagePath, JPanel target) {
    try {
        BufferedImage img = ImageIO.read(new File(imagePath));

        if (img == null) {
            SwingUtilities.invokeLater(() -> JOptionPane.showMessageDialog(this,
                    "Nie mozna wczytać obrazu: " + imagePath, "Błąd",
                    JOptionPane.WARNING_MESSAGE));
            return;
        }

        ImageIcon imageIcon = new ImageIcon(img);
        JLabel imageLabel = new JLabel(imageIcon);

        Image imgScaled = img.getScaledInstance(target.getWidth(),
target.getHeight(), Image.SCALE_SMOOTH);
        imageIcon = new ImageIcon(imgScaled);
        imageLabel.setIcon(imageIcon);

        target.removeAll();
        target.add(imageLabel);
        target.revalidate();
        target.repaint();
    } catch (IOException e) {
        SwingUtilities.invokeLater(() -> JOptionPane.showMessageDialog(this,
                "Błąd wczytywania obrazu: " + imagePath, "Błąd",
                JOptionPane.ERROR_MESSAGE));
    }
}
```

Następnie wykorzystać powyższe funkcje do wyświetlenia podglądu załadowanego szablonu w funkcji *handleSelectTemplate*.

```
String imagePath = getCorrespondingImagePath(file);
if (imagePath != null) {
    setIrisView(imagePath, panelIrisRight);
} else {
    SwingUtilities.invokeLater(() -> JOptionPane.showMessageDialog(this,
            "Nie odnaleziono obrazu dla tego szablonu: " + file.getPath(),
"Uwaga",
            JOptionPane.WARNING_MESSAGE));
}
```

Wynik powinien wyglądać następująco:



Rysunek 11: Aplikacja z podglądem wybranego szablonu

2. W wyniku kliknięcia przycisku *Identyfikuj* niech zostanie uruchomiona funkcja identyfikacyjna SDK. W tym celu należy uzupełnić funkcję *identify*. Pierwszym etapem jest inicjalizacja klienta `NBiometricClient`. Następnie tworzone jest nowa operacja UPDATE, polegająca na dodaniu do klienta osobników (NSubject) z bazy szablonów. Potem ustawiany jest próg dopasowania, sterujący minimalną wartość parametru *score* do uznania dwóch szablonów za zgodne.

```
updateIrisesTools();

NBiometricClient client = IrisesTools.getInstance().getClient();
client.clear();

NBiometricTask updateTask =
client.createTask(EnumSet.of(NBiometricOperation.UPDATE), null);
for (NSubject s : baseSubjects) {
    updateTask.getSubjects().add(s);
}

client.performTask(updateTask);
client.setMatchingThreshold(48);

NBiometricStatus status = client.identify(subjectToIdentify);
```

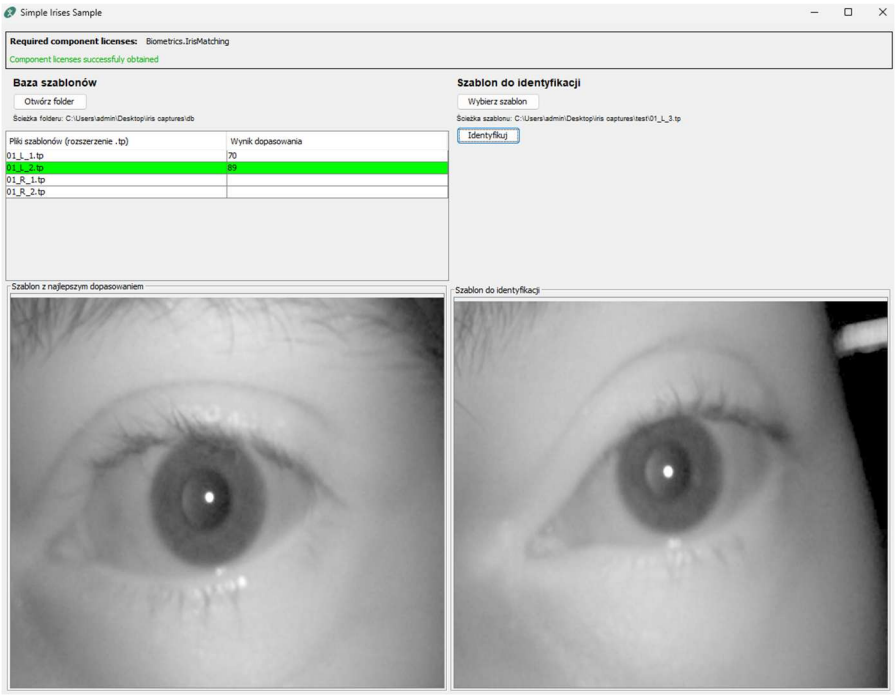
3. Po uzyskaniu wyników niech stworzona będzie mapa, przechowująca dla każdego osobnika (id to nazwa pliku, np. 01_L.pt) wartość *score* dopasowania.

```
HashMap<String, Integer> resultsPerSubject = new HashMap<>();
if (status == NBiometricStatus.OK) {
    for (int i = 0; i < subjectToIdentify.getMatchingResults().size(); i++) {
        NMatchingResult result = subjectToIdentify.getMatchingResults().get(i);
        System.out.format("Matched with ID: '%s' with score %d\n",
result.getId(), result.getScore());
        resultsPerSubject.put(result.getId(), result.getScore());
    }
} else if (status == NBiometricStatus.MATCH_NOT_FOUND) {
    System.out.format("Match not found");
} else {
    System.out.format("Identification failed. Status: %s.\n", status);
}
```

4. W oparciu o wyznaczone dane, można zaktualizować widok tabelki wyników. W tym celu należy iterować po każdym wczytanym osobniku i pobierać jego *score*, po czym aktualizować

dane w tabelce. Jednocześnie można wyznaczać indeks osobnika o najwyższym dopasowaniu do podświetlenia w tabeli. Podobnie jak omówiono to wcześniej, można wyświetlić podgląd dopasowania.

```
int maxIndex = -1;
int maxValue = Integer.MIN_VALUE;
for (int i = 0; i < baseSubjects.size(); i++) {
    NSubject subject = baseSubjects.get(i);
    Integer result = resultsPerSubject.get(subject.getId());
    tableModel.setValueAt(result, i, 1);

    if (result != null && result > maxValue) {
        maxValue = result;
        maxIndex = i;
    }
}

highlight(maxIndex);
String imgPath = getCorrespondingImagePath(this.baseSubjectsFiles.get(maxIndex));
setIrisView(imgPath, panelIrisLeft);
updateControls();
```



Rysunek 12: Finalna postać aplikacji

5. Należy przetestować rezultat działania programu dla różnych wartości matchingThreshold.