Akademia Górniczo-Hutnicza

Wydział Informatyki

Projekt nr 4
Z przedmiotu Informatyka Medyczna

"Porównywanie odcisków Palców"

Autor: *Jacek Tyszkiewicz*
Kierunek studiów: *Informatyka*

Kraków, 2025

Spis treści

1. Segmentacja obrazów medycznych

1.1 Cel projektu:

Celem projektu było zapoznanie się z przetwarzaniem i analizą obrazów medycznych w formacie DICOM. Należało zaimplementować interaktywne narzędzia do wizualizacji oraz analizy danych obrazowych. Podczas realizacji tego zadania wykonano:

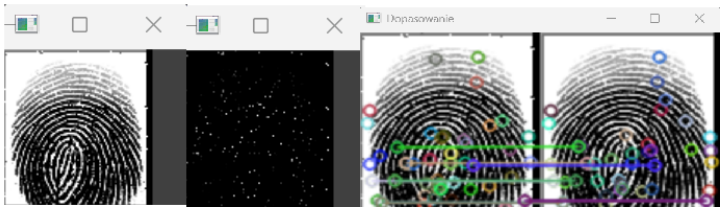
1.2 Zadanie 1

- Uruchom kod z pliku `zad1.py`
- Porównaj działanie przy użyciu innych parametrów
- Umieść w raporcie:
  - czas obliczeń dopasowania,
  - końcowy wynik metryki **MattesMutualInformation**,
  - obraz różnicowy (reprezentatywny przekrój) i szachownica (reprezentatywny przekrój),
  - wizualizację 3D wyników (porównanie przed i po)

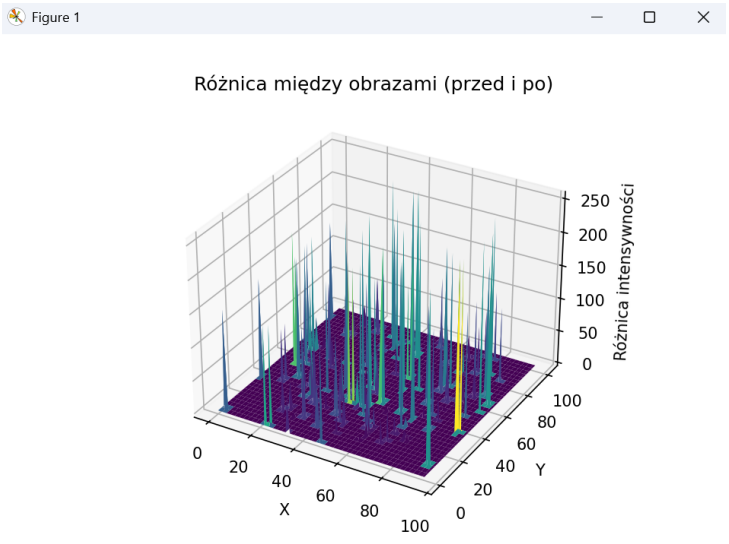
Program porównuje odcisk palca z obrazu testowego z odciskami znajdującymi się w bazie (Real_subset). Wykorzystuje do tego algorytm SIFT do wykrywania cech charakterystycznych (kluczowych punktów) oraz algorytm FLANN (Fast Library for Approximate Nearest Neighbors) do ich dopasowania.

FLANN to szybka biblioteka wyszukiwania sąsiadów, która umożliwia efektywne porównywanie dużych zestawów cech. Dzięki metodzie KNN (k-nearest neighbors) oraz filtrowaniu wyników za pomocą testu Lowe'a, dopasowywane są tylko najbardziej wiarygodne punkty wspólne między obrazami.

Dla każdego obrazu w bazie wyznaczany jest procent poprawnych dopasowań względem obrazu testowego. Jeśli dopasowanie przekracza ustalony próg, program wyświetla wynik oraz wizualizację z zaznaczeniem pasujących punktów.



Rys. 1.1. Opis obrazka (np. Przykładowy odcisk palca)



Rys. 1.2. wyniki

Opis najważniejszych części kodu na końcu sprawozdania.

1.3 Opis najważniejszych części kodu

1.4 ZADANIE 2 – Registration challenge (3 pkt)

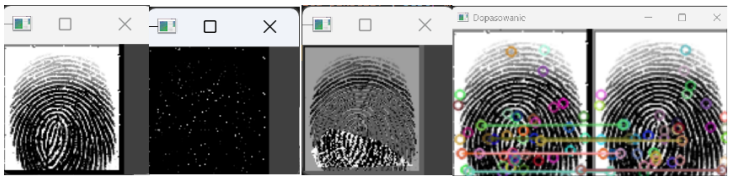
- **Multi-resolution framework** (podejście *wielorozdzielczościowe / wieloskalowe*)
- Inne metody transformacji, lub ich złożenie:
  - Free Form Deformation
  - Demons Based Registration
  - Thin Plate Splines
  - Inne
- Różne interpolatory
- Różne optymalizatory
- Umieścić w raporcie:
  - czas obliczeń dopasowania
  - końcowy wynik metryki **MattesMutualInformation**
  - obraz różnicowy (reprezentatywny przekrój)
  - szachownica (reprezentatywny przekrój)
  - wizualizacja 3D wyników (porównanie przed i po)

W kodzie mamy:

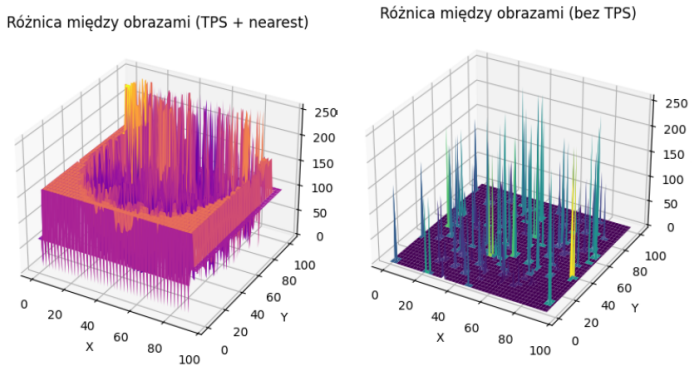
– **SIFT** – do ekstrakcji punktów kluczowych i cech,

– **FLANN** – do dopasowania punktów kluczowych (matcher),

– **PiecewiseAffineTransform** – jako przybliżenie transformacji typu Thin Plate Spline (TPS).



Rys. 1.3. wynik



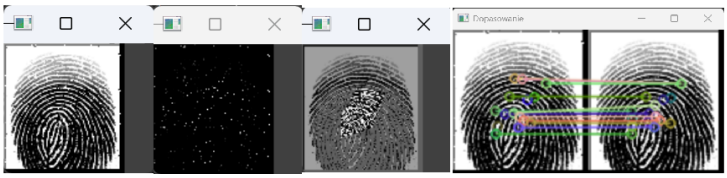
Rys. 1.4. wykres

1. Transformacja TPS (Piecewise Affine):

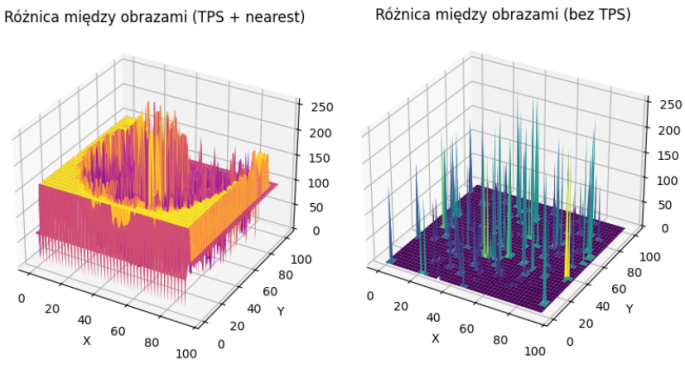
```
tps_transform = PiecewiseAffineTransform()
tps_transform.estimate(src_pts, dst_pts)
warped = warp(test_preprocessed, tps_transform, order=0, mode='
    edge')
```

W kodzie mamy:

- **AKAZE** – do detekcji punktów kluczowych i ekstrakcji cech,
- **Brute-Force Matcher (Hamming)** – do dopasowania cech między obrazami,
- **Piecewise Affine Transform** – jako przybliżenie transformacji typu Thin Plate Spline (TPS).



Rys. 1.5. wynik



Rys. 1.6. wykres

1. Ekstrakcja cech – AKAZE:

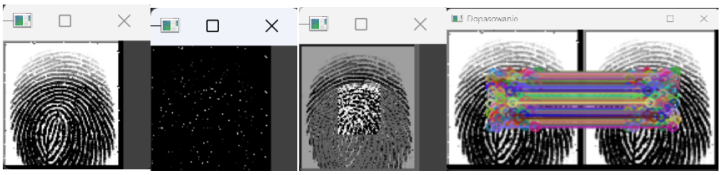
```
akaze = cv2.AKAZE_create()

def features_extraction(image):
    return akaze.detectAndCompute(image, None)
```

2. Dopasowanie cech – Brute-Force z Hammingiem:

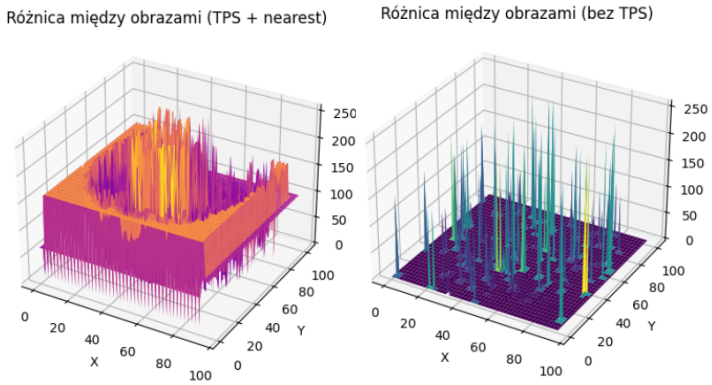
```
bf = cv2.BFMatcher(cv2.NORM_HAMMING)

matches = bf.knnMatch(desc1, desc2, k=2)
good_matches = []
for m, n in matches:
    if m.distance < 0.8 * n.distance:
        good_matches.append(m)
```

W kodziemy mamy:

- **ORB** – szybka metoda ekstrakcji cech (binarne deskryptory),
- **Brute-Force Matcher (Hamming)** – dopasowanie cech z ratio testem,
- **Piecewise Affine Transform (TPS-like)** – nieliniowa lokalna transformacja obrazu,



Rys. 1.7. wynik



Rys. 1.8. wykres

1. Ekstrakcja cech – AKAZE:

```
akaze = cv2.AKAZE_create()

def features_extraction(image):
    return akaze.detectAndCompute(image, None)
```

2. Dopasowanie cech – Brute-Force z testem ratio:

```
bf = cv2.BFMatcher(cv2.NORM_HAMMING)

matches = bf.knnMatch(desc1, desc2, k=2)
good_matches = []
for m, n in matches:
    if m.distance < 0.8 * n.distance:
        good_matches.append(m)
```

1.5 Opis najważniejszych części kodu

1.Ekstrakcja cech - SIFT:

```
sift = cv2.SIFT_create()
def features_extraction(image):
    return sift.detectAndCompute(image, None)
```

Funkcja zwraca punkty kluczowe oraz deskryptory obrazu.

2.Dopasowanie cech - FLANN:

```
flann = cv2.FlannBasedMatcher(dict(algorithm=1, trees=10), dict()
   )
matches = flann.knnMatch(descriptors_1, descriptors_2, k=2)
```

Ratio test odrzuca słabe dopasowania:

```
if p.distance < 0.1 * q.distance:
    match_points.append(p)
```

3.Wybór najlepszego dopasowania

```
match_ratio = len(match_points) / keypoints_count

    if match_ratio > len(best["matches"]) / keypoints_count:
        best.update({
            "file": file,
            "image": db_image,
            "keypoints_2": keypoints_2,
            "matches": match_points,
            "match_time": match_time,
            "mean_diff": np.mean(cv2.absdiff(test_preprocessed,
               preprocess(db_image)))
        })
```

4. Transformacja TPS (Piecewise Affine):

```
tps_transform = PiecewiseAffineTransform()
tps_transform.estimate(src_pts, dst_pts)
warped = warp(test_preprocessed, tps_transform, order=0, mode='
   edge')
```