Algoritmes.

Welke er zijn, hoe ze werken, waar ze goed in zijn en hoe er een is geïmplementeerd.

Vingerafdrukanalyse

…door middel van een Python script.

Brent Schouten

**Verschillende algoritmes**

Voor het vergelijken en matchen van de vingerafdrukken hebben wij besloten een programma te gebruiken dat is geschreven in de codetaal Python. Hierdoor hebben wij de vingerafdrukken veel sneller kunnen vergelijken dan andere groepen. Het programma werkt door een foto te vergelijken met een groep andere foto’s in een database.

Bij het schrijven van dit programma was er keuze tussen 3 verschillende algoritmes, die ieder hun eigen voordelen, nadelen en eigenschappen hebben. Die drie zijn **M**ain **S**quared **E**rror (**MSE**), **S**tructural **S**imilarity **I**ndex **M**easure (**SSIM**) en **Root S**cale-**I**nvariant **F**eature **T**ransform (**RootSIFT**).

* Eigenschap

**+** Positief

**-** Negatief

**MSE**

**+** Makkelijk te implementeren. **+** Rekenkundig erg simpel. **+** Erg lichtgewicht voor de computer. **+** Erg snel.

**-**  Beschouwt complexere foto’s anders dan mensen dat doen; geen patroondetectie.  
 **-**  Legt geen nadruk op structurele verschillen zoals andere algoritmes.

* Vergelijkt alle pixelverschillen (contrast, kleur, etc.) en behandelt ze gelijk.
* Best voor rekenkundig lichte en snelle vergelijkingen van eenvoudige afbeeldingen.

**SSIM**

**+** Houdt rekening met hoe mensen foto’s waarnemen. **+** Houdt rekening met structuren. **+** Analyseert ook aangrenzende pixels in plaats van alleen de pixels zelf zoals bij MSE.

**-**  Meer rekenkracht nodig dan MSE, wat het langzamer maakt.  
 **-**  Moeilijker te implementeren dan MSE.  
 **-**  Slecht met kleine details en verschillen.  
 **-**  Resultaten kunnen verschillen door de grootte van de foto’s.

* Best voor het vergelijken van complexere foto’s dan bij MSE.

**RootSIFT**

**+** Let niet op grootte of draaiing van de foto’s. **+** Kan goed omgaan met ruis, vervormingen en veranderingen in belichting. **+** Erg goed in het analyseren van simpele en complexe foto’s. **+** Erg goed in objectdetectie.

**-**  Vereist erg veel rekenkracht en tijd relatief tot andere algoritmes.  
 **-**  Erg moeilijk te implementeren.  
 **-**  Kan minder goed omgaan met kleine foto’s of foto’s met een lage resolutie.

* Negeert kleur compleet en focust alleen op helderheid.
* Best voor het vergelijken van erg complexe foto’s waar kleur geen rol speelt en als tijd niet van belang is.

**Keuze en uitleg.**

Uiteindelijk hebben we besloten dat RootSIFT het beste is om de vingerafdrukken te vergelijken, omdat het goed om kan gaan met ruis, vervormingen, veranderingen in belichting, draaiing, grootteverschillen en complexe foto’s. Ook spelen rekenkracht en tijd geen rol vanwege een voldoende krachtige CPU en de aanwezigheid van genoeg RAM.

(Root)SIFT werkt door als eerst de foto’s te vergroten op verschillende schalen, ze te vervagen en vervolgens de foto’s met elkaar te vergelijken. Het vervagen van de foto’s zorgt ervoor dat het algoritme makkelijker punten van kleine verschillen en details kan detecteren in de foto’s (deze punten heten **keypoints**, en liggen vaak op hoeken en randen). Dit doet het op verschillende schalen zodat het vast kan stellen dat de keypoints niet veranderen met schaal.  
Dit proces heet **Scale-Space Extrema Detection**.

   

[Een visualisatie van de vervaging die plaats vindt bij Scale-Space Extrema Detection op een enkele vergroting.](https://en.wikipedia.org/wiki/Scale_space)

Na het detecteren van verschillende keypoints worden deze nu verfijnd om nauwkeuriger de schaal en locatie te bepalen. Hierbij worden ruis en instabiele keypoints verwijderd.

Om ervoor te zorgen dat (Root)SIFT niet verschillen in draaiing tussen foto’s detecteert wordt ook naar omgeving van elk keypoint gekeken, specifiek hoe het er relatief uit ziet. Hiermee kan de draaiing van elk keypoint worden bepaald.

SIFT maakt nu rondom elk keypoint een beschrijvende vector, ook wel een **descriptor**, die samenvat wat de lokale oriëntaties en intensiteiten in de buurt van een keypoint zijn.

Zodra het algoritme al de verschillende keypoints en descriptoren van 2 afbeeldingen heeft kan het ze met elkaar gaan vergelijken en overeenkomsten herkennen, ook al zijn de foto’s gedraaid, geschaald en/of zijn er verschillen in belichting.

RootSIFT is een variant van SIFT dat beter is in accurate resultaten te verkrijgen. Waar het varieert van zijn originele versie is dat eerst de descriptor L1-genormaliseerd wordt. Dit betekent dat het de som van alle onderdelen van de descriptor gelijk stelt aan 1, waardoor grote verschillen in de descriptor worden verminderd. Vervolgens neemt het algoritme de wortel ervan (vandaar de Root- in RootSIFT) om de Hellinger-afstand te gebruiken, in plaats van de standaard Euclidische afstand (een rechte lijn). Dit verbetert resultaten vooral als er grote variaties in belichting of ruis aanwezig zijn.

De Hellinger-afstand is een maat om de verschillen of overeenkomsten tussen 2 afbeeldingen te meten. Het heeft altijd een waarde tussen 0 en 1, waarbij een waarde van 0 indiceert dat de afbeeldingen perfect identiek zijn, en een waarde van 1 indiceert dat ze volledig verschillend zijn met geen overeenkomsten. De reden dat de Hellinger-afstand hier wordt gebruikt is omdat het gevoeliger is voor kleine verschillen in de foto’s en het beter in staat is om variaties in intensiteit, zoals ruis en belichtingsverschillen, eruit te filteren.

**Het complete programma.**

rootsift.py

# kps = keypoints

# descs = descriptors

import numpy

import cv2

class RootSIFT:

    def \_\_init\_\_(self):

        self.extractor = cv2.SIFT\_create()

    def compute(self, image, kps, eps=1e-7):

        (kps, descs) = self.extractor.compute(image, kps)

        if len(kps) == 0:

            return ([], None)

        descs /= (descs.sum(axis=1, keepdims=True) + eps)

        descs = numpy.sqrt(descs)

        return (kps, descs)

class RootSIFT definieert hoe RootSIFT de berekeningen voor descriptoren uit moet voeren.  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

driver.py

# kps = keypoints

# descs = descriptors

import cv2

import os

import concurrent.futures

from rootsift import RootSIFT

from heapq import nsmallest

from time import sleep

rs = RootSIFT()

def extract\_rootSIFT\_features(image\_path):

    image = cv2.imread(image\_path)

    if image is None:

        raise FileNotFoundError(f"Image not found or unable to load at path: {image\_path}")

    gray = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

    detector = cv2.SIFT\_create()

    kps = detector.detect(gray)

    (kps, descs) = rs.compute(gray, kps)

    return (kps, descs)

def match\_image(image\_path, descs\_input):

    try:

        (kps\_dataset, descs\_dataset) = extract\_rootSIFT\_features(image\_path)

        bf = cv2.BFMatcher()

        matches = bf.match(descs\_input, descs\_dataset)

        match\_score = sum([m.distance for m in matches])

        if descs\_dataset is None:

            return (image\_path, float('inf'))

        return (image\_path, match\_score)

    except Exception as e:

        print(f"Error processing {image\_path}: {e}")

        return (image\_path, float('inf'))

def get\_matches(input\_image\_path, dataset\_directory):

    print(f"Processing input image: {input\_image\_path}")

    (kps\_input, descs\_input) = extract\_rootSIFT\_features(input\_image\_path)

    dataset\_images = [os.path.join(dataset\_directory, img) for img in os.listdir(dataset\_directory)]

    print(f"Found {len(dataset\_images)} images in dataset directory: {dataset\_directory}")

    for img\_path in dataset\_images:

        print(f"Processing dataset image: {img\_path}")

    with concurrent.futures.ThreadPoolExecutor() as executor:

        futures = [executor.submit(match\_image, img\_path, descs\_input) for img\_path in dataset\_images]

        results = [f.result() for f in concurrent.futures.as\_completed(futures)]

    matches = nsmallest(26, results, key=lambda x: x[1])

    return matches

input\_image = r"Insert directory here" # Needs to include file name, for example "C:\...\picture.jpg".

dataset\_directory = r"Insert directory here" # Only requires the directory to a folder, for example "C:\...\picture\_database".

top\_matches = get\_matches(input\_image, dataset\_directory)

print("Matches ranked:")

for i, (image\_path, score) in enumerate(top\_matches):

    image\_name = os.path.basename(image\_path)

    print(f"{i+1}. {image\_name}, score: {score}")

sleep(10 \* 60)

Bij rs = RootSIFT() wordt de RootSIFT klasse uit rootsift.py gehaald.

Onder def extract\_rootSIFT\_features(image\_path): wordt er een afbeelding ingelezen en omgezet in grijswaarden, waarna SIFT de keypoints eruit haalt en RootSIFT de descriptoren eruit haalt.

def match\_image(image\_path, descs\_input): heeft als doel om de descriptoren van een afbeelding in de database te vergelijken met de descriptoren van de te vergelijken foto. Hier worden ook de waardes van de foto’s berekend, wat de som is van alle afstanden.

Het stukje def get\_matches(input\_image\_path, dataset\_directory): zorgt ervoor dat het alle waardes die het als uitkomst krijgt in een lijst zet van laagste score naar hoogste score. Een lagere score betekent hier namelijk meer overeenkomsten. Hier wordt ook de functie ThreadPoolExecutor geroepen zodat het Python script de hele CPU tegelijk kan gebruiken in tegenstelling tot een klein stukje zoals standaard het geval is bij Python. Dit zorgt ervoor dat de berekeningen een stuk sneller klaar zijn.

Bij input\_image en dataset\_directory worden de locaties van de onbekende foto en de database gespecificeerd. In dit geval moet het nog ingevoerd worden.

Alles vanaf top\_matches = get\_matches(input\_image, dataset\_directory) en daaronder is om de resultaten te tonen en zichtbaar te houden.

**Bronnen.**

Informatie:  
<https://pyimagesearch.com/2014/09/15/python-compare-two-images/>  
<https://pyimagesearch.com/2015/04/13/implementing-rootsift-in-python-and-opencv/>

Foto’s:  
<https://en.wikipedia.org/wiki/File:Scalespace0.png>  
<https://en.wikipedia.org/wiki/File:Scalespace1.png>  
<https://en.wikipedia.org/wiki/File:Scalespace2.png>  
<https://en.wikipedia.org/wiki/File:Scalespace3.png>