University of Studies of Enna “Kore”

Engineering and Architecture Department

Degree course in Artificial Intelligence and Cyber Security Engineering

Exam’s Disertation

Biometric Authentication for IT Systems Security Project

***Students*:**
Stefano Incardone, Giuseppe Capizzi

***Professor*:**
Prof. Vincenzo Conti

ACADEMIC YEAR 2024 - 2025

INDEX

I. Introduction ***1***

I.1. Project structure 1

II. Project ***3***

II.1. Configuration parameters 3

II.2. Features extraction and matching 5

II.2.1. Features extraction 5

II.2.2. Matching 6

II.3. Enrollment and database creation 7

II.4. Identification 8

II.5. Verification 8

II.6. Performance evaluation 9

References ***10***

I. Introduction

The aim of the project is to develop an automatic multi-instance recognition system based on fingerprints, providing a quantitative estimate of the accuracy performance of the FVC2006 dataset.

Fingerprint recognition is the study of fingerprints to uniquely identify a user. The approach used in this project is based on a recognition that starts from the analysis of the ridges and valleys of a fingerprint, extracting the relative *minutiae*.

The goal of this approach is to recognize the set of minutiae that are common to both fingerprints considered, to determine if the fingerprint to be identified corresponds with the one stored in the system's databases, therefore, it is a point matching, which is complicated due to low-quality images or distortions related to the angles of the fingerprints during the registration stage.

I.1. Project structure

- **Features extraction**:
  - `config.py`: configuration parameters used during features extraction.
  - `fingerprint.py`: algorithms and structures used to extract features.
  - `references/`: papers and documents used as references for the features extraction process.

- **Enrollment and database creation**:
  - `enrollment.py`: features extraction of all the fingerprints in the folder specified with the "`DATASET_DIR_PATH`" configuration key, to create the database folder specified with the "`DATABASE_DIR_PATH`" configuration key.
  - Used datasets FVC2006, the naming convention followed with the filenames is the following:
    - *database full tag*: "`DATASET_DIR_PATH/FVC2006/db1_b/101_1`".
    - *full tag*: "`FVC2006/db1_b/101_1`".
    - *database tag*: "`FVC2006/db1_b/`".
    - *finger tag*: "`101`".
    - *acquisition tag*: "`1`".
    - *full finger tag*: "`FVC2006/db1_b/101`".
    - *fingerprint tag*: "`101_1`".

- **Identification and Verification**:
  - `identification.py`, identification module: gets the declared identity as input in order to identify it and returns the identity it represents;
  - `verification.py`, verification module: gets the declared identity as input and an expected identity to verify if the declared identity is genuine or an impostor.

- **Performance evaluation**:
  - `performance_evaluation.py`: evaluates the performance of the matching algorithm in order to measure it's **FAR** and **FRR** characteristics.

- **Interactive visualization**:
  - `interactive_visualization.ipynb`: interactive notebook to visualize in real time the features extraction and matching process.

II. Project

II.1. Configuration parameters

The `config.py` file stores configuration keys used to configure the behaviour of the different modules, providing the following keys:

- **Enrollment**:
  - `DATASET_DIR_PATH`: Path to the folder containing the fingerprint dataset.
  - `FINGERPRINTS_IMAGE_FILE_EXTENSION`: Fingerprint image file extension, every file not matching this extension will be ignored.
  - `DATABASE_DIR_PATH`: Path to the folder containing the registered fingerprint database.

- **Feature Extraction**:
  - `GRADIENT_SOBEL_FILTER_LENGTH`: Length of the Sobel filter for calculating the gradient, used to highlight the crests of the fingerprint.
  - `GRADIENT_MODULE_BLOCK_LENGTH`: Size of blocks used to calculate the gradient modulus.
  - `SEGMENTATION_MASK_THRESHOLD_SCALE`: Segmentation threshold to distinguish the useful area of the fingerprint from the background.
  - `DIRECTIONAL_MAP_BLOCK_LENGTH`: Size of blocks used to calculate the directional map of the ridges.
  - `DIRECTIONAL_MAP_BLUR_FILTER_LENGTH`: Length of the blur filter applied to the directional map (if `-1`, no filter is applied).
  - `LOCAL_RIDGE_BLOCK_ROWS, LOCAL_RIDGE_BLOCK_COLUMNS`: Size of blocks used for local ridge analysis.
  - `GABOR_FILTERS_COUNT`: Number of Gabor filters used for ridge enhancement.
  - `GABOR_FILTERS_SIGMA, GABOR_FILTERS_GAMMA`: Gabor filter parameters for improving contrast and ridge definition.
  - `BINARIZATION_BLOCK_SIZE`: Block size used in fingerprint binarization.
  - `SINGULARITIES_MIN_DISTANCE_FROM_BORDER`: Minimum distance from the edge to detect singular points.
  - `MINUTIAE_MIN_DISTANCE_FROM_BORDER`: Minimum distance from the edge to detect minutiae.

- `MINUTIAE_FOLLOWED_LENGTH_MIN`, `MINUTIAE_FOLLOWED_LENGTH_MAX`: Minimum and maximum length followed for the detection of minutiae.
- `MCC_RADIUS`, `MCC_DENSITY`, `MCC_GAUSSIAN_STD`, `MCC_SIGMOID_TAU`, `MCC_SIGMOID_MU`: Parameters used in the MCC (Minutia Cylinder-Code) representation for encoding and comparing minutiae.

- **Matching**:
  - `MATCHING_SCORE_GENUINE_THRESHOLD`: Matching score threshold to distinguish genuine from impostor matches.
  - `MATCHING_ALGORITHM`: Matching algorithm used (Local Structures, Hough (Ratha) or Hough (Chouta)).
  - `LOCAL_STRUCTURES_MATCHING_PAIR_COUNT`: Number of minutiae pairs considered in matching based on local structures.
  - `HOUGH_MATCHING_PIXELS_DISTANCE_THRESHOLD`: Distance threshold in pixels for Hough transform based matching.
  - `HOUGH_MATCHING_ANGLE_DISTANCE_THRESHOLD`: Angular distance threshold for matching.
  - `HOUGH_RATHA_MATCHING_ALIGNMENT_ANGLE_FREEDOM`: Angular freedom allowed for alignment in the Hough (Ratha) algorithm.
  - `HOUGH_RATHA_MATCHING_ALIGNMENT_SCALE_FREEDOM`: Freedom in the alignment scale for Hough (Ratha) algorithm.
  - `HOUGH_CHOUTA_MATCHING_ERR_FREEDOM`: Error tolerance for alignment in the Hough (Chouta) algorithm.

II.2. Features extraction and matching

The feature extraction and matching module are implemented in the `fingerprint.py` script, which, based on the configuration keys previously described, will extract all the relevant features for recognition purposes, using the appropriate classes and methods, to carry out the matching.

II.2.1. Features extraction

The feature extraction process begins within the `Fingerprint` class, which reads, at the specified path, the image relating to the desired fingerprint in greyscale, improving and analyzing it through a sequence of steps.

- **Preprocessing**: a normalization step is applied to the pixels, resizing the image to a fixed size of 255x255 pixels, and applying a scaling of the pixel values to ensure that they have values in the intensity range between 0 and 255.
- **Gradients' calculation**: by applying a differential operator such as the Sobel filter we can extract the gradients related to the crests and valleys to recognize the points where the image pixels pass from a crest to a valley. The gradients will be later be used to determine the direction of the crests and valleys.
- **Segmentation Map**: by calculating the integral of the gradients and applying a threshold to this integral it is possible to extract the areas of the image that belong to the crests and valleys relating to the fingerprint, generating a segmentation map that isolates the region of interest (ROI) of the fingerprint, namely the region that contains the fingerprint.
- **Estimation of orientations**: through the gradients the directions of the valleys and crests are estimated, forming a directional map relating to the angles that are formed with respect to the horizontal axis in the interval $[0, \pi]$ radians. A smoothing pass can be then applied to the directional map to improve its quality.
- **Estimation of frequencies**: by analyzing a portion of the fingerprint it is possible to extract the average frequency of the crests and valleys to determine the parameters that will configure the characteristics of the Gabor filters.
- **Gabor filters' application**: taking into account the frequencies of the crests, Gabor filters are generated for different angles, configurable using the "`GABOR_FILTERS_COUNT`" configuration key. Once Gabor filters are applied, taking into account the directions of the ridges, the image will be filtered, producing an improved representation with very clear and precise details.
- **Binarization and Thinning**: binarization is a process that consists of transforming the image with grayscale values between 0 and 255 into a black and white binary image, therefore with values 0 or 1. This is followed by a process of

skeletonization, or thinning, which consists in thinning the previously binarized ridges to obtain an image where each ridge is exactly one pixel wide.

- **Singularity points**: the singularity points are characteristic points of each fingerprint which are extracted through an iterative process based on the evaluation of the orientations present within the directional map in order to recognize patterns such as *cores*, *deltas* and *whorls*, through the application of metrics such as that of the *Poincaré index*.
- **Extraction of minutiae**: the minutiae points are the points most used for the purpose of fingerprint recognition, as they describe the characteristics of the ridges and are classified into *terminations*, the points where a ridge ends, and *bifurcations*, the points where a ridge splits following two separate roads. The method used to recognize these minutiae is based on the *crossing number* recognition method, according to which a termination is recognized if a lit pixel of a crest is close to only another lit pixel, while a bifurcation is recognized if a lit pixel of a crest is close to only three other lit pixels.
- **Generation of Local Structures**: MCC representation associates a local structure to each minutia. This structure encodes spatial and directional relationships between the minutia and its fixed-radius neighbourhood and can be conveniently represented as a cylinder whose base and height are related to the spatial and directional information, respectively.

II.2.2. Matching

The matching module is implemented within the `fingerprint.py` file using different matching algorithms specified using the "`MATCHING_ALGORITHM`" configuration key which can select one of the following algorithms:

- **Local structures**: the algorithm, implemented in the "`matching_score_local_structures()`" method, returns the matching score by using the Euclidean norm to calculate the distance between the local structures of the two fingerprints; the indices of the pairs of minutiae with minimum distance are then selected and the matching score will be calculated as the complement of the average of the distances.
- **Hough Transforms**: the algorithm returns the matching score by calculating the alignment parameters between two sets of minutiae which will minimize the distances between them:
  - **Ratha's algorithm**: the algorithm that calculates the alignment parameters according to the algorithm developed by Ratha et al. is implemented in the "`hough_alignment_parameters_ratha()`" function.

    The algorithm considers each pair of minutiae to find the three parameters of translation $(\Delta x, \Delta y)$, rotation $(\Delta \phi)$ and scale (s) that will align the two sets of

minutiae. These parameters will be extracted based on tolerances that control how precise the alignment parameters will be.

- **Chouta's algorithm**: the algorithm that calculates the alignment parameters according to the algorithm developed by Chouta et al. is implemented in the "`hough_alignment_parameters_chouta()`" function.

  The algorithm considers each pair of minutiae to find the two parameters of translation $(\Delta x, \Delta y)$ and rotation $(\Delta \phi)$ that will align the two sets of minutiae. The exact parameters will be extracted and corrected according to a tolerance to reduce alignment errors.

After calculating the alignment parameters, these will be applied to the set of original minutiae to align them with those of the template, using the "`align_minutiae_hough()`" function, and subsequently the aligned minutiae will be compared in such a way as to extract those that have the minimum distance and angle difference between them, in the "`match_minutiae()`" function, and the matching score *m* is calculated as:

$$m = \frac{M}{L}, \qquad M \leq L$$

Where M represents the number of minutiae sufficiently close and aligned with each other, and L represents the minimum between the quantity of minutiae belonging to the declared identity and the quantity of minutiae belonging to the identity of the comparison template.

II.3. Enrollment and database creation

The enrollment phase is part of the database module, implemented in the `enrollment.py` script, which is responsible for storing the previously extracted features in a database to be able to retrieve them during the later stages. A single database entry is made of some information about the user, which in our case is the fingerprint acquisition tag, and the features relative to that fingerprint acquisition.

The process starts by collecting all the file paths contained in the folder specified with the "`DATASET_DIR_PATH`" configuration key that match the expected file kind identified with the extension specified with the "`FINGERPRINTS_IMAGE_FILE_EXTENSION`" configuration key. During this stage every file with a different extension will be ignored and every file not matching the *fingerprint tag* structure will result in the termination of the program.

After collecting all valid file paths, each corresponding image will be processed to extract its features and saved in a database as feature templates, which will be later saved to disk in the folder specified with the "`DATABASE_DIR_PATH`" configuration key. Each file represents a set of fingerprints acquisitions relating to a single individual and is stored as a data structure that allows access and manipulation.

II.4. Identification

The identification module, implemented in the `identification.py` script, is responsible for identifying an *unknown identity* using a one-to-many comparison strategy against every other identity stored in the database to determine if said identity has been previously enrolled or not.

The script starts by validating the command line arguments, and if none are presented or erroneous ones are encountered it will display helpful messages.

The script then looks for the fingerprint image with the *unknown identity* tag specified at the command line in the "`DATASET_DIR_PATH`" directory and will fail execution if it cannot find it. It will then extract the features as if a fingerprint has been just registered by a fingerprint sensor, and the fingerprint features will then be compared to the ones stored in the previously enrolled fingerprints stored in the "`DATABASE_DIR_PATH`".

The identification process starts by comparing the *unknown fingerprint's* features to every template in the database to get the corresponding matching score, skipping the template corresponding with the *unknown identity* to emulate real world usage scenarios.

At the end of the matching process of each database the average matching score is used to determine if the *unknown identity* matches the database identity base on the "`MATCHING_SCORE_GENUINE_THRESHOLD`" configuration key, and once every comparison has been done it will display the identification result. The possible results are that the *unknown fingerprint* matches or not the stored identities, and in the case of a failed identification the most likely identity will be returned, meaning the one that resulted in the highest matching score out of all the non-matching identities.

II.5. Verification

The verification module, implemented in the `verification.py` script, is responsible for verifying the *declared identity* using a one-to-one comparison strategy against the provided *expected identity* to determine if the *declared identity* matches or not the *expected identity*.

The script starts by validating the command line arguments, and if none are presented or erroneous ones are encountered it will display helpful messages.

As in the case of the identification process the *declared identity* is compared to the ones stored in the database, with the difference that the matching score is calculated based only on the comparison with the same declared identity fingerprint acquisitions, producing an average matching score that will determine if the fingerprint matches or not the *expected identity*.

II.6. Performance evaluation

To ensure the reliability of the fingerprint-based biometric recognition system, a *performance evaluation* procedure was implemented in the "`performance_evaluation.py`" script. This phase aims to measure the accuracy of the system through the calculation of fundamental metrics such as the **False Acceptance Rate (FAR)** and the **False Rejection Rate (FRR)**.

It starts by getting the fingerprints' data stored in the database directory, which is scanned recursively, identifying files with the correct extension ("`.npy`").

To evaluate performance, the system carries out two types of tests:

1. **Matching between fingerprints of the same identity** (Intra-Class Matching):
   - For each fingerprint in the database, a comparison is made with all other fingerprints of the same identity.
   - If the matching score exceeds the predefined threshold, the system considers the comparison a **True Positive (TP)**, otherwise it classifies it as a **False Negative (FN)**.
2. **Matching between fingerprints of different identities** (Inter-Class Matching):
   - The fingerprint is compared with those of other identities.
   - If the matching score exceeds the predefined threshold, a **False Positive (FP)** is recorded, indicating that the system wrongly accepted a mismatch, otherwise, a **True Negative (TN)** is recorded.

At the end of the tests, the system calculates two key metrics:

- **False Acceptance Rate (FAR)**: measures the percentage of fingerprints belonging to different identities that were incorrectly accepted as matching, following this formula:

$$FAR = \frac{FP}{FP + TN} \times 100$$

- **False Rejection Rate (FRR)**: measures the percentage of fingerprints from the same identity that were erroneously rejected, following this formula:

$$FRR = \frac{FN}{FN + TP} \times 100$$

These values are fundamental to evaluate the balance between security and usability of the system. A high FAR implies a risk of unauthorized access, while a high FRR reduces the convenience of use as legitimate users may be rejected.

References

[1] F. Magalhães, H. P. Oliveira and A. C. Campilho, "A new method for the detection of singular points in fingerprint images," *2009 Workshop on Applications of Computer Vision (WACV)*, Snowbird, UT, USA, 2009, pp. 1-6, https://doi.org/10.1109/WACV.2009.5403106.

[2] N. K. Ratha, K. Karu, Shaoyun Chen and A. K. Jain, "A real-time matching system for large fingerprint databases," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 8, pp. 799-813, Aug. 1996: https://doi.org/10.1109/34.531800.

[3] Mlambo, C. S. (2015). *A study on Hough transform-based fingerprint alignment algorithms*, University of Johannesburg. https://ujcontent.uj.ac.za/esploro/outputs/doctoral/A-study-on-Hough-transform-based-fingerprint/9910732607691#file-0.

[4] Lin Hong, Yifei Wan and A. Jain, "Fingerprint image enhancement: algorithm and performance evaluation" in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 8, pp. 777-789, Aug. 1998, https://doi.org/10.1109/34.709565.

[5] R. Cappelli, M. Ferrara and D. Maltoni, "Minutia Cylinder-Code: A New Representation and Matching Technique for Fingerprint Recognition," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 12, pp. 2128-2141, Dec. 2010, https://doi.org/10.1109/TPAMI.2010.52.