University of Arkansas, Fayetteville

ScholarWorks@UARK

Computer Science and Computer Engineering Undergraduate Honors Theses

Computer Science and Computer Engineering

12-2018

Fingerprint Database Privacy Guard: an Open-source System that Secures Fingerprints with Locality Sensitive Hashing Algorithms

Enrique Sanchez
University of Arkansas, Fayetteville

Follow this and additional works at: https://scholarworks.uark.edu/csceuht

 Part of the Computational Engineering Commons

Citation

Sanchez, E. (2018). Fingerprint Database Privacy Guard: an Open-source System that Secures Fingerprints with Locality Sensitive Hashing Algorithms. *Computer Science and Computer Engineering Undergraduate Honors Theses* Retrieved from https://scholarworks.uark.edu/csceuht/63

This Thesis is brought to you for free and open access by the Computer Science and Computer Engineering at ScholarWorks@UARK. It has been accepted for inclusion in Computer Science and Computer Engineering Undergraduate Honors Theses by an authorized administrator of ScholarWorks@UARK. For more information, please contact scholar@uark.edu, uarepos@uark.edu.

Fingerprint Database Privacy Guard: an Open-source System that Secures Fingerprints with Locality Sensitive Hashing Algorithms

Fingerprint Database Privacy Guard: an Open-source System that Secures Fingerprints with Locality Sensitive Hashing Algorithms

An Undergraduate Honors College Thesis

of the

Department of Computer Science and Computer Engineering
College of Engineering
University of Arkansas
Fayetteville, AR

By

Enrique J. S. Headley

December 2018

University of Arkansas

ABSTRACT

Fingerprint identification is one of the most accurate sources of identification, yet it is not widely used in public facilities for security concerns. Moreover, the cost of fingerprint system is inaccessible for small-budget business because of their high cost. Therefore, this study created an open-source solution to secure fingerprint samples in the database while using low-cost hardware components. Locality Sensitive Hashing Algorithms such as ORB and Image hash were compared in this study as a potential alternative to SURF. To test the design, fifteen samples were collected and stored in a database without verifying the quality of the samples. Then, thirteen other samples were read from the sensor and forty-five permutations were created from the first fifteen samples. The results showed that a low-cost system can secure fingerprint sample in a database using Open-source technologies, but the identification process needs some improvement. Also, the study showed that image hash is a good alternative to SURF when the sensors readings are a force to one position.

ACKNOWLEDGEMENTS

I would like to thank Dr. Pat Parkerson for his continued support and the opportunity to start doing research since I was a sophomore. Also, I would like to thank Dr. Alexander Nelson for his feedback and guidance throughout this research. Not only their encouragement but also their laboratory equipment provided a clear path that made possible the completion of this project. Furthermore, I would like to thank Dr. John M. Gauch for his technical feedback and recommendations about image processing and feature extractors.

Moreover, I would like to thank my friends and laboratory partners who provided technical, writing, and presentation feedback. Their point of view and their peer review helped me make the documentation easier to understand. Additionally, I would like to thank my family for their love and support, which helps me to stay motivated. Finally, I would like to thank the Honor College for providing the grant that made this project possible.

TABLE OF CONTENTS

ABSTRACT iii
ACKNOWLEDGEMENTS iv
LIST OF TABLES 1
LIST OF FIGURES 2
1. INTRODUCTION 3
1.1 Problem 3
1.1 Thesis Statement 4
1.2 Approach 4
2. BACKGROUND 5
2.1 Key Concepts 5
2.1.1 Image Hashing 5
2.1.2 Key points Detection and Description Algorithms 6
2.2 Related Work 7
3. ARCHITECTURE 8
3.1 Design 8
3.1.1 High-Level Design – Hardware 8
3.1.2 High Level Design – Software 10
3.2 Implementation 11
3.2.1 Template Pre-processing 11
3.2.2. Feature Extractors 12
3.2.3. Storing Features 14
3.2.4. Matching 15
4. METHODOLOGY, RESULTS, AND ANALYSIS 17
4.1 Methodology 17

4.2 Results 18
4.2.1. Fingerprint Template Reconstruction 18
4.2.2. Matching 19
4.2.3 Performance 21
4.2.4. Fingerprint Quality 23
4.3 Analysis 24
5. CONCLUSIONS 25
5.1 Summary 25
5.2 Future work 25
6. REFERENCES 27
Tables 30
Hardware Sketch Configuration 31

LIST OF TABLES

Table 1 Data set representation with description 30

Table 2 Number of samples skipped by SURF 30

LIST OF FIGURES

Figure 1. High-level software architecture 8

Figure 2. High-level software design 10

Figure 3. template preprocessing stages 11

Figure 4. Fingerprint extraction SURF vs ORB. 12

Figure 5. Permutations of the samples. 17

Figure 6. Maping matching points between samples. 18

Figure 7. Matching performance of ORB. 20

Figure 8. Matching performance of SURF 20

Figure 9. Matching performance of Image Hash. 21

Figure 10. Average position 22

Figure 11. Raspberry Pi 3 B+ Pins schematics 31

Figure 12. Caged Fingerprint 31

1. INTRODUCTION

1.1 Problem

Fingerprints are one of the most reliable sources of identification [9] that has been used for hundreds of years [19], and the improvement in computer efficiency has made fingerprints easier to process [1]. However, storing full fingerprint samples on databases is a risk because if the database gets compromised, all the users' fingerprints on the database will get compromised even if their fingerprints were encrypted [9],[3]. This will cause individuals to lose the ability to be identified using their fingerprints for the rest of their life [3]. Compared with a password, a secret combination of characters, fingerprints are unique and non-resettable, which is the main reason that federal laws prohibit storing fingerprint templates on a public database [9]. Moreover, the compromised database may cause individuals to suffer identification theft that may lead to problems like the thief accessing the individuals' personal and entering others country with their identity. Similarly, the person that comprised the database may use the fingerprint templates to access enterprise data and government facilities.

Furthermore, there are companies that provide a complete fingerprint identification system; however, the technologies used by this company are usually expensive which make their equipment and software solutions inaccessible for low budget companies. For instance, Speeded-Up Robust Features (SURF) algorithm is patented and require a commercial license to be used which cause an increase in yearly fixed cost for a company that will like to use one of the top technologies to secure fingerprint on databases. Similarly, the cost of state-of-the-art fingerprint scanners may cost a few hundreds of dollars which increase the front cost of implementing a fingerprint system beside the yearly cost which makes this security measure not appealing for low-profit companies. Thus, it is important to have a low-cost system that can be reliable and

efficient. One of the biggest problems of not having a solution for these problems are being faced by government public services such as public hospitals and community clinics. These medical facilities have a history of applying medical procedures to the wrong patient [11] that result in 200,000 to 400,000 patients life lost every year [21].

1.1 Thesis Statement

A low-cost fingerprint system with accurate recognition rate and secure database can be achieved while using open source image processing techniques and Internet of Things (IoT) technologies.

1.2 Approach

The proposed solution in this paper is to develop a system that uses open source image processing techniques to extract features from a fingerprint — storing only the extracted features on the database. If the database gets compromised, features stored on the database will not be enough to recreate the full fingerprint template. Thus, the system database can be reconstructed if compromised, and users will not lose the opportunity to be identified using their fingerprints. Similarly, image hashes were also studied as an alternative to feature extractor algorithm to compare performance and reliability.

To eliminate the cost of paying for patented technologies, this work leverages free technologies such as Oriented FAST and Rotated BRIEF (ORB), and it was compared against state-of-the-art technologies like SURF in the process of extracting features and matching fingerprint templates. Also, to reduce the cost of the hardware, a Raspberry Pi B+ will be used to collected templates from an inexpensive optical scanner. In addition, the Raspberry Pi will be used to process the template, extract features, store, and match features in the same system.

Doing this, the need for an external server and an expensive computer to process the information was removed.

2. BACKGROUND

2.1 Key Concepts

For storing string-based passwords on databases, the standards are using a cryptographic hash algorithm such as SHA-4. This type of algorithm is bits sensitive based [5] which will be affected by any variation on color, position or contrast of the sample read from the sensor. Therefore, trying to use this method to secure images is impossible since the small variation in the image will change the entire hash value. The variation in bits from the fingerprint scanner readings will make it impossible to properly identify the proper finger because cryptographic hash algorithm will return a completely different hash. For this reason, there are some hashing algorithms that are contents based, usually referred to as Locality-sensitive hashing (LSH), which help matching images and locality sensitive data. Thus, using LSH algorithms will help secure fingerprints in the database while keeping the matching capabilities.

2.1.1 Image Hashing

Image hashing is a content-aware hashing algorithm that has been used to identify repeated multimedia in databases, retrieval of image content-based data, and image indexing using a set of string [6]. Image hashing has a similar property to the cryptographic hashing which is the algorithm make it hard to transform the hash value back to the original data. There are six versions of this algorithm hashing (a-hash), perception hashing (p-hash), perception hashing simple (p-hash_simple), difference hashing (d-hash), difference hashing horizontal (d-hash_h), and Wavelet hashing (w-hash). The two most promising type for this application is p-hash and wash because both of them used an advance mathematical operation that will help maintain

consistency in similar images. The p-hash will check each pixel and [4] take the discrete cosine transformation (DCT) and will output true if the pixel is greater or equal to the average pixel value. The main reason a-hash does not show potential to detect small changes in the fingerprint template compared to p-hash is that a-hash is missing the DCT which is "efficient when image blocks contain arbitrarily shaped discontinuities" [7]. The w-hash functions similar to p-hash, but instead of using DCT it will use discrete wavelet transform (DWT) which evaluates un-stationary and stationary signals to accurately identify the local disruption of the signals [5].

2.1.2 Key points Detection and Description Algorithms

Unlike image hashing, key point and descriptors detector algorithm can detect the same image with a different size or rotation [1] [13]. There are several algorithms on MATLAB and OpenCV that can perform feature detections such as SIFT [15], BRIEF [16], DAISY [16], and BRISK [18]. However, for this application, extracting features from fingerprints, the most commonly used are Speeded-Up Robust Features (SURF) [1] and Oriented FAST and Rotated BRIEF (ORB) [13] because of their speed and robustness.

SURF, which is a fast version of SIFT, is a feature and descriptor extractor that is tolerant to change is rotation and escalation in images which show a good performance with respect of "repeatability, distinctiveness, robustness, yet computed and compare much faster" [1]. SURF uses hessian detectors to analyze points of interest that will later be evaluated by a Hessian Matrix that will extract the strongest key points.

ORB is a combination of Binary Robust Independent Elementary Features (BRIEF) and Features from Accelerated Segment Test (FAST), but with the ability to be rotation resistance which is not present in either of those algorithms. This combination makes ORB able to extract features by measuring the binary difference between each pixel and identifying the intensity of

the center pixel compared to the other pixel around its radius. All this is done simultaneously with the calculation of the corners' intensity using the Intensity Centroid equation that will give a vector that will represent the orientation of the sample.

2.2 Related Work

The authors in [1] perform an experiment that demonstrates that fingerprint identification using local robust features extraction and matching — such as SURF — is faster and more reliable than other technologies present at the time. Their experiment shows 88.3% rate success while identifying fingerprint even for rotated, scaled, and blurred samples. However, this solution only improves fingerprint-matching performance and reliable but does not show how to improve image-storing security. This paper will prove that using implementations similar to [1], it is possible to store key points, image hash and descriptor instead of storing fingerprint templates. The reason this will improve the security of fingerprint is that most of the technology used in this paper has multiple variables that can be changed such as thresholds, matching parameters, number of key points, and edit pre-processing image techniques. This will make it harder for hackers to try to break the system since there are plenty of unknown parameters, and also it will provide the user the ability to still be able to use their fingerprint if the system is compromised.

The work performed in [2] is more similar to the achievements in this work. The authors show the benefits on SURF and how to store the key points' information using a Locality-Sensitive hashing (LSH). Nonetheless, the solution in [2] is using SURF — which is now patented and needs a license for commercial use — whereas this paper will use ORB, an alternative to SURF that claims to be faster and free. Furthermore, the experiments in [2],[1] used the Fingerprint Verification Competition (FVC) databases [12] which is publically available

with fingerprint samples. In contrast, this paper will create a complete system that will collect, process, store, and match the fingerprints. Additionally, the FCV databases are populated fingerprint images with a resolution of 500 dpi compared to 300 dpi from the sensor used in this paper. Similarly, the size image, 388x374, from the FCV database is larger than the image size, 256x288, obtained from the sensor used in this paper. Therefore, the FCV database has samples that cover the entire fingerprint which results in more minutiae captured in each sample compared with the samples collected with the sensor used in this paper. The fingerprint sensor in this project will record about half of the thump fingerprint which may lead to some drawback in matching rate compared to the solution in [1],[2]. However, having a free alternative to SURF and using a cheap fingerprint scanner may balance the matching rate loss.

3. ARCHITECTURE

3.1 Design

3.1.1 High-Level Design – Hardware

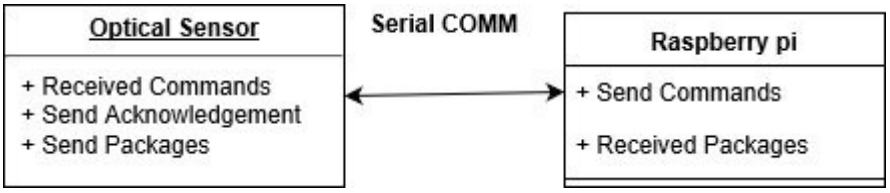

Figure 1. High-level software architecture that represents the functions each component can perform and how they communicate with each other.

The hardware design for this paper is straightforward and it consists of two parts. First, an optical sensor is used to collect the fingerprint template. This system can receive commands through universal asynchronous receiver-transmitter (UART) as well as send packets and acknowledge signals. The size of the packets is 137 bytes where 128 bytes correspond to the fingerprint data. The Raspberry Pi 3 B+, besides performing the image processing, will handle

the communication through UART with the sensor using a library called Pyfingerprint [10]. It is important to mention that the Raspberry Pi model 3 B+ uses the main UART bus, which is faster and stable than the mini UART bus, to handle Bluetooth communication. However, in this application Bluetooth was not required, so the mini UART was set to handle Bluetooth communication while the main UART was set to handle the communication between the Pi and the fingerprint sensor to avoid connection and data loss.

3.1.2 High Level Design – Software

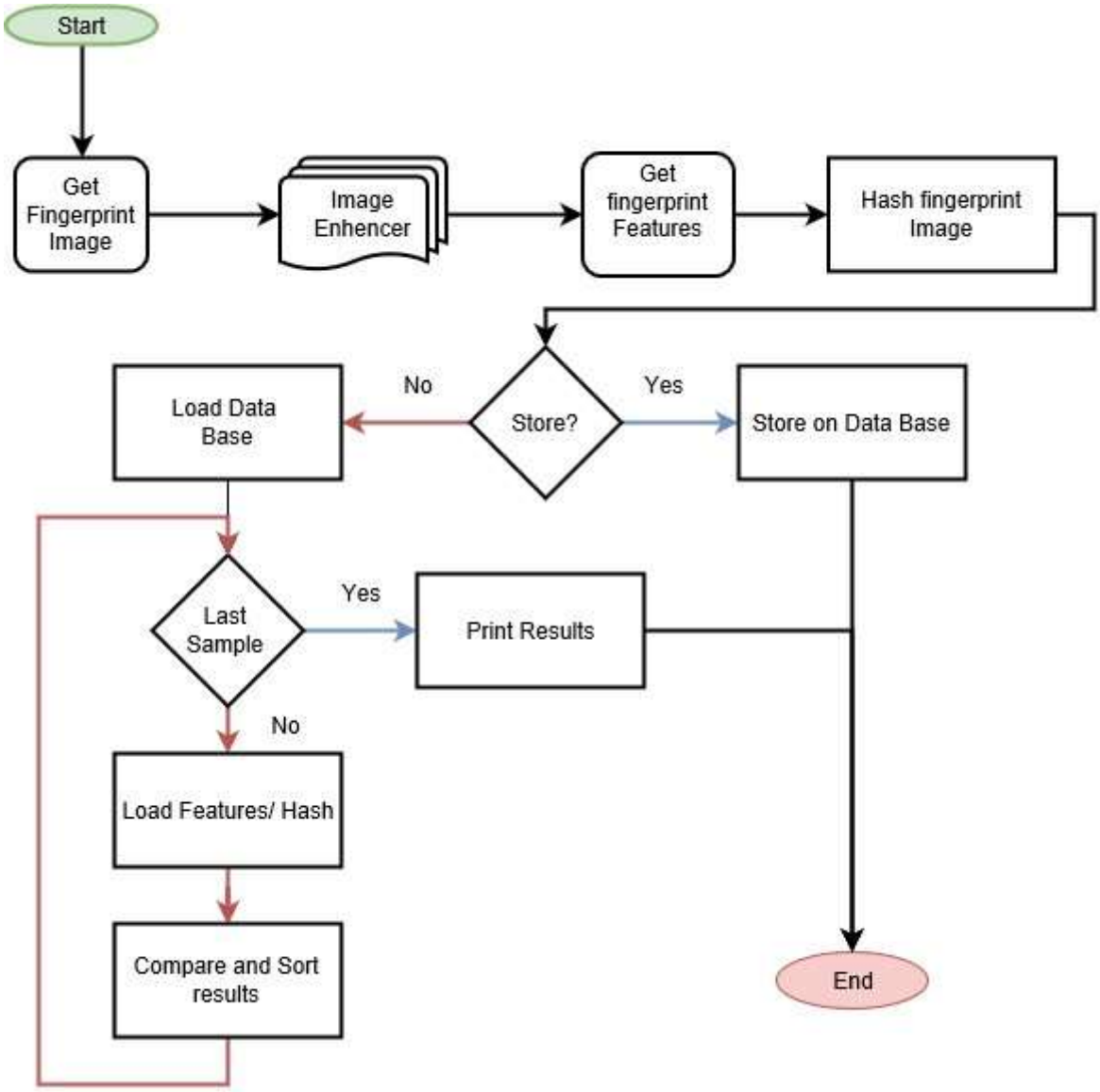

Figure 2. High-level software design that shows the diagram flow how the Secure Fingerprint library will work.

The software design consists of four main parts: getting the fingerprint template, pre-processing the template, extracting features/hashes from a template, and storing or matching features. As mention above, the image collection will be taken care of using the Pyfingerprint library that will return a PIL Image object. Then, the object will be processed to remove any kind

of disruption by assuring image is only black and white using, enhance template contrast, and skeletonize it. After that, the feature extractor will process the image to get all key points, descriptors, and hashes. Finally, if the operation is to store the fingerprint template, the information will be stored on the respective database; otherwise, the fingerprint template features will be compared to each element on the database and a sorted list will be printed with the matching accuracy in percentage. It is important to note that during the entire process the full fingerprint template is maintained in transient memory only and it will be deleted after extracting all the features.

3.2 Implementation

3.2.1 Template Pre-processing

Figure 3. template preprocessing stages represented from left to right where the first image represent the original template and the other represents the stage of making sure is only black and white, improve contrast and transform to a skeletonize image. Fingerprint template extracted from FVC 2004 [12]

Before being able to extract the feature from a fingerprint sample, the image must meet certain requirements that will allow the proper extraction of key features of the fingerprint template. It is important to mention that this process is not required for image hashing because the algorithm has its own pre-processing functions built in. However, for feature extraction, it is important to reduce noise on the fingerprint template and transform the template to a simple black and white image. Figure 3 shows the process the image which are verifying the image is black and white (B&W), increasing contrast, and to skeletonize the image. The B&W

verification and contrast increase reduces noise in the fingerprint template which is crucial before passing the template to the skeletonization function, also known as thinning function. This function generates a skeleton representation of the sample by reducing the number of pixels [20].

3.2.2. Feature Extractors

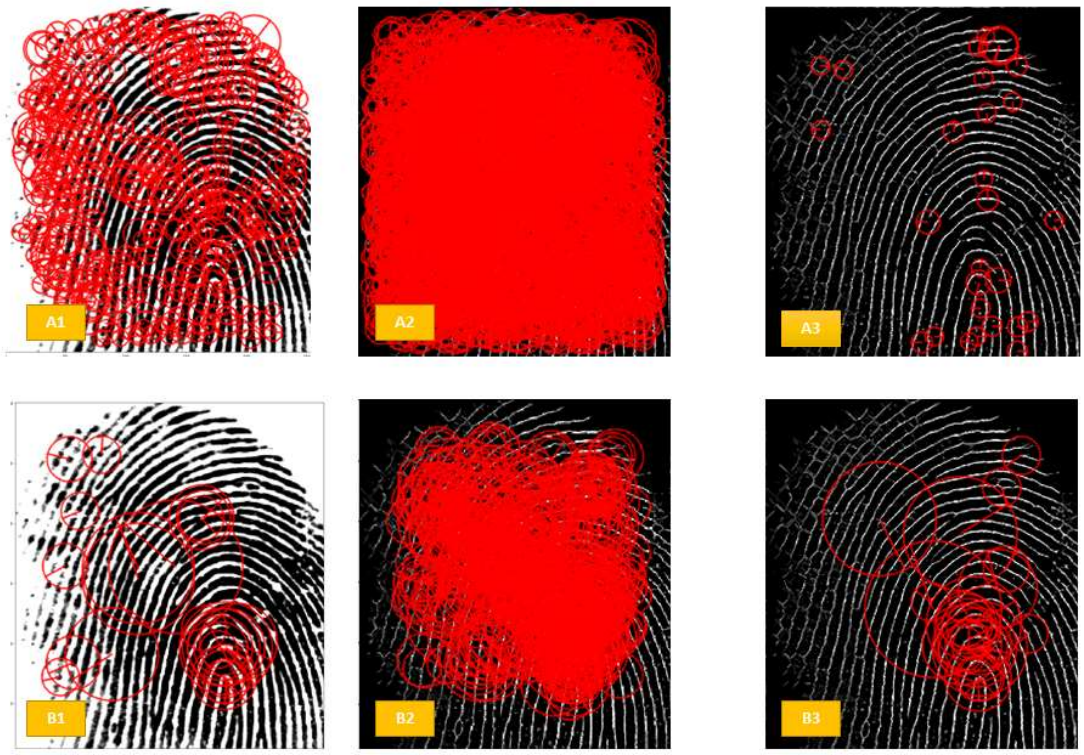

Figure 4. Fingerprint extraction SURF (A) vs ORB (B). Images label (1) are the original images with the same threshold has the one labeled (3). Image label (2) represent the maximum number of key points that can be extracted from the sample.

The number of features extracted depends on the type of image. For instance, a plain B/W image may have thousands of features while the same image normalized may only have a fraction of the features of the original. A quick test was done to make sure the database can be filled with a feature that was relevant for the study. In figure 4, sample A1 and sample B1 are a representation of features extracted without a constrained threshold that will limit the number of features that can be extracted. These samples are irrelevant to the study since the sample is filled with key points instead of identifying only the strong ones. Also, Figure 4 shows the comparison

between an image that has been preprocessed (labeled 3) and the original (labeled 2). Here SURF's original sample has more features than the processed whereas ORB maintains almost the same number of features. However, to maintain a fair comparison, this paper only used the processed sample and a maximum number of the key point was set. Therefore, a threshold for computing SURF features was set to 700 hessian keypoint detector which returned an 'x' numbers of key points, but only the first 80 were stored. The maximum of 80 key points was set based in [2] where 100 features were enough to describe the template; however, the samples in [2] had 20% more DPI and size than the samples used in this paper. Therefore, reducing the maximum number of features by 20% was considered enough for the samples used in this paper. In contrast to SURF, ORB was able to return the strongest 'x' number of features. In this paper x = 30 because ORB features have a longer radius compared to the ones provided by SURF. Also, more than thirty features will cover most of the fingerprint template. Furthermore, another parameter of the ORB object, such as scaleFactor =1.1 instead of 1.2. and WTK = 4, were changed to reduce the size of the features' radius. After identifying the right threshold, the feature and descriptor were extracted using the code below which returned a list of key points and an array of descriptors that represents the strongest point found in the template.

```
    def orb_features(self,img):
        orb = cv2.ORB_create(30,1.1,8,31,0,4) #30,1.2,8,31,4
        #find the key points
        kp_orb = orb.detect(img, None)
        # compute the descriptors with ORB
        kp_orb, desc = orb.compute(img, kp_orb)
        return kp_orb, desc

    def surf_features(self,img):
        surf = cv2.xfeatures2d.SURF_create(self.surf_th) #700
        kp_surf, des = surf.detectAndCompute(img, None)
        nFeatures = 75
        if len(kp_surf) != 0:
```

```
            kp_surf = kp_surf [:nFeatures]
            des = des[:nFeatures]
        return kp_surf,des
```

In contrast to the feature extractors, image hashing does not require any image preprocessing as it was mentioned before. Therefore, the original image will be passed to the function getAllHashing that will return an array of all possible hashing that can be obtained by the template. The hash size was maintained to the default, 4, to assure getting all hashing is perform as quickly as possible without any delays that can affect the system performances.

```
    def getAllHashing(self,img):
        array_hashes = []
        a_hash = imagehash.average_hash(img)
        d_hash = imagehash.dhash(img)
        d_hash_h = imagehash.dhash_vertical(img)
        p_hash = imagehash.phash(img)
        p_hash_simple = imagehash.phash_simple(img)
        w_hash = imagehash.whash(img)
        array_hashes = [a_hash, d_hash, d_hash_h, p_hash, p_hash_simple,
                        w_hash]
        return array_hashes
```

3.2.3. Storing Features

The extracted features consist of a multi-dimensional array which is not a primitive variable and it is not supported on SQL databases by default. However, objects in python can be serializable and a binary representation of an array can be stored in a file [8]. There is a python library called pickle that can serialize any type of python object and it has a good performance in storing and loading data. In this case, all the data are built into one multiple-dimensional array and the resulting array is store in using the function pickle.dump() and loaded using the function pickle.load(). The code below shows the pickle function called `pickle_keypoint` which is the one in charge of packing everything into one array to later be stored in the database. The

store_pickle_append function will take the list returned by the pickle function and append it to the database, and if the database does not exist then the function will create a new database to add the new entry. Also, the code shows the unpickle function which is the one in charge of unpacking the array read from the database and recreate all objects needed. The load_pickle will load all entry of the database and return an array of entries that will later be pass to the unpickle function to recreate the entry' object.

```
def pickle_keypoints(id, name, f_type, keypoints, descriptors):
    i = 0
    toReturn = [id, name, f_type]
    temp_array = []

    for point in keypoints:
        temp = (point.pt, point.size, point.angle, point.response, point.octave,
        point.class_id, descriptors[i])
        print(temp)
        i += 1
        temp_array.append(temp)
    toReturn.append(temp_array)
    return toReturn
def unpickle_keypoints(array):
    id_array = [array[0], array[1], array[2]] #id , name , pf_name
    keypoints = []
    descriptors = []

    for point in array[3]:
        temp_feature = cv2.KeyPoint(x=point[0][0],y=point[0][1],_size=point[1],
_angle=point[2], _response=point[3], _octave=point[4], _class_id=point[5])
        temp_descriptor = point[6]
        keypoints.append(temp_feature)
        descriptors.append(temp_descriptor)
    return id_array, keypoints, np.array(descriptors)
#from example: https://isotope11.com/blog/storing-surf-sift-orb-keypoints-using-
opencv-in-python
```

3.2.4. Matching

To compare each template of the database against the one read from the fingerprint sensor, all the feature and hashes are extracted and the load_picke function is called to get all

elements in the database. Then, the function called brute Matcher that will take the descriptor of the current template and compare it against each one stored in the database. The brute matcher is different for SURF and ORB because of the algorithms used by each one. The ORB matcher was set to detect each match using hamming distance because ORB descriptors are binary based, and the number of descriptors produced by BRIEF will be greater than 2. The reason for producing more than 2 descriptors is that the WTK parameter was set to 4 and it is recommended to use NORM_HAMMING2 FOR WTK values greater than 3. In contrast, SURF matcher was created without any parameter, but the ratio test threshold was set to 0.75 compared to 0.96 of ORB. The reason is that ORB will return the strongest feature by default whereas SURF feature was reduced to 80 without considering if those features were the strongest.

The equation below was used in the paper [1] to compare the matching rate. $Matches_{good}$ are the quantity of good matches that were found by the brute matcher after applying the ratio test. DesDB is the number of descriptors found in the template store in the database, and DesTPL is the number of descriptors found in the template that is being compared. The matching score will represent the percentage of the input template that match the ones in the database, the greater the percentage the better change both templates are equal.

$$Matching\ Score = \sqrt{\frac{(Match\quad good)^2}{(DesDb)(DesTPL)}} \times 100$$

To compare each hash stored in the database against the input template, a simple Hamming distance was calculated. The difference will demonstrate how close the input template is to the ones read from the database. The closer the hamming distance is to zero, the better the odds that the input template is the same as the one in the database.

$$difference = DBhash - TPLHash$$

4. METHODOLOGY, RESULTS, AND ANALYSIS

4.1 Methodology

Figure 5. represent the permutation of the sample stored in the database. A1 is the original sample, A2 is the sample moved down, A2 is the sample blurred, and A4 is the sample with some data loss.

To test the design, three sets of fingerprints were collected from the fingerprint sensor. The first set of samples will be stored in the database. Set 2 are samples collected in the same position as set 1, this will be used to test image hashing algorithm and see how it performs. Set 3 are samples that the subjects were not required to leave their fingers in the same position to get a third reading. Instead, the subjects were able to replace their fingers which will test the sensor ability to replicate similar samples to the ones in set 1. Also, set 3 will test how good the feature extraction algorithm will perform since the fingerprint samples may not have the same position as the samples in set 1. The purpose of the set from 4 to 6 is to create a simplified model to test the algorithm proposed in this paper. The simplified model will remove the variance in readings from the fingerprint sensor. These sets are only permutations of set 1. Figure 5 shows how the permutations of set 1 are represented on the fingerprint. The sample A1 is the original, sample

A2 show the original sample moved down which represent user not placing the finger in the same position. Sample A3 shows the original sample blurred at the top which represents dirt on the sensor, and sample A4 represents the original sample but with some pixel lost from the time data was collected.

Moreover, there was a total of fifteen samples in set 1 which is the one that was stored in the database. However, this paper also used some fingerprint samples from databases in [13] to add some level of difficulty when finding the fingerprints. To do this, the database B2 from [13] was cropped to match the size of the samples collected from the sensor used in this paper. Only the samples that show most of the fingerprint were saved which result in a total of 74 samples. Thus, the number of samples stored in the database was 89 counting the ones collected in set 1.

4.2 Results

4.2.1. Fingerprint Template Reconstruction

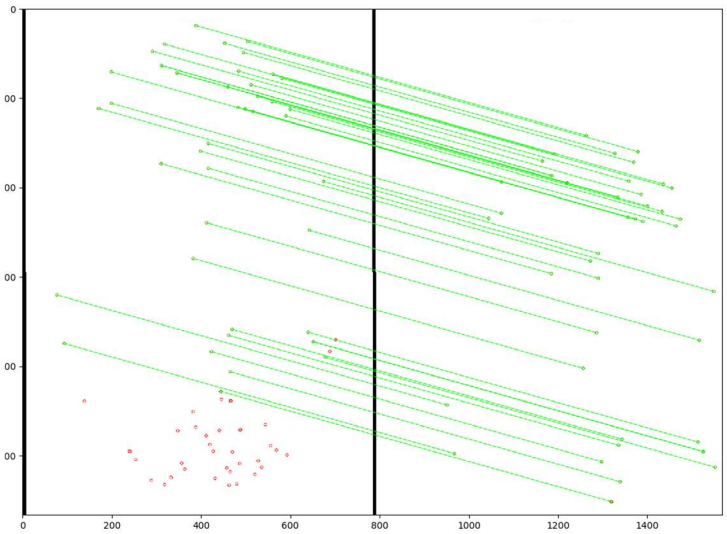

Figure 6. Matching points between samples from set 1 (left) and sample from set 4 (right) where the green lines represent match points and red dots represent the points with no match.

Figure 6 shows on the left side the graphical representation of the key features extracted by the SURF algorithm from the original template that was stored in the database. On the right

side, it shows the graphical representation of the features extracted from a sample in set 4. The green lines represent the distance between each match while the red points were not found matches. Also, figure 6 shows how the key point can be drawn without showing the full fingerprint sample.

4.2.2. Matching

To ensure the code was properly working, the first test performed was to compare the set stored in the database to itself. Figure 7 and 9 shows that all fifteen samples collected in set one were successfully matched. However, Figure 8 shows a total of twelve matched samples because the SURF algorithm did not find any features for three of the samples in set 1 as shown in table 1. Then, the results set 1 were discarded because this was only used as a verification test and it did not contribute any other meaningful information to the study. Thus, the resulted percentages matching for the overall experiment including sets 2-6 were 18.67% for ORB, 42.67% for image hash, and 30.67% for SURF. The results for the experiment excluding the externals samples were ORB with 36%, image hash with 68% and SURF with 42.67%. Lastly, each figure from 7 to 9 shows the resulted percentage match for each set.

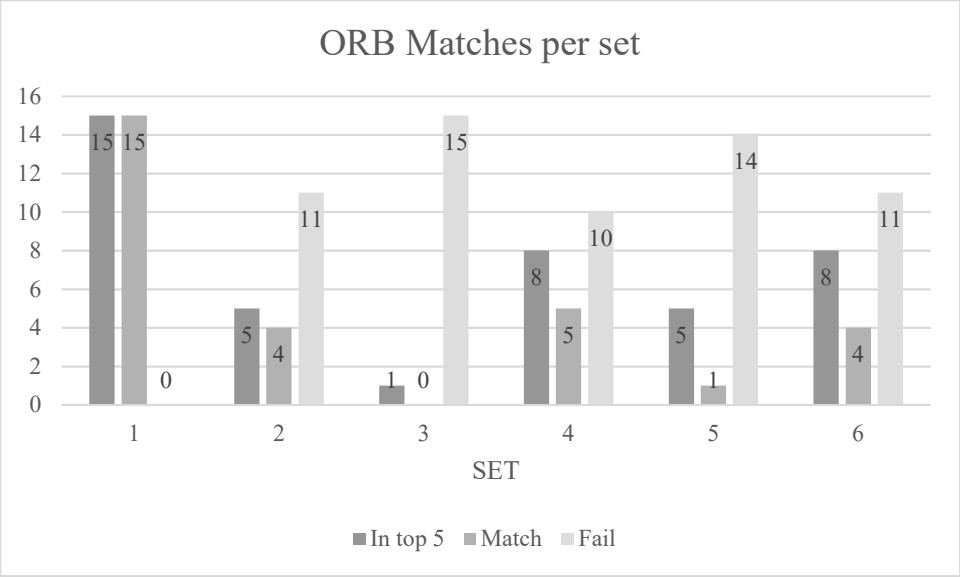

Figure 7 a graph that represents the matching performance of ORB in each set tested where "in top 5" means that the input sample was found between the top 5 matches.

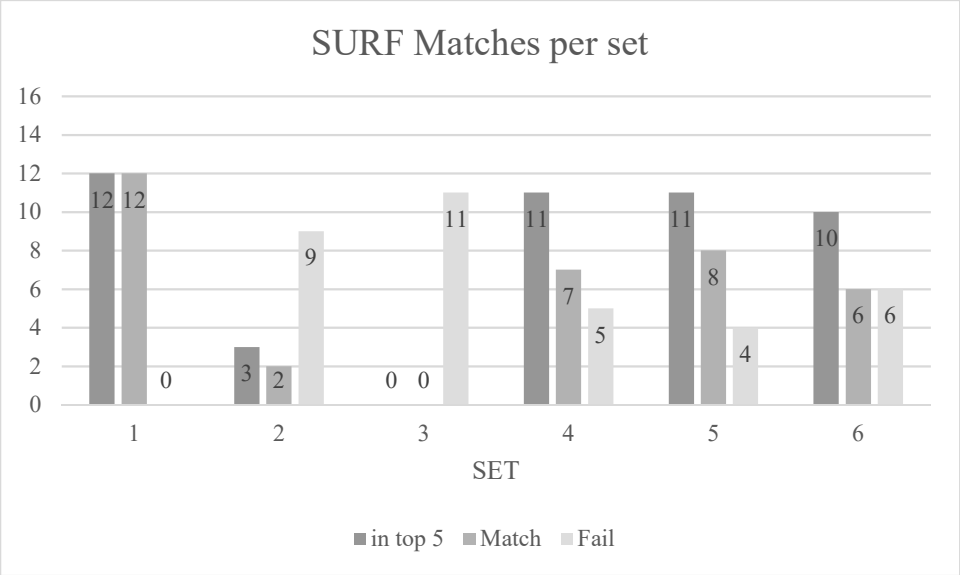

Figure 8. A graph that represents the matching performance of SURF in each set tested where "in top 5" means that the input sample was found between the top 5 matches.

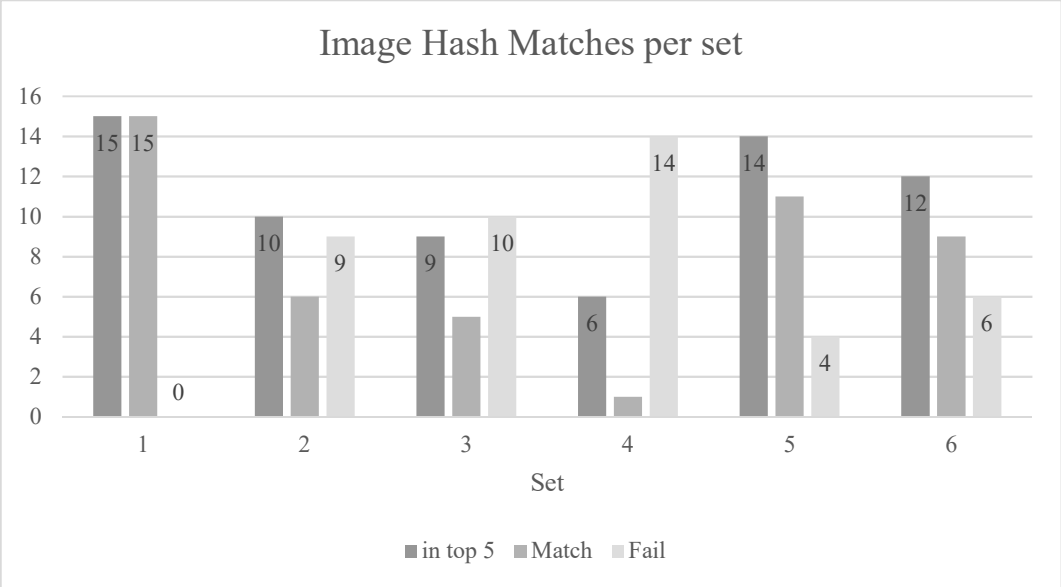

Figure 9. A graph that represents the matching performance of Image Hash in each set tested where "in top 5" means that the input sample was found between the top 5 matches.

4.2.3 Performance

The performance for this study is shown in figure 10 that represent the average position each set provided for each algorithm. The slope of the line represents the changes between each sample in each set compared to set 1. For example, set 4 to 6, which are permutations of set 1, have less dramatic changes compared to set 2 to 3, which are the other samples read from the sensor. Figure 11 shows the average Hamming distance for each algorithm to graphically represent which of the algorithm performed better. The lower the average Hamming distance, the better the accuracy of the specific hashing algorithm.

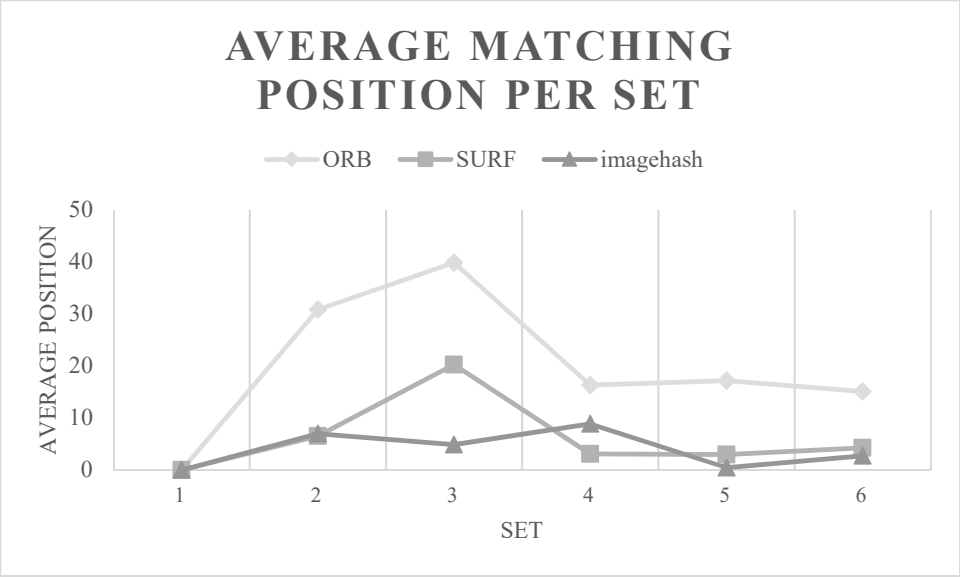

Figure 10 shows the average position that each algorithm achieved per set. This represents the accuracy of each set with the data collected. The lower the position the better where position 0 represents a match

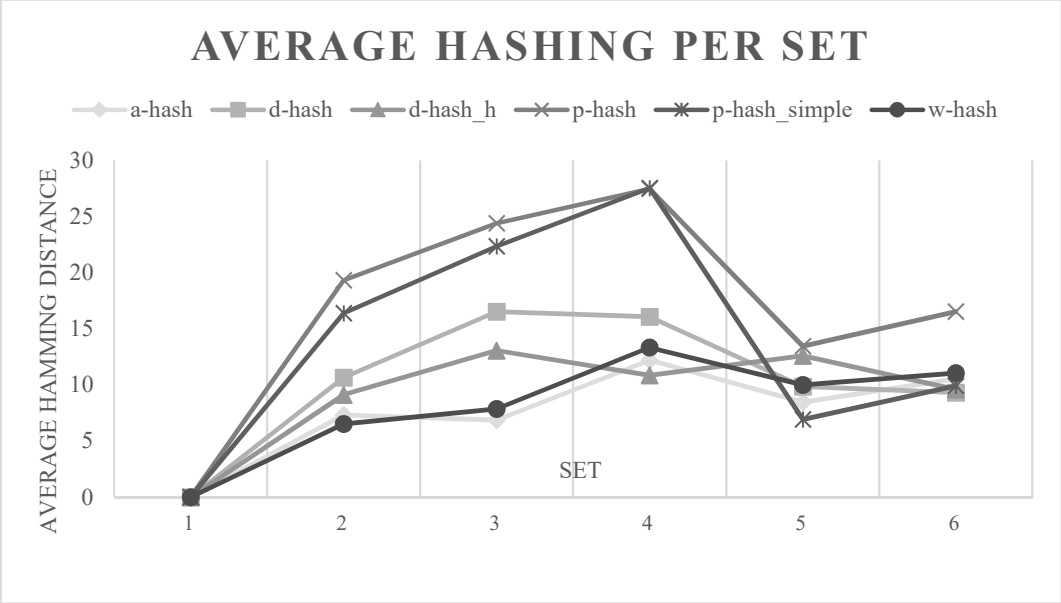

Figure 11. Performance of each image hash algorithm used in an experiment where the lower the value the close is the input sample to the one in the database. The comparison is done using the mean hamming difference in each set tested.

4.2.4. Fingerprint Quality

As mention before, image quality is important for feature extractors algorithms. Figure 12 shows that a good image quality results in a more accurate matching score. In figure 12, SURF and image hash algorithm matched the sample right for 83.3% of the time while ORB matched it right for 66.7% of the time. In contrast, Figure 13 shows that a bad image quality resulted in a poor matching performance. For instance, SURF with a bad quality image did not extract any features, and the fail rate for ORB and image hash were three times greater than a good image quality. Thus, there was a change of 67.7% that a good quality image was properly identified compare to 33% of a bad quality sample.

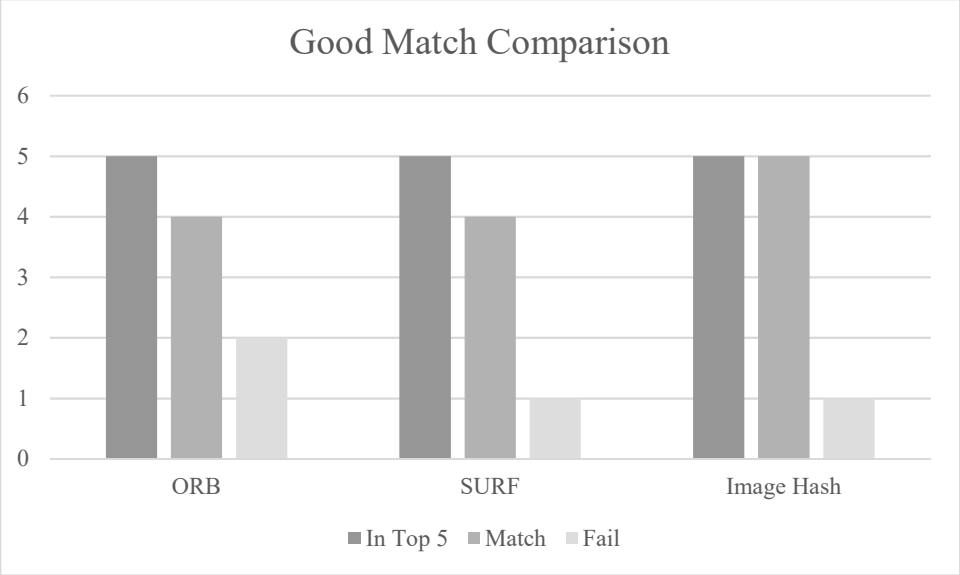

Figure 12. Comparison of each feature extractor using a good quality fingerprint reading where "in top 5" means that the input sample was found between the top 5 matches.

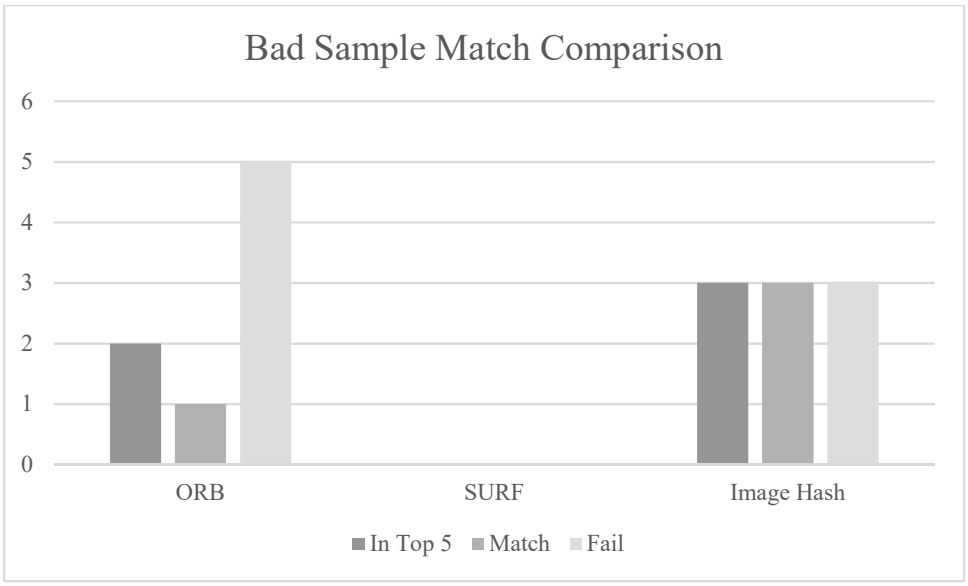

Figure 13. Comparison of each feature extractor using a bad quality fingerprint reading where "in top 5" means that the input sample was found between the top 5 matches.

4.3 Analysis

Securing fingerprints data using the technologies proposed in this paper is possible, but the precession of identifying the fingerprint is not accurate enough to conclude the system is reliable. As it was mentioned before, the algorithm used in this paper does not provide enough information to reconstruct the full fingerprint sample, as shown in figure 6. Also, the database can be reconstructed by modifying the method of extracting the fingerprint' features if the data gets compromised, as shown in figure 4. However, the resulted overall matching percentage of all algorithm used in this paper shows that the system lack of the ability to properly match fingerprints.

Image quality was one of the biggest factors that affect the percentage match of the system. The section 4.2.4 shows that there is almost twice the change of identifying a good quality image compared to a one that has a bad quality. If only good quality images were to use, the matching percentage would have improved. Thus, the percentage matching would have resulted in ORB with 37%, image hash with 85.34% and SURF with 21.34%. The bad quality

samples found in this paper were samples with high contrast (too black), low contrast (too white), and sample with less than half the fingerprint read. Similarly, some of the samples also show marks of the previous fingerprint resulted from not cleaning the sensor to simulate the way the system will be used in the real world.

In the result section, the top 5 positions were presented because most of the time an input sample appeared between the first five positions, the samples above it were samples from the external database. This is an indication that if the external data were to be removed, the accuracy of the system will be improved by around 20%. For instance, during the process of testing, the image hash algorithm ranked all the samples from sensor used in this paper first, and external samples after that. In contrast, SURF and ORB algorithm ranked any sample data in a random order. Thus, the system should only use one type of sensor to collect the fingerprint samples to have a more homogenized set of data.

5. CONCLUSIONS

5.1 Summary

Utilizing open source software and Internet of Things technology is a potential alternative to commercial base technology for fingerprint security. Even though the matching percentage of this paper does not show the desired accuracy for fingerprint indexing, the results show what went wrong during this experiment and the paper discuss how to approach those problems.

5.2 Future work

To improve the accuracy of the system, the quality of the fingerprint samples reading should be improved and the preprocessing techniques must be improved to have a better-skeletonized fingerprint sample. To improve the quality of the fingerprint sample, the cage around the fingerprint sensor, that forces the user to enter the image in one position, must be

improved. For example, this paper only considered the position of the object as a strong factor, but there are other factors such as illumination, cleanliness of the sensor, and bits lost that can also affect the quality of the fingerprint sample. Moreover, it is possible to train ORB with Support Vector Machine (SVM) that will help recognize good fingerprint sample [13]. Doing this, the system will be able to tell the user if the sensor is dirty or if the user needs to place his finger in a different position. Finally, an improved preprocessing function that can reduce the amount of noise in the fingerprint templet will help feature descriptors to extract features easier. For instance, A thinning method as the one in [15] will generate an image with sharp lines that will help feature extractors get strong key points easier.

6. REFERENCES

[1] Hany, U., & Akter, L. (2015). Speeded-Up Robust Feature extraction and matching for fingerprint recognition. 2015 International Conference on Electrical Engineering and Information Communication Technology (ICEEICT), 1-7.

[2] He, S., Zhang, C., & Hao, P. (2009, November). Comparative study of features for fingerprint indexing. In Image Processing (ICIP), 2009 16th IEEE International Conference on (pp. 2749-2752). IEEE.

[3] Tulyakov, S., Farooq, F., & Govindaraju, V. (2005, August). Symmetric hash functions for fingerprint minutiae. In International Conference on Pattern Recognition and Image Analysis (pp. 30-38). Springer, Berlin, Heidelberg.

[4] Dong, R., Di, Y., & Zhang, Q. (2013). A robust content authentication algorithm of speech based on perceptual hashing. *Sensors & Transducers, 161*(12), 375-382. Retrieved from http://0-search.proquest.com.library.uark.edu/docview/1509846734?accountid=8361

[5] Singh, S. P., & Bhatnagar, G. (2017). A robust image hashing based on discrete wavelet transform. Paper presented at the 440-444. doi:10.1109/ICSIPA.2017.8120651

[6] Tang, Z. J., Zhang, X. Q., Dai, Y. M., & Lan, W. W. (2013). Perceptual image hashing using local entropies and DWT. Imaging Science Journal, 61(2), 241–251. https://0-doi-org.library.uark.edu/10.1179/1743131X11Y.0000000039

[7] Fracastoro, G., Fosson, S. M., & Magli, E. (2017;2016;). Steerable discrete cosine transform. IEEE Transactions on Image Processing, 26(1), 303-314. doi:10.1109/TIP.2016.2623489

[8] Dalcın, L., Paz, R., Storti, M., & D'Elıa, J. (2008). MPI for Python: Performance improvements and MPI-2 extensions. Journal of Parallel and Distributed Computing, 68(5), 655-662.

[9] Maltoni, D., Maio, D., Jain, A. K., & Prabhakar, S. (2005). Handbook of fingerprint recognition (2nd ed.). New York: Springer.

[10] Raschke, B. (2016, March 6). Pyfingerprint. Retrieved September 4, 2018, from https://github.com/bastianraschke/pyfingerprint

[11] Sanchez, E (2018). Medical Digital Health Assistant (MDHA). Honor Project Proposal

[12] The Third International Fingerprint Verification Competition. (2004). Retrieved October/November, 2018, from http://bias.csr.unibo.it/fvc2004/download.asp
Biometric System Lab - University of Bologna

[13] Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011, November). ORB: An efficient alternative to SIFT or SURF. In *Computer Vision (ICCV), 2011 IEEE international conference on* (pp. 2564-2571). IEEE.

[14] Zhuo, L., Geng, Z., Zhang, J., & guang Li, X. (2016). ORB feature based web pornographic image recognition. *Neurocomputing*, *173*, 511-517.

[15] Burger, W., & Burge, M. J. (2016). Scale-Invariant Feature Transform (SIFT). In *Digital Image Processing* (pp. 609-664). Springer, London.

[16] Calonder, M., Lepetit, V., Strecha, C., & Fua, P. (2010, September). Brief: Binary robust independent elementary features. In *European conference on computer vision* (pp. 778-792). Springer, Berlin, Heidelberg.

[17] Tola, E., Lepetit, V., & Fua, P. (2010). Daisy: An efficient dense descriptor applied to wide-baseline stereo. *IEEE transactions on pattern analysis and machine intelligence*, *32*(5), 815-830.

[18] Leutenegger, S., Chli, M., & Siegwart, R. Y. (2011, November). BRISK: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on* (pp. 2548-2555). IEEE.

[19] Holder, E. H., Robinson, L. O., & Laub, J. H. (2011). *The fingerprint sourcebook*. US Department of Justice, Office of Justice Programs, National Institute of Justice.

[20] Parker, J. R. (2010). *Algorithms for image processing and computer vision*. John Wiley & Sons.

[21] Makary, M. A., & Daniel, M. (2016). Medical error-the third leading cause of death in the US. BMJ: British Medical Journal (Online), 353

Tables

Table 1

Data set representation with description

Set #	Description
1	First readings of the fingerprints that is going to be store in the database
2	Second reading of the fingerprint in the same position as set 1
3	Third reading of the fingerprint to place it in the same position as set 1
4	Permutation of set 1 moved around
5	Permutation of set 1with white lines drawn by an oil brush with 16px
6	Permutation of set 1 with 3 lines drawn by a marker with 30px

Table 2

Number of samples skipped by the SURF algorithm due to bad quality samples

Database	Skipped Sample	Total Samples	%
Collected (Set 1)	3	15	20
F2000 DB2 B	51	74	68.9
F2004 DB1 B	0	38	0
F2004 DB0 B	0	33	0

Hardware Sketch Configuration

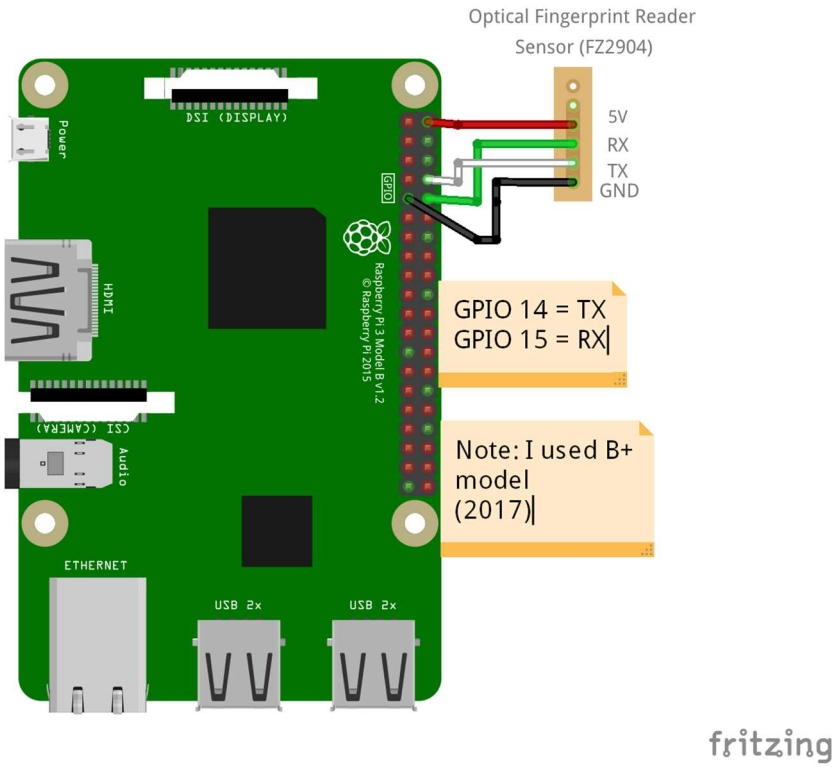

Figure 11. Raspberry Pi 3 B+ Pins schematics

Optical Fingerprint Reader Sensor (FZ2904)

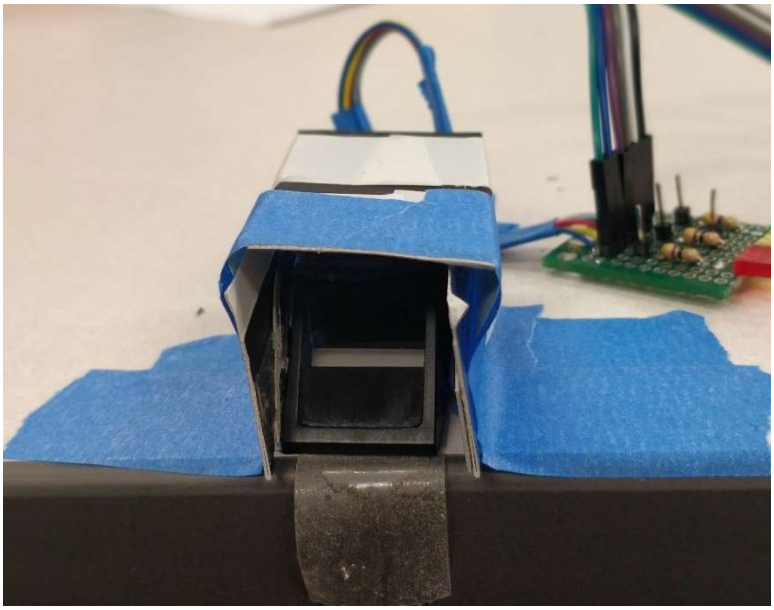

Figure 12. Caged Fingerprint