Finishing and Improving of Finger Vein System

Trethyn Trethyn
Electrical Engineering Bachelor Student

***Abstract*—Finger vein biometrics is a developing field, currently dominated by commercial research. Since commercial products and research are kept private, the academic community wishes to develop their own finger vein biometric systems, in order to conduct their own research. This report details a redesign of the finger vein scanner system at the University of Twente, making the system self contained and more coherent. The system consists of a strip of individually controllable Near InfraRed (NIR) LEDs, an InfraRed (IR) camera, a custom designed driver board to provide power to the LEDs and a Raspberry Pi 4 for control. The redesign is done by migrating the user interaction, signal processing, matching and enrolment systems from MatLab running on an external computer to a combination of OpenCV and C++, running directly on the finger vein scanner. The redesign also includes a new implementation of automatic LED adjustment, in order that the finger being captured is correctly exposed and can be properly processed. Furthermore touchscreen control is implemented for the system, making it completely standalone. The performance of the migrated system was evaluated with respect to the original, and vein extraction and recognition were found to be comparable.**

***Index Terms*—Near InfraRed(NIR), Finger Vein, OpenCV, C++, LED**

I. Introduction

There are several forms of biometric identification currently in common use. Fingerprint sensors on phones and computers, face recognition and retinal scans. There are, however, many more unique identifiers humans carry as a part of their body, such as the patterns of veins inside fingers.

Finger vein biometric devices do exist, but much less commonly than other forms of biometric recognition. There are a few on the commercial market, such as the Hitachi VeinID [1], but since these are commercial and hence the information on their systems private, scientific research into finger vein biometrics must be done separately. To this end, the University of Twente currently has a finger vein scanner device. However, the device's functionality could be improved. As of this writing it requires connectivity to a computer in order to capture images, or display those images, and all image processing and matching is currently done completely remotely, via Matlab. This is a nonideal situation, the device is not exactly easily usable in this state, so this text will describe an attempt to properly integrate the functionality of the device all in one physical setup, with the addition of a touchscreen for control.

II. Related Work

A. Finger Vein Recognition

Finger vein recognition is a member of the larger group of vascular biometric techniques, which includes hand, finger, wrist and eye recognition based on the unique patterns of blood vessels within these body parts [2]. As might be guessed from the name, finger vein recognition is specifically concerning the blood vessels in fingers.

The vascular patterns in fingers are measured by means of Near InfraRed (NIR) light, which is absorbed by haemoglobin [2]. When the area of interest is illuminated with NIR light, images captured with an NIR sensitive camera will show areas containing haemoglobin as dark, and other areas as light, allowing noninvasive capture of the vascular pattern of the finger [2] [3] [4].

There are multiple methods which have been used to capture finger vein images, namely transmission, reflection and side illumination [4] [2]. In transmission, NIR illumination is provided above the finger, which then shines through the finger and to a camera positioned below it [4]. In reflection, the illumination is provided below the finger along with the camera. This has an advantage over transmission, in that the user can see their finger while using the device and thus feels more comfortable [4], but also the disadvantage that most of the captured light does not penetrate the skin, leading to low contrast between veins and surrounding finger [4]. Lastly comes side illumination, in which light sources are placed on one or both sides of the finger, with camera below. The light would scatter inside the finger and the camera would pick up some of that scattered light [4] [5]. This method has the psychological advantage of a visible user finger, and the disadvantage that the side(s) of the user's finger will be over exposed [4].

Once the vascular pattern images have been captured, they then need to be processed and identified. The images are usually preprocessed, in order to combat factors such as blur or poor contrast [3] [2]. Then, once the images have been enhanced, the region of interest (finger area) must be determined. There are several different ways to do this, often involving the detection of some component with known position relative to the rest of the finger (e.g. edge detection) [3] [2]. Finally, then the pattern of veins can be extracted. There are many ways this is done, including Repeated Line Tracking, which tracks dark lines, the Gabor Filter, which performs texture analysis, or the use of trained Neural Networks to recognise vein patterns [6].

Finally, now that the vein pattern has been isolated, the vein images need to be matched and identified. Biometric comparison is done by use of algorithms which compare the images to images in a database and look for the best match(es) in terms of multiple parameters represented as comparison scores [3] [2].

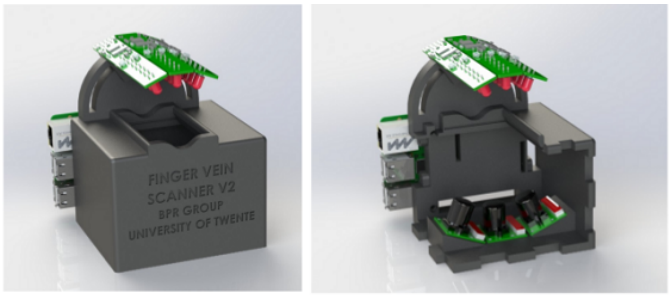


Fig. 1: Finger vein scanner, University of Twente [5]

B. The University of Twente Finger Vein Recognition Device

In 2012 the Services and Cybersecurity (SCS) group at the University of Twente designed and built their own finger vein scanner, in order to do research into what was primarily a commercial field, with little academically available information [5] [4]. This device was large, bulky and required an external computer in order to run. Therefore in 2017 [5], it was redesigned into a more compact form. This form, which is also, with minor physical changes (Currently the system only includes one LED board instead of the three shown), the current one, can be seen in figure 1.

The redesigned scanner featured transmission illumination, with the use of three cameras fitted with IR pass filters and placed below the finger, and three custom printed PCBs, each containing an array of IR LEDs placed in a hood above the finger. The orientation of these components can be seen in figure 1. Each camera was controlled by a separate Raspberry Pi, two Raspberry Pi Zeros and one Raspberry Pi 3, with the Zeros acting as slaves to the 3. The 3 also controls the LEDs via I^2C and an ATMEGA microcontroller. The LED lighting was custom made, in order to achieve the specific radiant intensity required, and to use a constant current driver (desirable to avoid flickering affecting intensity). [5].

In order to optimise the contrast of the images taken for best finger vein detection, an optimal grey value for finger tissue was taken as 80, based on the previous device. In order that this grey value is achieved, the LEDs are individually controllable by the Raspberry Pi, and can be adjusted until this optimal grey value is measured in the captured image. [5].

A problem that the previous design suffered from was over exposure of the camera, so the new design included an LED cover to make the light from each LED significantly more directional. Distortion in the photographs due to the wide angle camera was compensated for, and algorithms were designed to control the light intensity of each LED in order to achieve more optimum grey values. [7].

Further work was then done by the master student Thomas van Zonneveld, with a redesign of the system using ArduCam IR cameras, three Raspberry Pi 4s and a C++ based interface to allow image capture. This redesign is the current state of the device, with camera control and manual LED control implemented on the scanner, and image processing, feature extraction and matching done on a computer via Matlab.

III. Design

In order to improve the system, the Matlab functionality can be moved to the scanner. This Matlab functionality can be considered to consist of two parts, namely the interaction with the user, and the actual image processing.

A. Image Processing

When a vein image is captured, it is sent to Matlab, which performs one of two operation streams on it.

If the image is to be enrolled as the template for a new user in the system, it is first cropped to the region of interest (ROI), and then saved with a unique identifier as a template. If, on the other hand, the image captured is to be compared to an already enrolled template to authenticate a returning user, both it and the template will undergo the same pre-processing and feature extraction processes, and then be matched. A matching score can then be found, and this is used to determine whether or not the user is the person they claim to be.

The pre-processing consists of two processes. First the edges of the finger are identified and the finger is normalised to a baseline. Then the actual feature extraction can be performed, where the positions of the veins themselves are found and the image binarised. After these stages have been completed, the binarised vein images can be used to calculate matching scores.

To implement this on the Raspberry Pi, it needs to be converted to another form. Matlab cannot run on a Raspberry Pi, and even if it could the inefficiency and slow speed of Matlab scripts on such a device would be a severe problem. It was decided to rewrite the pre-processing, feature extraction and matching methods in C++, with the help of OpenCV. This library was chosen because it is a widely used, competent image processing and computer vision library that is easily available and has many resources available for support. C++ will be used because it is fast, efficient, and interfaces easily and well with OpenCV.

B. User Interaction

Ideally, there should be two main ways the user can interact with the system, to enrol as a new user or to attempt to authenticate themselves as a returning user. So that the device is completely standalone, user interaction will be via a touchscreen. This touchscreen will display a GUI that allows the user to enrol and to perform matching with a pre-enrolled template. A simple LED control algorithm will be included with image capturing, so that the images used are optimal.

The design of the system is perhaps easier to understand with the help of figures 2 and 3, which illustrate the plan for the redesigned finger vein scanner.

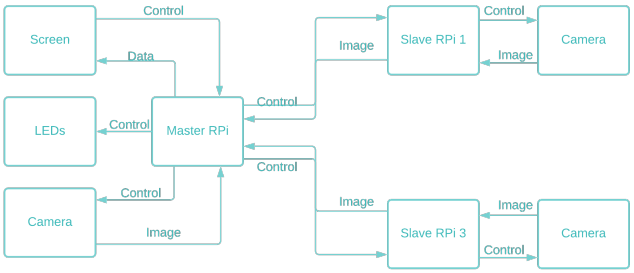


Fig. 2: Hardware Design Diagram (fullsize version in appendix)

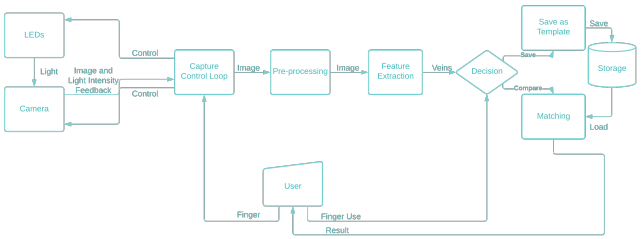


Fig. 3: Software Design Diagram (fullsize version in appendix)

C. Hardware

For the purpose of these improvements, not many physical changes need to be made, with hardware additions mostly consisting of the addition of a touchscreen. Of the options available from the lab, it was decided to use the Adafruit 2.4" PiTFT Hat [8]. This decision was made because this screen is of a useful size where the information and options required can be neatly displayed, and because it does not interfere with the operation of I^2C. The other available screen was both larger and considerably dirtier, with its signals interfering with the I^2C bus and hence the LED and camera control.

D. Evaluation

It is all very well to design a system, but that design does not matter very much unless it works. The new integrated finger vein scanner should perform at least as well as the previous version, with equivalent or better finger recognition and image capturing.

Finger recognition performance can be evaluated and compared by performing image processing and matching on the same images with both implementations, then comparing the results. There should not be a loss of matching accuracy, and vein images generated should be of sufficient quality to provide useful results.

Image capture performance can be evaluated by ensuring that the finger vein images captured are of a good quality to be used. Based on previous iterations of the scanner, and information from previous work, the finger itself should be uniformly at an acceptable grey value, and the veins should be reliably extractable [5] [7].

IV. Method

Now that plans have been laid, the improvement and integration of the device can be performed.

A. Pre-processing, Feature Extraction and Matching

In the original Matlab, the finger region identification, image normalisation, vein extraction, binariasation and matching were split into separate functions. No reason was seen to change this, so the titles and usages were carried over to the C++ implementation.

1) Region identification: Called *lee_region*, this function is based on the region detection method described in a paper by Lee et al. [9]. It works by convolving the image with the 4x40 mask shown in equation 1, splitting the image in half vertically, and then taking the maximum point in each column of the upper half, and the minimum in each column of the lower half, to be the edges of the finger. No major changes were made in the working of this function during its linguistic migration, and any differences in output between the versions are expected to be due to differences in filter implementation between OpenCV's *Filter2D* and MatLab's *imfilter*.

$$\begin{bmatrix} -1 & -1 & -1 & \dots & -1 \\ -1 & -1 & -1 & \dots & -1 \\ 1 & 1 & 1 & \dots & 1 \\ 1 & 1 & 1 & \dots & 1 \end{bmatrix} \tag{1}$$

2) Image normalisation: This function is called *huang_normalise*, and is based on the normalisation method in a paper by Huang et al., though without the elliptic projection described there [10]. The transform matrix differs slightly in appearance between the implementations, due to different conventions for it between Matlab and OpenCV. The baseline projection is also different, since in Matlab a baseline is determined using *robustfit*, in the C++ implementation a self written implementation of least squares linear regression is used, named *linfit*. They produce similar, but not identical results.

3) Vein extraction: The function used for vein extraction is called *miura_max_curvature*, and is based on a method described in the paper by Miura et al. [11]. Vein patterns are identified by filtering the normalised image with five different kernels, determined according to equations based on an input parameter *sigma*. These filtered results are then combined into four images. Each image shows changes between light and dark in the original image, though sensitive in different directions. One shows horizontal changes, one vertical, one top left to bottom right diagonal, one bottom left to top right diagonal. These images are multiplied with the binary finger region mask obtained in *lee_region*, and then each of these images is then scanned over, each curve being used to mark one point on the vein image. These points are in the place of maximum curvature, and weighted by the length of the curve. Once all points have been found and weighted, veins are isolated by finding vein centres. The main difference between the C++ and Matlab implementations of this function is in the treatment of the bottom left to top right diagonal curvature

extraction. OpenCV has a function which is used to find diagonals of matrices, so to use this the image in question is flipped over the vertical axis before diagonal extraction.

4) Binarisation: Once this vein image has been generated, it is binarised. In both languages this is done by finding the median value of the non zero parts of the image and using this as a threshold for binarisation.

5) Matching: The matching function, named *miura_match* and once again based on the paper by Miura et al. [11]. Matching is performed quite simply by finding the cross correlation of the template image and the input image, taking the maximum result and then normalising over a specified at input search area. The result of this will be a score between 0 and 0.5. This is multiplied by 200 when output to the user, so it can be represented as a percentage. The main differences between the two implementations of this function are in Matlab and OpenCV's approaches to cross correlation, which are unlikely to produce exactly the same results. The image had to be manually zero padded in the C++ implementation, while Matlab does this implicitly.

B. Touchscreen and User Interaction

Since one of the main ideas of this project is to make the finger vein scanner standalone, all interaction with the system will now have to be via the touchscreen that will be installed.

1) Physical implementation of the touchscreen: The touchscreen used was designed as a Raspberry Pi hat [8]. As such, although it only makes use of the SPI pins [8], it is provided with a full header to sit on top of the Raspberry Pi board. The screen does provide an access point for 26 out of 40 pins which it does not use. A problem with this is that in the current design the header of the Pi is already in use, being entirely connected to the LED driver board. However, in practice only the I^2C, 5 V and GND pins are needed for the connection to the driver board, so it should be possible to connect all three devices at once. An attempt was made to run the screen remotely via jumper cables, but the screen then did not display any information that was sent to it, so the system was changed so that the Pi and driver board were connected by jumper cables and the screen sat on top of the Pi. This did work, although not well since the jumper cables could not provide enough power to the Pi.

2) GUI for the touchscreen: As was described in section III-B, the touchscreen GUI should be able to allow the enrolment of new users, and to authenticate returning users again by comparing their finger to a pre-enrolled template. The GUI that was designed and realised can be seen in figure 4. The capture button allows for a new picture to be taken, enrolling a user in the system. The arrow buttons allow the list of saved images to be cycled through (current image will be shown in the rectangle on the right), and the identify button will match the currently selected image to a finger presented to the device, then report the score beneath the rectangle, as well as display the identified vein pattern for interest. A preview showing the direct information stream from the camera is displayed overlaying the GUI. This is done because of the way raw camera footage is handled by the Raspberry Pi - it's done low level by the GPU [12], and as such cannot be integrated with the rest of the GUI, which is made with GTKmm 3 [13]. A quit button is also included, so the user can escape to the desktop if needed.

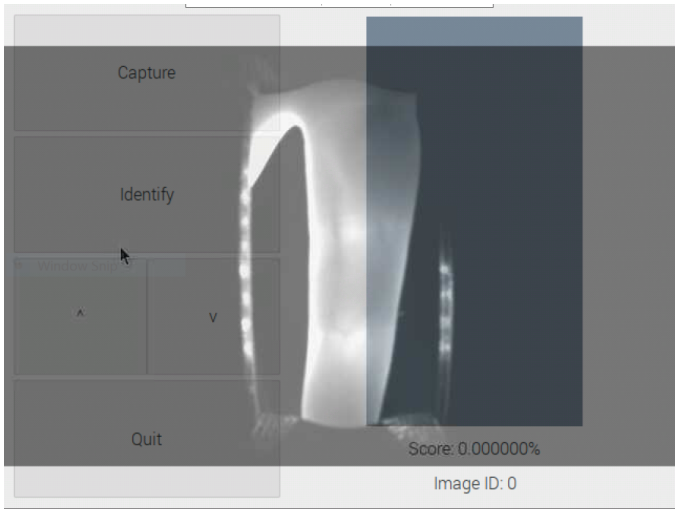


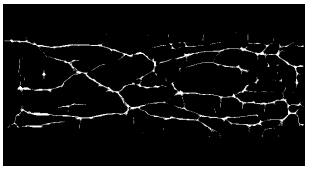
Fig. 4: Final version of the finger vein scanner GUI

C. LED Control

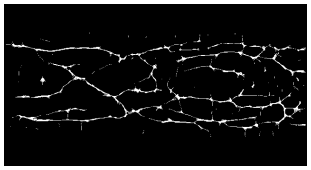
In order that pictures taken with the device can be reliably usable, a form of LED control was implemented. A simple control function, *adjust_leds_auto*, was written. This function captures an image with the camera, then runs *lee_region* on that image to identify the finger area. Then a rectangle one pixel wide and as long as the image is taken from the image at the centre of the finger (based on the edge positions returned by *lee_region*). The mean grey value of this rectangle is then found, and if it is below the lower acceptable threshold of 70 the lights are brightened by a step, while if it is above the upper acceptable threshold of 90 the lights are dimmed by a step. These acceptable thresholds were chosen based on the mean grey values (as found by the same method as described above) for several images in the University of Twente's finger vein dataset [4].

V. Results

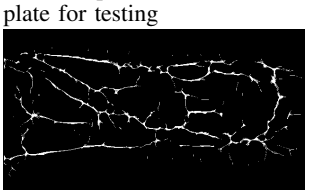
The image processing functions were tested first using vein images from the University of Twente dataset [4], second with images captured by the system itself. In order to determine the quality of the system, tests were performed in which a selected finger vein image would be compared to a different image of the same finger, a different finger of the same user, and a completely unrelated finger. In order to compare implementations of the system, these tests were performed with the same images and the same comparisons in Matlab and in C++. Some of the vein images generated during these tests can be seen in figure 6 (Matlab) and 7 (C++).



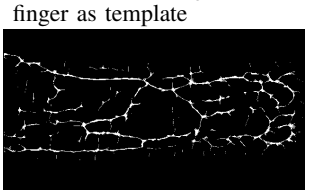
(a) Vein pattern used as template for testing



(b) Different image of same finger as template

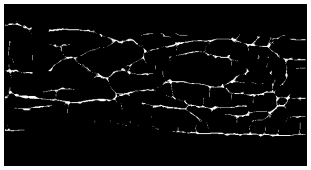


(c) Vein pattern from a different finger of the same user

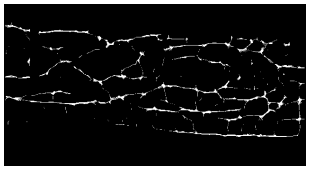


(d) Vein pattern of a different user

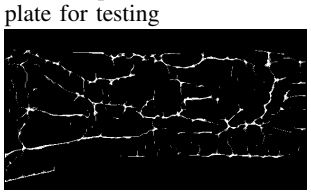
Fig. 6: Resulting veins from Matlab finger vein comparison



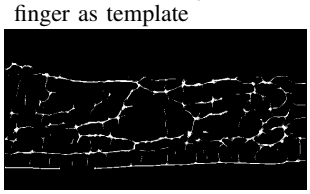
(a) Vein pattern used as template for testing



(b) Different image of same finger as template



(c) Vein pattern from a different finger of the same user



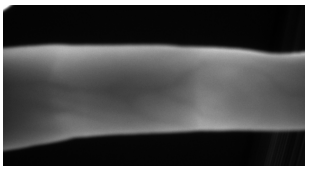
(d) Vein pattern of a different user

Fig. 7: Resulting veins from C++ finger vein comparison

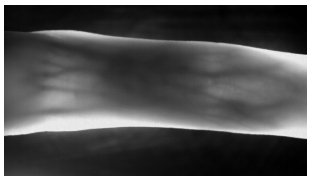
These test sets were performed ten times, each time with different sets of images. The results of these tests can be seen summarised in table I.

Then, in order to test the ability of the system to capture useful images, a small dataset of three individuals was captured anonymously, and the same tests were performed using these images. Due to the smaller size of this dataset, six sets of tests were performed rather than ten. Some of the vein images captured can be seen in figures 8 and 9. The results of this second set of tests can be seen in table II. It would have been preferable for the sample size to be larger, but the number of both available and willing participants was somewhat limited.

An example of the images captured by the system can be seen in figure 5a.

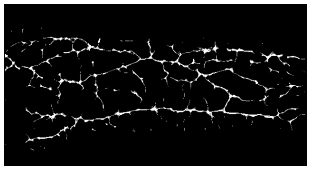


(a) Finger image captured by the new finger vein system

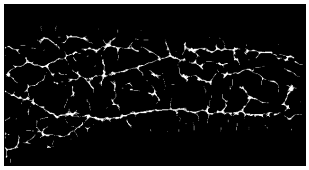


(b) Finger image captured by the previous finger vein system

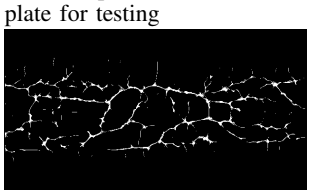
Fig. 5: Comparison of images from the previous and redesigned finger vein systems



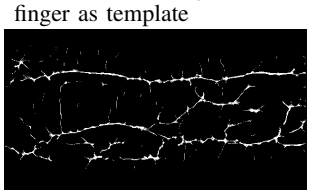
(a) Vein pattern used as template for testing



(b) Different image of same finger as template

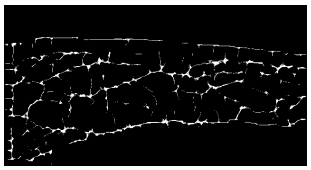


(c) Vein pattern from a different finger of the same user

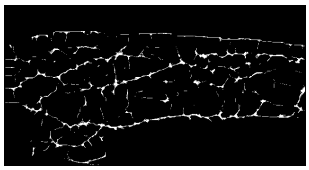


(d) Vein pattern of a different user

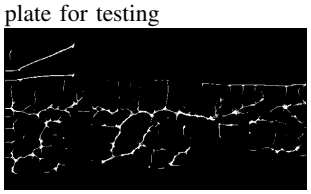
Fig. 8: Resulting veins from Matlab finger vein comparison of images captured with the device



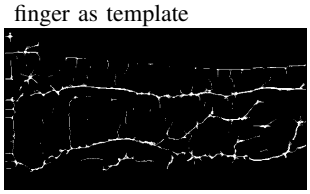
(a) Vein pattern used as template for testing



(b) Different image of same finger as template



(c) Vein pattern from a different finger of the same user



(d) Vein pattern of a different user

Fig. 9: Resulting veins from C++ finger vein comparison of images captured with the device

TABLE I: Results of implementation comparison performed on dataset images

	Matching score self		Matching score self, other finger		Matching score other		
Name	Matlab	C++	Matlab	C++	Matlab	C++	Notes
0001_1_1_120509-135315	34.5 %	27.9 %	11.8 %	24.8 %	11.6 %	18.9 %	
0002_1_1_120509-140611	38.5 %	54.1 %	10.4 %	21.4 %	11.1 %	19.1 %	
0003_1_2_120509-141420	18 %	43 %	13.7 %	22.4 %	13.6 %	31.4 %	Matlab baseline extrapolation reached iteration limit
0004_1_2_120509-141934	24.2 %	30.6 %	10 %	26.9 %	13.4 %	26 %	Matlab baseline extrapolation reached iteration limit
0005_1_1_120509-142157	49 %	74.6 %	11.8 %	21.6 %	14.7 %	19.6 %	
0006_1_1_120509-143643	28.9 %	48.4 %	11.1 %	27.9 %	13 %	33 %	
0007_1_2_120509-144552	22.6 %	34.6 %	13.7 %	23.8 %	10 %	18.5 %	
0008_1_2_120509-145152	22 %	28.9 %	14.1 %	21.1 %	11.5 %	28 %	
0009_1_4_120523-144626	36.7 %	53.8 %	16 %	28.6 %	11.9 %	22.2 %	Matlab baseline extrapolation reached iteration limit
0010_1_2_120509-150320	32.7 %	58.1 %	10.7 %	27.7 %	9.8 %	22.4 %	

TABLE II: Results of implementation comparison performed on images captured with the new device

	Matching score self		Matching score self, other finger		Matching score other		
Name	Matlab	C++	Matlab	C++	Matlab	C++	Notes
A_1_1_1	40.3 %	94.2 %	10.6 %	22.7 %	11.8 %	28.4 %	
B_1_1_1	22.4 %	57.3 %	12.1 %	19.5 %	11 %	20.7 %	Matlab baseline extrapolation reached iteration limit
C_1_1_1	23 %	38.6 %	10.7 %	20.1 %	11.3 %	21.1 %	
A_2_2_1	32.2 %	79.2 %	13 %	31.1 %	10.6 %	30.1 %	
B_3_1_1	37.3 %	28.9 %	12.1 %	18.8 %	10.4 %	17.9 %	
C_2_1_1	24.7 %	39.6 %	11.7 %	20.9 %	10.1 %	24.2 %	Matlab baseline extrapolation reached iteration limit

VI. Discussion

A. Vein Extraction

To comparison by the eye, the vein extraction as performed by C++ is, generally speaking, of slightly lower quality than the original Matlab. Some smaller veins are not picked up (see for example figures 6b and 7b), and sometimes the edges of the fingers are identified as veins (see for example figures 8c and 9c).

The issues with picking up the edges of the fingers as veins is due to the region identification. For reasons likely due to the differences between filter implementations of Matlab and C++, the region identified by the C++ *lee_region* implementation tends to be slightly larger than that identified by the Matlab implementation. This results in the detection of the edges of the finger as veins. If the algorithm were more optimised for OpenCV, this issue could likely be improved.

Some of the smaller veins are not picked up by the C++ implementation. This, again, is likely due to differences in filter implementations. Again, the issue could likely be solved by further altering the algorithm with the working of OpenCV's Filter2D in mind.

B. Image Quality

In figure 5 two images are shown for comparison, figure 5b from the dataset and figure 5a, which was captured with the redesigned scanner. Vein data can be seen in both images, and both are of a sufficient mean grey value. Figure 5a is a little out of focus. This is due to the camera used, the monochrome version of the OV9281 from ArduCam. The OV9281 is a manually focussable camera, but the monochrome version has had the lens glued in place [14]. This might be rectified with the proper tools to remove the glue, but as of now it was decided not to do this, for fear of damaging the camera and rendering the system unusable.

Despite this, veins in a level of detail comparable to the dataset images could still be extracted (figures 8, 9). Therefore this level of blur was considered to be a point to improve, but currently acceptable.

C. Matching Reliability

The results of the matching tests can be seen in tables I and II. For most of the image sets tested on, Matlab and C++ were able to provide a noticeable level of distinction between another image of the same finger, and images of different fingers. For both implementations, there was not much of a difference between the matching score of a different finger belonging to the same user, and the matching score of a completely unrelated finger. The scores reported by C++ tend to be higher, but in general proportionally higher than the Matlab scores. Sometimes Matlab's normalisation function did not perform correctly, being unable to find a baseline, and in those cases the matching score reported is often much lower than it was expected to be, based on the other test sets.

Since the purpose of the system is to determine whether or not a finger presented to the scanner matches the template it is being compared to, once the matching has been performed there are essentially two possible outcomes. Either the finger is considered to match the template, or it is not. Therefore it makes sense that for each system there should be a threshold in place, above which fingers match and below which they do not. Based on the results for Matlab, it makes sense to place that threshold at around 20% (the correct finger is usually in the range of 30%, while incorrect fingers are usually around 10 %). For C++ it makes sense to place the threshold higher, at around 30%, since here correct fingers are usually identified in the range of 30 - 40%, while incorrect ones are usually somewhere near 20%.

TABLE III: Sensitivity and specificity of implementations

Implementation	Sensitivity	Specificity
Matlab	93.75 %	100 %
C++	81.25 %	87.5 %

With these thresholds in mind, there are some cases where the score reported by OpenCV for a finger being matched with itself is low enough that it looks more like a finger that would be rejected. This is also the case in Matlab, sometimes but not always for the same fingers. Interestingly enough, these are all cases in which images from the dataset that were captured of the same finger, but on different days, were compared, making this the likely reason. None of these cases exist in table II, since due to the limited availability of willing subjects in these days of plague, the images captured with the redesigned scanner were all taken on the same day.

There are also some cases where the score reported by OpenCV for fingers that are not the same is high enough that it looks more like a finger that should be accepted. This is never the case for Matlab. A reason for this might be the reduced number of smaller veins picked up by OpenCV's vein extraction.

To put this in a more quantifiable way, taking both measurement sets into account and using the thresholds of 20% and 30% for Matlab and C++ respectively, the sensitivities and specificities of the two implementations can be found (shown in table III.

It can be seen in table III that while Matlab has a higher sensitivity and specificity that C++ (and is hence a better indicator in general), the sensitivity and specificity of C++ are still usefully high. In the majority of cases C++ will give the correct result.

In general, both systems have their false negatives. The C++ implementation also has false positives, and results in higher overall numbers than Matlab. This means that a higher threshold should be used for deciding whether or not a finger matches the template. The system could also stand some improvement to reduce the false positives, but as it stands it does, in the majority of cases, produce reliable indicative results.

VII. Conclusion

In the beginning, plans were made to redesign the existing finger vein scanner into a more compact and coherent form.

Those plans included the migration of the image processing and control from Matlab to C++ on the actual device, the addition of a touchscreen to allow user interaction, and the design of a simple GUI for that touchscreen. These things were accomplished, and a small dataset was collected with the device. Tests were performed, both with the dataset of the university and the newly captured dataset, to determine how well the new version of the system was able to perform. Images captured by the system were sufficient for use, as was the vein extraction, although both could stand some further optimisation before they can be considered equivalent to the original.

VIII. Recommendations

Further improvements that could be made to the device include further optimising the code around the filtering of images, so that the finer veins are picked up and the edges of the fingers are more reliably ignored. Redesigning the case and the circuitboard of the system, so that the touchscreen is stably included in the system. Improving the LED control, and improving the focus of the camera. Finding a nicer way to display the camera preview on the screen.

References

[1] Hitachi. Veinid, finger vein authentication technology. [Online]. Available: https://www.hitachi.co.jp/products/it/veinid/global/index.html

[2] A. Uhl, "State of the art in vascular biometrics," *Handbook of Vascular Biometrics*, 2020. [Online]. Available: https://www.springerprofessional.de/en/state-of-the-art-in-vascular-biometrics/17383680?fulltextView=true

[3] S. Kirchgasser, C. Kauba, and A. Uhl, "Towards understanding acquisition conditions influencing finger vein recognition," *Handbook of Vascular Biometrics*, 2020. [Online]. Available: https://www.springerprofessional.de/en/towards-understanding-acquisition-conditions-influencing-finger-/17383710?fulltextView=true

[4] R. Veldhuis, L. Spreeuwers, B. Ton, and S. Rozendal, "A high-quality finger vein dataset collected using a custom-designed capture device," *Handbook of Vascular Biometrics*, 2020. [Online]. Available: https://www.springerprofessional.de/en/a-high-quality-finger-vein-dataset-collected-using-a-custom-desi/17383698?fulltextView=true

[5] S. Rozendal, "Redesign of a finger vein scanner," *University of Twente Student Theses*, 2017.

[6] E. Jalilian and A. Uhl, "Improved cnn-segmentation-based finger vein recognition using automatically generated and fused training labels," *Handbook of Vascular Biometrics*, 2020. [Online]. Available: https://www.springerprofessional.de/en/improved-cnn-segmentation-based-finger-vein-recognition-using-au/17383712?fulltextView=true

[7] B. Peeters, "Finishing and improving of finger vein system," *University of Twente Student Theses*, 2020.

[8] adafruit. Adafruit 2.4" pitft hat with resistive touchscreen mini kit. [Online]. Available: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-2-4-pitft-hat-with-resistive-touchscreen-mini-kit.pdf?timestamp=1624277069

[9] E. C. Lee, H. C. Lee, and K. R. Park, "Finger vein recognition using minutia-based alignment and local binary pattern-based feature extraction," *International Journal of Imaging Systems and Technology*, vol. 19, no. 3, pp. 179–186, 2009. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/ima.20193

[10] B. Huang, Y. Dai, R. Li, D. Tang, and W. Li, "Finger-vein authentication based on wide line detector and pattern normalization," pp. 1269–1272, 2010.

[11] N. Miura, A. Nagasaka, and T. Miyatake, "Extraction of finger-vein patterns using maximum curvature points in image profiles," *Ieice Transactions - IEICE*, vol. E90D, pp. 347–350, 01 2005.

[12] Arducam. [Online]. Available: https://www.arducam.com/docs/pi/ov9281/tutorial/

[13] The GNOME Project, "Gtkmm." [Online]. Available: https://developer.gnome.org/gtkmm/stable/

[14] Arducam. [Online]. Available: https://www.arducam.com/product/arducam-ov9281-1mp-mono-global-shutter-mipi-camera-with-130degree-850nm-only-m12-mount-for-raspberry-pi/

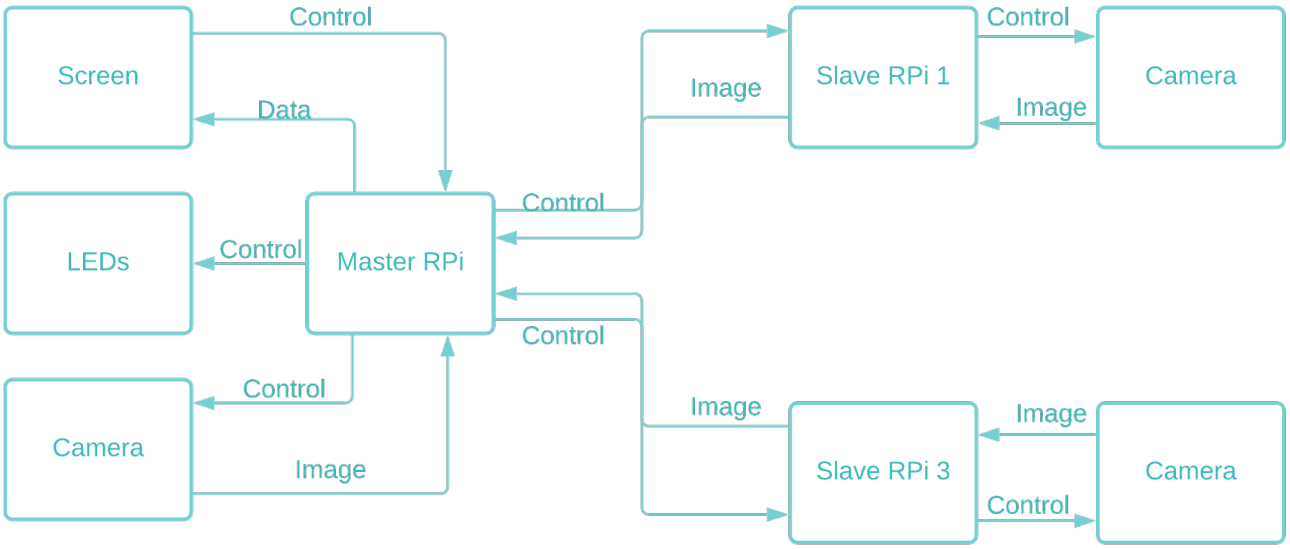


Fig. 10: Hardware Design Diagram

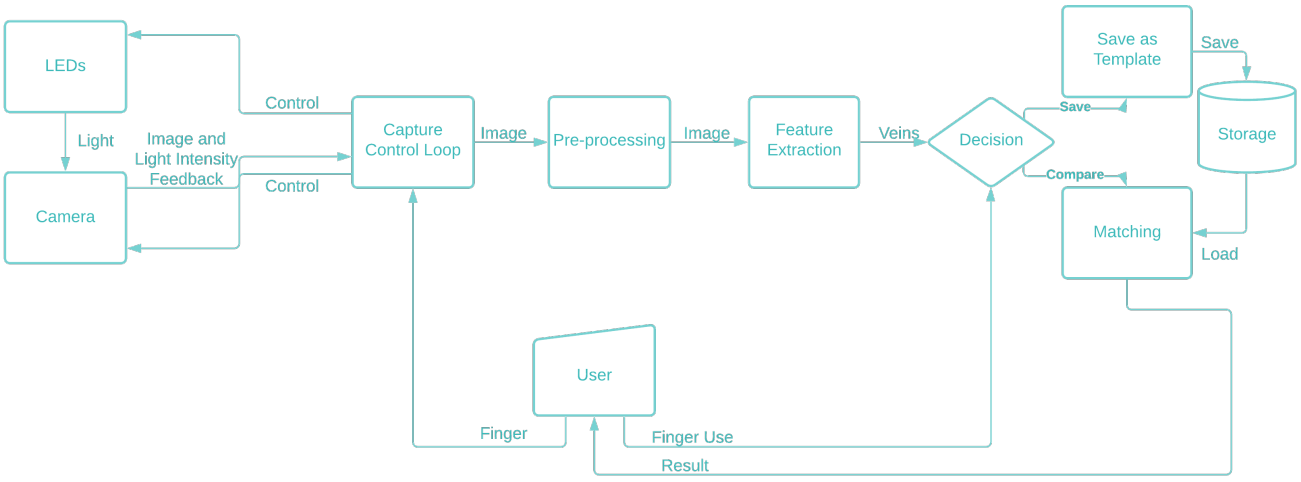


Fig. 11: Software Design Diagram