Course: MSc Student Project

Project Report

"Identification of Finger-based Signatures Using Computer Vision Techniques"

Student name and surname:	Augustin MASCARELLI
Student code:	62722
Student name and surname:	Yona BITTON
Student code:	62737
Student name and surname:	Quentin AMIEL
Student code:	62720
Student name and surname:	Kelig LEFEUVRE
Student code:	62724
Student name and surname:	Enzo COGNÉVILLE
Student code:	62733
Student name and surname:	Elias DÈVE
Student code:	62714
Supervisor:	Prof. Alexander Grakovski
Date performed:	16/04/2023

Riga

April 17, 2023

Contents

1 Introduction

1.1 Purpose of the Document

The purpose of this document is to outline the project plan for the development of an application that uses computer vision techniques to identify finger-based signatures. The application will be designed to address the challenge of signing documents in a world where physical contact is discouraged due to the COVID-19 pandemic.

This document will serve as a reference guide for the project team, detailing the project objectives, scope, and context, as well as the group composition and the technologies that will be used. It will also provide a description of the methodology that will be followed to achieve the project objectives and a summary of the expected outcomes.

The intended audience for this document includes the project team members, the supervisor, and any other stakeholders who may be interested in the project's progress and outcomes.

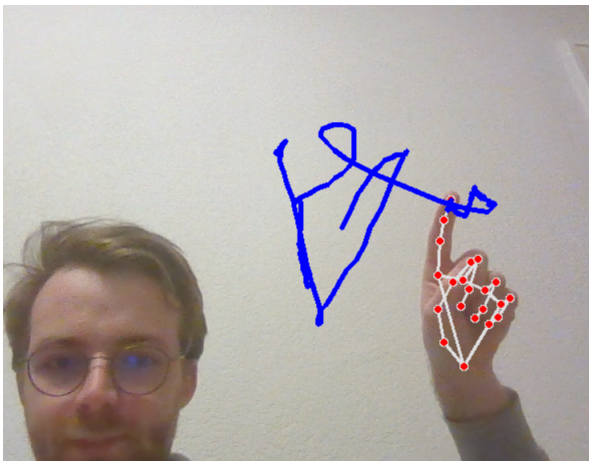

Figure 1: Air Signing

1.2 Objectives

The main objectives of this project are to:

1. Develop an application that uses computer vision techniques to identify a user's finger and capture the depth of the finger for use in creating a digital signature.
2. Implement a solution to recognize and transcribe the user's signature in 3D and 2D.
3. Test the application to ensure it is accurate and reliable in identifying and transcribing finger-based signatures.
4. Develop a user-friendly interface to allow for easy signature creation and storage.

These objectives will be achieved through a series of tasks and subtasks that will be outlined in the methodology section of this document. The successful completion of these objectives will result in an application that is both useful and innovative, providing a new solution for individuals and organizations seeking a secure and contactless way to sign documents.

1.3 Scope

The application will be designed to operate on both desktop and mobile devices with a camera, allowing users to create digital signatures on the go. The application will also include a user-friendly interface for easy signature creation and storage.

The project will focus on the development of the software application and will not involve the creation of any hardware components. The application will be tested to ensure it is accurate and reliable in identifying and transcribing finger-based signatures.

The project will be completed within the allocated time frame and with the resources available to the project team. Any changes to the scope of the project will be evaluated and approved by the project supervisor before implementation.

1.4 Technologies

To implement the proposed solution, we will leverage existing technologies and frameworks that are commonly used in computer vision, image processing, and software development. In this section, we will discuss some of the potential technologies that we could use to implement the different components of our application.

1.4.1 Technologies and frameworks

OpenCV[1]

OpenCV (Open Source Computer Vision) is an open-source library of computer vision algorithms and tools that can be used for image processing, machine learning, and other computer vision applications. OpenCV has a variety of built-in functions that can be used for feature detection, image segmentation, and object recognition, making it an ideal technology for identifying fingers in images captured by the camera.

TensorFlow[2]

TensorFlow is a popular open-source framework for building machine learning models. It provides a variety of tools for training and deploying machine learning models, as well as pre-trained models that can be used out-of-the-box. In our application, we could use TensorFlow to train a machine learning model that can accurately identify finger positions and orientations in images captured by the camera.

PyTorch[3]

PyTorch is an open-source machine learning library that is widely used for building and training deep learning models. It offers a dynamic computational graph that allows for real-time experimentation and a wide range of built-in modules and tools for data loading, model building, and optimization. PyTorch supports both CPU and GPU acceleration and is highly optimized for distributed computing, making it ideal for large-scale applications. It has a large and active community that provides extensive documentation and support, as well as a vast library of pre-trained models and tools.

Tkinter[4]

Tkinter is a standard Python library for creating graphical user interfaces (GUIs). It is based on the Tk GUI toolkit, which was developed by John Ousterhout at the University of California, Berkeley in the late 1980s. Tkinter provides a simple way to create windows, buttons, text boxes, and other GUI elements, as well as to respond to events such as button clicks or key presses.

1.4.2 Advantages and Disadvantages

Each of the above technologies has its own advantages and disadvantages. OpenCV[1], for example, provides powerful tools for image processing and feature detection, but may require significant effort to integrate into a larger application. TensorFlow[2], on the other hand, provides pre-trained models and high-level APIs for building machine learning models, but

may require more specialized knowledge and resources to train custom models. Tkinter[4] is a simple way to build a friendly UI.

1.4.3 Technology Selection

Based on our analysis, we have selected OpenCV[1] and TensorFlow[2] as the primary technologies for implementing our application. OpenCV[1] will be used for finger detection and tracking, while TensorFlow[2] will be used for machine learning-based finger identification. We believe that these technologies provide a good balance of functionality, performance, and ease of integration, and will enable us to achieve our project goals in a timely efficient manner.

Overall, we believe that our technology choices will enable us to develop a robust and effective solution for finger-based signature recognition that can be used in a variety of settings and applications. By combining the strengths of multiple technologies and frameworks, we are confident that we can deliver a high-quality and innovative product that meets the needs of our users.

2 Work Schedules

In this section, we describe the work schedules for the two main groups within our team: the finger recognition group and the signature capture group. The finger recognition group will focus on developing the algorithms and software necessary to detect and track a user's finger in real time, while the signature capture group will focus on capturing the user's signature using their finger and obtaining depth information to create a 3D representation of the signature.

The general schedule was as follows:

- Part 1: Research and familiarization with relevant technologies and software libraries.
- Part 2: Prototype finger detection and signature capture software.
- Part 3: Refine software and integrate the two main groups' work.
- Part 4: Testing, debugging, and finalization of the software.

3 State of the art

The state of the art concerning air signing with fingers involves the use of computer vision and machine learning techniques to recognize and interpret hand gestures. This technology is typically referred to as computer vision-based sign language recognition, and it relies on complex algorithms to detect and track the movement of hands and fingers in real-time.

Recent advances in deep learning and neural networks have greatly improved the accuracy and speed of sign language recognition systems. These systems can recognize and translate sign language into spoken language, enabling communication between people who are deaf or hard of hearing and those who are not. [7] In addition, wearable devices such as gloves and bracelets with embedded sensors can capture hand movements. [8]

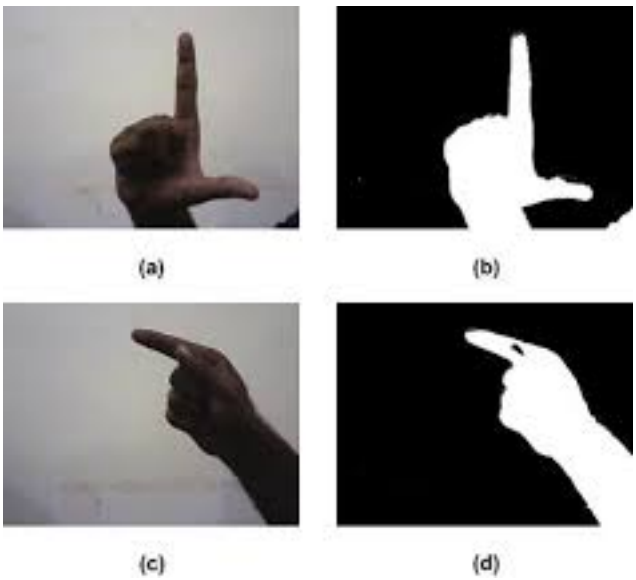

Figure 2: Translate sign language into spoken language[6]

Overall, the state of the art in air signing with fingers is rapidly advancing, and there is a great deal of potential for this technology to improve the lives of people.

Air finger drawing and tracking to sign typically require some form of technology or device to accurately track and interpret hand movements. While AI may be used in some cases to improve accuracy or recognize specific gestures, it is not necessarily required.
One example of a non-AI-based approach to air finger drawing and tracking to sign is using a motion tracking device such as a Microsoft Kinect[9] or Leap Motion [10]. These devices use infrared cameras to track the movement of your hands and fingers in real-time, allowing you to draw or sign in the air and have your movements translated into digital form.

Another approach is using a touchless interface that can detect the movement of your fingers through sensors or motion detectors. This can include devices like the Myo armband[11], which uses electromyography sensors to detect muscle movements in your arm and translate them into commands for controlling a computer or device.

Figure 3: Myo Armband[11]

It is important to note that while these non-AI-based approaches can be effective, they may not be as accurate or reliable as AI-based methods, especially for more complex gestures or sign language. Additionally, the technology required for these approaches may be less common or more expensive than AI-based alternatives. There are several traditional methods of hand recognition that do not rely on AI, including:

- Template matching[12]: This method involves comparing the shape of a hand to a predefined set of templates to determine the most likely match. The templates can be created based on the average shape of a hand or specific hand shapes of interest.
- Edge detection[13]: This method involves detecting the edges of a hand in an image and using them to define the shape of the hand. Various edge detection algorithms can be used to accomplish this, including the Canny edge detector and the Sobel operator.
- Contour analysis[14]: This method involves extracting the contour of a hand in an image and analyzing its shape and features. Various techniques can be used to extract the contour, including the watershed algorithm, the active contour model, and the level set method.

4 Finger Recognition and drawing with finger

We made a solution to recognise the hand and each finger and then we wanted to add the features to draw following our index.

4.1 Functioning

This code uses OpenCV[1] and MediaPipe[15] to create a program that detects the position of the index finger in the webcam feed and uses it to draw a curve on the screen.

Here's a brief explanation of each module used in the code:

- Cv2: OpenCV[1] (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision.
- Mediapipe: MediaPipe[15] is a cross-platform framework for building multimodal applied machine learning pipelines.
- Numpy[5]: NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

4.1.1 How it works

MediaPipe[16] Hands is a hand tracking solution developed by Google as part of the MediaPipe[16] framework. It uses a combination of computer vision techniques and machine learning models to detect and track the positions of individual hand landmarks in real-time video streams.

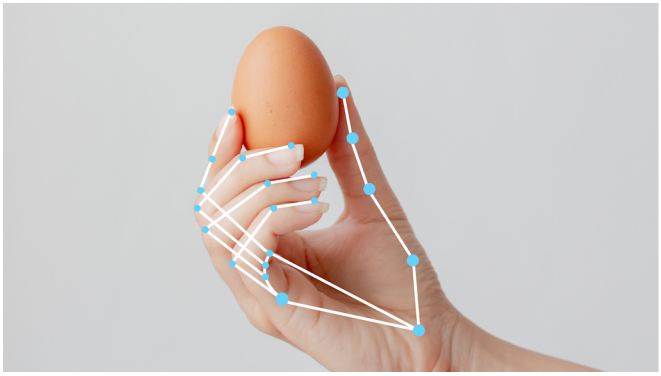

Figure 4: Hand detected with MediaPipe

The MediaPipe[15] Hands pipeline consists of several stages:

- Hand Detection: The first stage involves using a machine learning model to detect whether there is a hand in the input video stream. The model identifies potential hand regions in the video frames and generates a hand detection bounding box.
- Hand Landmark Estimation: The second stage involves using another machine learning model to estimate the 3D positions of 21 key landmarks on the detected hand. These landmarks include finger tips, joints, and the palm center.
- Tracking: The third stage involves using a Kalman filter to track the 3D positions of the hand landmarks over time. The filter predicts the next location of each landmark based on the previous positions and updates the predictions based on the new input data.

- Gesture Recognition: The final stage involves using the tracked hand landmarks to recognize specific hand gestures. For example, it can detect whether the hand is making a fist or an open palm.

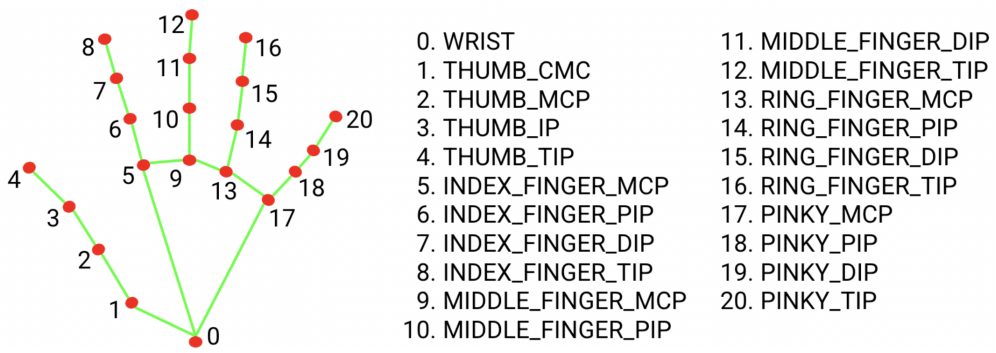

Figure 5: Hand Landmark Estimation

MediaPipe[15] Hands uses a deep neural network architecture called BlazePalm[**?**] for hand detection and a second architecture called BlazeHand for landmark estimation. Both models are lightweight and optimized for real-time performance on mobile and embedded devices.
Palms can be modelled using only square bounding boxes and therefore reducing the number of anchors by a factor between 3 and 5. Then an encoder-decoder feature extractor for a larger scene-context awareness even for small objects. Lastly, it minimize the focal loss during training to support a large amount of anchors resulting from the high scale variance

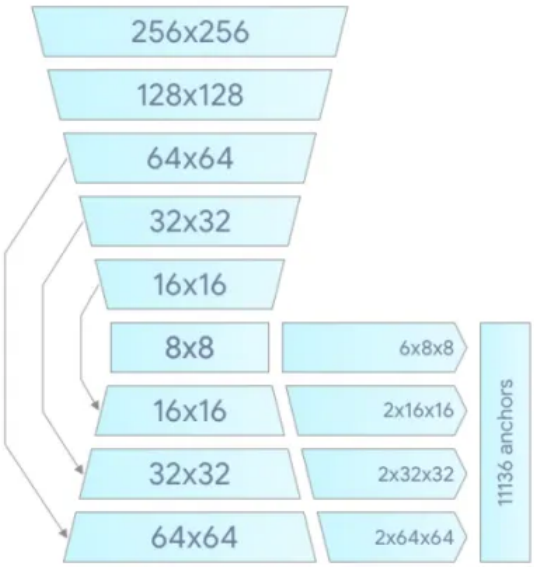

Figure 6: BlazePalm model architecture [15]

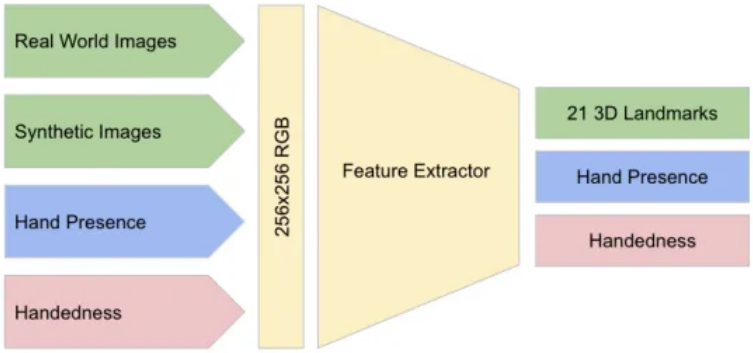

Figure 7: Hand Landmark model architecture [15]

MediaPipe[15] Hands also supports multi-hand tracking, meaning it can track the positions of multiple hands in a video stream. Additionally, it provides various configuration options that allow developers to customize the pipeline for specific use cases and hardware platforms.

Model Variation	Average Precision
No decoder + cross entropy loss	86.22%
Decoder + cross entropy loss	94.07%
Decoder + focal loss	95.7%

Table 1: Model variation on precision. [15]

Overall, MediaPipe[15] Hands is a powerful and efficient solution for hand tracking that can be used in a variety of applications, such as virtual reality, augmented reality, sign language recognition, and gesture-based interfaces.

4.1.2 The process of the algorithm

Here's how our algorithm works:

1. First, we use cv2 to provide computer vision capabilities for video processing. We continuously captures frames from the video stream.

2. Then, we initializes MediaPipe's hands[15] module to detect hand landmarks in the video stream. We then draw the landmarks and the connections between them.

3. If the program detects a hand in the captured image, it gets the landmark coordinates of the index finger that we also mark with a circle to be sure of his position.

4. We then draw a curve on the screen following the index finger and finally get an output image of our drawn signature.

Figure 8: Finger recognition and drawing without depth detection

This code can be modified to draw various shapes or lines. With this we can air-sign with our index and see our signature.

5 Depth estimation for writing on and off

The objective is to estimate the depth of the writing hand in order to determine whether the finger movement should be taken as writing or not, in the same way as when you lift the pen from the paper.

5.1 ANN Model - MiDaS

The most logical approach would have been to create and train our own model. Indeed, datasets on depth prediction and estimation are available, for example the NYU V2 Depth dataset[17]. Unfortunately, these datasets are more than 400,000 images long and weigh more than 400 GB. As we do not have a supercomputer at our disposal, training would take months. Indeed, training with images is much faster using GPUs instead of CPUs, another point is that training requires the model to learn the dataset more than a hundred times. We therefore turned to models already trained and made available in open source by companies on platforms such as Hugging Face.

We found the Midas model[18]. It has been trained partly on the NYU V2 Depth dataset[17] and partly on other datasets. This model has been trained in part by the company INTEL and Pytorch[3] which has supercomputers. MiDaS (Mixed Depth Estimation of absolute Scale)[18] is a deep learning-based approach for monocular depth estimation, which means it can estimate the depth of an image from a single 2D image input, without requiring stereo or other depth input.
MiDaS[18] is based on a convolutional neural network (CNN) architecture that has been trained on a large dataset of approximately 2 million images with depth information with more than 70 million parameters. The network takes an input image as input and produces a corresponding depth map, which represents the distance from the camera to each pixel in the image. This depth map can be used for a wide range of computer vision tasks, such as virtual and augmented reality, autonomous driving, robotics, and more.

Figure 9: MiDaS samples

What sets MiDaS[18] apart from other depth estimation techniques is that it uses a mixed-scale approach, where the network learns to estimate depth at multiple scales simultaneously, and combines these estimates to produce a final depth map. This allows MiDaS[18] to capture

both large-scale and small-scale depth features in the image, resulting in more accurate depth estimates.
MiDaS[18] has a lightweight version, MiDaS Small[18], that offer the opportunity to compute real-time depth estimation. With only 1 million parameters, that makes it much faster to run on devices with limited computational resources. However, this also reduces the accuracy, as MiDaS Small[18] cannot capture as much fine-grained detail in the depth map as the larger model.

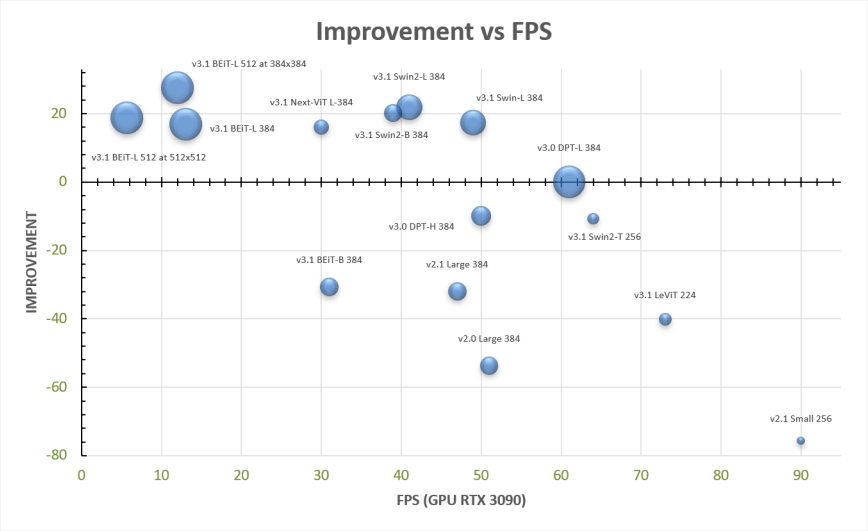

Figure 10: MiDaS version comparison

Initially, we aim to use the MiDaS[18] model to estimate the frame-by-frame depth of a live video stream performed by the user signing at a distance. We thus obtain the relative depth between the different objects present in the image. The writing boundary was determined as the threshold at which the user's finger exceeded half the distance between his face and the camera. Thus, we can rely on a fixed object close enough to the camera to have an accurate estimation of the depth: the user's face is our reference plane. We then implemented a bounding box face detection model using MediaPipe[15]. This allows us to roughly delimit the area in which the user's face is located, our reference plane. The depth video produced using MiDaS[18] has 30 frames per second with each frame in grayscale. By calculating the arithmetic mean between the intensity value of the pixels in the area detected as the face and the maximum intensity value of the pixels: 255 (which theoretically corresponds to an object stuck to the camera), we obtain a threshold at equal distance between the head and the camera lens. The advantage is that this threshold is not fixed and therefore it adapts to the user more or less close.

Figure 11: Depth estimation B&W

The next step is to determine if the user's finger is in the writing area or not. For this, we take the intensity value of the pixels located on the finger detected thanks to MediaPipe[15], described in the finger tracking part, and we compare it with the threshold value calculated for each frame. This process is repeated for each frame, i.e. 300 times for a 10 second video: estimation of the image depth, detection of the face, calculation of the threshold, comparison with the intensity on the finger.

5.2 Image processing method

This other solution is based triangle similarity results from the processing of the image to determine the distance with the finger. Triangle similarity is a defines as follow.
Two triangles are similar if :

- Two pairs of corresponding angles are equal,
- Three pairs of corresponding sides are proportional,
- Two pairs of corresponding sides are proportional and the corresponding angles between them are equal.

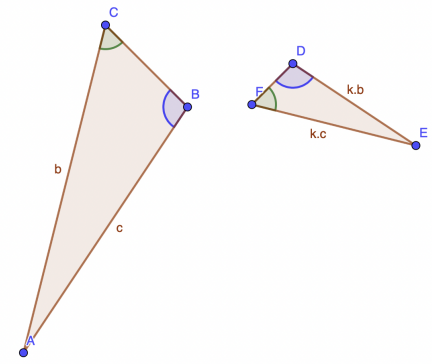

Figure 12: Triangle similarity

Let's define an object of fixed width W and place it at a distance D from the camera lens. According to the computer and the camera, the object has a width P in pixel. We can compute perceived focal length F (F is a fixed value depending on the camera) :

$$F = \frac{D.P}{W} \tag{1}$$

As the object moves, the width P change and by applying the triangle similarity, we can calculate the new distance D' between the object and the camera lens.

$$D' = \frac{F.W}{P} \tag{2}$$

Figure 13: Depth calculation [20]

In our case, the object to detect the distance from the camera is the hand of the user. Therefore we need to detect the user's hand and measure its width. For this purpose, we use the hand detection feature of MediaPipe[15] to compute a bounding box of the hand.

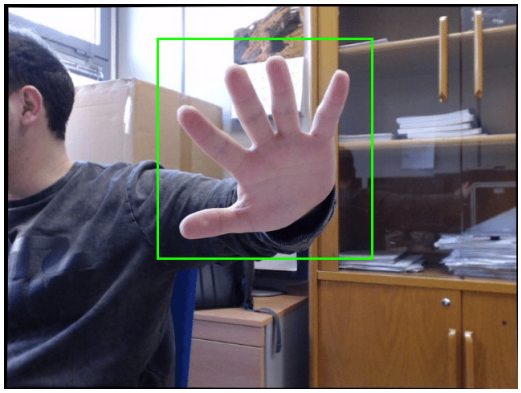

Figure 14: Hand bounding box

This method requires a calibration phase. During this phase the user positions his hand in front of the camera and clicks on the Start button to record the reference distance. Thereafter, the distance will be calculated at each image captured by the video stream. When the calculated distance is less than the reference distance, the user can write with his finger. Otherwise, the writing will be disabled and will resume only when the finger is back in the writing area.

5.3 Comparison

After studying both methods, we chose the image processing method which is faster and less resource consuming. Indeed, the methods based on deep learning did not correspond to one of the most important criteria of the specifications: the speed of execution. The lightest and fastest model alone takes more than a minute (68.4 seconds) to compute a 20 second video. On average, in a live stream, this model is able to process a video stream with a frame rate of 8.7 fps. Nevertheless, we could process the video not live but in post-processing, after it has been recorded. Unfortunately, the performance provided by MiDaS Small model[18] is too low and there is too much loss of precision to be able to correctly determine the writing area. The risk of producing an unreadable and unreliable signature is very high. In order to improve the performance, we could use one of the two other versions of MiDaS[18], Hybrid and Large[19], which are slower but more reliable. The Hybrid version, moderately fast, offers acceptable performances for our use, in particular a frank and precise segmentation of the various depth planes of the image. However, this model is far too time and resource consuming for an efficient use: almost an hour for 20 seconds of video. Deep Learning methods do not seem to be adapted to a general public use of our solution, without hardware configuration allowing to improve the computing capacities of the computer. Finally, the image processing method fulfills all the criteria: a speed of execution allowing to process a live stream and optimal precision which offers a reliable and efficient use of our solution.

Model Variation	Processing time
MiDaS v3 - Large	12.923 s/image
MiDaS v3 - Hybrid	5.984 s/image
MiDaS Small	0.114 s/image
Triangle similarity OpenCV	3.02×10^{-6} s/image

Table 2: Model variation on processing time

6 Conclusion

To conclude, we have used the different techniques mentioned above. To detect the hand we used MediaPipe[15] and to detect the depth we used the image analysis technique. Indeed, these are the methods that provided us with the best results in both live and non-live video. Then we developed, thanks to Tkinter, a simple user interface that allows to quickly take control of the application.
Thus the first step, once the program is launched, is to calibrate the application to locate the hand position spatially. The second step is to draw the signature by raising the finger to mark the cuts, then to validate. The signature is then displayed and can be saved to be used in a document for example(15).

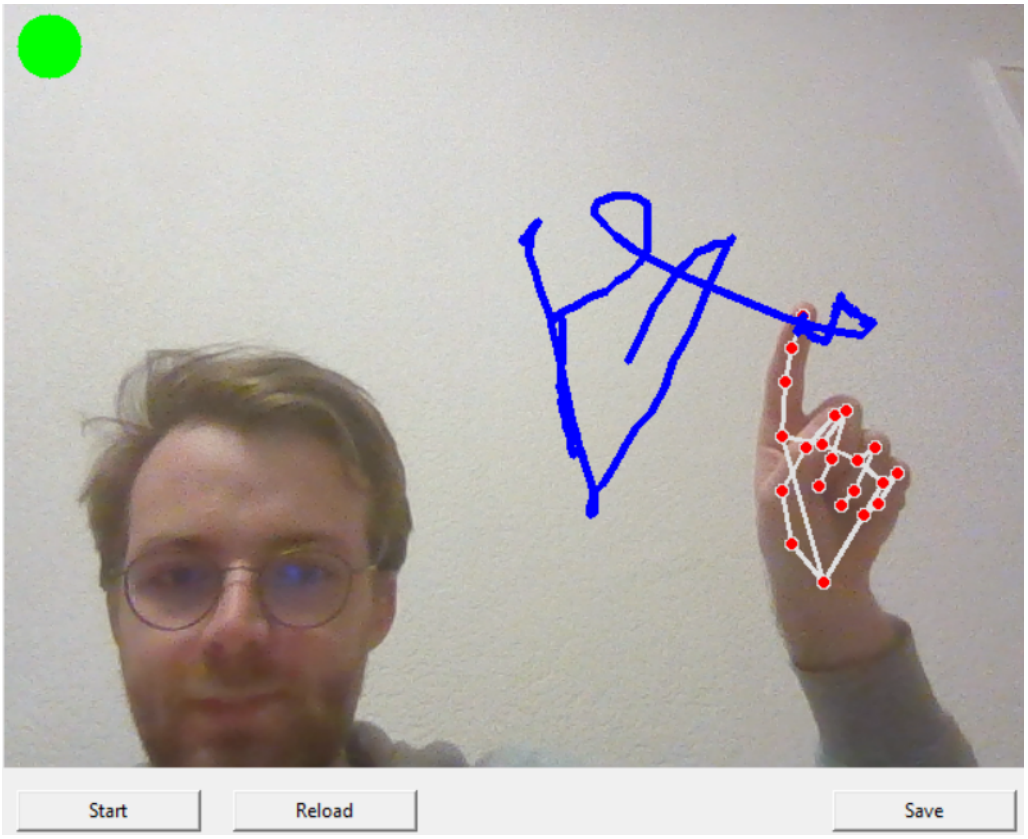

Figure 15: UI presentation

To test yourself you can visit this github directory, the front page explains the installation:

`https://github.com/FallenElias/Finger_AirSigning_Project`

We are satisfied with the results, it has a good precision that allows to make a precise signature and that can be really used in documents.

References

[1] OpenCV: Bradski, G., 2000. The OpenCV Library. Dr. Dobb's Journal of Software Tools. URL: https://opencv.org/

[2] TensorFlow: Abadi, Martin et al., 2016. Tensorflow: A system for large-scale machine learning. In 12th *USENIX* Symposium on Operating Systems Design and Implementation (*OSDI* 16). pp. 265–283. URL: https://www.tensorflow.org/

[3] PyTorch: Paszke, A. et al., 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[4] Tkinter: Lundh, F., 1999. An introduction to tkinter. URL: www.pythonware.com/library/tkinter/introduction/index.htm.

[5] Harris, C.R., Millman, K.J., van der Walt, S.J. et al. Array programming with NumPy. Nature 585, 357–362 (2020). DOI: 10.1038/s41586-020-2649-2.

[6] Karan Bhavsar, Raj Ghatiya, Aarti Gohil, Devanshi Thakkar, Bhumi Shah. (2021). International Journal of Research Publication and Reviews : Sign Language Recognition. ISSN 2582-7421. https://ijrpr.com/uploads/V2ISSUE9/IJRPR1329.pdf URL : https://ijrpr.com/uploads/V2ISSUE9/IJRPR1329.pdf

[7] Artificial Intelligence Technologies for Sign Language. I. Papastratis, C. Chatzikonstantinou, D. Konstantinidis, K. Dimitropoulos, P. Daras. URL: https://www.mdpi.com/1424-8220/21/17/5843

[8] Haratian, R. Motion Capture Sensing Technologies and Techniques: A Sensor Agnostic Approach to Address Wearability Challenges. Sens Imaging 23, 25 (2022). URL: https://doi.org/10.1007/s11220-022-00394-2

[9] Kinect : Zhang, Zhengyou. (2012). Microsoft Kinect Sensor and Its Effect. IEEE Multimedia - IEEEMM. 19. 4-10. 10.1109/MMUL.2012.24. URL: https://www.researchgate.net/publication/254058710_Microsoft_Kinect_Sensor_and_Its_Effect

[10] Leap Motion : The world's most advanced hand tracking. URL: https://www.ultraleap.com/tracking/

[11] Myo Armband : Visconti, Paolo & Gaetani, Federico & Zappatore, Giovanni & Primiceri, Patrizio. (2018). Technical Features and Functionalities of Myo Armband: An Overview on Related Literature and Advanced Applications of myoelectric Armbands Mainly Focused on Arm Prostheses. International Journal on Smart Sensing and Intelligent Systems. 11. 1-25. 10.21307/ijssis-2018-005.

[12] Sibiryakov, Alexander. (2011). Fast and high-performance template matching method. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 1417-1424. 10.1109/CVPR.2011.5995391.

[13] Canny, John. (1986). A Computational Approach To Edge Detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on. PAMI-8. 679 - 698. 10.1109/TPAMI.1986.4767851.

[14] Gong, Xin-Yi & Su, Hu & Xu, De & Zhang, Zhengtao & Shen, Fei & Yang, Hua-Bin. (2018). An Overview of Contour Detection Approaches. International Journal of Automation and Computing. 15. 1-17. 10.1007/s11633-018-1117-z.

[15] MediaPipe: Camillo Lugaresi, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, ChuoLing Chang, Ming Guang Yong, Juhyun Lee, et al. Mediapipe: A framework for building perception pipelines. arXiv preprint arXiv:1906.08172, 2019. URL: https://arxiv.org/pdf/2006.10214.pdf

[16] MediaPipe: Hand landmarks detection guide. URL: https://developers.google.com/mediapipe/solutions/vision/hand_landmarker

[17] NYU DEPTH V2:Nathan Silberman, Derek Hoiem, Pushmeet Kohli and Rob Fergus. Indoor Segmentation and Support Inference from RGBD Images. ECCV . 2012

[18] MiDaS : Ranftl and Katrin Lasinger and David Hafner and Konrad Schindler and Vladlen Koltun. Ranftl2020. Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI). 2020. https://pytorch.org/hub/intelisl_midas_v2/

[19] DPT: Ranftl and Alexey Bochkovskiy and Vladlen Koltun. Vision Transformers for Dense Prediction. DBLP:journals/corr/abs-2103-13413, 2021. https://arxiv.org/abs/2103.13413

[20] Depth : Adrian Rosebrock https://pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/