INF573 FINAL REPORT

Interactive Games with Gesture Detection

December 2021

Vincent Geffroy, Xiyao Li

1 Introduction

We decided to go into the field of gesture detection with some Machine Learning solutions. We began with hand detection for a very classical game, Rock Paper Scissors, which aims to create a game platform to detect players' gestures and determine players' choice. So the player could play Rock Paper Scissors with the computer. We found a very well-trained Google library Mediapipe to do the face/hand/object detection by Machine Learning solutions. And at the same time, we tried to develop our method for gesture detection by using a colour filter.

The Google library Mediapipe returns high-fidelity 3D hand keypoints with short processing time, then we realised that we could implement some more complex projects with it. Therefore, we were motivated to implement a drawing board with different gestures representing pencil and eraser and a Kalimba for music playing.

2 Main Methods Used

2.1 Hand Detection using OpenCV

At first, we tried to implement our hand and finger detection using OpenCV without Machine Learning/Deep Learning methods. The idea is to ask the user to click several times on their hand with different gestures for extracting its colour scale. Then with the scale obtained, we build a Skin Mask with a similar colour and find its contour. Theoretically, one convexity defect can be detected between two fingers. So by counting the number of defects, the algorithm can tell the number of open fingers. For example, for no defect, it is a rock gesture ; for one or two defects, it is a scissors gesture ; and for more than three defects, it is a paper gesture.

To go into more technical details, here is a more precise description : [Mis]

— **Skin Mask :** We first ask the user to click multiple times on different parts of his hand in different gestures. We store the values of the pixels clicked (HSV value) in a list. We then take the 5 percentile value as a lower bound and the 95 percentile as a upper bound. This allow to filter the results and cut off some noise.

We then use this lower and upper bound in order to obtain a mask of the hand. We first convert the BGR image to an HSV image, and then we search pixels whose HSV value fall between the lower and upper bound. This pixels are considered as skin. Then we apply blurring and threshing functions to improve the mask quality. Our results can be seen on figure 1.

— **Contours and Convexity Defects :** With the functions already integrated into OpenCV, we can find the contours and the convex hull easily. The convex hull of a shape is the smallest convex set that contains it, which helps detect the convexity defects in the next step.

The convexity defect is defined as the points furthest from the convex points. It can be depicted as the calculated difference between the convex hull and the contour. So, if the finger tips are considered as the convex points, the trough between the fingers can also be considered as convexity defects.[Fio+19]

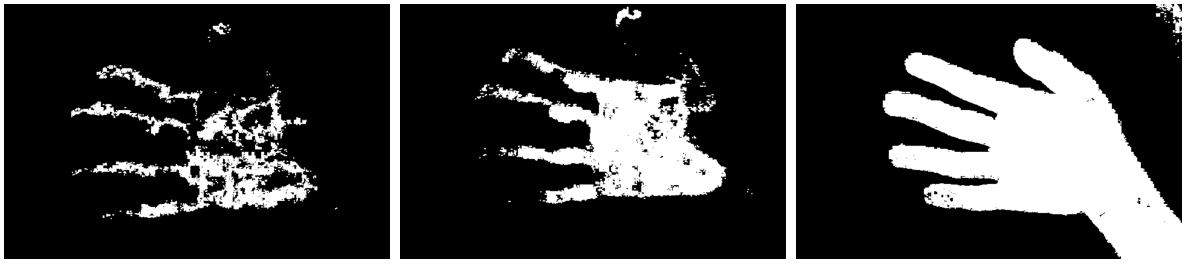


Figure 1 – Skin Mask with 3 clics / 10 clics / 30 clics

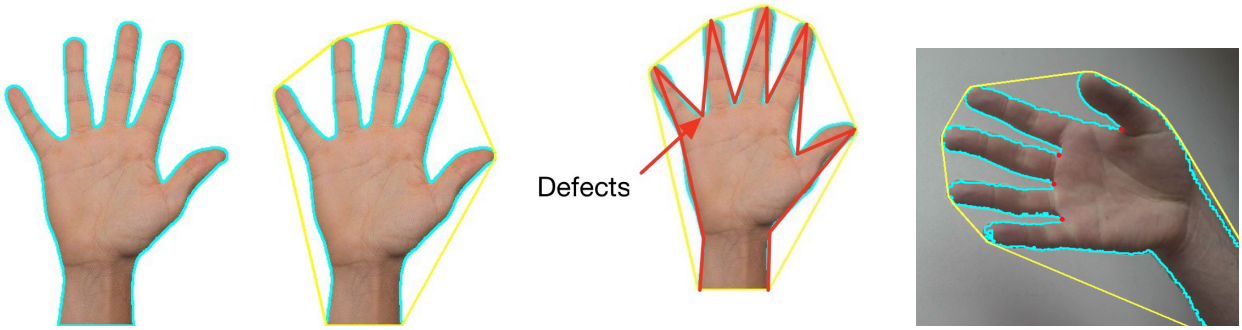


Figure 2 – Contours (blue), Convex Hull (yellow) and Convexity defects (red)

— **Fingers Recognition :** The OpenCV function `convexityDefects()` returns an array whose each row contains four values `(start point, end point, furthest point, approximate distance to furthest point)`, corresponding to `(convexity defect, tip of one finger, tip of the other finger, approximate distance between two tips)`. From these values and with regard to the notations of figure 3, we can obtain the lengths of a, b and c and then compute the angle θ with :

$$\theta = \arccos\left(\frac{a^2 + b^2 - c^2}{2ab}\right)$$

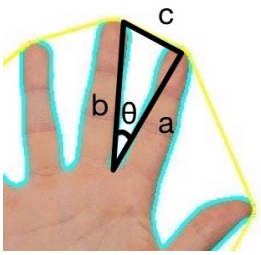


Figure 3 – Recognise fingers by calculating the angle

In our algorithm, if the angle θ is less than a given value θ_0, it is considered as a point between two open fingers.

We find that $\theta_0 = \frac{\pi}{4}$ is a good choice. For a smaller θ_0 we miss some points and for a larger value of θ_0 we find points that are not in between fingers. (see Figure 4)

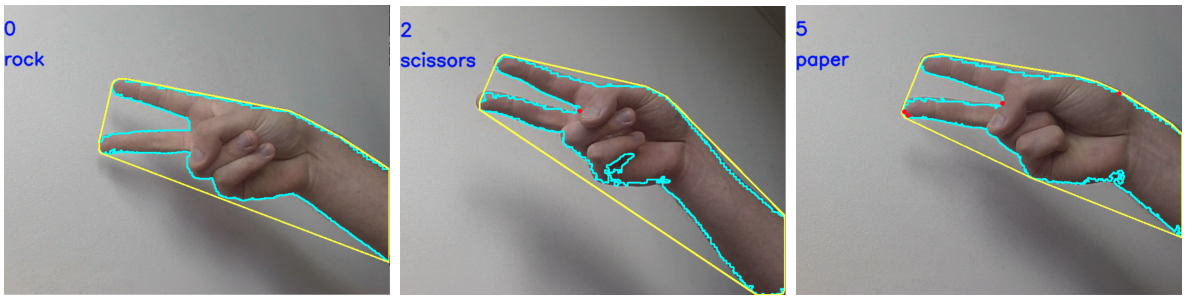


Figure 4 – Number of computed points in between finger for $\theta_0 = \frac{\pi}{2}/\frac{\pi}{4}/\frac{\pi}{8}$

This algorithm is very restricted. Determining the user's gestures is only based on the skin mask and on the number of convexity defects. The simplicity of the model is source of many deficiency. In particular the use of a colour-scale based detection lack in robustness. When one's head and hands are both in the camera, or even when some colours in the background are near the skin tone of the person, the mask will take everything into account and the result won't be useful. This algorithm is also dependant on lighting and the environment, when the external condition change, the lower and upper bounds need to be computed again. (see figures 5-6)

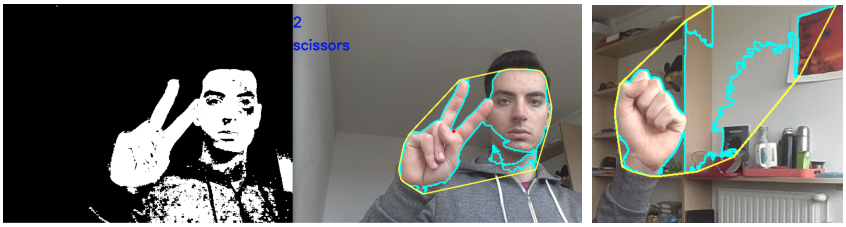


Figure 5 – Errors when colors near the hand skin tone are found elsewhere

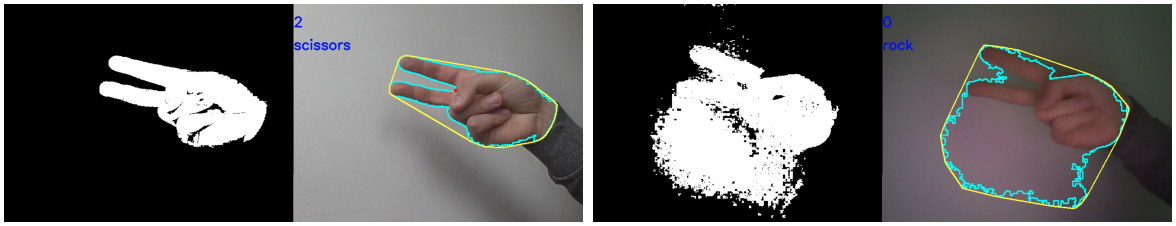


Figure 6 – Errors when the lighting changes

To resolve these problems, we work out three ideas. The first is to detect the background and give it a different colour. The second is to detect the faces for making the distinction with the hands. And the third is to normalise the lighting. But each of these purposes will take a long time, so we found a well-developed library Google Mediapipe to carry out our project.

2.2 Google Library Mediapipe

Mediapipe is a library that integrates many Machine Learning solutions for live and streaming media. It contains Face Detection, Hand Detection, Pose Classification, 3D Object Detection, etc. Here we only use the Mediapipe's Hand Detection.

Perceiving the shape and motion of hands comes naturally to people. However, robust real-time hand perception is a challenging computer vision task, as hands often occlude themselves or each other (e.g. finger/palm occlusions and handshakes) and lack high contrast patterns. Mediapipe employs machine learning methods to infer 21 3D landmarks of a hand from just a single frame to offer a high-fidelity hand and finger tracking solution. [Goo]

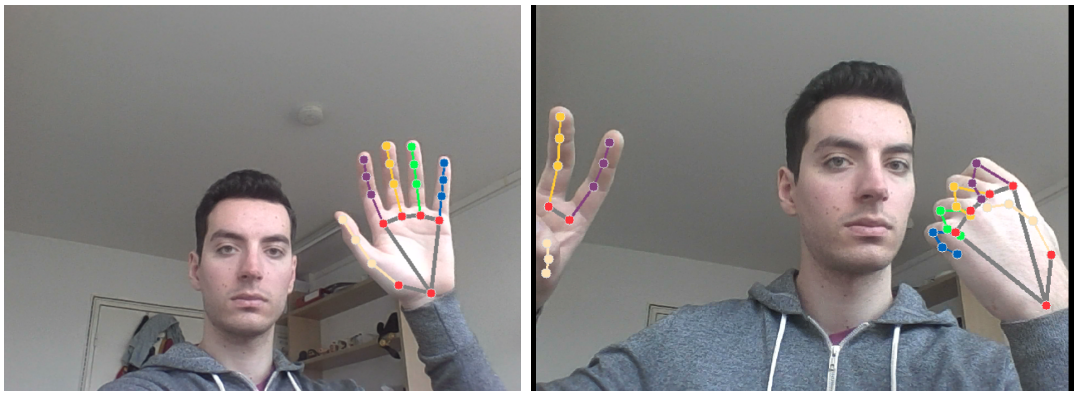


Figure 7 – Mediapipe's Hand Detection : detection of one hand, of multiple hands, of partial hands, of backward hands

2.2.1 Palm Detection Model

Detecting hands is a complex task : whereas faces have high contrast patterns (i.e. in the eye and mouth region), the lack of such features in hands makes it comparatively difficult to detect them reliably from their visual features alone. Instead, providing additional context, like arm, body, or person features, aids accurate hand localisation.

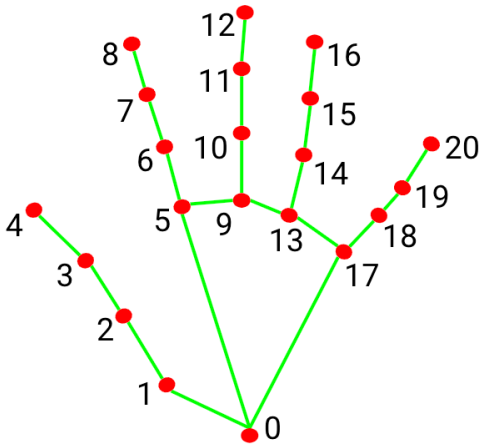
So the library combines different strategies. First, it trains a palm detector instead of a hand detector, since estimating bounding boxes of rigid objects like palms and fists is significantly simpler than detecting hands with articulated fingers. In addition, as palms are smaller objects, the non-maximum suppression algorithm works well even for two-hand self-occlusion cases, like handshakes. Moreover, palms can be modelled using square bounding boxes (anchors in ML terminology), and therefore reducing the number of anchors by a factor of 3-5. Second, an encoder-decoder feature extractor is used for bigger scene context-awareness even for small objects. Lastly, the algorithm minimises the focal loss during training to support a large number of anchors resulting from the high scale variance.

With the above techniques, the package achieves an average precision of 95.7% in palm detection, while using a regular cross-entropy loss and no decoder gives a baseline of just 86.22%.

2.2.2 Hand Landmark Model

After the palm detection over the whole image, the landmark model performs precise keypoint localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression. The model learns a consistent internal hand pose representation and is robust even to partially visible hands.

To obtain ground truth data, the developer team have manually annotated 30K real-world images with 21 3D coordinates. The Z-valueis taken from image depth map. To better cover the possible hand poses and provide additional supervision on the nature of hand geometry, they also render a high-quality synthetic hand model over various backgrounds and map it to the corresponding 3D coordinates.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figure 8 – Hand Landmarks

We have access to the 21 points of the hand (or both hands if two are present). For each points we have access to their (x,y,z) coordinates. With these informations we can do a lot of things. We created three small games using Mediapipe.

3 Implementations

For the layout, we used the library Pygame, a python library that allows the creation of simple games. As a starting point, we have initially taken the code : [Can], in which we learned how to use Pygame and Mediapipe. But at the end of the project, most of the code has been changed.

To describe Mediapipe's integrated methods, we will use the notation Vector(i, j) representing the vector from hand-landmark i to hand-landmark j.

3.1 Home Page

When you launch the game (with command - `python3 main.py`), a menu appears with the 4 games we created. The `Quit` button closes the terminal.

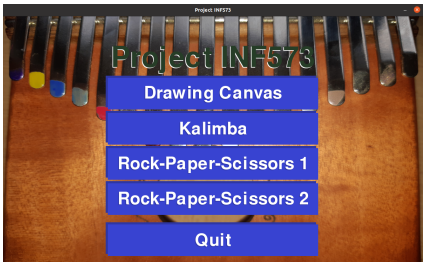


Figure 9 – Home Page / Hand Landmarks

3.2 Games 1 : Drawing Canvas

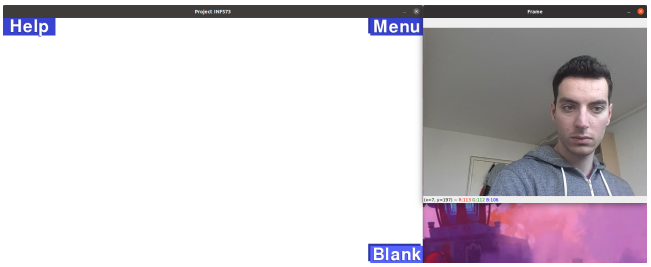


Figure 10 – Drawing Canvas with no hand detected

When the drawing canvas opens, we get a whiteboard to draw. Three buttons are available : the "Help" button gives information on how to play this game, the "Menu" button gets back to the homepage and the "Blank" button erases every drawing.

We use Mediapipe to detect the presence of one or two hands. At the same time, the algorithm distinguishes the left and right hand. Then we look at the hands' position for deciding what tool to use.

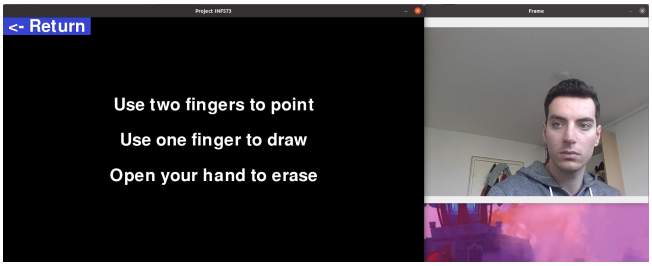


Figure 11 – Example of "Help" window

Aiming tool :

If index finger and middle finger are straight ($\text{Vector}(0,6)\cdot\text{Vector}(6,8) > 0$ and $\text{Vector}(0,10)\cdot\text{Vector}(10,12) > 0$) and ring finger and little finger are bent ($\text{Vector}(0,14)\cdot\text{Vector}(14,16) < 0$ and $\text{Vector}(0,18)\cdot\text{Vector}(18,20) < 0$) then we consider that this hand is using the aiming tool.

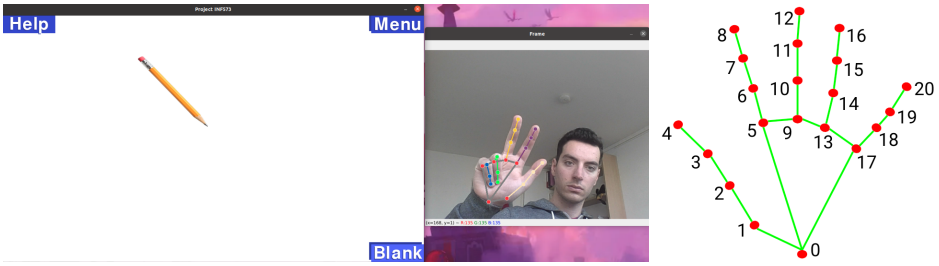


Figure 12 – Using the aiming tool / Hand Landmarks

In this tool, a big pencil appears on the screen, which is oriented corresponding to the hand used, the tip of to pencil corresponds to the cursor. It allows the user to aim at some point without drawing.

Drawing tool :

If the index finger is straight, the middle finger, ring finger and little finger are bent, we consider this hand using the drawing tool.

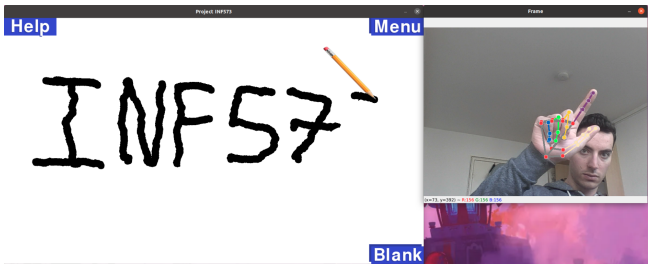


Figure 13 – Using the drawing tool

In this tool, a small pencil appears on the screen, which is oriented corresponding to the hand used, the tip of to pencil corresponds to the cursor. It allows the user to draw.

Problem : if we only add a point where the pencil is, when we have a quick movement of the hand, we get a succession of points and not a line.

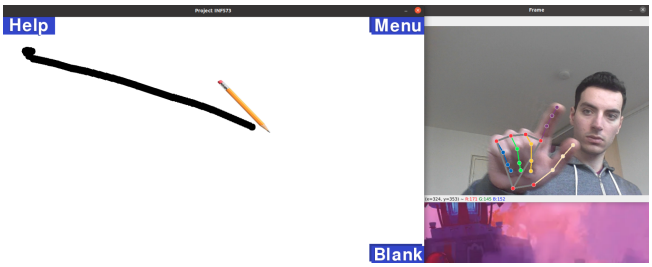


Figure 14 – The hand can move a lot between frames !

To solve this problem, we store the state of the pencil. If we were already drawing in the previous frame, we link two points by a line. If not, we draw just one point.

Eraser tool :

If the index finger, middle finger, ring finger and little finger are bent, then we consider that this hand is using the eraser tool.

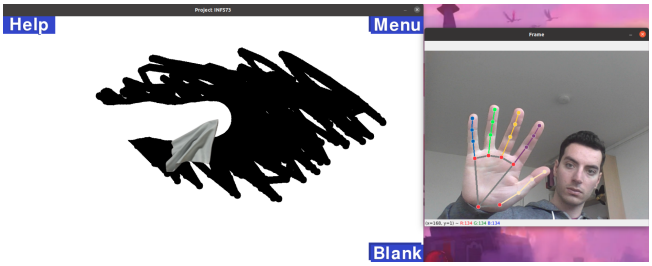


Figure 15 – Using the eraser tool

In this tool, tissue paper appears on the screen, which is oriented corresponding to the hand used, the tip of to towel corresponds to the cursor. It allows the user to erase. As for the drawing tool, we draw a line and not simply a succession of the white dot.

Problem : How to deal with two hands at the same time ?

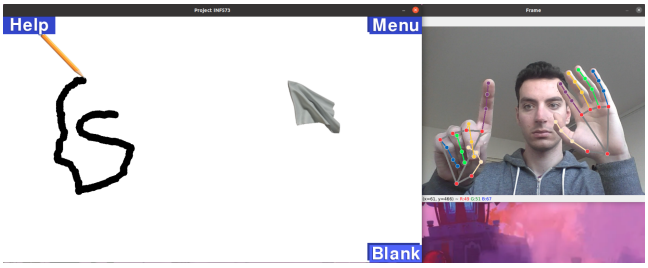


Figure 16 – Two hands in action with different tools for each hands

Mediapipe allows using two hands at the same time, with a function that distinguish the left and right hand. However, the problem is that we need to be careful when we draw lines

with one hand and do something else with the other. Therefore we need to store independently the position of each hand.

3.3 Games 2 : Kalimba

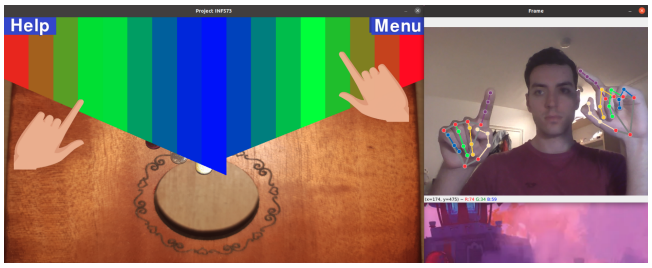


Figure 17 – Kalimba with two hands detected

The second game allows you to play the kalimba. As for the drawing canva, you can find a "Help" and a "Menu" button.

In this game, we divide the space into two parts : the upper part with keys and the lower part without keys. We then divide the upper space into 17 parts along the x-axis, one part for each key. For each key, we have recorded its sound on an actual kalimba.

Mediapipe will find the tip of the index finger (hand-landmark 8) and place the cursor there. The corresponding hand appears on the screen according to what the user puts. For each hand and frame, we will look if the tip of the index finger is in the lower space and if the index finger in the previous frame was in the upper part. If so, we play the note corresponding to the key on which the finger was in the previous frame.

3.4 Games 3 : Rock-Paper-Scissors 1

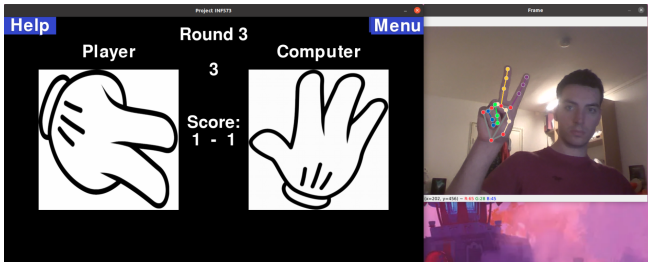


Figure 18 – Playing Rock-Paper-Scissors with Mediapipe

This third game allows you to play Rock-Paper-Scissors using Mediapipe.

The gesture detection of rock, paper and scissors is very straightforward. It depends only on the hand landmarks coordinates computed from Mediapipe. In our algorithm, we detect the

open and close status of each finger (see Game 1 to know how scalar product can tell you if one finger is open or not).

If the algorithm detects that the index, middle, ring and little fingers are all in close status, the player's gesture is rock. If the index, middle, ring and little fingers are all in open status, it is paper. And if the index and middle fingers are open and the ring and little fingers are closed, it is scissors. In case that no hand or a wrong configuration of fingers is detected, the game marks it as an error.

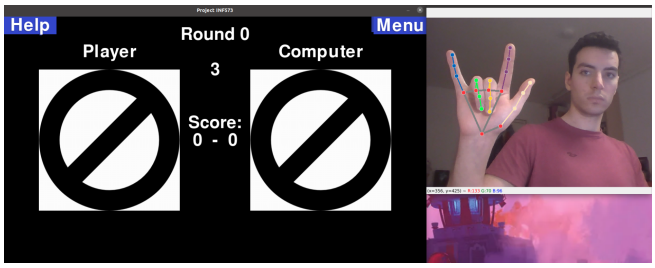


Figure 19 – Wrong configuration of fingers

In each round of this game, we give 3 seconds to the player to choose a sign. After three seconds, the computer selects a sign randomly and then compute the score. The first to arrive with 3 points wins. If an error occurs at the end of the 3 seconds, we restart the round.

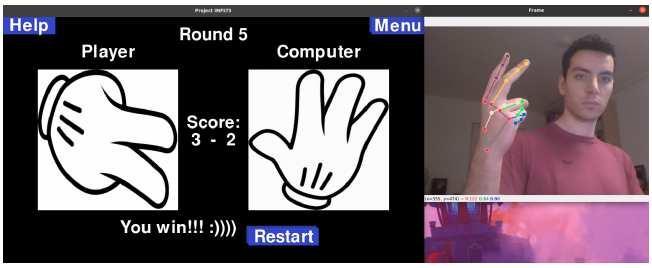


Figure 20 – You win!!!

At the end of the game, whether you won or lost, a restart button appears, allowing you to start the game again. Else, you can go back to the menu.

3.5 Games 4 : Rock-Paper-Scissors 4

This fourth game allows playing Rock-Paper-Scissors with the naive method explained in 2.1. We first need the user to calibrate the lower and upper bound of the skin tone colour. Then when Game 4 opens, we start with the user's calibration.

Once the results convince us and the right-hand gesture is detected, we can click on the "Start" button. If a problem occurs during the calibration, we can click on the "Reset" button to start the calibration again. Then we use the same mechanics as for Game 3.

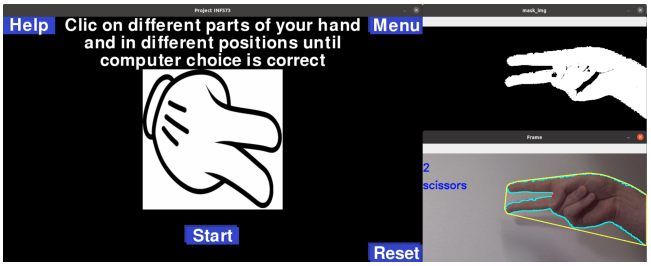


Figure 21 – Calibration off the color range

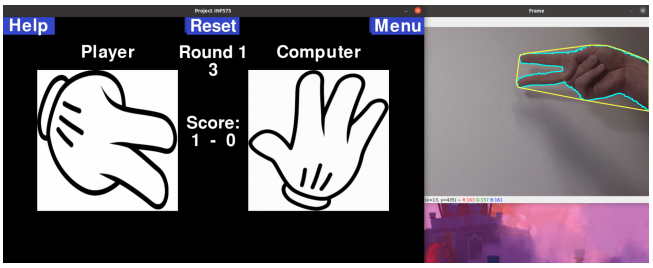


Figure 22 – Playing Rock-Paper-Scissors without Mediapipe

4 Conclusion

In this project, we confront the difficulty of hand gesture recognition for multiple applications. We found a simple colour-based hand detection that gives good results in a controlled environment. But this algorithm has too many weaknesses to be usable in practice. While trying alternative methods, we came across Mediapipe, a library that gives a stable functionality that can be used in different scenarios.

The 3D hand landmarks can be widely used in many more complex problems, like sign language translation. While the sign is static (like the paper or scissor sign), we can apply Machine Learning to these already saved landmarks. And wile the sign is dynamic, we can store these landmarks every x second and do machine learning on them. The possibility of Mediapipe seems endless.

Références

[Fio+19] Caroline El Fiorenza et al. “Hand Gesture Recognition using Convexity Defect”. In : *International Journal of Innovative Technology and Exploring Engineering* 9 (2019), p. 1161-1165.

[Can] Sergio Canu. *How I built a Computer Vision Game with Opencv, Mediapipe and Python*. URL : https://pysource.com/2021/08/24/how-i-built-a-computer-vision-game-with-opencv-mediapipe-and-python/.

[Goo] Google. *MediaPipe Hands*. URL : https://google.github.io/mediapipe/solutions/hands.

[Mis] Madhav Mishra. *Hand Detection and Finger Counting Using OpenCV-Python*. URL : https://medium.com/analytics-vidhya/hand-detection-and-finger-counting-using-opencv-python-5b594704eb08.