Hand Detection Applications

Final project CS 415

Davide Bartoletti

UIN 662965513

December 2, 2022

1 Introduction

Computer vision is a field of artificial intelligence that trains computers to interpret and understand the visual world. Using digital images from cameras and videos and deep learning models, machines can accurately identify and classify objects — and then react to what they "see". I was always attracted by this field. My main purpose would be to exploit computer vision in order to control a robotic arm. For this reason I chose the field of detection, that may include body detection and hand or face detection. The project has the aim to explore different applications based on hand detection, thanks the usage of OpenCV and Mediapipe libraries. In this project we develop an interface where a user can choose between different fancy usages of the hand detection module. We propose a finger painter, the game rock paper scissor and the mouse controller .

1.1 Libraries

Our work is based on some important libraries:

- **OpenCv**: Huge open-source library for the computer vision, machine learning, and image processing and now it plays a major role in real-time operation which is very important in today's systems. By using it, one can process images and videos to identify objects, faces, or even handwriting of a human.

- **Mediapipe**: a Framework for building machine learning pipelines for processing time-series data like video, audio, etc. This cross-platform Framework works on Desktop/Server, Android, iOS, and embedded devices like Raspberry Pi and Jetson Nano. It was initially developed for real-time analysis of video and audio on YouTube. Gradually it got integrated into many more products. In our case we exploit MediaPipe Hand, a high-fidelity hand and finger tracking solution. It employs machine learning (ML) to infer 21 3D landmarks of a hand from just a single frame.

- **Pyautogui**: lets Python scripts to control the mouse and keyboard to automate interactions with other applications. The API is designed to be simple and we can take actions like click the left or right botton of the mouse, or move the pointer in a specified point of the screen.

- **Pycaw**: Python Core Audio Windows Library, used in order to control the audio of computer in the mouse controller modality

2 Hand Tracking Module

The ability to perceive the shape and motion of hands can be a vital component in improving the user experience across a variety of technological domains and platforms. For example, it can form the basis for sign language understanding and hand gesture control, and can also enable the overlay of digital content and information on top of the physical world in augmented reality. While coming naturally to people, robust real-time hand perception is a decidedly challenging computer vision task, as hands often occlude themselves or each other. MediaPipe Hands utilizes an ML pipeline consisting of multiple models working together: A palm detection model that operates on the full image and returns an oriented hand bounding box. A hand landmark model that operates on the cropped image region defined by the palm detector and returns high-fidelity 3D hand keypoints. Providing the accurately cropped hand image to the hand landmark model drastically reduces the need for data augmentation (e.g. rotations, translation and scale) and instead allows the network to dedicate most of its capacity towards coordinate prediction accuracy. The fundamental block in the code of this project is the handDetector class that allows to find hands and the relative positions of the different landmarks with respect to the image captured from the camera. First of all we have imported the cv2 and mediapipe modules. The first one is used to obtain frames from the camera (*VideoCapture(0)*), the second one to get the landmarks (thanks the class *HandDetector*). The HandDetector class is characterized by different parameters:

- **static_image_mode**: is used to define if the image has to be considered as a static image or as a video stream.

- **max_num_hands**: indicates the max number of hands that can be detected.
- **min_detection_confidence**: indicates the value of confidence that the detection module has to reach to be considered successful. This is a value between 0 and 1. For some applications is left with the default values, for the painter one the value is increased in order to have a more accurate drawing.
- **min_tracking_confidence**: indicates the value of confidence to track the landmarks in a successful way.

First of all this module has to see if any hand is present in the image and, if this condition is satisfied, it draws a circle for each landmark and a line connecting them in the correct way, as shown in the image below. Each hand is characterized by 21 different detected landmarks, in order to be able to track many different movements. This module is very accurate thanks to the training on 30000 hand images. The fundamental step is to track continuously the important points involved in the different applications, the *findPosition* function has this role, and will return a list containing all the landmarks and the specific x,y and z locations. Another important function is the one returning a list containing 5 binary values to see what fingers are up: *fingersUp*. An example is the output [0,1,1,0,0] when index and middle fingers are up.

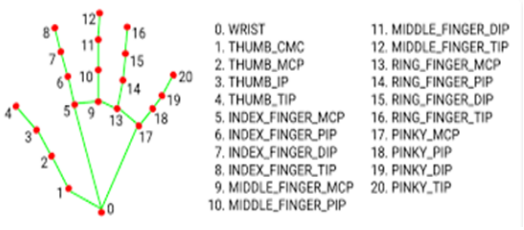

Figure 1: Landmarks

Figure 2: Detected Hand

3 Starting Interface

When a user starts the application, the menu interface shown in the following pictures will appear displaying the possibility to choose one of the three modes. This interface is already recognizing the gesture of the hand and raising the fingers (except for the pinkie finger used to run the chosen modality) is possible to select one of the three applications. In this way, raising one finger will select the first mode, two fingers the second, and three of them the third one. When a mode is selected, instructions for the correct usage of it are displayed on the right side of the name of the application. To start the run of a modality the user has to raise the pinkie tip. All the interface exploits images created with Canva. To change the image according to the selection, when the gesture is done, a new image is displayed instead the previous one.

Figure 3: Starting Interface

Figure 4: Painter modality selected

Figure 5: Rock Paper Scissor modality selected

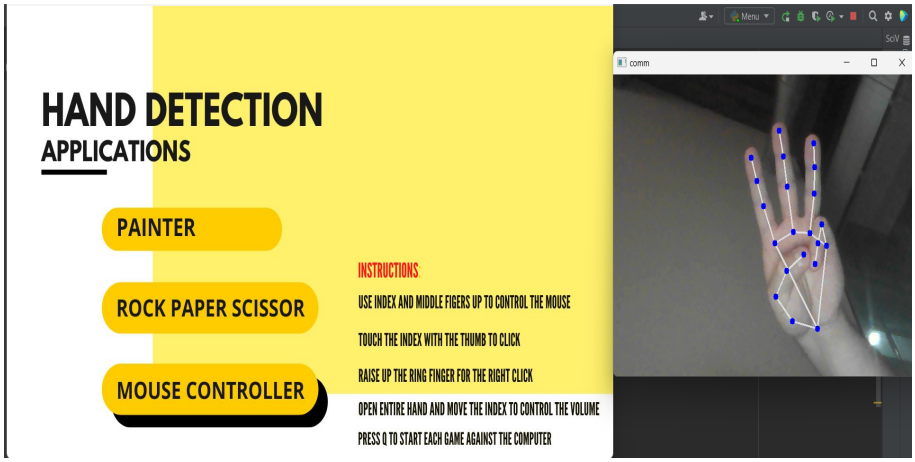

Figure 6: Mouse controller modality selected

4 Painter Modality

This is a fancy application that allow the user to draw on the screen selecting between different brush colors and sizes. As is possible to see in the following images on the upper and left side of the screen is inserted a se-

lection part, created on Canva. When two fingers up are detected through the HandDetection module, the application goes in the selection mode and when a specific area of the screen is reached, a new image, corresponding to the selection is provided. The draw is provided by the position of the eighth landmark, the one specific for the finger tip. Th:e application works in two differnt modes:

- SELECTION MODE: the user can select the color or the brush size raising two fingers and going over the chosen option. Three different colors and three different brush sizes are provided.
- DRAWING MODE: raising only one finger is possible to draw on the screen. Is provided also a deleting modality thanks which is possible to cancel drawings just made. Also for the robber the user can select different sizes.

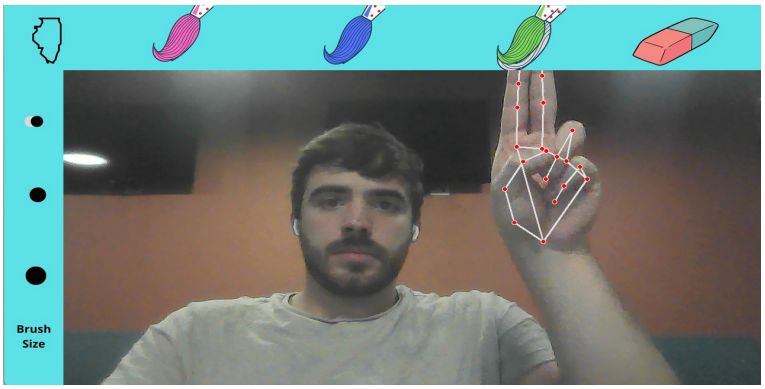

Figure 7: Selection modality

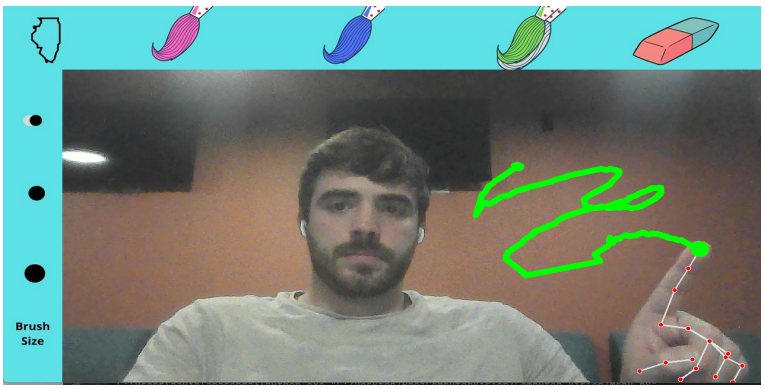

Figure 8: Drawing modality

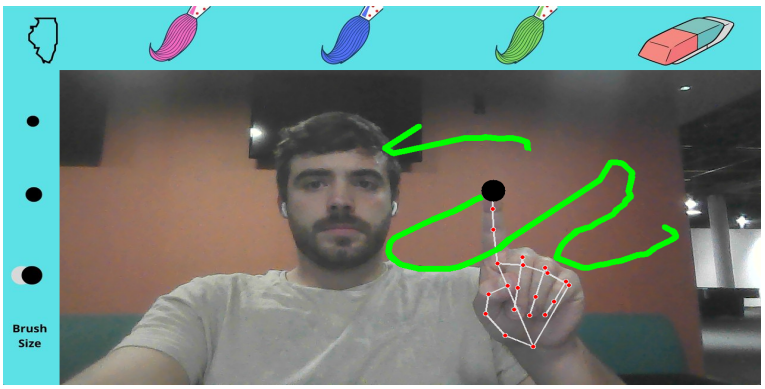

Figure 9: Rober modality

4.1 Implementation

First of all, images for the interface were created on Canva and moved to a specific folder. Secondly we define the hand detector with a detection confidence level equal to 0.85, in order to draw with a better accuracy. Then a black image was created (*imgCanvas*) on which we'll apply the drawings. The image captured from the webcam is flipped such that, when we move the hand on the right, the same movement will be reproduced on the

screen, and not the mirror one. The two different modalities, selection and drawing mode are underlined by the presence of a rectangle between index tip and middle tip in the first mode, and by a circle on the index tip in the second case. The color of the rectangle and the circle reflects the selected brush color. When robber is chosen the color will be black. To understand the right modality the *fingersUp()* function is imported from the handDetection module and the drawing is made possible thanks the usage of the *cv2.line* function on the *imgCanvas*, the black one. In order to make the drawing appear on the video image, a mask is applied between the *imgCanvas* and the image got from the video. First the *imgCanvas* is transformed into a 2GRAY image, than a *cv2.bitwise_and* and after a *cv2.bitwise_or* is applied between the previous modified image and the image provided by the camera. This procedure has the aim to show the drawing not on a black image, but on the image captured by the camera. The first variant tried to overlay the two images regulating the opacity, but the result was not as good as the last one. As expressed in the menu interface, is possible to quit the application pressing 'q' on the keyboard.

5 Rock Paper Scissor

This modality is related to the famous game Rock Paper Scissor. The interface was created on Canva. Anytime the key 's' is pressed in the keyboard, a counter starting from 0 to 3 is provided in order to guide the user when to do his move. Randomly the computer outputs a sign and if the user chooses the winning one, the score is increased for him. This application uses the finger-Up function to distinguish the different moves:

- Zero fingers = Rock
- Index and middle fingers up = Scissor
- Five fingers up = Paper

Also in this application the run of the program is interrupted pressing the ‘q’ on the keyboard.

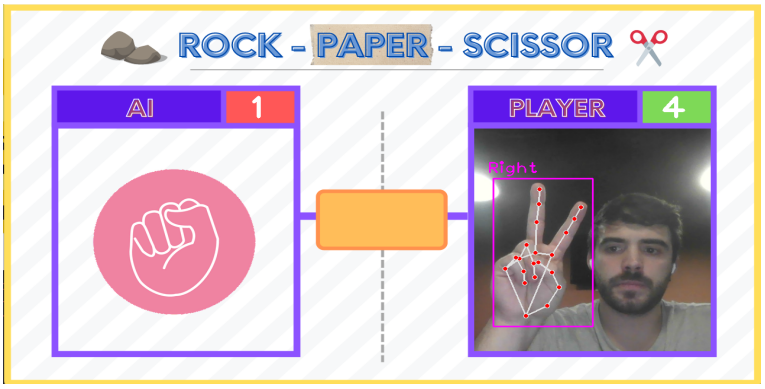

Figure 10: Waiting time

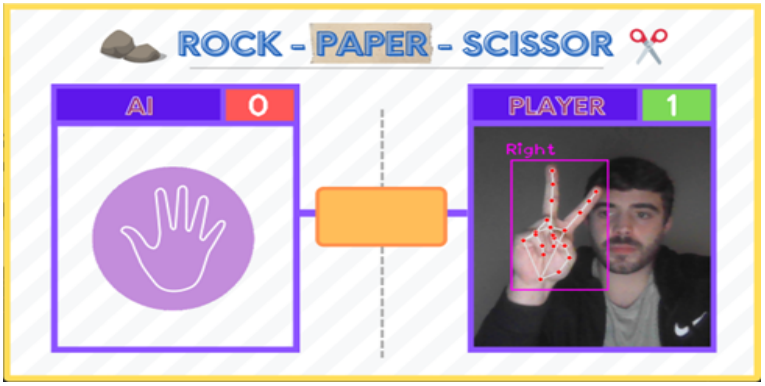

Figure 11: Winning case

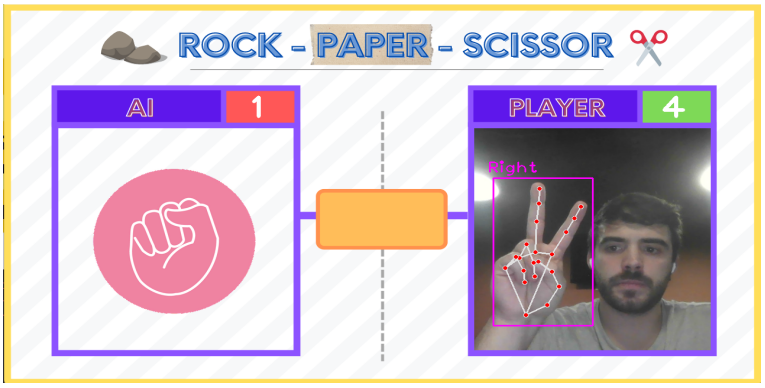

Figure 12: Loss case

5.1 Implementation

The implementation starts detecting the hand and finding the different landmarks. The *handDetector* class in this case is slightly different because a box containing the hand is also displayed on the screen, and a word indicating if the hand is the right one or the left one. Then the image provided by the camera is resized in order to fit the size shown in the previous figures. When the game is started (pressing 's'), a counter is started to mark the time. When the counter reaches a value equal to 3 the result of the user is saved, according to how many and which fingers are up. For this part is used the function *fingersUp*. For the computer result, a random number is generated, from 1 to 3, that is associated to a specific symbol between rock, paper and scissor. At the end, according to the result, the score is updated or not in case of draw. Also in this case there is the possibility to quit the application pressing 'q' on the keyboard.

6 Mouse Controller

The last modality is used to control the mouse position when index and middle fingers are up. This implementation allows the user to click thanks the usage of the thumb. When the distance between the index and the thumb is less than a specific number, the left click takes action. The other two functionalities allow to right click when the ring finger is raised up and control the computer volume when all fingers are up and the index is moved up and down. To allow the user to move on the entire screen easily, a rectangle is drawn on the image captured from the screen indicating the display area, as shown in the following pictures.-

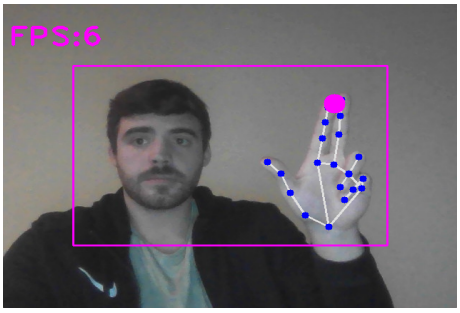

Figure 13: Mouse Controller

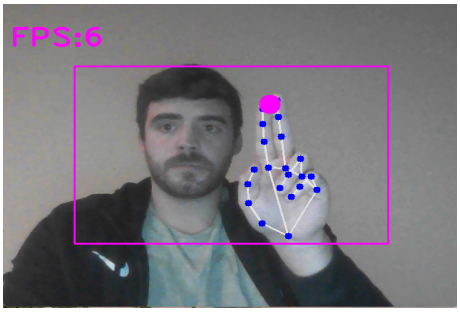

Figure 14: Left Click

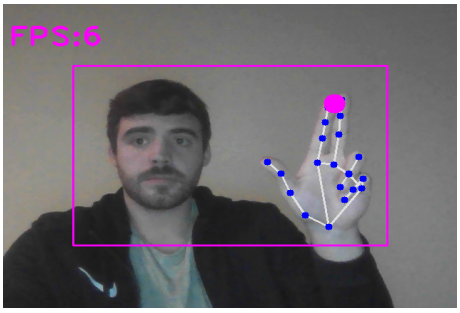

Figure 15: Mouse Controller

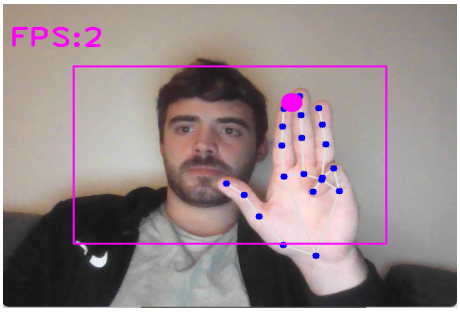

Figure 16: Right Click

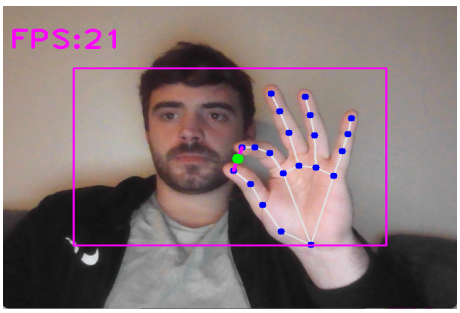

Figure 17: Setting Volume Low

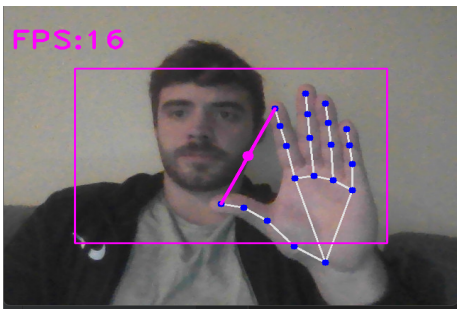

Figure 18: Setting Volume High

6.1 Implementation

This modality is based on the usage of the pyautogui library, that has functions allowing to move the mouse in a specific point and make actions like right or left click. First of all we set the values of width and height of the camera, and other variables in order to smooth the movement of the cursor. When the function fingersUp detects index and middle fingers up, the x and y values of the middle point between the two fingers is calculated and passed to the pyautogui function in order to move the mouse in that specific position. Thanks an interpolation we are capable to reduce the area of movement to the one represented by the purple rectangle, shown in the pictures before. The next step is to smooth the values in order to reduce the noise in the movement of the pointer. According to the movements displayed in the figure before is possible to get different actions:

- left click: when the distance between the third and fifth landmark is less than 25
- right click: when the ring finger is raised

- volume controller: when all fingers are up and we move the index in order to change the volume.

To implement the last functionality, the function *VolumeController* is imported from the *Utilities* module. In this function we use a command in order to be able to control the computer speakers. A line between the index and the thumb fingers is created and the volume is regulated according to the change in the line length, thanks an interpolation function, similar to the one used to move the mouse according to the rectangle shown in the image captured from the camera. In the image taken from the camera is also displayed the number of frames per second; as is possible to notice this value decrease when we are moving the mouse, due to the smoothing technique implemented.

7 Remarks and future work

This project could be improved thanks the introduction of new applications, like different games. Another approach could be to add some gestures in order to add functionalities to the mouse controller modality as: drag and drop, zoom in and zoom out, multiple selection and scrolling. In the painter application would be possible to add different colors or implement the possibility to draw with both hands. My final objective is to implement the hand detection module in a robotic arm thanks the usage of Arduino. In conclusion, regarding the course, I think the topics covered allow students to have a perfect overview of what means working in the field of computer vision. I would like to explore more image segmentation, on which an artificial intelligence architecture assigns a specific label to each single pixel

of an image, this because I'm fascinated by the topic of autonomous driving.

References

[1] https://analyticsindiamag.com/how-to-create-a-virtual-painting-app-using-opencv/

[2] https://towardsdatascience.com/tutorial-webcam-paint-opencv-dbe356ab5d6c

[3] https://opencv.org/

[4] https://mediapipe.dev/

[5] https://pyautogui.readthedocs.io/en/latest/

[6] Ahmad Puad Ismail1, Farah Athirah Abd Aziz. Hand gesture recognition on python and opencv

[7] https://www.section.io/engineering-education/creating-a-hand-tracking-module/

[8] https://www.analyticsvidhya.com/blog/2021/07/building-a-hand-tracking-system-using-opencv/

[9] https://www.computervision.zone/

[10] ttps://www.youtube.com/watch?v=k2EahPgl0ho

[11] https://how2electronics.com/gesture-controlled-virtual-mouse-with-esp32-cam-opencv/

[12] S.Shriram, B.Nagaraj, J.Jaya, S.Shankar, and P.Ajay. Deep Learning-Based Real-Time AI Virtual Mouse System Using Computer Vision to Avoid COVID-19 Spread

[13] https://google.github.io/mediapipe/solutions/hands.html

[14] https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html

[15] https://techtutorialsx.com/2021/04/20/python-real-time-hand-tracking/