Hand Tracking and gesture actions in Windows 10

Data Mining

David Campero Maña

Universidad de Huelva
Escuela Técnica Superior de Ingeniería

MediaPipe

Index

Introduction

This project is based on an intelligent system capable of analyzing in real time the hands of a person, which waits for a pincer gesture on the last hand to appear in the scene, in order to control the volume of the computer in the Windows operating system. 10.

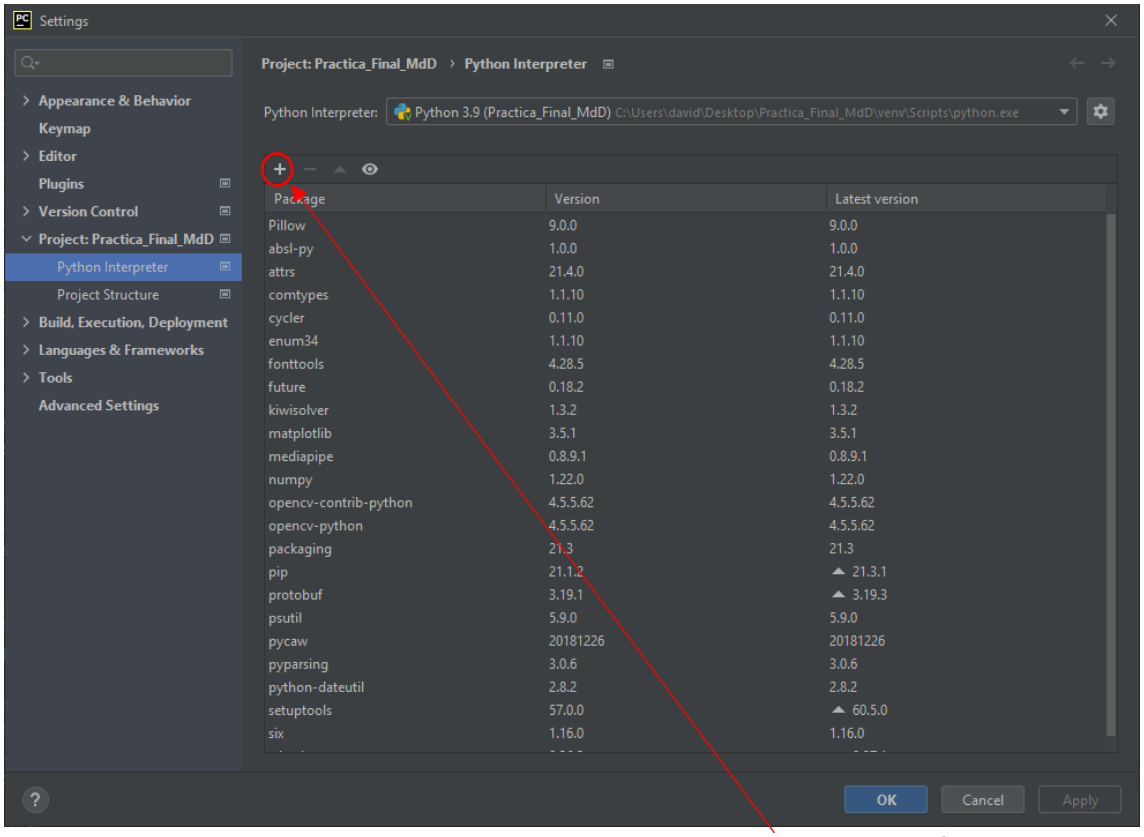
For the realization of this project, I have used libraries such as Mediapipe and OpenCV. All of them are free and easy to implement.

In addition, the development environment where I have done this project is PyCharm, an IDE created by JetBrains.

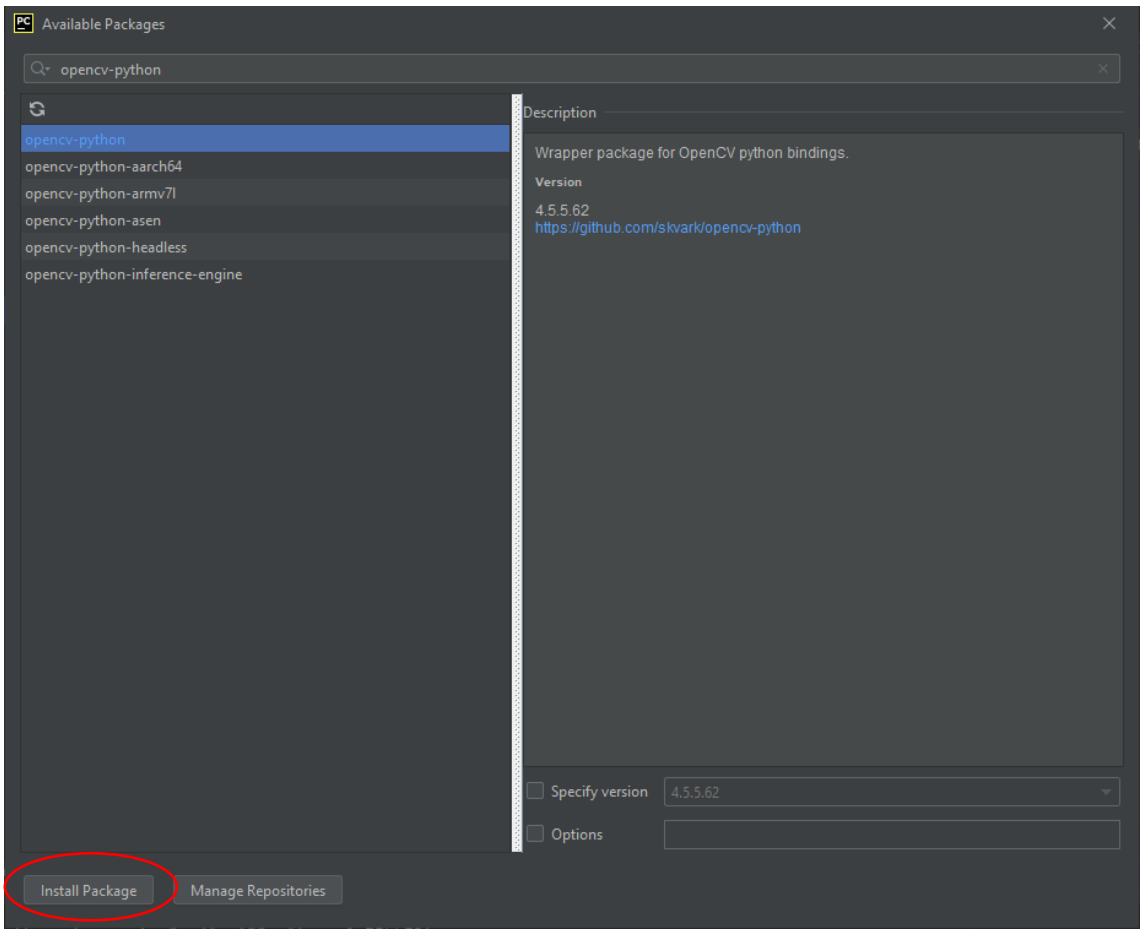
Development environment and pre-configuration

As I mentioned before, the IDE I have used is PyCharm. This IDE offers a number of features so good, that make it much easier for the programmer in question, when making Machine Learning project using Python.

This IDE offers the ability to import packages and libraries without having to install them from the cmd with Python. We simply go to the project settings and click on Project: … → Python interpreter. There, we will get a window like this:



To add packages to our interpreter, we only have to click on the **+**, and search for the desired package. Once that, select the package and click on "Install Package".



We will have to wait a little while for the IDE itself to install them and add them to the project.

For our program to work, we will have to add the following packages:

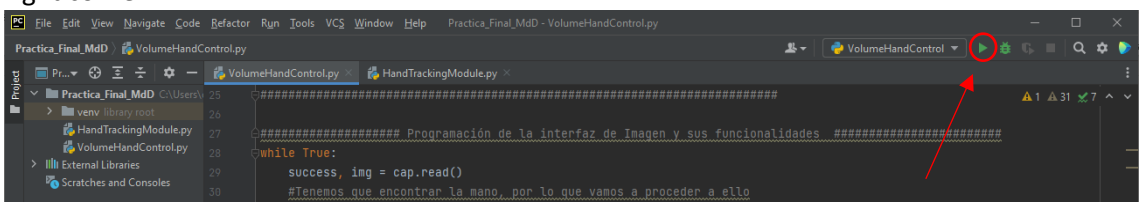
- Opencv-python
- Mediapipe
- Keyboard
- Pycaw

For the last package, we will have to install an additional library in the cmd to do the volume down or volume up action in Windows. With Python installed (version 3.9 on the PC I developed the project), we open the cmd and put the following command line:

```
python -m pip install pycaw
```

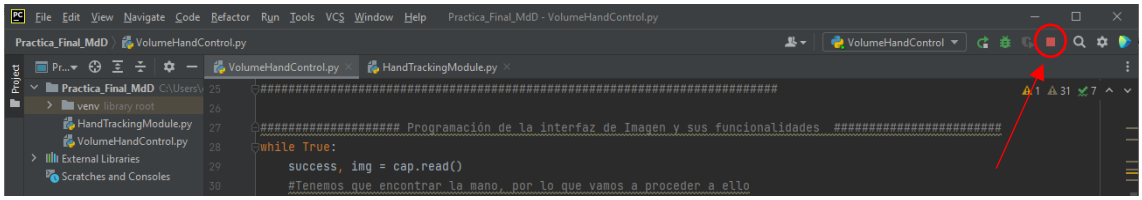
We let it install and once finished, we close the cmd window and ready, we can compile and use the program.

To compile and run the program, simply click on the compile, and run symbol in the upper right corner.



To stop the program, click on the new window that opens with the image interface and when you want to exit the program, press the Esc (Escape) key.

If we do not click on the image interface window and press the Esc key, it may close with some delay and with a negative numeric output message, since PyCharm by default, has the behavior of killing the process and not closing the application as it constantly analyzes an image for study. It would be like pressing the stop button next to the compile and run button.



If we run and compile the program, the image interface window will pop up and the program will start running.

Packages and libraries

1. **Mediapipe**
   MediaPipe is a framework that provides open source, cross-platform, customizable ML solutions for live and streaming media. It was created by Google and is one of the most widely used ML frameworks due to its easy implementation and hardware acceleration.
   Mediapipe uses TensorFlow Lite to be able to do hand prediction using hardware acceleration, either by CPU or GPU.

2. **OpenCV**
   OpenCV is a library of programming functions intended primarily for real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage and then Itseez. The library is cross-platform and free to use under the open source Apache 2 license.

3. **Keyboard**
   Small Python library to take full control of the keyboard. It captures global events, registers hotkeys, simulates keystrokes and much more.

4. **Pycaw**
   Small Python library created by Andre Miras, programmer in Barcelona, which acts as a communicator between the operating system and the program we create to manage the Windows sound.

5. **Time**
   This module provides several functions for manipulating time values.

6. **Numpy**
   - Provides an array object of arbitrary homogeneous elements.
   - Fast mathematical operations on arrays.
   - Linear algebra, Fourier transforms, random number generation.

7. **Math**
   This module provides access to the mathematical functions defined by the C standard.

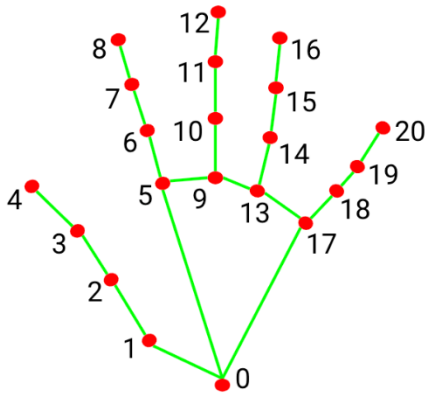
8. **Sys**
   This module provides access to some objects used or maintained by the interpreter and to functions that interact strongly with the interpreter.

Program and its coding

The program consists of a continuous analysis in search of 2 hands. The most recent hand we place will be considered the Key Hand and it will be when we can control the volume of the PC.

Once we initialize the program we can place the hand or hands, in the frame of the image interface, and it will analyze the hand or hands, and place the skeleton points on them.

According to Mediapipe documentation, the skeleton of the hand is divided into the following points:



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

To know what data Mediapipe is dealing with, I have created a parameter called lmList that contains the real-time detection of the hand position in the image interface. I cascade this parameter so that I can see the information it contemplates and thus know what to play.

For the tracking of the gesture to be able to translate it to distance, we will take the points 4 and 8.

VolumeHandControl class

We create two coordinate variables for the representation of the points on the hands.

The variables x1 and y1 are the values of the x and y coordinates of point 4 of our hand, that is, the thumb.

The variables x2 and y2 are the values of the x and y coordinates of point 8 of our hand, i.e., the index.

```
x1, y1 = lmList[4][1], lmList[4][2]
x2, y2 = lmList[8][1], lmList[8][2]
```

Now, I want a circle in the center of the stretching line to indicate when the minimum volume is reached.

```
cx, cy = (x1 + x2) // 2, (y1 + y2) // 2
```

With the aforementioned x and y coordinates, we pass them as parameters in order to create the control circles on the thumb and index finger.

```
cv2.circle(img, (x1, y1), 9, (255, 0, 255), cv2.FILLED)
cv2.circle(img, (x2, y2), 9, (255, 0, 255), cv2.FILLED)
```

Now we create the line that stretches dynamically with the fingers and the circle in the center.

```
cv2.line(img, (x1, y1), (x2, y2), (255, 0, 255), 3)
cv2.circle(img, (cx, cy), 5, (255, 0, 0), cv2.FILLED)
```

To know the distance in real time between the finger points, we do it in the following way:

```
length = math.hypot(x2 - x1, y2 - y1)
```

We want to obtain this because, depending on the distance that is reflected when we separate or join the fingers, it will be the one that we will interpret for the volume. In my case, I have indicated that my minimum volume (0%) is when the distance is at 35 and maximum (100%) when it reaches 250.

Then, with one of the functions provided by Andre Miras, we pass this data by parameter to volume.SetMasterVolumeLevel, which will reflect it in Windows.

```
vol = np.interp(length, [35, 250], [minVol, maxVol])
volume.SetMasterVolumeLevel(vol, None)
```

To control the volume, we only have to take what Andre Miras provides in his GitHbub, although I have made modifications.

```
devices = AudioUtilities.GetSpeakers()
interface = devices.Activate(IAudioEndpointVolume._iid_, CLSCTX_ALL,
None)
volume = cast(interface, POINTER(IAudioEndpointVolume))
volRange = volume.GetVolumeRange()
# volume.SetMasterVolumeLevel(-20.0, None)
minVol = volRange[0]
maxVol = volRange[1]
```

I have created two auxiliary variables to control the volume, minVol and maxVol, which has the value 0 if it is the minimum and 1 if it is the maximum, and I have removed some functions that were not useful or that I was not going to use.

To give the button effect to the middle circle when I join the two fingers, I make the following condition:

```
if length < 35:
    cv2.circle(img, (cx, cy), 5, (0, 255, 0), cv2.FILLED)
```

Finally, we add the FPS of the image in the image interface:

```
cTime = time.time()
fps = 1 / (cTime - pTime)
pTime = cTime

# Ahora vamos a mostrarlo por pantalla
# A donde queremos aplicarlo
# Que queremos que aparezca
```

```
# La posicion en la ventana
# La fuente que queremos
# La escala del texto
# El color
# Y la anchura del texto

cv2.putText(img, f'FPS: {int(fps)}', (20, 50), cv2.FONT_HERSHEY_PLAIN,
2, (0, 0, 0), 2)

cv2.imshow("Volume Hand Control - David Campero Mana", img)
cv2.waitKey(1)
```

HandTrackingModule class

The previous class was used to visualize the points, to point out those that we will use and also to see in a representative way the human-computer interaction. Now in this class, we will see how we declare the initializers and the classes needed to do the hand tracking.

First, we initialize the parameters:

```
def __init__(self, mode=False, maxHands=2, complexity = 1,
detectionCon=0.5, trackCon=0.5):

    self.mode = mode
    self.maxHands = maxHands
    self.complexity = complexity
    self.detectionCon = detectionCon
    self.trackCon = trackCon

    self.mpHands = mp.solutions.hands
    self.hands = self.mpHands.Hands(self.mode, self.maxHands,
self.complexity, self.detectionCon, self.trackCon)
    self.mpDraw = mp.solutions.drawing_utils
    self.tipIds = [4, 8, 12, 16, 20]
```

Second, we created the method for finding the hands:

```
def findHands(self, img, draw=True):
    imgRGB = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    self.results = self.hands.process(imgRGB)
    # print(results.multi_hand_landmarks)

    if self.results.multi_hand_landmarks:
        for handLms in self.results.multi_hand_landmarks:
            if draw:
                self.mpDraw.draw_landmarks(img, handLms,
self.mpHands.HAND_CONNECTIONS)
    return img
```

And thirdly, and lastly, to find the position of the hands:

```
def findPosition(self, img, handNo=0, draw=True):
    lmList = []
    if self.results.multi_hand_landmarks:
        myHand = self.results.multi_hand_landmarks[handNo]
        for id, lm in enumerate(myHand.landmark):
            h, w, c = img.shape
            cx, cy = int(lm.x * w), int(lm.y * h)
            lmList.append([id, cx, cy])
            if draw:
```

```
                cv2.circle(img, (cx, cy), 5, (255, 0, 255),
cv2.FILLED)

    return lmList
```

I have left in this class, auxiliary methods that I have created to be able to make tests with the distance, when we raise the fingers, etc., but it is something that is not important for this project. What is documented in this memory, is what we need.

Application

As I mentioned earlier, this is a window containing an image interface that, in real time, analyzes the scene for the key hand. It can track both hands, but the key hand will be the one that has recently appeared and is the volume controller of the computer.

Please find attached a video to see how it works live, by clicking here.