Proyecto Final

Hand Tracking y acciones con gestos en Windows 10

Minería de Datos

David Campero Maña

Universidad de Huelva
Escuela Técnica Superior de Ingeniería

MediaPipe

Índice

Introducción

Este proyecto se basa en un sistema inteligente capaz de analizar en tiempo real las manos de una persona, que espera un gesto de pinza en la última mano que aparezca en la escena, para poder controlar el volumen del ordenador en el sistema operativo Windows 10.

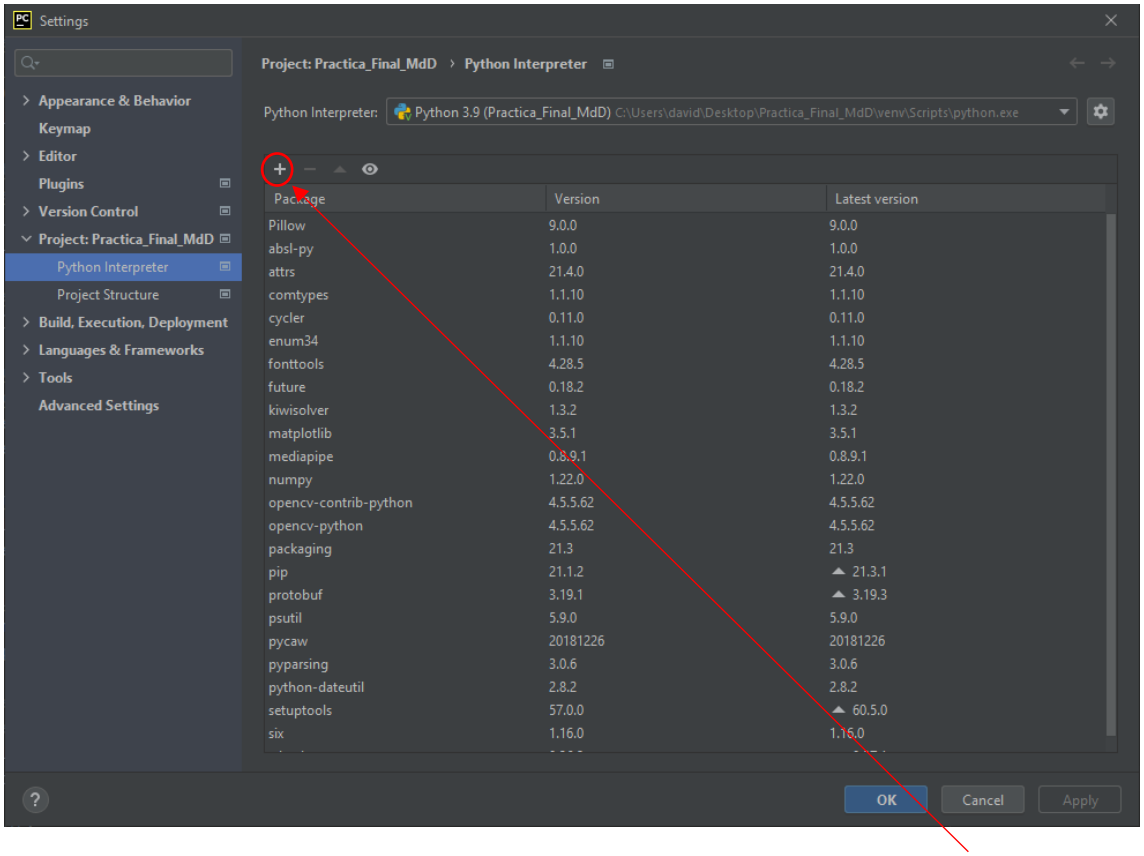
Para la realización de este proyecto, he usado librerías como Mediapipe y OpenCV. Todas ellas libres y de fácil implementación.

Además, el entorno de desarrollo a dónde he realizado este proyecto es en PyCharm, un IDE creado por JetBrains.

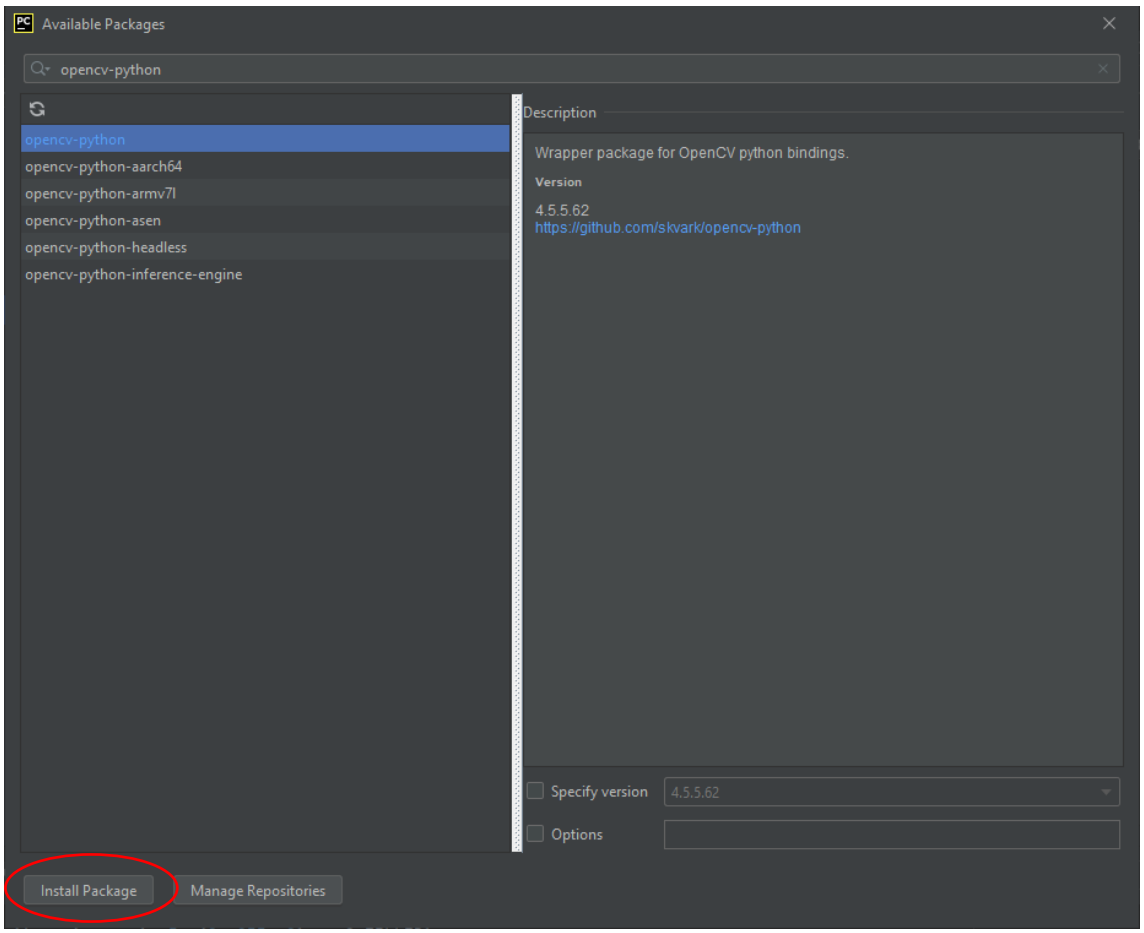
Entorno de desarrollo y configuración previa

Como anteriormente he mencionado, el IDE que he usado en **PyCharm**. Este IDE ofrece una cantidad de funcionalidades tan buenas, que facilitan mucho el trabajo al programador en cuestión, a la hora de hacer proyecto de Machine Learning usando Python.

Este IDE ofrece la capacidad de poder importar paquetes y librerías sin tener que instalarlas desde el cmd con Pyhton. Simplemente vamos a los ajustes del proyecto y hacemos clic en Project: … → Python interpreter. Ahí, obtendremos una ventana como esta:



Para añadir paquetes a nuestro intérprete, tan solo tenemos que hacer clic en el **+**, y buscar el paquete deseado. Una vez eso, seleccionamos el paquete y hacemos clic en “Install Package”.



Tendremos que esperar un poco a que el propio IDE los instale y los añada al proyecto.

Para que nuestro programa funcione, tendremos que añadir los siguientes paquetes:

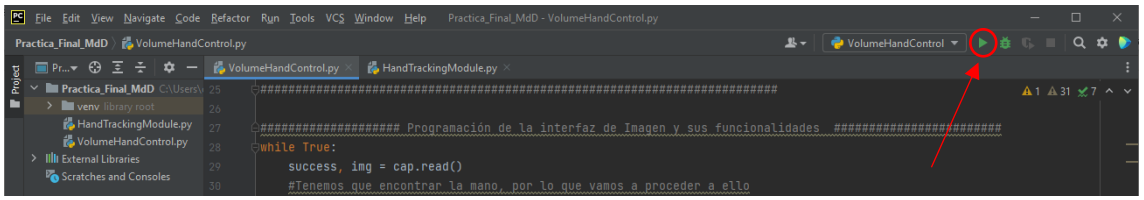
- Opencv-python
- Mediapipe
- Keyboard
- Pycaw

Para el último paquete, tendremos que instalar una librería adicional en el cmd para que haga la acción de bajar o subir volumen en Windows. Con Python instalado (versión 3.9 en el PC que he desarrollado el proyecto), abrimos el cmd y ponemos la siguiente línea de comando:

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

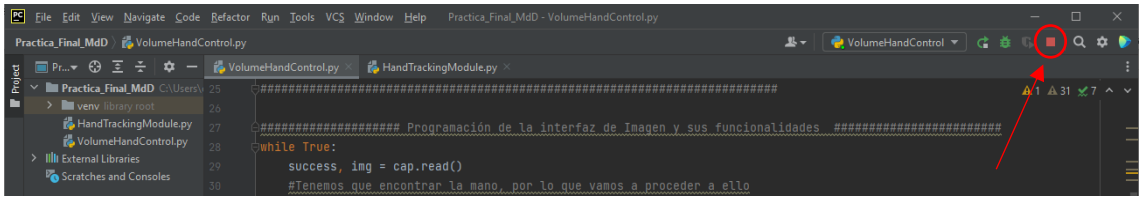
Dejamos que se instale y una vez finalizado, cerramos la ventana de cmd y listo, ya podemos compilar y usar el programa.

Para compilar y ejecutar el programa, simplemente hacemos clic en el símbolo de compilar y ejecutar que se encuentra en la esquina superior de la derecha.



Para parar el programa, tendremos que hacer clic a la nueva ventana que se abre con la interfaz de imagen y en el momento que se quiera salir del programa, presionar la tecla Esc (Escape).

Si no hacemos clic en la ventana de la interfaz de imagen y presionamos la tecla Esc, puede que se cierre con algo de retardo y con un mensaje de salida numérico negativo, ya que PyCharm por defecto, tiene el comportamiento de matar al proceso y no cerrar la aplicación ya que analiza de manera constante una imagen para su estudio. Sería como presionar el botón de parada que se encuentra al lado del botón de compilar y ejecutar.



Si ejecutamos y compilamos el programa, saldría la ventana de la interfaz de imagen y comenzaría el funcionamiento del programa.

Paquetes y librerías

1. **Mediapipe**
   MediaPipe es un framework que ofrece soluciones ML personalizables y multiplataforma de código abierto para medios en directo y en streaming. Fue creado por Google y es uno de los framework de ML más usados por su fácil implementación y su aceleración por hardware.
   Mediapipe usa TensorFlow Lite para poder hacer la predicción de las manos mediante la aceleración hardware, ya sea por CPU como por GPU.

2. **OpenCV**
   OpenCV es una biblioteca de funciones de programación destinada principalmente a la visión por ordenador en tiempo real. Desarrollada originalmente por Intel, fue apoyada posteriormente por Willow Garage y luego por Itseez. La biblioteca es multiplataforma y de uso gratuito bajo la licencia Apache 2 de código abierto.

3. **Keyboard**
   Pequeña librería para Python, para tomar el control total del teclado. Capta eventos globales, registra hotkeys, simula la pulsación de teclas y mucho más.

4. **Pycaw**
   Pequeña librería de Python creada por Andre Miras, programador en Barcelona, que actúa como comunicador entre el sistema operativo y el programa que creemos para poder manejar el sonido de Windows.

5. **Time**
   Este módulo proporciona varias funciones para manipular los valores del tiempo.

6. **Numpy**
   - Proporciona un objeto array de elementos homogéneos arbitrarios.
   - Operaciones matemáticas rápidas sobre arrays.
   - Álgebra lineal, transformaciones de Fourier, generación de números aleatorios.

7. **Math**
   Este módulo proporciona acceso a las funciones matemáticas definidas por el estándar C.

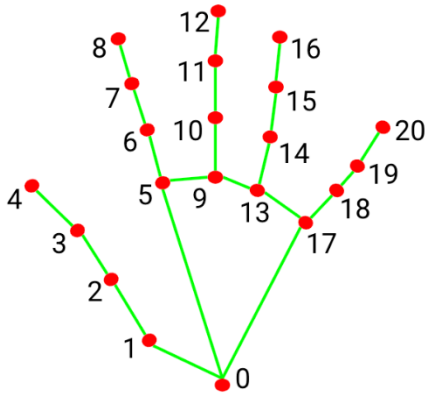
8. **Sys**
   Este módulo proporciona acceso a algunos objetos utilizados o mantenidos por el intérprete y a funciones que interactúan fuertemente con el intérprete.

Programa y su codificación

El programa consiste en un análisis continuo en busca de 2 manos. La mano que más reciente ponemos, se considerará la Key Hand y será cuando podamos controlar el volumen del PC.

Una vez que inicialicemos el programa podremos la mano o las manos, en el frame de la interfaz de imagen, y analizará la o las manos, y pondrá los puntos del esqueleton en ellas.

Según la documentación de Mediapipe, el esqueleton de la mano se divide en los puntos siguientes:



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

Para saber los datos que trata Mediapipe, he creado un parámetro llamado **lmList** que contiene la detección en tiempo real de la posición de la mano en la interfaz de imagen. Hago un casteo de este parámetro para poder ver la información que contempla y así saber qué tocar.

Para el tracking del gesto para poder traducirlo a distancia, cogeremos los puntos 4 y 8.

Clase VolumeHandControl

Creamos dos variables de coordenadas para la representación de los puntos en las manos.

Las variables x1 e y1, son los valores de las coordenadas x e y del punto 4 de nuestra mano, es decir, del pulgar.

Las variables x2 e y2, son los valores de las coordenadas x e y del punto 8 de nuestra mano, es decir, del índice.

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

Ahora, quiero que en el centro de la línea que se estira, haya un círculo que me indique cuando alcanza el mínimo de volumen.

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

Con las coordenadas x e y anteriormente mencionadas, los pasamos como parámetros para poder crear los círculos de control en el pulgar y en el índice.

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

Ahora creamos la línea que se estira dinámicamente con los dedos y el círculo del centro

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

Para saber la distancia en tiempo real entre los puntos de los dedos, lo hacemos de la siguiente forma:

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

Esto lo queremos obtener por que, dependiendo de la distancia que se vaya reflejando cuando vamos separando o uniendo los dedos, será aquella que interpretaremos para el volumen. En mi caso, he indicado que mi volumen mínimo (0%) es cuando la distancia esté en 35 y máxima (100%) cuando alcanza 250.

Luego, con una de las funciones que nos da Andre Miras, pasamos estos datos por parámetro a volumen.SetMasterVolumeLevel, que se encargará de reflejarlo en Windows.

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

Para controlar el volumen, tan solo tenemos que coger lo que Andre Miras nos facilita en su GitHbub, aunque yo haya hecho modificaciones.

```
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]
```

Yo he creado dos variables auxiliares para controlar el volumen, minVol y maxVol, que tiene el valor 0 si es el mínimo y 1 si es máximo, y he quitado algunas funciones que no me eran de utilidad o no iba a usar.

Para darle el efecto de botón al círculo de en medio cuando junto los dos dedos, hago la siguiente condición:

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

Por último, añadimos los FPS de la imagen en la interfaz de imagen:

```
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)
```

Clase HandTrackingModule

La clase anterior se utilizaba para visualizar los puntos, señalar aquellos que usaremos y además ver de manera representativa la interacción persona-ordenador. Ahora en esta clase, veremos como declaramos los inicializadores y las clases necesarias para hacer el tracking de la mano.

En primer lugar, inicializamos los parámetros:

```
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]
```

En segundo lugar, creamos el método para encontrar las manos:

```
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
```

Y en tercer, y último lugar, para encontrar la posición de las manos:

```
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
```

He dejado en esta clase, métodos auxiliares que he creado para poder hacer pruebas con la distancia, cuando levantamos los dedos, etc., pero es algo que no es importante para este proyecto. Lo que está documentado en esta memoria, es lo que nos hace falta.

Aplicación

Como anteriormente he mencionado, se trata de una ventana que contiene una interfaz de imagen que, en tiempo real, analiza la escena en busca de la Key Hand. Puede seguir ambas manos, pero la Key Hand será aquella que ha aparecido recientemente y sea el controlador del volumen del ordenador.

Adjunto un vídeo para que se vea su funcionamiento en directo, pinchando aquí.