



---

**MotionFinder**

# **Detección de gestos en Python**

---

Nombre del alumno: Fernando Dorado Gutiérrez

Curso académico: 2º DAM

Tutor del proyecto: Miguel Ángel Romero Pasamontes



# Índice:

<b>Resumen</b>	<b>1</b>
<b>Abstract</b>	<b>1</b>
<b>1. Justificación del proyecto</b>	<b>2</b>
<b>2. Objetivos</b>	<b>3</b>
<b>A. Objetivos generales</b>	<b>3</b>
a.a. Objetivo general I	3
a.b. Objetivo general II	4
<b>B. Objetivos específicos</b>	<b>5</b>
<b>3. Desarrollo</b>	<b>6</b>
3.1. Fundamentación teórica	6
3.1.1.Tecnologías utilizadas	6
3.1.2.Herramientas	7
3.2. Materiales y métodos	12
3.2.1. Diagrama de Gantt	13
3.2.2.Explícacion completa del código	14
Dentro de un archivo llamado img_colector.py	14
Dentro del siguiente archivo create_dataset.py	18
Dentro de un archivo llamado train.py	22
Dentro de un archivo llamado main.py	24
3.3. Resultados y análisis	28
<b>4. Conclusiones</b>	<b>31</b>
<b>5. Líneas de investigación futuras</b>	<b>32</b>
<b>6. Bibliografía</b>	<b>33</b>
<b>7. Anexos</b>	<b>36</b>
Anexo 1	36
Anexo 2	37
Anexo 3	38
Anexo 4	39
Anexo 5: Glosario de términos	39
<b>8. Otros puntos</b>	<b>41</b>

## Resumen

La idea principal en torno a la que girará todo el transcurso de este proyecto es la detección de gestos en Python.

En un inicio se plantea una aplicación con la capacidad para detectar mediante una señal de cámara una serie de diferentes gestos con posibles aplicaciones a discutir. Según este contexto se han abordado múltiples puntos de vista para llegar a una decisión final. Se presentan desde un inicio varias opciones de aplicación que han resultado vitales para obtener el resultado último pero para evitar extender los siguientes puntos con explicaciones elaboradas de cosas que no se han llevado completamente a cabo esto figurará en el apartado de Anexos.

Aún a pesar de utilizar anexos para anotar toda esta información adicional, habrá apartados en los que sí se incluirán explicaciones tanto del desarrollo ya realizado como de los posibles objetivos a cumplir del prototipo construido.

## Abstract

The main focus of this project is gesture detection using Python.

The initial concept involves developing an application capable of detecting various gestures through a camera signal, with potential applications to be discussed.

In this context, multiple perspectives have been considered to arrive at a final decision. Various application options were presented from the outset, which have been crucial in achieving the ultimate result. To avoid extending the following sections with detailed explanations of aspects that have not been fully developed, this information will be included in the "Anexos" section.

Despite separating this additional information, there will be sections with explanations of both, the development carried out and the potential objectives to be achieved with the constructed prototype.

## 1. Justificación del proyecto

En primera instancia, el motivo con más peso por el cual es posible decir que este proyecto tiene cierta viabilidad, es la increíble variedad de opciones existentes a la hora de desarrollar una idea como el reconocimiento de gestos en tiempo real. Hay una cantidad prácticamente incontable de campos que se pueden beneficiar de algo como esto.

Podríamos empezar hablando de cómo se podría revolucionar la industria médica. Solucionando problemas de latencias, se captan gestos de un doctor en Berlín y se reproducen para operar a un paciente en Tokio.

En casa por ejemplo, podríamos captar con una cámara suficiente información como para hacer de teclado y ratón sin necesidad de dejarnos dinero en esos dispositivos físicos.

Además de personalizar controles con la finalidad explicada anteriormente, podríamos pensar en la gente que tiene problemas motrices y que por los motivos que sea, son incapaces de usar un ratón o teclado convencional, la detección de gestos en este caso podría resultar en una opción mejor cuando hablamos de accesibilidad, personalizando un sistema de control mediante lenguaje de signos específico que permita facilitar el aprendizaje y uso para este tipo de contextos.

Se podría reinventar desde cero la forma de controlar todos los dispositivos que usamos a diario, cada persona podría personalizar una serie de gestos para que sus dispositivos tengan un propietario único e intransferible, sería como ponerle un cifrado a la forma en la que conduces tu coche para que nadie excepto tú sepa arrancarlo, poner marcha atrás o girar a la derecha.

Podría seguir enumerando escenarios posibles en los que la detección de gestos demuestra cuán útil puede llegar a ser, pero se trata de una tecnología aplicable en tantos aspectos de nuestro día a día, que tenemos posibilidades casi infinitas, y aunque sea cliché, realmente el límite es nuestra imaginación.

## 2. Objetivos

Tal como se explica en el apartado primero, hay varios enfoques que no se desarrollarán finalmente en estos puntos. Se debe aclarar que comenzaron como objetivos principales y figuran en el Anexo 1 debido a la investigación previa realizada para determinar su viabilidad, aunque en un caso específico se realizó un prototipo de aplicación que sí debe quedar redactado a continuación, así como en los siguientes apartados de desarrollo y conclusiones gracias a la importancia que esto ha tenido en la toma de decisiones acerca de la dirección a seguir.

### A. Objetivos generales

La verdadera razón por la que el primer intento de aplicación figura dentro de los objetivos generales a pesar de funcionar de forma completamente distinta a la aplicación final es porque ayuda a entender la forma en la que se debe procesar la información captada en cámara para traducirla a lenguaje humano y aplicarla en diferentes contextos. Al fin y al cabo hay que dejar muy claro que siempre se ha buscado esto como objetivo, partiendo siempre desde la misma base: Interpretar dentro de una señal de video una serie de diferentes signos realizados con las manos.

#### a.a. Objetivo general I

Prototipo de juego

Desarrollar a modo de demostración un programa para aplicar los gestos reconocidos en cámara en tiempo real para representar acciones en un entorno virtual.

Los subapartados (objetivos específicos) de este punto son los siguientes:

- Implementar un reconocimiento básico y enviar un string del gesto detectado por consola en cada instancia de tiempo.
- Crear un pequeño entorno de prueba mediante el cual se pueda emular movimiento en diferentes direcciones.
- Conector que detecte los strings generados en consola del reconocimiento de gestos y se los pase al entorno para reproducir movimientos.

## a.b. Objetivo general II

### Traducción de signos

El objetivo final de este proyecto consiste en determinar de forma eficaz mediante el análisis concurrente de fotogramas de un video en directo, una serie de señales recogidas en el sistema de signos internacional (SSI).

Para facilitar el desarrollo del proyecto y tener una mejor comprensión de la memoria del mismo se va a optar por realizar una división del proyecto fragmentándose según la siguiente serie de pasos estructurados (objetivos específicos) :

- Crear un método para recolectar imágenes de los diferentes gestos con los que entrenar al modelo .
- Creación de un dataset a partir de las coordenadas encontradas en las imágenes recolectadas.
- Entrenar un modelo de aprendizaje automático para identificar los diferentes signos del alfabeto del sistema de signos internacional a partir del dataset generado.
- Detectar en tiempo real los gestos recogidos en el modelo de entrenamiento.
- Imprimir en pantalla los gestos reconocidos con éxito para crear palabras completas.

## B. Objetivos específicos

### Prototipo

Reconocimiento básico: para detectar señales básicas con las manos, hay que trazar una serie de puntos con líneas interconectadas y utilizarlas para ubicar la posición de las falanges de los dedos mediante la librería mediapipe. Para cada instancia de tiempo en la que se detecta correctamente una posición concreta de los puntos dibujados hay que mandar el texto correspondiente por consola. \*\*\*Usamos un modelo preentrenado\*\*\*

Entorno de testeo: para simular movimiento direccional hay que pintar en pantalla un objeto que responda a comandos por teclado y responda con los desplazamientos indicados en base a los comandos enviados.

Conektor: para conectar el reconocimiento gestual con el entorno virtual hay que crear una vía de comunicación que extraiga la información del reconocimiento y se la pase al entorno de forma ordenada y precisa para no generar ningún tipo de fallo a la hora de ejecutar los comandos a realizar.

### Traducción de signos

Captar imágenes para conjunto de entrenamiento: debemos desarrollar un método para captar una serie frames durante un espacio de tiempo determinado y almacenarlo en carpetas.

Preparar imágenes y entrenar un modelo de predicción: se debe poder transformar las imágenes recolectadas en un tipo de dato con el que trabajar para crear un modelo de predicción capaz de generar predicciones eficaces.

Emplear modelo en tiempo real: se debe emplear el modelo creado en tiempo real para obtener resultados de las predicciones.

Captar el resultado para construir palabras: a partir de las predicciones hay que construir palabras completas y mostrarlas en pantalla.

## 3. Desarrollo

### 3.1. Fundamentación teórica

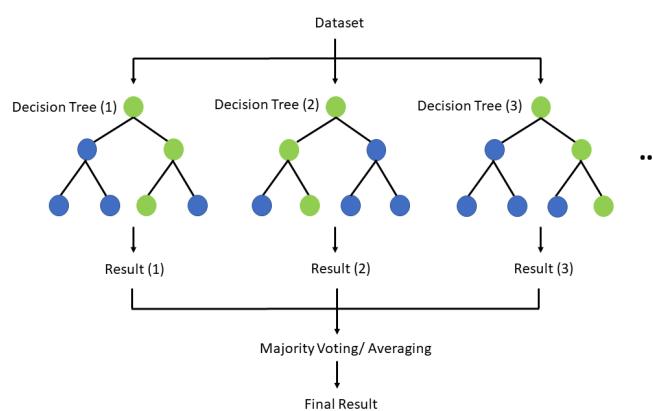
Aprendizaje automático: consiste en una herramienta basada en algoritmos que permite enseñar mediante datos cómo se pueden resolver problemas específicos aprendiendo y mejorando con el tiempo.

Es una mejora respecto a las herramientas tradicionales de análisis, se puede trabajar con grandes y complejos volúmenes de datos de forma más sencilla

#### 3.1.1. Tecnologías utilizadas

Random Forest:

Se utilizará un modelo de predicción de árbol que explicado de forma breve servirá para recordar una serie de coordenadas. Cuando usamos una cámara y captamos los puntos de referencia de una mano, las coordenadas que se captan para un determinado gesto se quedan bastante cerca de las que nuestro modelo tenía que recordar. Acto seguido se comparan los “recuerdos” de nuestro modelo con las coordenadas que vemos en tiempo real para decidir qué signo es más probable a ser el correcto. A mayor volumen de datos de entrenamiento en el modelo, mayor certeza tiene a la hora de adivinar correctamente cada signo.



### 3.1.2.Herramientas

#### IDEs

En este apartado hablamos sobre los dos IDEs utilizados a lo largo del desarrollo del proyecto. En un inicio se comenzó con PyCharm y posteriormente se cambió a IntelliJ por preferencias estéticas entre otras cosas, principalmente preferencias visuales dentro de menús de ajustes, comodidad a la hora de manipular repositorios, selección de librerías, etc...

#### IntelliJ IDEA:

Es un IDE desarrollado por JetBrains. Su desarrollo comenzó en el año 2000, con la primera versión de IntelliJ IDEA y en enero de 2001 la variante Community Edition llegó al público en formato open-source.

JetBrains ha mantenido la propiedad de IntelliJ a lo largo de los años, lo que ha permitido un control constante sobre su desarrollo y actualizaciones.

Es completamente gratuito y sin restricciones, su desarrollo activo y la participación de una gran comunidad de desarrolladores aseguran una constante mejora y actualización.

Está principalmente desarrollado en Java, lo que garantiza su portabilidad y compatibilidad con múltiples sistemas operativos que soportan la JVM (Java Virtual Machine). Es por esto, que su enfoque principal es la programación en Java, pero adicionalmente también ofrece un soporte excepcional para una gran variedad de lenguajes entre los que podemos encontrar Python, cosa que amplía su utilidad y lo convierte en una opción versátil y perfecta para nosotros.

Es innegable que tiene muchas funcionalidades de las cuales no vamos a hacer uso y en adición, como ya hemos mencionado con anterioridad, no está enfocado principalmente a Python, pero todo esto no supone ningún impedimento, de hecho se

trata de un IDE perfectamente capaz para el contexto que tenemos y es una gran opción a tener en cuenta.

Pycharm:

Desarrollado en el año 2010, PyCharm ha mantenido una posición destacada en el mundo del desarrollo de software gracias a su enfoque centrado en el usuario y su continua innovación. Con una interfaz intuitiva y poderosas herramientas de desarrollo, PyCharm puede resultar una herramienta esencial para programadores de todos los niveles de experiencia.

Es un entorno de desarrollo integrado excepcionalmente valorado, desarrollado también por JetBrains. Desde su creación, PyCharm ha sido una opción muy robusta preferida por gran parte de la comunidad de programadores, ofreciendo una buena plataforma muy fácil de usar para el desarrollo en Python, porque este sí que ha estado centrado desde el principio en este lenguaje.

Está disponible en varias ediciones, una versión Community gratuita y de código abierto y una versión Professional con características adicionales para equipos y proyectos comerciales, ambas, con varias funcionalidades como la edición de código inteligente, depurador avanzado o la integración con sistemas de control de versiones, cosas que dejan a PyCharm en buen lugar y nos han facilitado mucho el proceso de desarrollo antes de migrar a IntelliJ.

### Lenguajes de programación

Desde el inicio se han realizado muchos cambios, se han tomado diferentes decisiones, pero siempre se ha mantenido una cosa clara, el lenguaje a utilizar debía ser Python.

Python: Python fue desarrollado originalmente por Guido van Rossum y lanzado en 1991. Desde entonces, ha crecido en popularidad y ahora es uno de los lenguajes de programación más utilizados en el mundo.

Conocemos Python como un lenguaje de alto nivel, diseñado para ser fácil de leer y escribir por lo que tiene un sintaxis limpia y fácil de entender, con un fuerte énfasis en la legibilidad. Está orientado a objetos y esto significa que permite a los programadores definir sus propios tipos de datos en forma de clases, con atributos y métodos propios, facilitando la organización del código y la creación de programas más complejos.

También es un lenguaje de programación interpretado, lo que significa que el código se ejecuta línea por línea, facilitando la depuración y haciendo un desarrollo más interactivo. A diferencia de los lenguajes compilados, no necesitas compilar tu programa antes de ejecutarlo.

Esto puede ser una preferencia personal pero el código resulta más fácil de leer y mantener que casi con cualquier otro lenguaje. Es muy claro y conciso y adicionalmente el amplio abanico de bibliotecas que tiene proporciona funcionalidades adicionales bastante cómodas de aprender a usar.

A menudo puedes evitar tener que escribir código desde cero, tenemos una comunidad activa de desarrolladores que permite aprender su uso de forma sencilla, encontrando soluciones a errores en foros y aprendiendo nuevas cosas de forma constante. Y no solo tenemos a nuestra disposición foros, hay miles de recursos disponibles en línea para aprender, tutoriales, cursos, tenemos multitud de formatos pero todos ellos son útiles por igual.

Python es una opción popular para una variedad de aplicaciones. Se utiliza en desarrollo web, donde frameworks como Django y Flask hacen que sea fácil construir sitios. También es popular en análisis de datos y aprendizaje automático, con bibliotecas como pandas, numpy y scikit-learn. Estas tres, son las que más se hacen sonar o por lo menos en mi caso de las que más he oído hablar. Se tendrá esto muy en cuenta porque hablaremos más de ellas.

## Librerías

Hacemos uso de 3 librerías de código abierto.

OpenCV: es una biblioteca de software desarrollada por intel en C++ y C, mantenida en la actualidad por una organización con el mismo nombre. Se trata de la librería de computer vision<sup>1</sup> más grande del mundo, resulta extremadamente útil en una variedad increíblemente extensa de campos pero de todas sus aplicaciones, el área que nos interesa más es la enfocada en reconocimiento gestual en tiempo real.

Mediapipe: es la solución proporcionada por Google para abordar el mismo tipo de situaciones que enfrenta OpenCV. Nos otorga una forma de trabajar obteniendo un resultado que se puede conectar con técnicas de inteligencia artificial y aprendizaje automático.

Scikit-learn: es una herramienta muy útil para predecir conjuntos de datos basándose en 2 tipos de aprendizaje, supervisado<sup>2</sup> y no supervisado<sup>3</sup>.

En el primero, el algoritmo recibe un conjunto de datos de entrenamiento que incluye las entradas y salidas esperadas correspondientes, su objetivo es aprender una función que mapee las entradas a las salidas, hay ejemplos como problemas de clasificación<sup>4</sup> (predecir etiquetas de clase) y regresión<sup>5</sup> (predecir valores numéricos).

Por otro lado el segundo, que solo tiene las entradas y debe encontrar patrones o estructuras interesantes en los datos sin tener salidas esperadas. Los algoritmos de aprendizaje no supervisado en scikit-learn incluyen técnicas de clustering<sup>6</sup> (agrupar datos similares) y reducción de dimensionalidad<sup>7</sup> (encontrar representaciones más compactas de los datos).

Pickle: El módulo pickle implementa protocolos binarios para serializar<sup>8</sup> y deserializar<sup>9</sup> una estructura de objetos Python. Pickling es el proceso mediante el cual se serializa y unpickling el proceso inverso.

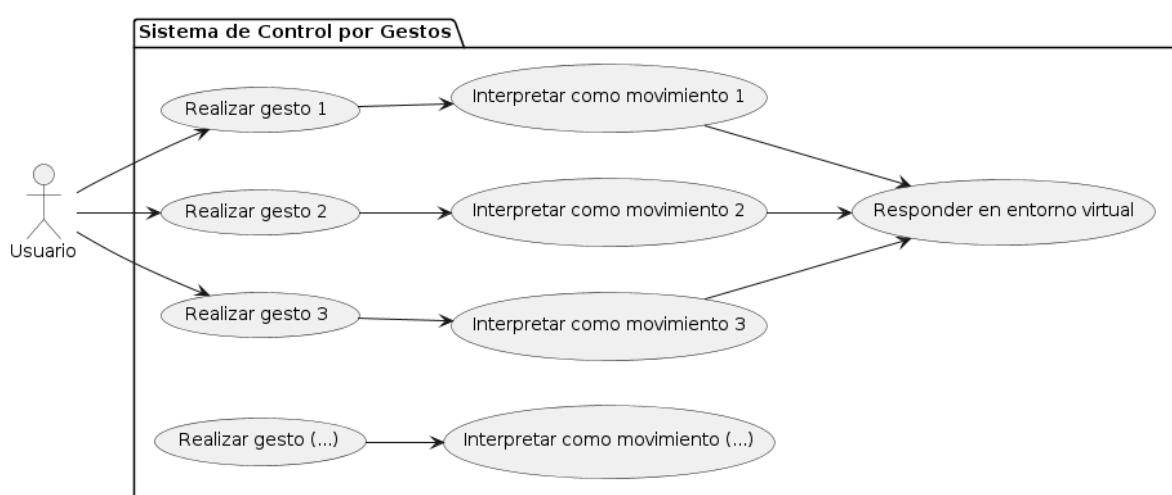
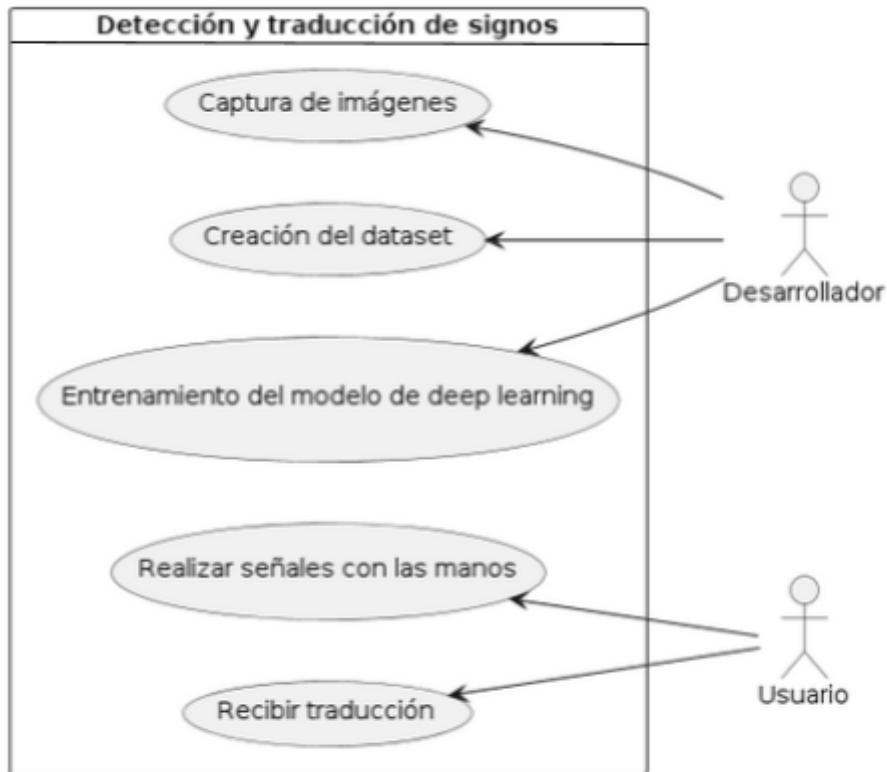
Numpy: es una librería de vital importancia a la hora de trabajar con cálculos complejos. No me voy a extender mucho porque en mi caso, no se emplean cálculos

o funciones avanzadas, pero de igual forma Numpy me resulta necesario, ya que los algoritmos de scikit-learn utilizan preferiblemente arrays Numpy como entrada, y lo mismo para la salida.

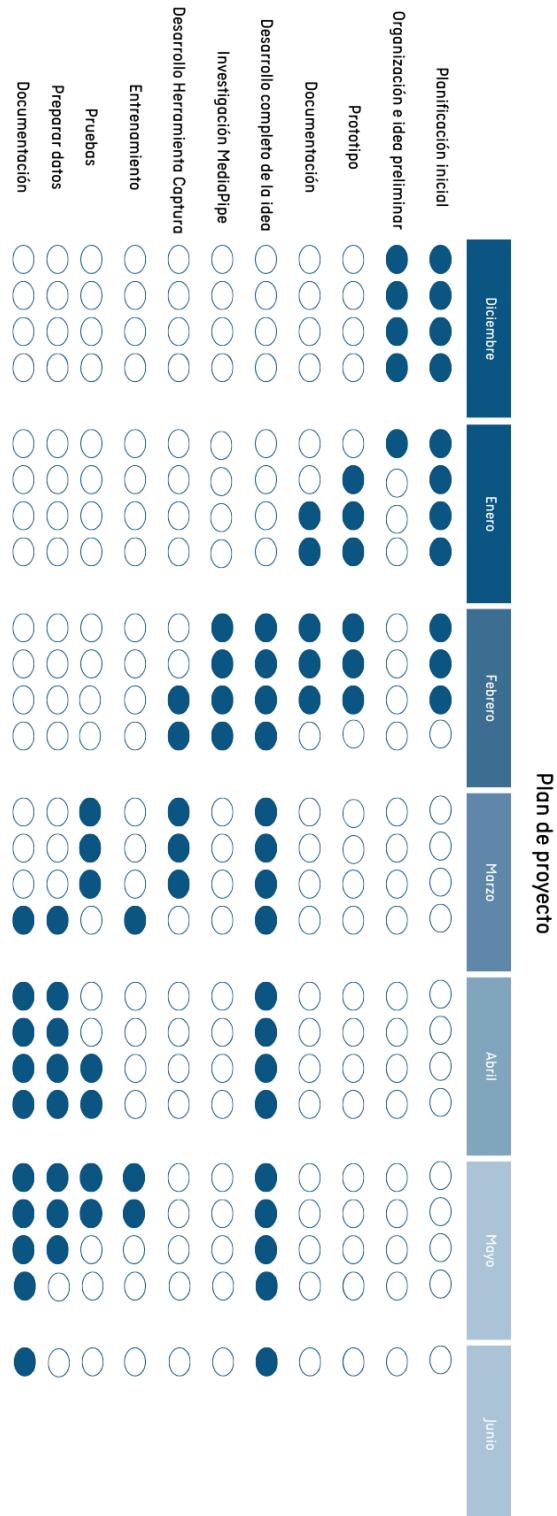
OS: por último queda el módulo OS, que proporciona formas de interacción con nuestro sistema operativo, se usa para manipular o crear rutas, archivos y directorios.

### 3.2. Materiales y métodos

Diagramas de casos de uso



### 3.2.1. Diagrama de Gantt



### 3.2.2. Explicación completa del código

Traducción de Signos:

Empezamos con el primer objetivo definido, la recolección de imágenes para el dataset.

Dentro de un archivo llamado img\_colector.py

```
1 import os
2 import cv2
```

Importamos las librerías a utilizar. La primera OS, para crear 26 carpetas dónde se guardarán las imágenes sobre las que trabajaremos más tarde. La siguiente cv2, conocida mejor como OpenCV, utilizada para acceder a la cámara y captar los frames.

```
#Ruta y nombre del directorio
DATA_DIR = './data'

#Si el directorio no existe se crea
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)
```

2 pasos sencillos. Número uno, se establece la ruta y nombre donde crearemos las carpetas,. Número dos, verificar si el directorio existe, en caso de no encontrarlo se crea.

```
#Aquí establecemos el numero de veces que queremos crear una carpeta de imgs y el número de imgs a captar
number_of_classes = 1
dataset_size = 500
```

Continuamos con las carpetas a crear, en el caso mostrado se elige 1 debido a que estábamos sustituyendo datos inservibles. Si quisieramos hacer varias letras de golpe se establecería el número elegido junto con el número de frames a captar, en este caso 500.

```
# Obtener una lista de los nombres de las carpetas

existing_dirs = [i for i in os.listdir(DATA_DIR) if os.path.isdir(os.path.join(DATA_DIR, i))]
```

Aquí pasamos a algo más avanzado, en primera instancia se crea una lista para tener todos los nombres de las carpetas existentes en el directorio indicado “existing\_dirs”, para encontrarlas se hace lo siguiente:

Para cada item “i” dentro de la lista de items en el directorio “os.listdir(DATA\_DIR)” se realiza la comprobación de si el item es una carpeta “if os.path.isdir”. Si lo es, se añade a la lista “os.path.join(DATA\_DIR, i)”

```
# Definir una función para convertir nombres de carpetas a enteros
2 usages
def get_dir_number(dir_name):
    try:
        return int(dir_name)
    except ValueError:
        # Retornamos -1, indicando que el nombre de la carpeta no es un número
        return -1
```

Aquí vemos una función que convierte en int los strings que tenemos como nombre en las carpetas (aunque la carpeta se llame “0” sigue siendo un string).

En caso de fallar el try por encontrar dir\_name con un valor que no puede pasar a entero el except devuelve un -1 indicando el fallo.

```
# Ordenar las carpetas por su nombre numérico
existing_dirs.sort(key=get_dir_number)
```

Se ordena la carpeta mediante los números obtenidos con la función anterior.

```
#Si ya hay dirs
if existing_dirs:
    #Se mira cuenta cual es el último y se le suma 1
    starting_index = get_dir_number(existing_dirs[-1]) + 1

#Si no hay dirs
else:
    #Se empieza en 0
    starting_index = 0
```

Y aquí, sencillo, repito lo explicado en comentarios, si `existing_dirs` contiene algo, se cuenta cuántas cosas tiene dentro, se elige la que está en última posición y se suma una para saber a partir de qué número empezamos a añadir.

En caso de encontrar `existing_dirs` vacío se pone 0 como posición inicial.

```
#Iniciamos la captura de video
cap = cv2.VideoCapture(0)
```

Para capturar nuestra cámara.

```
# Iterar a través de un rango de índices, comenzando en starting_index
for j in range(starting_index, starting_index + number_of_classes):
    if not os.path.exists(os.path.join(DATA_DIR, str(j))):
        os.makedirs(os.path.join(DATA_DIR, str(j)))

    print('Creando carpeta grupo de frames {}'.format(j))
```

Queremos ir creando carpetas numeradas secuencialmente contando los elementos que hay en el rango desde que empiezan las carpetas nuevas hasta que acaban.

Si tenemos 3 carpetas y queremos añadir 10 tendríamos: 0, 1, 2. El `starting_index` es el 2, `starting_index + 10` es 12. El rango sería del 2 al 12.

Si no existe un directorio con el nombre del índice (en string) indicado, se crea.

Después se imprime un mensaje con el número de la carpeta que se está creando.

```
# Esperar a que el usuario presione la barra espaciadora
while True:
    ret, frame = cap.read()
    cv2.putText(frame, 'Espacio para captar una señal', (20, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 0, 0), 2, cv2.LINE_AA)
    cv2.imshow('frame', frame)
    if cv2.waitKey(25) == 32:
        break
```

A continuación, bajo una condición que se cumple siempre, se captura frame en cámara, se añade un texto indicando que uses el espacio para comenzar la captura de frames.

```
cv2.putText(frame, 'Espacio para captar una señal', (20, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (255, 0, 0), 2, cv2.LINE_AA)
```

Se pone la ubicación del texto, la fuente, tamaño del texto, color, grosor y tipo de línea.

```
cv2.imshow('frame', frame)
```

Se muestra una ventana con los frames.

```
if cv2.waitKey(25) == 32:
    break
```

Tras 25ms si se capta el código ASCII para el espacio se sale del bucle

```
# Capturar las imágenes
counter = 0
while counter < dataset_size:
    ret, frame = cap.read()
    cv2.putText(frame, 'Captando frames', (20, 20), cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 2, cv2.LINE_AA)
    cv2.imshow('frame', frame)
    cv2.waitKey(25)
    cv2.imwrite(os.path.join(DATA_DIR, str(j), '{}.jpg'.format(counter)), frame)
    counter += 1
```

Al salir del bucle anterior se comienza con el siguiente y se realiza el mismo número de veces que imágenes se quiera en el dataset.

Se captura frame en cámara, se pone un texto indicando que se están recolectando los frames, ahora en un color rojo. Se muestran los frames en ventana. Se esperan 25ms.

Y se guardan en carpeta con el nombre del índice sumándose una unidad en cada pasada mediante un contador que al llegar al valor del dataset\_size servirá para parar.

```
cap.release()
cv2.destroyAllWindows()
```

Y por último cerramos correctamente el programa.

Dentro del siguiente archivo create\_dataset.py

```
1 import os
2 import mediapipe as mp
3 import cv2
4 import pickle
```

De nuevo librerías, Os, archivos y directorios, Mediapipe para seguimiento de manos y Pickle para serialización y OpenCV para pasar a RGB.

```
DATA_DIR = './data'
```

Ruta del directorio con los datos.

```
# Arrays para coordenadas
# data para info en bruto
# labels para categorías
data = []
labels = []
```

2 listas vacías, data para almacenar las coordenadas de los puntos de las manos, y labels para almacenar valores asociados.

```
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
```

Módulos necesarios para trabajar con Mediapipe, es lo que nos permite trazar los puntos de referencia.

```
hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.3)
```

El objeto hands sirve para inicializar el rastreo, tiene 2 parámetros, el primero, que toma como valor true para indicar que espera una imagen como entrada en lugar de video y el segundo, que toma como valor 0.3 para indicar cuán seguro se debe estar para considerar que el objeto encontrado en cada imagen se trata de una mano. Si tuviéramos mejor iluminación y cámara se optaría por subir el valor.

```
# Entrar en cada carpeta del directorio especificado
for dir_ in os.listdir(DATA_DIR):

    dir_path = os.path.join(DATA_DIR, dir_)
    if os.path.isdir(dir_path):
```

Se recorren directorios, explicado anteriormente.

```
#Para ir accediendo a cada imagen de cada carpeta
for img_path in os.listdir(dir_path):
    # -----Array para cada img-----
    data_aux = []
```

Para cada imagen dentro de cada carpeta se crea un array.

```
img = cv2.imread(os.path.join(dir_path, img_path))
```

Primero se lee la imagen obteniendo la ruta completa como ruta, nombre.

```
img = cv2.imread(os.path.join(dir_path, img_path))
```

Luego se emplea esto para cambiar el formato de color de BGR a RGB, Mediapipe usa RGB.

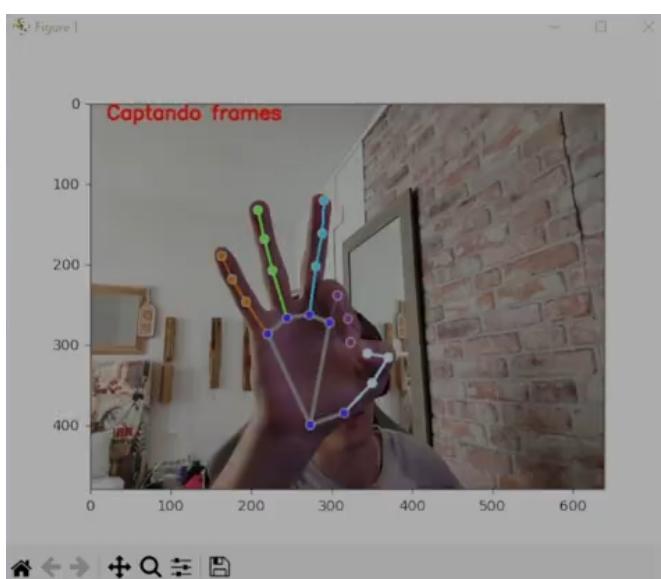
```
results = hands.process(img_rgb)
if results.multi_hand_landmarks:
    for hand_landmarks in results.multi_hand_landmarks:
```

Dentro de results tenemos los resultados de la detección de la imagen que se acaba de pasar.

Después se verifica si se han encontrado puntos de referencia, si se da el caso, para todos los puntos de referencia de cada conjunto de puntos ocurre lo siguiente:

```
# DIBUJAR IMÁGENES PARA EXPLICAR PUNTOS Y TRAZADO
#
# mp_drawing.draw_landmarks(
#     img_rgb, # Imagen sobre la cual dibujar
#     hand_landmarks, # Salida del modelo
#     mp_hands.HAND_CONNECTIONS, # Conexiones de la mano
#     mp_drawing_styles.get_default_hand_landmarks_style(), # Estilo de puntos de referencia
#     mp_drawing_styles.get_default_hand_connections_style()) # Estilo de conexiones
```

Podríamos pintar los puntos para comprobar o explicar cómo se realiza el proceso, el código para hacerlo es el mostrado arriba y da como resultado algo como esto:



```
# COORDENADAS DE LOS PUNTOS DE REFERENCIA PARA METER LUEGO EN UN ARRAY
for i in range(len(hand_landmarks.landmark)):
    # Imprimir todas las coords
    print(hand_landmarks.landmark[i])

    # Coordenadas de altura y anchura X e Y
    x = hand_landmarks.landmark[i].x
    y = hand_landmarks.landmark[i].y
    # guardar cada coordenada en un conjunto
    data_aux.append(x)
    data_aux.append(y)
```

Todo eso en el programa final se omite porque no hay necesidad de mostrarlo, simplemente queremos guardar la ubicación de cada punto y eso se hace aquí tal que así:

Primero se itera sobre todos los puntos de referencia de la mano detectada.

Se imprimen en consola los puntos detectados, esto se puede dejar comentado, realmente sirve para explicar y ver el progreso del funcionamiento.

Y más tarde se extraen las coordenadas x e y para guardar dentro de data\_aux ordenadamente.

```
# guardar conjuntos de coords en el array de cada img
data.append((data_aux))
# guardar en cada categoria (nuestras carpetas de img representan categorias)
labels.append(dir_)
```

Se van añadiendo las coordenadas en conjunto a la lista de data, cada item de data es una lista de coordenadas.

En labels se guarda la etiqueta correspondiente al directorio actual.

Resumidamente, aunque sea mucho lío, tenemos una lista con más listas dentro, estas contienen coordenadas para los puntos que hay en cada imagen de cada carpeta.

```
# abrir el conjunto de datos 'wb' w -> escritura b -> bytes
f = open('data.pickle', 'wb')

# El objeto a guardar es un diccionario de los datos en bruto y las categorías
pickle.dump( obj: {'data': data, 'labels': labels}, f)

f.close()

# -----Mostrar resultado para explicaciones-----
#
#         plt.figure()
#         plt.imshow(img_rgb)
#
# plt.show()
```

Se abre un archivo en modo escritura (si no existe se crea solo) y se guarda un diccionario que contiene los datos (data) y las etiquetas (labels) en formato binario.

Después se cierra.

Dentro de un archivo llamado train.py

```
import pickle

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
```

Pickle para deserializar, numpy para convertir los arrays en arrays de NumPy y sklearn para entrenar un modelo de predicción.

```
data_dict = pickle.load(open('./data.pickle', 'rb'))
```

Abrir el archivo binario en modo lectura y cargarlo dentro de data\_dict.

```
#Queremos convertir los datos que tenemos
data = np.asarray(data_dict['data'])
labels = np.asarray(data_dict['labels'])
```

Se convierten las listas en arrays de NumPy.

```
#Se separan los datos que tenemos en set de entrenamiento y set de testeo uno para enseñar al algoritmo y otro para comp
#test_size=0.2 indica que pillamos el 80% para entrenamiento y el 20 restante para testeo, shuffle=True aleatoriza la mu
x_train, x_test, y_train, y_test = train_test_split(*arrays: data, labels, test_size=0.2, shuffle=True, stratify=labels)
```

Se dividen los datos en dos sets, un set de pruebas y otro de entrenamiento.

Se indican los arrays con los que se trabaja, el porcentaje de datos que van a pruebas y a entrenamiento (20-80), se indica que se debe aleatorizar la muestra y que debe haber una proporción similar en cuanto a los conjuntos de datos para mantener resultados equitativos.

```
#RandomForestClassifier
model = RandomForestClassifier()

#Entrenar
model.fit(x_train, y_train)

#Predecir
y_predict = model.predict(x_test)

#Éxito?
score = accuracy_score(y_predict, y_test)

print('{}% de las muestras se clasificaron correctamente!'.format(score * 100))
```

Se emplea el clasificador elegido, se entrena el modelo utilizando el set de entrenamiento x\_train para data e y\_train para labels.

Se utiliza el set de prueba para hacer predicciones.

La precisión se obtiene comparando los labels predichos con los reales.

Multiplicando por 100 el score tenemos un porcentaje de muestras clasificadas.

```
##GUARDAR MODELO PARA MIRAR RENDIMIENTO CON DATOS REALES
f = open('model.p', 'wb')
pickle.dump( obj: {'model': model}, f)
f.close()
```

Por último se abre un archivo, se guarda el modelo en formato binario y se cierra.

Dentro de un archivo llamado main.py

```
import cv2
import mediapipe as mp
import pickle
import numpy as np
```

OpenCV, MediaPipe, Pickle y NumPy para repetir procesos realizados previamente.

```
# Cargar el modelo
model_dict = pickle.load(open('model.p', 'rb'))
model = model_dict['model']
```

Cargamos el modelo guardado anteriormente

```
# Inicializar la captura de video
cap = cv2.VideoCapture(0)
```

Capturamos señal de cámara

```
# Inicializar MediaPipe Hands
mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles

hands = mp_hands.Hands(static_image_mode=True, min_detection_confidence=0.8)
```

De nuevo mediapipe para inicializar la detección de manos, ahora con un grado de confianza algo más alto para que si hay algún gesto raro que no conoce directamente no detecte la mano.

```
# Diccionario de etiquetas
labels_dict = {0: 'A', 1: 'B', 2: 'C', 3: 'D', 4: 'E', 5: 'F', 6: 'G', 7: 'H'}
```

Un diccionario que nos servirá para interpretar el número detectado como la letra correspondiente.

```
# Variables para contar los resultados
current_result = None
result_count = 0
threshold = 30 # Umbral para agregar la letra al array
word = [] # Lista para almacenar las letras detectadas
```

Queremos un contador para detectar cuantas veces se detecta una letra, al detectarla 30 veces la letra se añadirá a una lista para imprimirla poco a poco.

```
while True:
    data_aux = []
    ret, frame = cap.read()
    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    results = hands.process(frame_rgb)
```

Se captura cada frame, se convierte a RGB y se procesa para detectar las manos.

```
if results.multi_hand_landmarks:
    if len(results.multi_hand_landmarks) > 1:
        cv2.putText(frame, 'Too many hands detected', (50, 50),
    else:
```

Si el resultado de la detección se obtiene y se comprueba que tiene una longitud mayor a 1 es porque se ha detectado más de 1 mano en pantalla, en ese caso se imprimirá el mensaje “Too many hands detected”.

```

else:
    for hand_landmarks in results.multi_hand_landmarks:
        mp_drawing.draw_landmarks(
            frame, # Imagen sobre la cual dibujar
            hand_landmarks, # Salida del modelo
            mp_hands.HAND_CONNECTIONS, # Conexiones de la mano
            mp_drawing_styles.get_default_hand_landmarks_style(), # Es
            mp_drawing_styles.get_default_hand_connections_style() # E
)

```

En caso contrario, si solo hay una mano se dibujan los puntos de referencia y las líneas entre puntos.

```

# Coordenadas de los puntos de referencia para meter luego en un array
for i in range(len(hand_landmarks.landmark)):
    x = hand_landmarks.landmark[i].x
    y = hand_landmarks.landmark[i].y
    data_aux.append(x)
    data_aux.append(y)

```

Adicionalmente se cogen las coordenadas de x e y se van añadiendo en las listas correspondientes.

```

# Predicción
prediction = model.predict([np.asarray(data_aux)])[0]
result = labels_dict[int(prediction)]

if current_result == result:
    result_count += 1
else:
    current_result = result
    result_count = 1

```

Se realiza la predicción utilizando el modelo entrenado sobre las coordenadas de los puntos de referencia en data\_aux como array numpy.

El resultado se interpreta mediante el diccionario que tenemos con las 26 letras.

Se cuenta el número de veces que se obtiene un resultado repetido.

```
print(f"Resultado: {result}, N° de detecciones"
      f": {result_count}")
```

Se imprime por consola el resultado y el número de veces seguidas que se ha detectado.

```
if result_count >= threshold:
    # Añadir la letra detectada a la palabra
    word.append(current_result)
    result_count = 0 # Reiniciar el contador
    print(f"Letra añadida: {result}, Palabra: {''.join(word)}")
```

Si el contador supera el umbral se añade nuestra letra detectada para ir formando palabras.

```
# Dibujar la palabra construida en la pantalla
cv2.putText(frame, ''.join(word), (50, 150), cv2.FONT_HERSHEY_SIMPLEX, 2, (0, 255, 0), 5

cv2.imshow('frame', frame)
```

Se dibuja en pantalla la letra añadida y se muestra el frame

```
key = cv2.waitKey(50)
if key == 8: # Presionar tecla de borrar (Backspace)
    if word:
        word.pop()

if key == 27: # Presionar 'ESC' para salir
    break
```

Borramos con la tecla 50 en ASCII (backspace) la última letra agregada en pantalla por si nos equivocamos. Esc para salir.

```
cap.release()
cv2.destroyAllWindows()
```

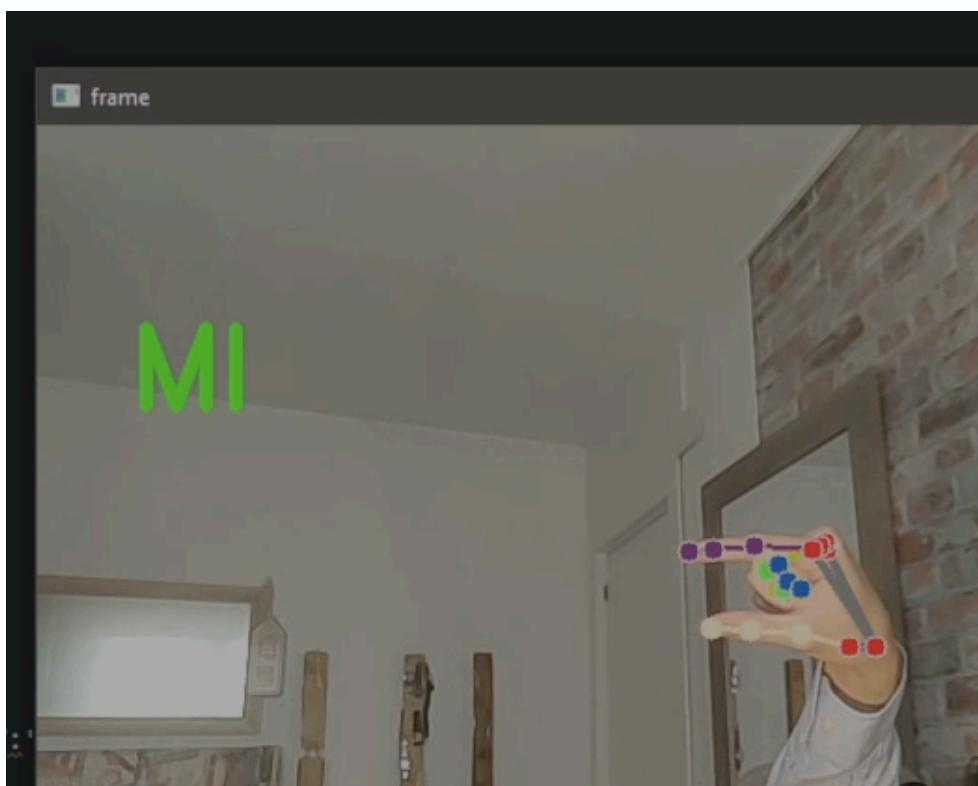
Finalmente se cierra.

### 3.3. Resultados y análisis

Tras un muy largo proceso de captura de imágenes, un proceso de limpieza bastante avanzado (porque no siempre salen correctamente todos los puntos en los frames capturados), la creación del dataset, el entrenamiento para generar el modelo y en última instancia generar palabras detectando correctamente signos en directo, podemos decir que tenemos un resultado más que adecuado para la magnitud del proyecto.

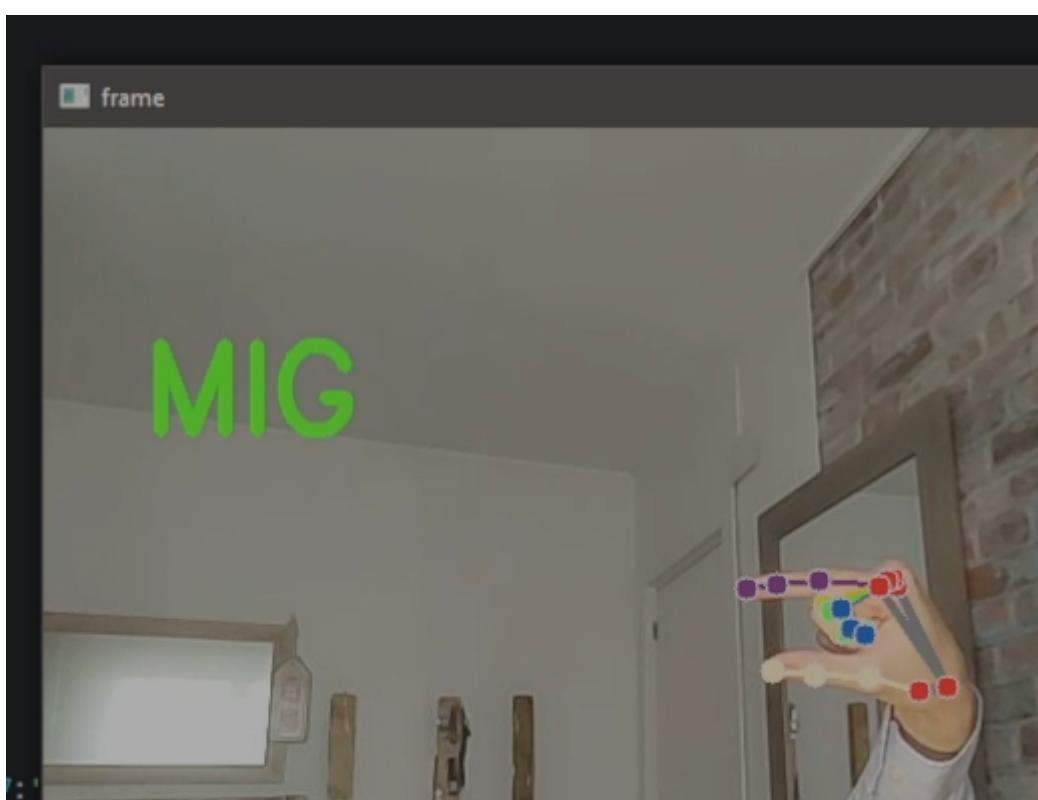
Se puede emplear de forma eficaz el programa, se detectan correctamente letras desde la A a la Z y es posible con total perfección formar palabras en condiciones de luz tanto normales como un poco adversas.

Bien es cierto que podríamos mejorar un poco más la precisión, que aunque no es escasa puede ser mejorable, algo de lo que se hablará en el apartado 6, a cambio por supuesto de una inversión en tiempo y esfuerzo que actualmente no se podía realizar. Se puede replantear también código para mejorar su estructura y acortar tiempos de espera a la hora de crear datasets, aun a pesar de ellos, el resultado obtenido es satisfactorio y a continuación dejo unas imágenes.

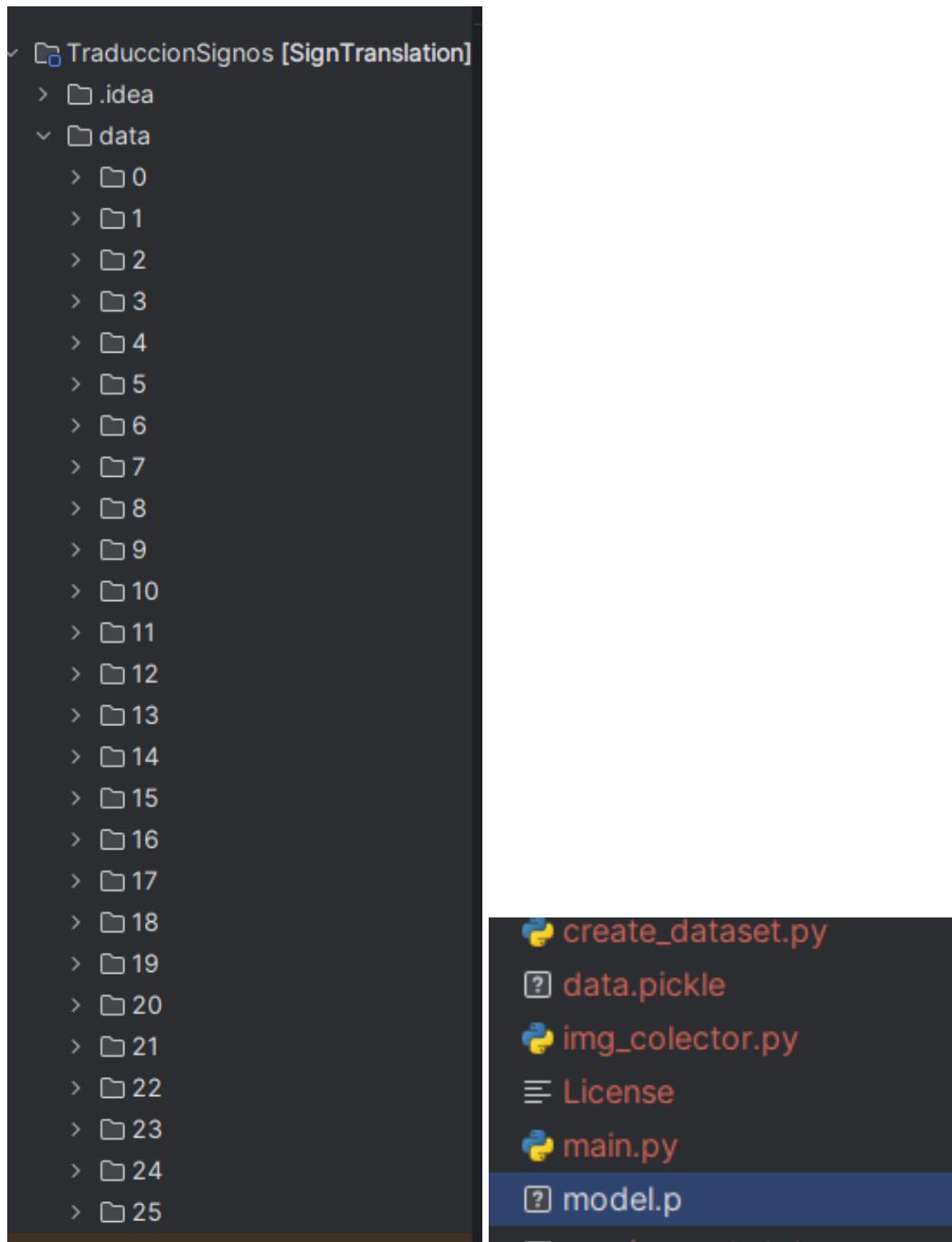


```
Run main x

Resultado: G, N° de detecciones: 10
Resultado: G, N° de detecciones: 17
Resultado: G, N° de detecciones: 18
Resultado: G, N° de detecciones: 19
Resultado: G, N° de detecciones: 20
Resultado: G, N° de detecciones: 21
Resultado: G, N° de detecciones: 22
Resultado: G, N° de detecciones: 23
Resultado: G, N° de detecciones: 24
Resultado: G, N° de detecciones: 25
Resultado: G, N° de detecciones: 26
Resultado: G, N° de detecciones: 27
Resultado: G, N° de detecciones: 28
Resultado: G, N° de detecciones: 29
Resultado: G, N° de detecciones: 30
Letra añadida: G, Palabra: MIG
Resultado: G, N° de detecciones: 1
```



Dejo también constancia de las imágenes de entrenamiento, imágenes que para utilizar el programa ya no son necesarias, y puedes omitir, de la misma forma que se puede omitir el dataset, solo necesitas el modelo, pero son imágenes que llevan tiempo recopilar porque no solo es el proceso de captura también sumamos el proceso de limpieza que se debe realizar.



## 4. Conclusiones

Tenemos la oportunidad de poder decir que este proyecto ha sido completado eficazmente cumpliendo los objetivos que se habían propuesto.

El prototipo sirvió para entender cómo captar señas, manejar strings y enviar señales en consecuencia. Esto ha sido aplicable a la hora de crear bucles y también cuando hemos recopilado letras para ir las añadiendo por pantalla, ha servido de inspiración también a la hora de encontrar el nicho en el que enfocar el proyecto. No me cansaré decir que ha sido una parte del desarrollo vital, en los anexos se explica algo sobre el código de este prototipo y se adjunta también a parte de la documentación.

En cuanto a la aplicación real del proyecto, repito que se ha obtenido un resultado satisfactorio, y además, lo interesante aquí es que el programa es válido para cualquier lengua de signos, es reutilizable y puedes emplear el sistema de signos francés si te tomas el tiempo de realizar las capturas de frames y todos los pasos que van después. Incluso puedes crear si quieras tu propio alfabeto en lengua de signos, solo tienes que inventarte tus señas a la hora de captar los frames, puedes asignar las letras que quieras luego.

## 5. Líneas de investigación futuras

### **Tensorflow:**

Tensorflow es el camino a seguir, no he tocado mucho el tema debido a la implementación del algoritmo RandomForest de scikitLearn, el aprendizaje profundo puede dar mejores resultados, así que tengo por seguro un camino en mente y es el siguiente:

- Seguir investigando acerca del tema para profundizar y encontrar formas de mejorar el rendimiento y la eficacia de la aplicación.
- Plantear un posible port a móvil con Tensor Flow Lite.
- Tener una batería de pruebas muy grande realizada por varios usuarios en dispositivos móviles y abrir la aplicación a un público más extenso cuando sea posible.

## 6. Bibliografía

#### Derechos de autor en imágenes generadas con IA

Business Insider. (2023). ¿Tienen derechos de autor las imágenes creadas por IA?

Business Insider. Recuperado de

[<https://www.businessinsider.es/tienen-derechos-autor-imagenes-creadas-ia-1223714>]

#### NumpyArray

NumPy. (2023). numpy.array. NumPy Documentation. Recuperado de

[<https://numpy.org/doc/stable/reference/generated/numpy.array.html>]

#### Investigación inicial

Ravi8x. (2023). AndroidCamera. GitHub. Recuperado de

[<https://github.com/ravi8x/AndroidCamera>]

Dev47Apps. (2023). DroidCam - OBS. Dev47Apps. Recuperado de

[<https://www.dev47apps.com/obs/>]

VDO.Ninja. (2023). VDO.Ninja. Recuperado de

[<https://vdo.ninja/>]

#### Detección de gestos y resize con OpenCV y MediaPipe

OpenCV. (2023). OpenCV. Recuperado de

[<https://opencv.org/>]

Google Developers. (2023). Gesture Recognizer - MediaPipe. Google Developers.

Recuperado de

[[https://developers.google.com/mediapipe/solutions/vision/gesture\\_recognizer/python#live-stream](https://developers.google.com/mediapipe/solutions/vision/gesture_recognizer/python#live-stream)]

Google Developers. (2023). Hand Landmarker - MediaPipe. Google Developers.

Recuperado de

[[https://developers.google.com/mediapipe/solutions/vision/hand\\_landmarker#models](https://developers.google.com/mediapipe/solutions/vision/hand_landmarker#models)]

Computer Vision Zone. (2023). Computer Vision Zone. Recuperado de  
[<https://www.computervision.zone>]

#### Juego de prueba con Pygame y OpenGL  
Pygame. (2023). Pygame Documentation. Recuperado de  
[<https://www.pygame.org/docs>]

PyPI. (2023). PyOpenGL. Recuperado de  
[<https://pypi.org/project/PyOpenGL>]

#### Threads, subprocess y queue  
Python Software Foundation. (2023). subprocess — Subproceso de gestión. Python Documentation. Recuperado de  
[<https://docs.python.org/es/3/library/subprocess.html>]

Python Software Foundation. (2023). threading — Hilos. Python Documentation. Recuperado de  
[<https://docs.python.org/es/3.8/library/threading.html>]

Python Software Foundation. (2023). queue — Colas. Python Documentation. Recuperado de  
[<https://docs.python.org/es/3/library/queue.html>]

#### Información importante sobre términos técnicos  
Microsoft. (s.f.). ¿Qué es la visión por computadora? Azure. Recuperado de  
[<https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-computer-vision#clasificación-de-objetos>]

IBM. (s.f.). Aprendizaje supervisado. Recuperado de  
[<https://www.ibm.com/es-es/topics/supervised-learning>]

Google Cloud. (s.f.). ¿Qué es el aprendizaje automático (machine learning)?.

Recuperado de [<https://cloud.google.com/learn/what-is-machine-learning?hl=es-419>]

#### #### TFGs de ejemplo

Cordobés Delgado, J. (2021). TFG - Jorge Cordobés Delgado. Universidad Politécnica de Madrid. Recuperado de

[[https://oa.upm.es/71433/1/TFG\\_JORGE\\_CORDOBES\\_DELGADO.pdf](https://oa.upm.es/71433/1/TFG_JORGE_CORDOBES_DELGADO.pdf)]

Padrón Castañeda, R. (2019). TFG - Ruymán Padrón Castañeda. Universidad Politécnica de Madrid. Recuperado de

[[https://oa.upm.es/56101/1/TFG\\_RUYMAN\\_PADRON\\_CASTANEDA.pdf](https://oa.upm.es/56101/1/TFG_RUYMAN_PADRON_CASTANEDA.pdf)]

Molinaro, G. (2022). Desarrollo de una aplicación de comunicación multimedia a través de internet. Universidad Politécnica de Valencia. Recuperado de

[<https://riunet.upv.es/bitstream/handle/10251/188692/Molinaro%20-%20Desarrollo%20de%20una%20aplicacion%20de%20comunicacion%20multimedia%20a%20traves%20de%20internet.pdf?sequence=1>]

#### #### Ideas similares e inspiración

Reddit. (2023). I made a hand gesture-controlled prototype game. Recuperado de

[[https://www.reddit.com/r/developersIndia/comments/13ro9up/i\\_made\\_a\\_hand\\_gesture\\_controlled\\_prototype\\_game/](https://www.reddit.com/r/developersIndia/comments/13ro9up/i_made_a_hand_gesture_controlled_prototype_game/)]

## 7. Anexos

Al ser este documento un documento de guía y memoria debe quedar registrada toda la información importante referente al proyecto, así como todas y cada una de las decisiones que se han ido tomando en relación al mismo, pero todo eso no cabe dentro de lo anterior, por lo que ampliamos contenido que no entra en ningún sitio mediante los anexos presentados a continuación. Adicionalmente se añade un apartado de vocabulario técnico.

En el primer caso a tratar en los anexos se dispondrán todas las opciones que se han ido barajando desde el comienzo. Aunque no se hayan podido continuar por diferentes causas, han sido de vital importancia para establecer el rumbo del proyecto.

*Muchas de las ilustraciones empleadas para dar una mejor explicación de lo que se habla, están sacadas de una herramienta de inteligencia artificial.*

*Las licencias creative common nos permiten usar muchas imágenes, pero las imágenes que se generen mediante IA carecen de propietario por lo que no tienen licencia y resultan extremadamente útiles para este tipos de usos, explicación en bibliografía.*

### Anexo 1

- Control de interfaces de usuario: Se baraja la posibilidad de hacer una herramienta para controlar la navegación por diferentes menús sin la utilización de un teclado y un ratón, visualmente muy llamativo. Opciones como pausar o rebobinar contenido multimedia mediante gestos, subir el volumen de una pista de audio o pasar esa canción que andas escuchando moviendo un fader y apretando botones en una mesa de mezclas invisible. Se desarrollará un poco más con la investigación realizada en el apartado respectivo.



*-Un fader es “esa cosita que deslizas para arriba y para abajo”.-*

## Anexo 2

- Juegos: Aquí hay que decir la verdad, en este punto me entró la vena nostálgica e imaginé un juego de hace un tiempo cuyo nombre no voy a mencionar al que le vendría muy bien la posibilidad de controlarse mediante gestos. Se me vino a la mente porque lo jugábamos todos los años en navidad y siempre faltaban mandos. ¿Por qué limitarse a usar mandos cuando podemos indicar con las manos el movimiento de cada jugador?



*-Las imágenes del juego tienen copyright, esto es una imagen del concepto del juego generada con ia.-*

En este punto se explica el resultado obtenido tras el desarrollo del prototipo inicial, originalmente planificado con la posible aplicación orientada a juegos en el punto de mira. Tras estos resultados se replanteó el objetivo del proyecto y se tomó el rumbo especificado en los apartados anteriores.

Captar manos y detectar puntos que nos permiten trazar algunas líneas y detectar la posición de los dedos, a partir de ahí, se realiza un pequeño juego que consiste en un cuadrado blanco con un rectángulo rojo que realiza diferentes movimientos en función de los datos recibidos.

Lo primero que se tiene en cuenta es HandRecognition.py que de forma extremadamente resumida se encarga de captar gestos en tiempo real e imprimir en consola el gesto y la mano con la que se realiza. Más en profundidad, usamos HandRecognitionModule.py e interpretamos las manos como listas ocupando cada dedo una posición en la misma, cada dedo con valor 1 o 0, en función de que dedos tienen valor 0 y cuales 1 se almacenan diferentes gestos y se actúa en consecuencia.

En CommunicationTest.py simplemente hemos estado probando que se extraen correctamente los prints del archivo que se le especifique (HandRecognition.py), luego de la comprobación de su correcto funcionamiento se puede pasar a continuar el desarrollo.

Muy rápidamente, el archivo Main.py contiene una prueba de juego que resulta rendir exitosamente debido a varias funciones. Hay una función leer\_movimientos() que lee los datos de output en consola del archivo especificado y mete dato a dato en una cola (sigue la lógica de CommunicationTest), procesar\_datos() que en función de

los datos que se pillan de la cola, envía una señal u otra al juego, una función draw\_cube() para pintar en pantalla el cuadrado y establecer algunas partes de la lógica del juego y game() que contiene el juego en sí. En adición tenemos hilos para ejecutar las tareas de lectura y procesamiento a la vez que el juego.

GestureOutputEmulator.py que utilizamos para simular la detección de una serie de gestos y posteriormente emplearlo en ComunicationTest o en Main.

ImageResize.py que a partir de la altura o anchura redimensiona la imagen con la que trabajamos.

### Anexo 3

- Aprendizaje: En primer lugar aquí se había planteado la posibilidad de una aplicación para enseñar lenguaje de signos de forma interactiva, pero cuando más adelante me llegó la idea de hacer un traductor me pareció bastante mejor opción y finalmente resultó el punto elegido. ISL(International Signal Language).



-Alfabeto del lenguaje de signos internacional-

## Anexo 4

- Realidad Aumentada: La gran mayoría de cascos de realidad virtual tienen cámaras, y te toca hacer uso de unos mandos con forma rara, botones y gatillos. Me gustaría probar cuán práctico sería captar tus propias manos en primera persona como input directo.



-Quest 3 (Headset y mandos de VR)-

## Anexo 5: Glosario de términos

A pesar de dar una definición breve para alguna de las palabras citadas más abajo, según qué criterio puede resultar de importancia superlativa dejar una explicación más profunda para todos y cada uno de los términos que pueden ser complicados de entender.

1-Computer vision: se define como una rama de la informática que posibilita la identificación de objetos o personas en imágenes y video con el objetivo de simular la forma en que un ser humano comprende lo que ve.

2-Aprendizaje supervisado: compara salidas y entradas, tiene el objetivo de encontrar un función capaz de mapear características y asignarlas a sus etiquetas correspondientes

3-Aprendizaje no supervisado: no hay nada etiquetado, se buscan patrones y se intenta aprender la estructura de los datos.

4-Problemas de clasificación: para el aprendizaje supervisado, su objetivo es predecir la clase o categoría a la que pertenece una instancia de datos.

5-Problemas de regresión: para el aprendizaje supervisado, su objetivo es predecir un valor numérico basado en un conjunto de características

6-Clustering: para el aprendizaje no supervisado, agrupa un conjunto de datos en grupos similares, con el objetivo de encontrar estructuras ocultas.

7-Reducción de dimensionalidad: para el aprendizaje no supervisado, consiste en reducir el número de variables aleatorias, para simplificar modelos.

8-Serializar: proceso de convertir un objeto o estructura de datos en un formato que se puede almacenar o transmitir.

9-Deserializar: proceso inverso de la serialización, es decir, convertir un objeto serializado de vuelta a su forma original.

## 8. Otros puntos

- Retos profesionales

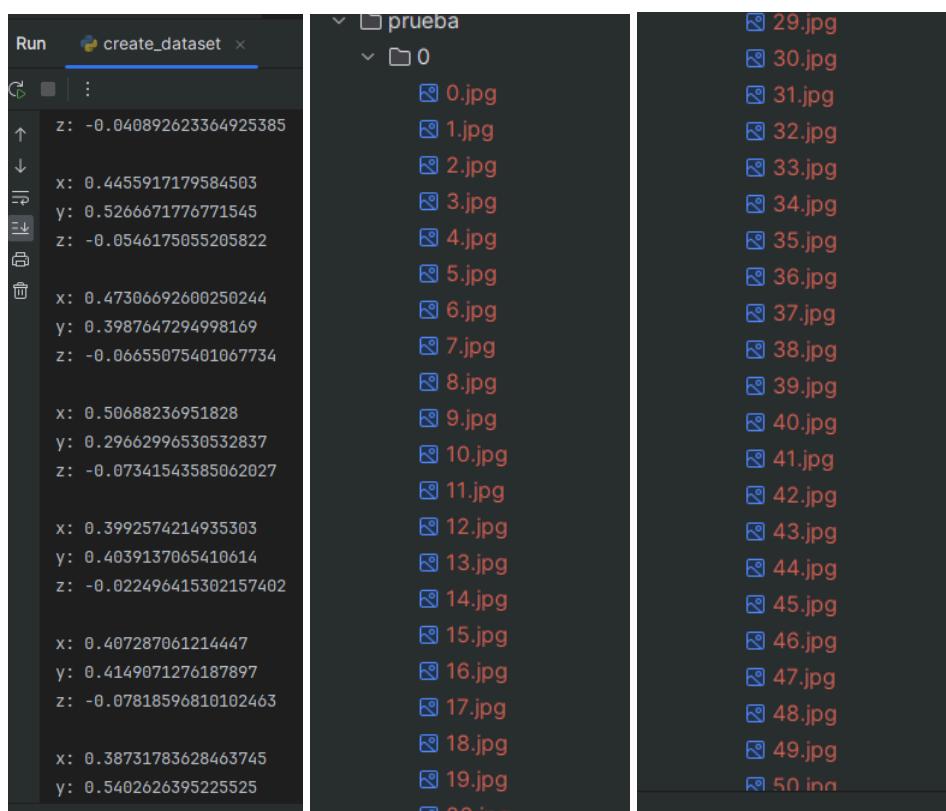
Un gran obstáculo ha sido el organizamiento de los datos recolectados. En un inicio se comienza con conjuntos pequeños de datos 5 letras, 42 puntos de coordenadas por cada imagen, con 50 imágenes para cada letra. Las coordenadas son algo así:

x: 0.379619836807251

y: 0.6882860660552979

z:-0.040892623364925385

Como decimos para cada imagen 42 coordenadas, la z la descartamos porque no hacemos uso de la profundidad, trabajamos solo con eje X e Y en 2D, salen 84 números que debemos almacenar, por 50 imágenes, 4.200 datos numéricos únicamente solo para la primera letra..



Nuestro primer testeо fue con 5 letras, 21.000 datos, pero hay que ir ampliando hasta las 26 letras donde salen 109.200 datos. Y claro una muestra de 50 imágenes es muy poco certera, poco a poco se ha ido usando un conjunto de entrenamiento más grande, pasando por 100, 200 y 500 imágenes en el estado final de la aplicación, 10 veces más

datos que al inicio, 10 veces más fallos y 10 veces más problemas, con una mejora cada vez menos notable. Con 100 imágenes se notaba mejora, el primer intento usaba un conjunto de datos muy escaso, con 200 imágenes se notaba un poco menos y con 500 digamos que aun menos todavía. Incrementar el volumen de datos es importante, siempre que tengas más imágenes vas a obtener mejores resultados, pero vas a acabar llegando a un límite para el que no merece la pena dedicar ni tiempo ni recursos.

Cada salto que das es más pequeño y gastas más tiempo creando un dataset, limpiando y comprobando las imágenes, que investigando y aplicando opciones de mejora para incrementar la certeza por otros medios.