# Reconocimiento de gestos con la mano. "Deep Learning"

*Curso 2k19/2k20*

**Apellidos, nombre** **Jorge Alcañiz Villanueva(joralvi1@inf.upv.es)**
**Zhihao zhang(zhizha1@inf.upv.es)**

**Dan Anitei (daan4@inf.upv.es)**

**Titulación** Grado de Ingeniería informática
**Fecha** 20 Abril 2020

# Índice

# Índice de figuras



# Índice de tablas

# 1 Resumen de las ideas clave

En este documento se presenta la memoria de trabajo del proyecto realizado en la asignatura de Sistemas Multimedia Interactivos e Inmersivos (SMI). El proyecto consiste en desarrollar un sistema en tiempo real, que proporciona al usuario una nueva forma de interactuar con un ordenador mediante el reconocimiento de gestos de la mano. El sistema está compuesto por una cámara web y una aplicación que a partir del contenido leído en tiempo real ofrece la funcionalidad descrita previamente.

Este documento sirve como una guía para el desarrollo de dicho sistema y se van a presentar las distintas fases del ciclo de vida del proyecto desde la planificación inicial con los objetivos a conseguir en este proyecto, la fase de análisis de los requisitos funcionales y no funcionales del sistema, el diseño del sistema en cuanto a recursos necesarios para llevar a cabo el proyecto, la fase de desarrollo de la solución y finalmente hasta la validación y despliegue de la misma.

Junto con este documento se van a suministrar los recursos digitales necesarios para llevar a cabo el proyecto, para que el lector pueda reproducir el experimento detallado a continuación.

# 2 Introducción

Con la progresiva expansión del mundo digital y la generación masiva de nuevo contenido (44 zettabytes en el año 2020 [1]), la interacción persona-computador es de suma importancia ya que esto permite poder ofrecer al usuario un entorno cómodo de usar y al mismo tiempo minimizar errores, incrementar la satisfacción, disminuir la frustración y, en definitiva, hacer más productivas las tareas que rodean a las personas y los computadores [2]. Los avances en la potencia de cómputo dan la posibilidad de crear nuevas formas de interaccionar con el computador, ofreciéndole al usuario nuevas experiencias en el mundo de los sistemas multimedia interactivos e inmersivos.

En este documento nos vamos a centrar en la interacción persona-computador mediante el uso de gestos de la mano. Actualmente existen varios tipos de sistemas que ofrecen esta forma de interaccionar con el computador. Los principales tipos de reconocimiento de gestos se basan en detectar el movimiento de la mano mediante el uso de guantes que cuantifican dicho movimiento usando acelerómetros, electromiograma, para estudiar el sistema nervioso periférico y así detectar el origen del movimiento ó mediante técnicas de visión para detectar y rastrear el movimiento de la mano para poder reconocer los gestos realizados [3]. Aunque hay tres principales clases de sistemas para reconocer los gestos, no todas son lo suficientemente maduras para usar en ámbitos comerciales, que según [3] se da el caso de los sistemas basados en electromiogramas.

El principal área de aplicación para los sistemas de interacción mediante gestos sería en la industria de los videojuegos. Esta manera de interaccionar con el computador podría revolucionar el mundo de los videojuegos, al ofrecer una alternativa con más facilidad de uso y más intuitiva que el típico uso del teclado y ratón [4].

Actualmente en el mercado hay aplicaciones como la desarrollada por el equipo de Mgestyk, que permiten el control de juegos mediante reconocimiento de gestos[1]. También existen soluciones hardware como el Leap Motion Controller[2] que emplean sensores para detectar y utilizar el movimiento de la mano y de los dedos como un dispositivo de entrada para el ordenador.

Aunque la industria de los juegos sería la más impactada por esta manera de interaccionar, no es la única industria que podría beneficiar de esta técnica, como sería

[1] Mgestyk wireless game control software at CES 2009: https://www.youtube.com/watch?v=C9WngYJoFVI
[2] Leap Motion Controller: https://www.ultraleap.com/product/leap-motion-controller/

el sector médico [5], [6] ó el sector militar [7].

# 3 Objetivos

En este proyecto vamos a crear un programa que simula el teclado mediante gestos programados y enlazarlo con un juego desarrollado previamente en otra asignatura. La idea es aislar la simulación del teclado de la implementación del videojuego para así poder utilizar el proyecto para interactuar con otras aplicaciones también.

Los principales objetivos a lograr en este proyecto se pueden ver en la Tabla 1.

| | |
|---|---|
| Objetivos Generales | Ser capaz de generar una aplicación de reconocimiento de gestos para ofrecer otra alternativa de interaccionar con una aplicación |
| Objetivos Específicos | 1. Familiarizarse con el funcionamiento de la librería OpenCV con el fin de:<br>● procesar imágenes en tiempo real<br>● aplicar filtros sobre las imágenes<br>● aislar características de interés de las imágenes<br>2. Saber los pasos necesarios para preprocesar y normalizar imágenes para crear un dataset.<br>3. Saber cómo crear una clasificador básico empleando un modelo de red neuronal convolucional.<br>4. Ser capaz de incorporar el clasificador de gestos en un programa en tiempo real.<br>5. Familiarizarse con los eventos de teclado para poder integrar el reconocedor de gestos con otras aplicaciones. |

Tabla 1. Principales objetivos del proyecto

# 4 Plan de actividades y planificación temporal

En este apartado se van a presentar las distintas actividades del proyecto y su correspondiente planificación temporal realizada después de una fase de análisis de la complejidad del problema. También se va a mostrar una diagrama de Gantt con el tiempo previsto y el tiempo real dedicado a cada una de las fases.

| No | Duración | Actividad | Subtareas |
|---|---|---|---|
| 1. | 5 dias | Análisis del problema y búsqueda de proyectos similares | |
| 2. | 7 días | Familiarizarse con el set de filtros de OpenCV para eliminar ruido o enfocar ciertas características de una imagen. | 1. Capturar imágenes con la cámara web<br>2. Enfocar la imagen para aumentar la claridad del contorno de los objetos<br>3. Suavizar la imagen para eliminar ruido<br>4. Convertir de un espacio de color a otro y aplicar umbrales de color para aislar ciertas características. |
| 3.. | 7 dìas | Trabajar con imágenes en tiempo real aplicando los filtros necesarios para aislar la forma de la mano que pueda ser reconocida | 1. Aplicar los filtros aprendidos anteriormente para eliminar el fondo de la imagen, aislando el contorno de la mano. |
| 4. | 7 días | Diseñar una colección de 4 gestos para simular las 4 flechas del teclado | |
| 5. | 21 días | Emplear un clasificador para reconocer los gestos previamente diseñados. | 1. Validar el funcionamiento de un clasificador de gestos, comprobando que efectivamente reconoce los gestos de las imágenes preprocesadas anteriormente. |
| 6. | 2 días | Integrar el clasificador en el programa y generar eventos de teclado | |
| 7. | 2 dias | Enlazar el programa de reconocimiento de gestos con el simulador de conducción. | |

| | | | |
|---|---|---|---|
| 8. | 2 dias | Hacer pequeñas modificaciones del videojuego (opcional) | 1. Añadir elementos de sonido al juego |

Tabla 2. Plan de Actividades Inicial.

A continuación se puede ver el diagrama de Gantt resultante:

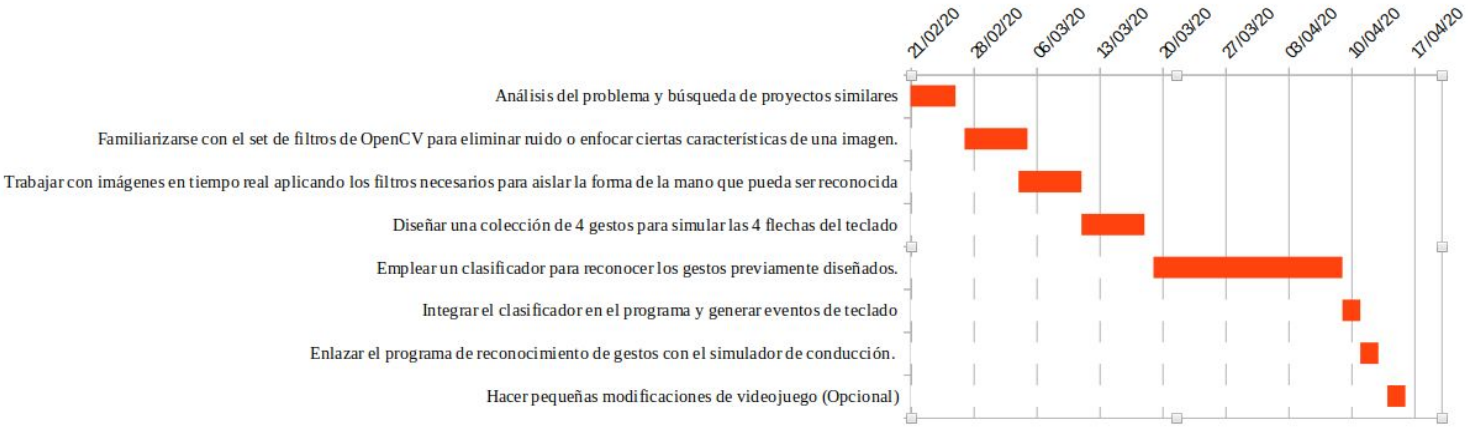



Figura 1. Diagrama de Gantt prevista.

La Figura 2 muestra el diagrama de Gantt real.

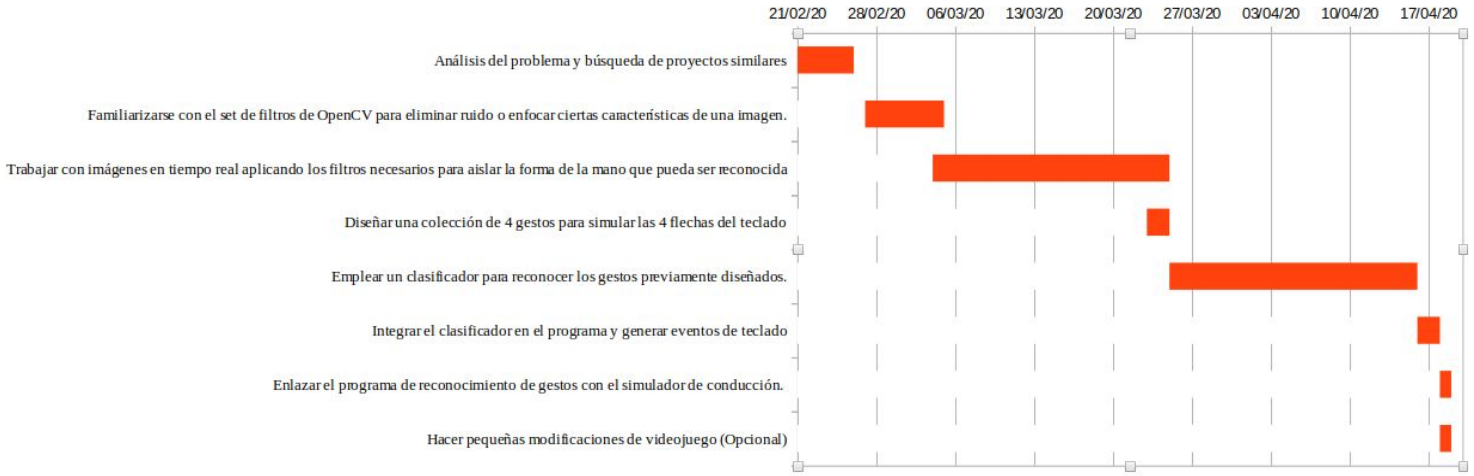



Figura 2. Diagrama de Gantt real.

Como se puede observar, hay una significante diferencia entre las dos gráficas. Las primeras dos actividades no se han visto afectadas, pero la actividad número 3 para aislar la forma de la mano, se ha alargado mucho más de lo previsto (de 7 días a 21

días). Esto ha afectado casi toda la planificación del proyecto. Para recuperar algo de tiempo, hemos hecho en paralelo la tarea de diseñar una colección de gestos y que finalmente se ha podido completar en sólo 2 días en vez de las 7 previstas.

Dado que el plazo de finalización del proyecto se ha alargado demasiado, hemos decidido no realizar la tarea número 8 que en principio era opcional.

# 5 Entorno de Desarrollo y Librerías necesarias

En la fase de análisis del programa se han estudiado los recursos necesarios para llevar a cabo el proyecto, y se ha decido realizarlo bajo el sistema operativo **Linux**, con el lenguaje de programación **Python,** ya que era el lenguaje de programación con el que estábamos más acostumbrados. Para poder capturar las imágenes desde la cámara web, se ha usado la Librería **OpenCV** de Python. El clasificador de imágenes ha sido construido con el módulo **Keras** proporcionado por **Tensorflow**, por tanto será necesaria su instalación.

Como se han requerido una serie de Librerías que permiten el correcto funcionamiento de este proyecto, supondremos que se tiene ya tanto descargado Python como el Paquete pip, pero el resto será necesario descargarlo e instalarlo, para ello haremos uso del instalador pip, de forma que en una sola línea se tendrán todas las librerías necesarias instaladas.

En la descarga del archivo, se tendrá un archivo de texto denominado “requirements.txt”. En dicho fichero están todas las librerías necesarias con sus respectivas versiones incluidas. En caso de que este fichero se viera corrupto y no se pudiera abrir, se mostraría aquí su contenido para poder descargar cada uno de ellos manualmente en caso de que hubiera algún problema.

*screeninfo==0.6.5*
*opencv_contrib_python==4.2.0.34*
*pynput==1.6.8*
*numpy==1.17.2*
*tensorflow==2.1.0*
*Pillow==7.1.1*

En el caso en el que se quiera instalar automáticamente, con el siguiente comando se

instalarán todos.

*pip install -r requirements.txt*

Dado que la instalación de la Librería **Tensorflow** es propensa a dar errores, se adjuntará un enlace con una guía detallada de como instalar **Tensorflow** y todas las librerías adicionales que este requiera[3].

# 6 Desarrollo

Una vez configurado el entorno de trabajo hemos pasado a la fase de desarrollo. Está fase está dividida en tres grandes bloques:

1. Crear una interfaz compuesta por 3 ventanas para facilitar la depuración de los filtros y de los procedimientos aplicados a las imágenes.
2. Crear un modelo de red neuronal convolucional y entrenar a partir de un conjunto de imágenes propias.
3. Integración del clasificador obtenido en el punto 2 para generar eventos de teclado y así poder controlar una aplicación externa.

## 6.1 Interfaz

Procedemos a desarrollar la interfaz para que, mediante la cámara poder obtener imágenes en tiempo real. Como se ha mencionado antes, la interfaz será compuesta por 3 ventanas, de cual dos tienen el objetivo de poder ver una imágen de cada mano por separado, y la tercera de mostrar la imagen obtenida por la cámara.

### 6.1.1 Imagen capturado por la cámara

En el primer paso, tenemos que conseguir obtener la imagen capturada de la cámara y mostrarla en la pantalla.

[3] Install Tensorflow 2.* on Ubuntu 19.10 with GPU support
https://medium.com/@Oysiyl/install-tensorflow-2-with-gpu-support-on-ubuntu-19-10-f502ae85593c

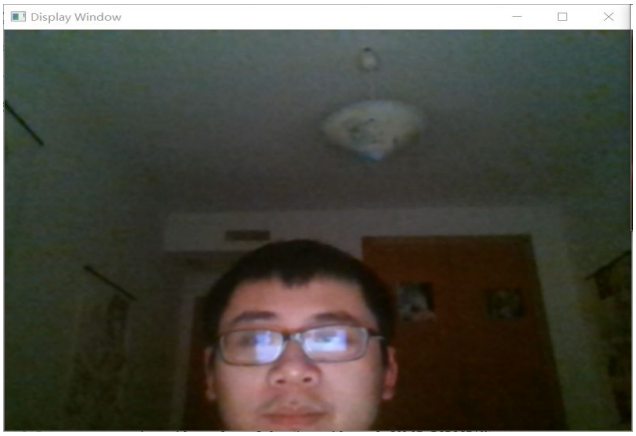



Figura 3.Captura de la cámara

Dado que la imágen obtenida por la cámara suele ser desenfocada, le hemos aplicado un primer filtro de erosión para mejorar la claridad. También ha sido necesario aplicarle una rotación en el eje X a la imagen al obtener la imagen desde la cámara con un efecto espejo.

### 6.1.2 Ventanas Auxiliares

Con la finalidad de facilitar el reconocimiento de los gestos, hemos creado las dos ventanas auxiliares para limitar la región de reconocimiento. Dichas ventanas se han recortado a partir de la imagen central y dado que tienen un tamaño reducido, ofrecen al usuario una retroalimentación de donde debería mantener las manos para poder controlar la aplicación. La posición de dichas ventanas ha sido pensada para ofrecer al usuario una posición lo suficientemente cómoda y natural.

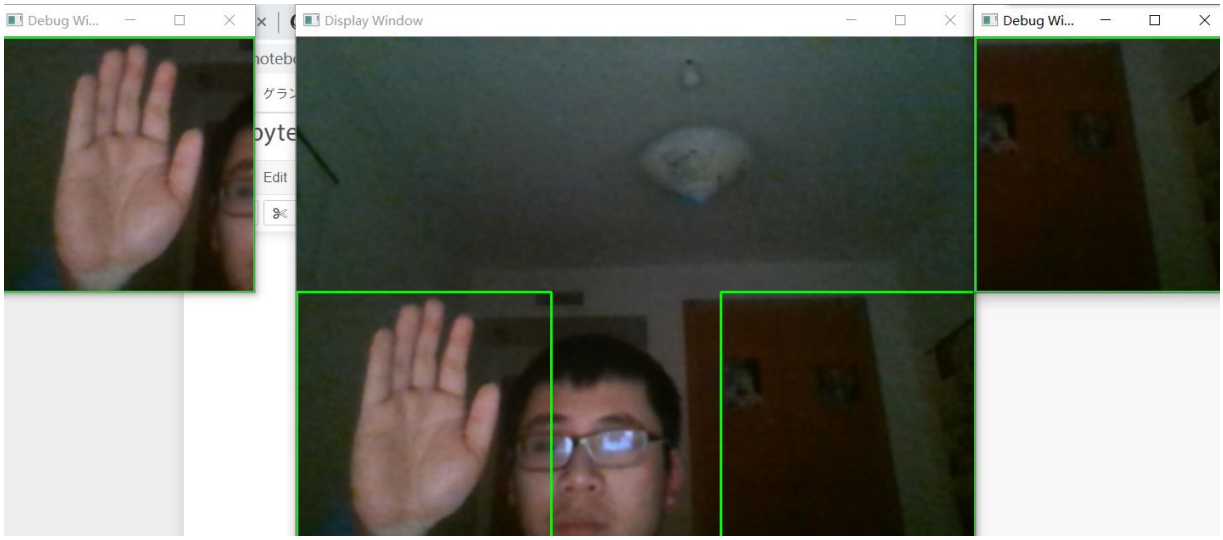



Figura 4.Regiones de reconocimiento

### 6.1.3 Eliminar Fondo

Ahora que ya tenemos las ventanas correspondientes a cada mano, tenemos que preprocesar las imágenes obtenidas por la cámara para tenerlas listas para la fase de reconocimiento. Para ello hemos desarrollado dos alternativas, ambas son válidas y dan buenos resultados.
En la primera tenemos una barra en la ventana principal que se denomina Threshold, esta nos permite ir segmentando la imagen hasta crear una máscara binaria que usaremos para calcular la forma de la mano y eliminar el ruido que se nos pueda generar. Hay veces en que este ruido es imposible de eliminar. También se ha añadido una opción de invertir el color de la imagen por si se trabaja con la luz encendida o se trabaja en modo oscuro.
La segunda alternativa trata de eliminar el fondo empleando una función de calibrar la cámara. Dicha función guarda una imagen (Imagen Zero) al iniciar la cámara, donde las manos del usuario no deberían ser visibles y esta imagen se invierte y se le suma a lo que se está mostrando posteriormente. De esta forma se cancela lo que sería el fondo de la imagen y facilita la detección de las manos, ya que inicialmente las mano nos son visibles. [8], [9].

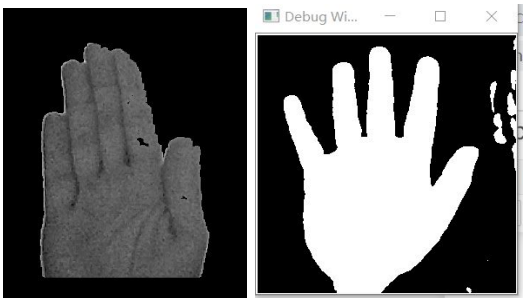



Figura 5. Resultados de las alternativas para eliminar el Fondo(Izquierda con Threshold)

Además de usar estos dos métodos, se ha intentado aislar la forma de la mano empleando filtros de color para detectar el color de la piel. Hemos decidido no emplear estos filtros al ser demasiado dependientes de la luz, y por tanto, no nos proporcionaban buenos resultados. Ejemplos de estos métodos se pueden encontrar en [10],[11].
Existen ventajas y desventajas de cada uno de los filtros para eliminar el fondo y aislar la forma de la mano. En el caso del Threshold si tienes una ventana delante, no afectaría mucho, pues generaría una pequeña sombra en la mano pero lo reconoce bien, sin embargo, en el caso de la imagen Zero, significaría que la luz que ha percibido al inicio será distinta a la actual, pues el reflejo genera ruido y el resultado final contendría demasiado ruido.

Por otra parte, sin luz no se observaría correctamente la mano en el caso del Threshold, pero , por el contrario, sí que se vería con la imagen Zero.

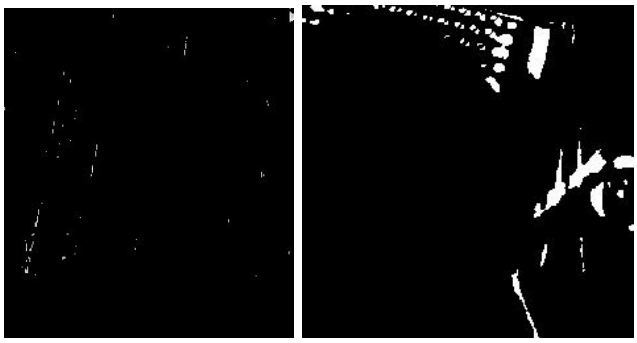


Figura 4. Ruido que se genera por la luz con Threshold

## 6.2 Reconocimiento de Gestos

Una vez eliminado el fondo y aislado la forma de la mano, las imágenes ya estarán listas para ser reconocidas. En este proyecto se han desarrollado dos técnicas distintas para el reconocimiento de los gestos.

La primera técnica que vamos a presentar es el empleo de un clasificador basado en redes neuronales convolucionales (CNN). En [12],[13] se muestra como se podría crear un clasificador de gestos basado en CNN. Hemos intentado integrar directamente los clasificadores presentados en estos dos tutoriales sin mucho éxito. El problema que se daba era por la diferencia entre las imágenes obtenidas por la cámara y de las imágenes con las que se ha entrenado la red propuesta. Dichas diferencias se debían al ángulo en el cual se han capturado las imágenes, y a una diferencia de luz y de filtros aplicados para aislar del fondo la mano. En la Figura 5 se pueden observar dichas diferencias.

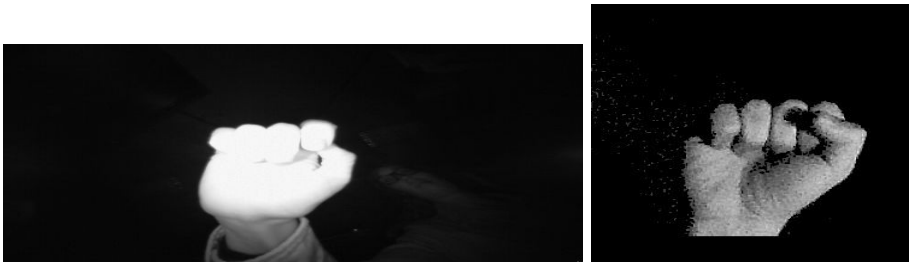


Figura 5. Dataset [13] (izquierda). Imagen propia (derecha)

Dado que el clasificador entrenado con la imagen izquierda de la Figura 5, sería capaz de reconocer imágenes con la misma iluminación y ángulo, hemos decidido crear un dataset de imágenes propios que aumentaría la probabilidad de reconocimiento de los gestos.

## 6.2.1 Red Neuronal Convolucional

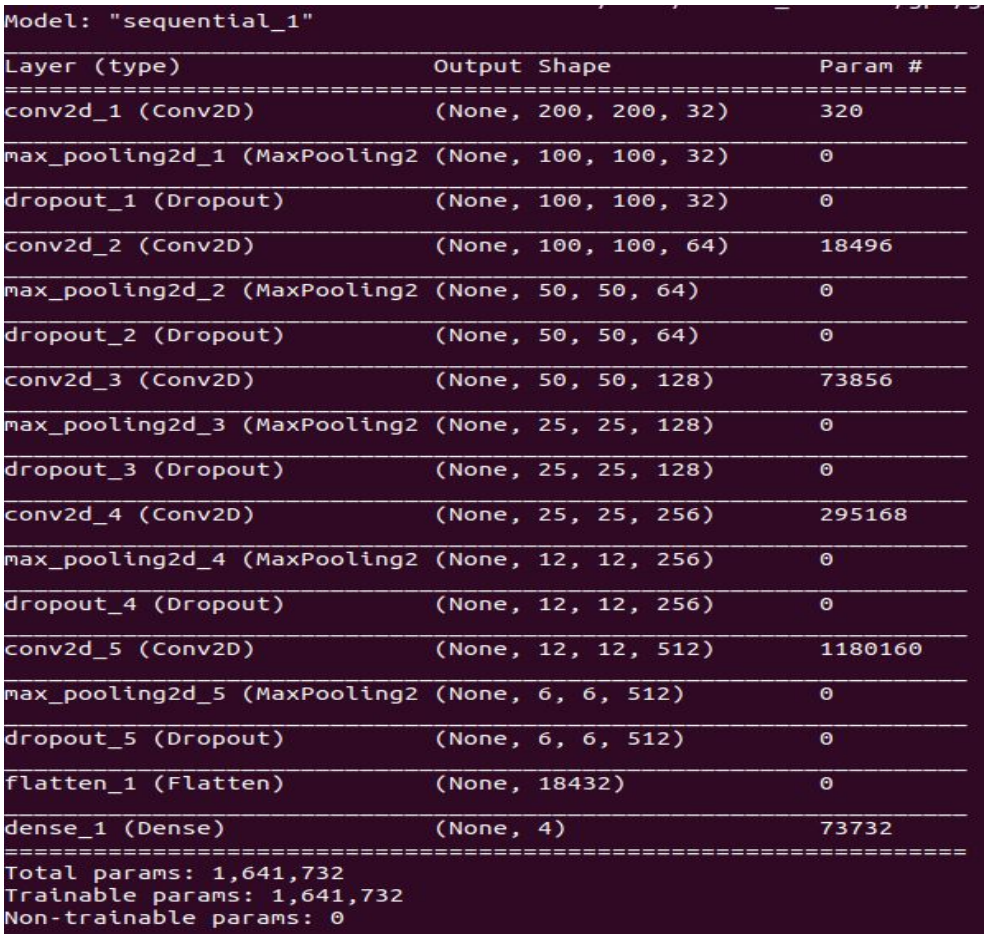
Los pasos a seguir en la creación de un clasificador CNN serían:

- Definir un modelo de red que permitiría aprender las características de las distintas clases de gestos y discriminar entre ellas
- Cargar los datos de entrenamiento y test, y normalizarlos si hace falta para adaptarlos a la red.
- Entrenar y evaluar los resultados obtenidos

A continuación se presentarán dichas fases.

### 6.2.1.1 Arquitectura del modelo

Para definir el modelo empleado en este proyecto, nos hemos basado en los ejemplos encontrados en [13] y [14]. Este modelo está formado por una sucesión de 5 capas convolucionales para aprender las distintivas que componen una imagen, seguidas por capas de Max Pooling para reducir el tamaño de las distintivas, para así llegar a aprender detalles cada vez más sutiles de las imágenes, y finalmente como una sexta capa, una capa totalmente conectada para obtener la salida del clasificador. Cada capa convolucional aplica una función de activación de tipo ReLU y usan una ventana de convolución de 3x3 píxeles.
En la Figura 6 se puede observar la arquitectura completa del modelo.



```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 200, 200, 32)      320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 100, 100, 32)      0
_________________________________________________________________
dropout_1 (Dropout)          (None, 100, 100, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 100, 100, 64)      18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 50, 50, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 50, 50, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 50, 50, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 25, 25, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 25, 25, 128)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 25, 25, 256)       295168
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 12, 12, 256)       0
_________________________________________________________________
dropout_4 (Dropout)          (None, 12, 12, 256)       0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 12, 12, 512)       1180160
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 6, 6, 512)         0
_________________________________________________________________
dropout_5 (Dropout)          (None, 6, 6, 512)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 18432)             0
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 73732
=================================================================
Total params: 1,641,732
Trainable params: 1,641,732
Non-trainable params: 0
```

Figura 5. Arquitectura del modelo CNN.

Como se puede observar en la Figura 5, después de cada capa de Max Pooling, se ha añadido una capa Dropout que desactiva un 25% de las neuronas de la capa anterior para reducir el sobreentrenamiento de la red.

## 6.2.1.2 Preprocesamiento

Una vez configurada la red, el siguiente paso sería cargar los datos y normalizarlos para poder entrenar. Dado que ha sido necesario crear un conjunto de imágenes propio, antes de entrar en los detalles de preprocesamiento vamos a explicar como se ha creado.

Todas las imágenes que hemos capturado a través de la cámara han pasado por los filtros explicados previamente para aislar la forma de la mano y minimizar el ruido en la medida de lo posible. Hemos tenido que ir aumentando el Threshold hasta que el fondo se quedase completamente negro o con cierto ruido, además, según la cámara del dispositivo, la imagen salía invertida, es decir, la mano era negra y el fondo blanco.

Para ello hemos habilitado un evento, al accionar “I” se invierte la imagen, ya que si sucede eso y no se invierte la imagen no se reconoce correctamente el gesto.

Figura 6. Imagen sin Invertir(en este caso sería necesario invertirlo)

Se han tomado fotos bajo 2 tipos de fuente de luz, para maximizar la probabilidad de reconocimiento ante imágenes con distintas iluminaciones. Cada foto ha sido capturada desde una posición ligeramente diferente para tener una mayor variabilidad en el conjunto. Para realizar una gran cantidad de imágenes, se ha realizado un evento con el botón “C” para poder realizar imágenes consecutivas, numeradas de forma ordenada y con el tipo de gesto correspondiente:

```
num_pic = 0
if key == 99:  #Presiono C
    num_class = "C_"
    cv2.imwrite("./data/" + num_class + str(num_pic) + ".jpg", left_hand)
    num_pic += 1
```

Por ejemplo en este caso, la primera imagen tendrá de nombre C_01.png y se guardará en la carpeta “data”. La idea se ha obtenido de uno de los proyectos que hemos encontrado en github, el cual hacía lo mismo, generaba su propio dataset. [13]
Para crear este conjunto de datos, es importante notar que se han usado solo capturas de la mano izquierda. No ha sido necesario crear imágenes de las dos manos, al poder aplicar una rotación en el eje X para invertir la foto de la mano derecha.
En total nuestra base de datos contiene 820 imágenes, cada una con su correspondiente etiqueta de gesto codificada en el propio nombre de la imagen. Hemos elegido codificar la clase en el nombre para facilitar el proceso de verificación de errores una vez entrenado el modelo. Hay 4 tipos de imágenes, donde cada tipo representa un gesto y dentro de cada tipo de gesto hay modelos de manos distintas, realizadas por cada uno de los alumnos del proyecto. Esto se realiza para no

sobreajustar el entrenamiento y permitir una mejor generalización del proceso de predicción, permitiendo tener cierta flexibilidad a la hora de clasificar.

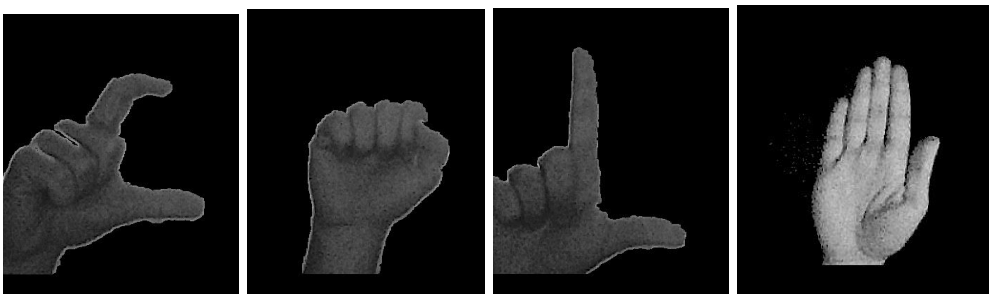


Figura 7. imágenes contenidas en la base de datos

Una vez creado el conjunto de imágenes, éstas se han tenido que cargar en un vector y redimensionar a un tamaño de 200x200 pixeles para poder entrenar la red. Cada imágen ha sido cargada 3 veces para aumentar el conjunto de datos. Todas las imágenes han sido convertidas a escala de grises y finalmente han sido normalizadas para trabajar con valores entre 0 y 1 para facilitar el proceso de entrenamiento. De cada imagen se ha extraído su correspondiente clase y se ha creado un vector de etiquetas con entradas para cada una de las imágenes. A continuación se han aleatorizado las posiciones de las imágenes y de sus clases dentro de los vectores (siguiendo el mismo criterio en los dos casos) para asegurarnos de no introducir un sesgo en el proceso de clasificación. El siguiente paso ha sido particionar el conjunto de datos, de los cuales se han guardado un 20% para test; de los 80% restantes el 20% se han guardado para validación. Ahora que tenemos listos los datos, podemos proceder con la fase de entrenamiento.

## 6.2.1.3 Entrenamiento y Resultados

Para entrenamiento hemos fijado lotes de 50 muestras y un total de 10 epochs. Hemos elegido un tipo de perdida de tipo “crossentropy” una función de optimización del tipo “stochactic gradient descent”. Finalmente hemos usado una métrica de “accuracy”.
En la Figura 8 y la Figura 9 se puede ver el proceso de entrenamiento.

```
_________________________________________________________________
Train on 1572 samples, validate on 393 samples
Epoch 1/10
1572/1572 [==============================] - 66s 42ms/step - loss: 1.6261 - accuracy: 0.4587 - val_loss: 1.6349 - val_accuracy: 0.45
55
Epoch 2/10
1572/1572 [==============================] - 68s 43ms/step - loss: 1.1471 - accuracy: 0.7156 - val_loss: 1.0521 - val_accuracy: 0.77
10
Epoch 3/10
1572/1572 [==============================] - 72s 46ms/step - loss: 0.7212 - accuracy: 0.8632 - val_loss: 0.8597 - val_accuracy: 0.80
41
Epoch 4/10
1572/1572 [==============================] - 78s 49ms/step - loss: 0.5626 - accuracy: 0.9109 - val_loss: 0.6308 - val_accuracy: 0.92
88
Epoch 5/10
1572/1572 [==============================] - 74s 47ms/step - loss: 0.4633 - accuracy: 0.9555 - val_loss: 0.4809 - val_accuracy: 0.98
47
Epoch 6/10
1572/1572 [==============================] - 75s 48ms/step - loss: 0.4010 - accuracy: 0.9765 - val_loss: 0.4334 - val_accuracy: 0.99
49
Epoch 7/10
1572/1572 [==============================] - 73s 46ms/step - loss: 0.3724 - accuracy: 0.9879 - val_loss: 0.5623 - val_accuracy: 0.89
82
Epoch 8/10
1572/1572 [==============================] - 73s 46ms/step - loss: 0.4029 - accuracy: 0.9733 - val_loss: 0.3841 - val_accuracy: 1.00
00
Epoch 9/10
1572/1572 [==============================] - 73s 46ms/step - loss: 0.3526 - accuracy: 0.9917 - val_loss: 0.3632 - val_accuracy: 0.99
75
Epoch 10/10
1572/1572 [==============================] - 80s 51ms/step - loss: 0.3399 - accuracy: 0.9968 - val_loss: 0.3561 - val_accuracy: 0.99
75
492/492 [==============================] - 7s 14ms/step
Test loss: 0.36207976743457765
Test accuracy: 0.9959349632263184
```

Figura 8. Detalles de entrenamiento de la CNN.

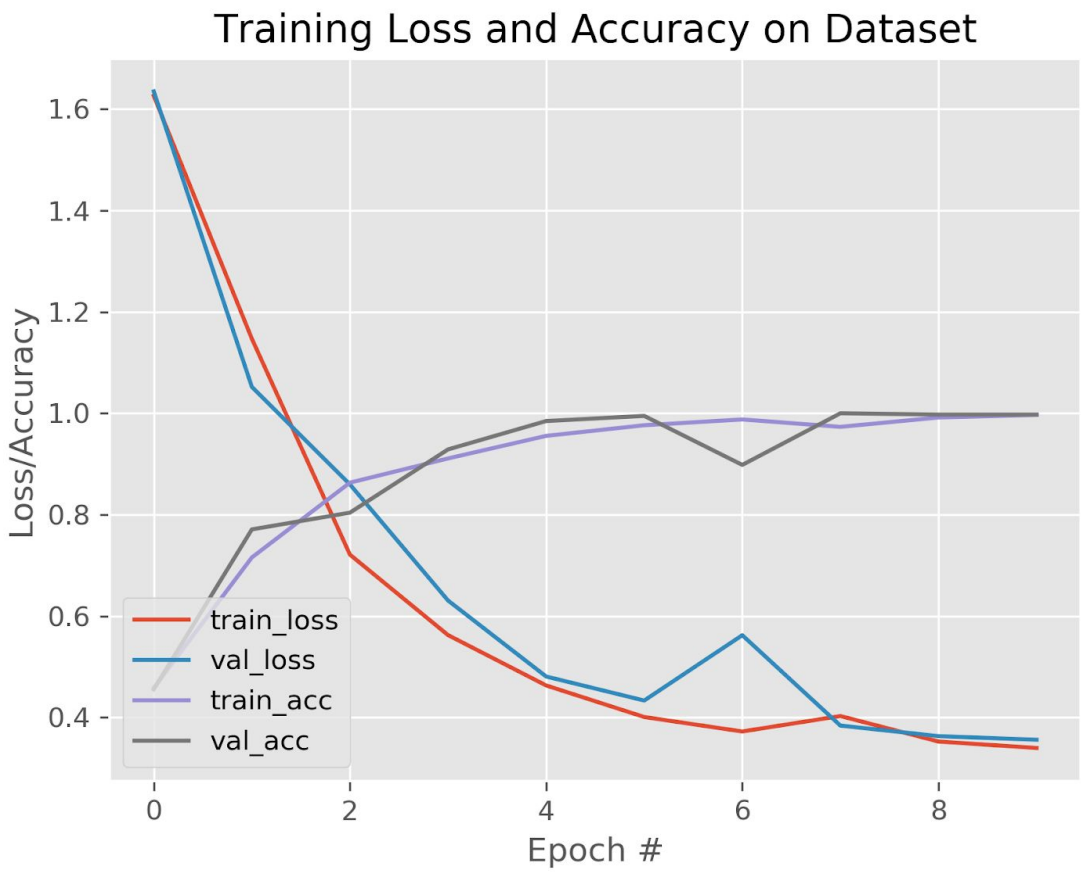



Figura 9. Representación gráfica del proceso de entrenamiento.

En las Figuras 8 y 9 se puede observar que se ha obtenido un resultado muy satisfactorio de 99.59% accuracy. En el epoch 6 se nota un sobreajuste a los datos de entrenamiento seguido de un reajuste de los parámetros del modelo.

### 6.2.2 Alternativa a la CNN

Aparte del reconocimiento con el modelo de red neuronal también hemos probado reconocer los gestos mediante otra técnica. En el apartado de aislar la forma de la mano del fondo hemos visto que hemos conseguido que en la mayor parte de la región esté solo la mano. Pero dicho método no es infalible y se puede ver como en la Figura 10 también se han detectado las orejas y los hombros.

En este apartado vamos a intentar obtener la información de la mano y reconocerla. para eso vamos a necesitar dos funciones que proporciona OpenCV, en concreto findContours [15] y convexHull [16].

Con la función findContours se han detectado cuántas figuras hay en la imagen y cuales son los píxeles que definen el contorno de esa figura. Como antes ya hemos conseguido filtrar la mayoría del fondo de la imagen y por tanto la mayor parte de la región contiene la forma de la mano, entonces dicha figura es la que tiene la mayor superficie de puntos y la pasamos a la función convexHull.

ConvexHull recibe los puntos que definen el contorno de la mano, y devuelve una serie de puntos que dibujan un polígono que delimitan la figura recibida. Pero, algunos puntos devueltos por convexHull están muy cerca, por ejemplo, en un dedo puede que hayan 5-6 puntos. Para resolver este problema hemos clasificado los puntos dependiendo de la separación. Si la distancia entre 2 puntos es menor que 30 distancia manhattan (fácil de calcular) entonces consideramos que los dos puntos representan el mismo dedo.

La mano derecha la hemos decidido usar para girar a izquierda y derecha. Cuando se muestra el pulgar, se simula un giro a la izquierda y cuando saca el meñique, se simula un giro a la derecha. Estos 2 gestos tienen una característica que consiste en tener un punto alejado de los otros para así poder identificar el gesto. Pero esta característica no es fiable, ya que depende de la distancia entre la mano y la cámara. Al no poder medir con una distancia constante, procedemos a calcular el centro del gravedad que hemos aprendido en la práctica 2 de la asignatura [8]. Por tanto, calculamos una distancia media de cada punto al centro de gravedad pero sin tener en cuenta el punto extremo izquierdo o derecho. Si la distancia del punto extremo izquierdo o derecho es más

lejano que una proporción entonces el gesto es reconocible como se puede observar en la siguiente figura:

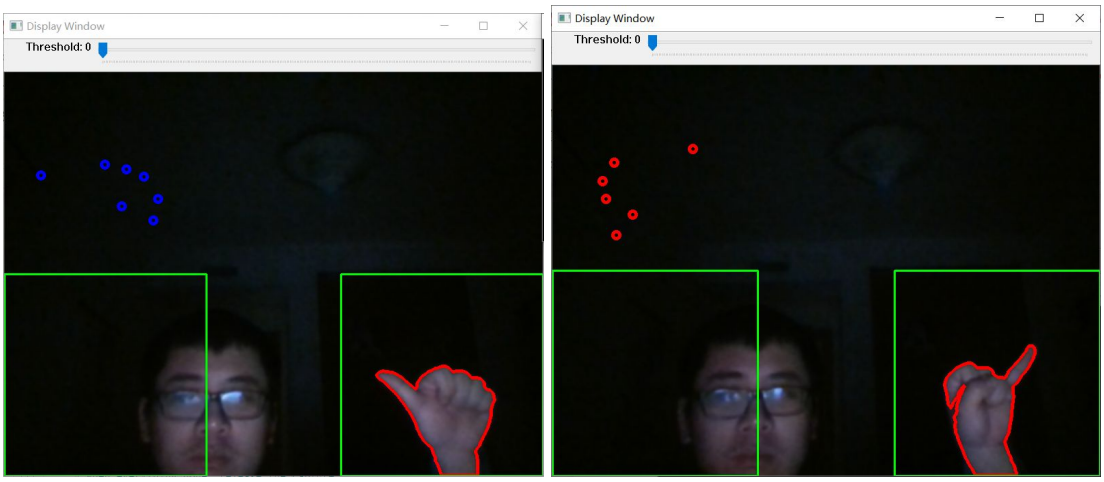
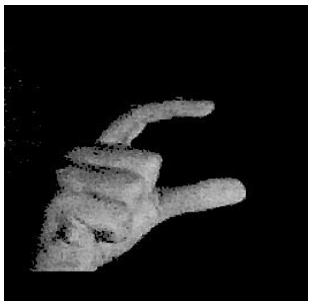



Figura 10. Pulgar (izquierdo) , Meñique(derecho)

Teniendo esto en cuenta, se pueden generar más gestos, entre los cuales, uno podría ser por ejemplo reconocer cada dedo de la mano o contar el número de dedos que hay en la imagen.

## 6.3 Generación de Eventos

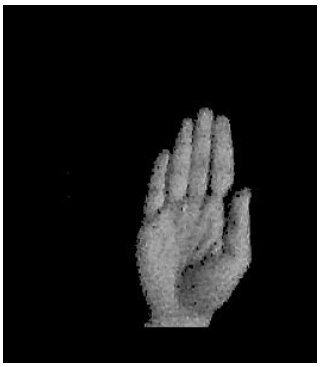
Teniendo ya el reconocedor de gestos, lo siguiente será según qué gesto ha reconocido, ejecutar una pulsación de teclado, para ello,utilizaremos la librería pynput, el cual nos permite “ejecutar” de teclado cualquier botón mediante la función press(). Esto hace que las teclas simuladas por los gestos sean fácil de configurar dependiendo de la aplicación con la cual se quiere interaccionar.
A continuación se muestra un ejemplo de una posible asignación entre un gesto y una tecla:



*if class_num[l_max] == '**C**':*
*keyboard.press(**Key.down**)*

*elif class_num[l_max] == **'fist'**:*
*keyboard.press(**Key.left**)*

*elif class_num[l_max] == **'L'** :*
*keyboard.press(**'L'**)*

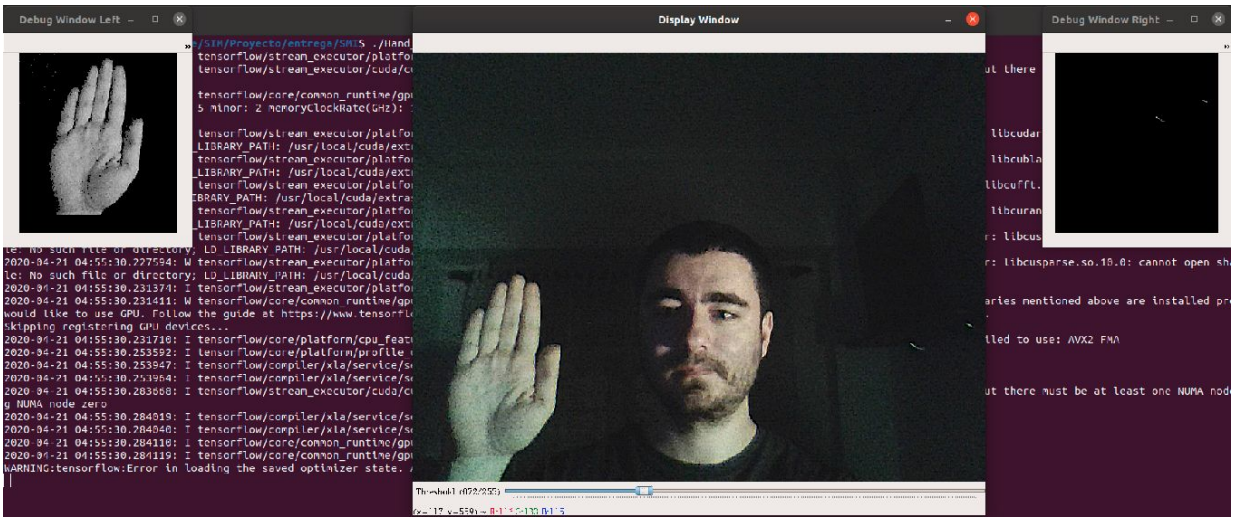


*elif class_num[l_max] == **'palm'** :*
*keyboard.press(**Key.up**)*

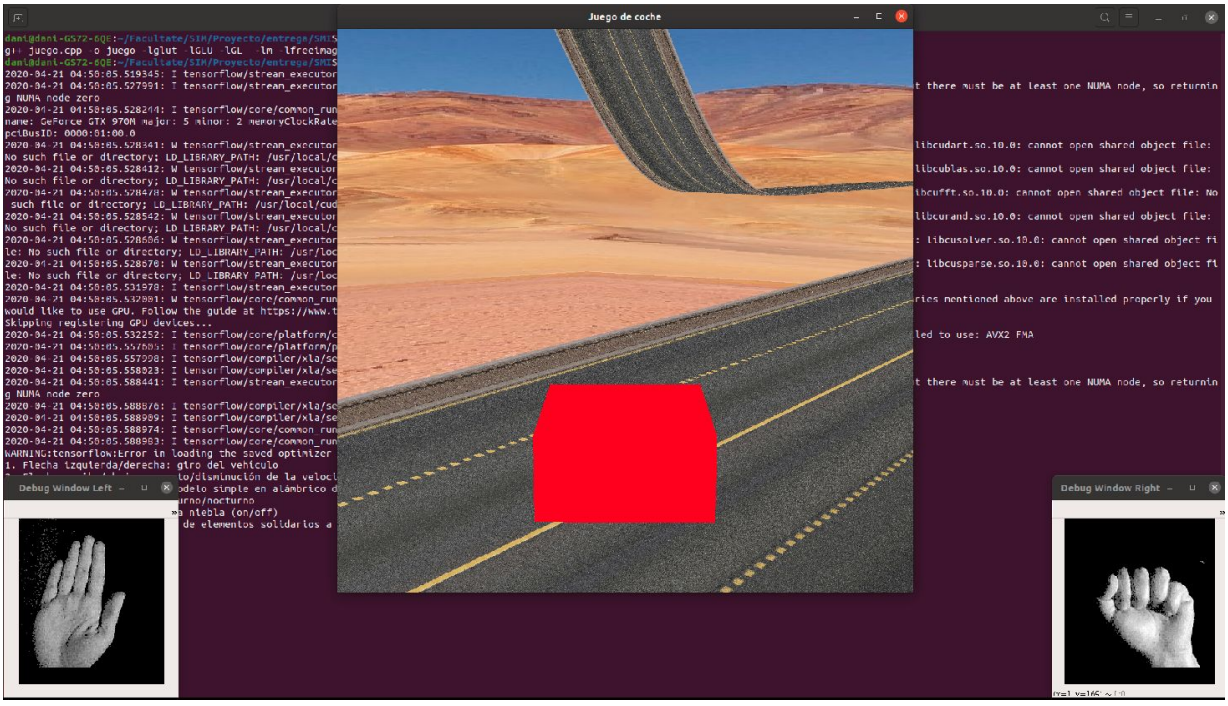
Como se observa en el código, si el predictor dice que el gesto es una C, el botón que presione será el de en nuestro caso sería la flecha hacia abajo en nuestro teclado [17]. Se tiene que tener en cuenta el hecho de que se puede diferenciar si el gesto proviene de la ventana asociada a la mano izquierda o de la mano derecha. Dicha diferenciación consigue doblar el número de teclas simuladas mediante gestos.

A continuación se puede ver un ejemplo del uso de esta manera de interaccionar con un juego desarrollado previamente en OpenGL:

Primer paso: Elegir el valor de Threshold que consigue aislar las manos:




Segundo paso, abrir e interaccionar con la aplicación proporcionada como argumento al programa:



# 7 Conclusión y Futuras Mejoras

Finalmente, podemos decir que aunque el proyecto tenga ciertas limitaciones se puede considerar un éxito, pues se consideraría un ejemplo más de que es lo que podríamos llegar a hacer mediante reconocimiento de gestos, pero ello no significa que el resultado siempre sea el mismo, pues existen una serie de factores que hacen difícil el reconocimiento, por ello queremos hacer una lista de futuras mejoras.

Una de las mejoras que aumentaría la precisión de la interacción mediante gestos sería poder aislar mejor la mano del fondo, ya que se genera bastante ruido y esto provoca un mal funcionamiento del programa. Mejorar este aspecto significaría una mejora en el proceso de detección y reconocimiento de gestos.

Otras de las posibles mejoras que podríamos realizar, sería el paso de mensajes a la aplicación externa, ya que se envían demasiados mensajes y esto sobrecarga la aplicación. Se ha intentado regular esperando diez frames consecutivos hasta tomar una decisión de que signo se ha interpretado, reduciendo el ruido producido por las transiciones entre gestos e introduciendo un pequeño delay entre el envío de eventos del teclado. También se ha introducido un temporizador para restringir la generación de nuevos eventos cada 100 milisegundos.

Finalmente, otra posible mejora sería encontrar unos gestos adecuados que permitirá al usuario no desplazar tanto la mano, ya que en el proceso de cambio de un tipo de gesto a otro, se reconocería incorrectamente algún que otro gesto indeseado y por tanto introducir efectos inesperados en la aplicación.

# 8 Bibliografía


[1] How Much Data Is Created Every Day? [Consultado 19/04/2020]

URL <https://seedscientific.com/how-much-data-is-created-every-day/>

[2] Human–computer interaction: [Consultado 19/04/2020] URL <https://en.wikipedia.org/wiki/Human%E2%80%93computer_interaction>

[3] Dardas, Nasser & Alhaj, Mohammad. (2011). Hand Gesture Interaction with a 3D Virtual Environment.

[4] How Gesture Control Could Transform Video Games [Consultado 02/03/2020] URL <https://www.livescience.com/22157-gesture-control-could-transform-video-games.html>

[5] Wachs J. et al. (2006) A Real-Time Hand Gesture Interface for Medical Visualization Applications. In: Tiwari A., Roy R., Knowles J., Avineri E., Dahal K. (eds) Applications of Soft Computing. Advances in Intelligent and Soft Computing, vol 36. Springer, Berlin, Heidelberg

[6] Jacob MG, Wachs JP, Packer RA. Hand-gesture-based sterile interface for the operating room using contextual cues for the navigation of radiological images. *J Am Med Inform Assoc*. 2013;20(e1):e183–e186. doi:10.1136/amiajnl-2012-001212

[7] Choi, Weonji. “Army Hand Signal Recognition System using Smartwatch Sensors.” (2018).

[8] Agustí, M.: “Práctica de introducción al empleo de técnicas de Visión por Computador: introducción a OpenCV ”. Universidad Politécnica de Valencia, Febrero 2020.

[9] How to Use Background Subtraction Method en:

https://docs.opencv.org/3.4/d1/dc5/tutorial_background_subtraction.html

[10] Skin Detection: A Step-by-Step Example using Python and OpenCV:

URL
<https://www.pyimagesearch.com/2014/08/18/skin-detection-step-step-example-using-python-opencv/>

[11] Skin Detection Using OpenCV Python:

URL <https://nalinc.github.io/blog/2018/skin-detection-python-opencv/>

[12] Build Hand Gesture Recognition from Scratch using Neural Network — Machine Learning Easy and Fun URL
<https://towardsdatascience.com/build-hand-gesture-recognition-from-scratch-using-neural-network-machine-learning-easy-and-fun-d7652dd105af>

[13] Tutorial: Using Deep Learning and CNNs to make a Hand Gesture recognition model URL
<https://towardsdatascience.com/tutorial-using-deep-learning-and-cnns-to-make-a-hand-gesture-recognition-model-371770b63a51>

[14] Convolutional Neural Networks for Beginners URL
<https://towardsdatascience.com/convolutional-neural-networks-for-beginners-practical-guide-with-python-and-keras-dc688ea90dca>

[15] Structural Analysis and Shape Descriptors URL
<https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=findcontours#structural-analysis-and-shape-descriptors>

[16] Convex Hull URL<:https://docs.opencv.org/3.4/d7/d1d/tutorial_hull.html>

[17] Keyboard Library URL <https://pythonhosted.org/pynput/keyboard.html>

[18] BFMacher Class Reference URL
<https://docs.opencv.org/master/d3/da1/classcv_1_1BFMatcher.html>