ULPGC
Escuela de Ingeniería Informática

Grado en Ingeniería Informática

Asignatura: Visión por Computador

# TRABAJO FINAL DE ASIGNATURA

## *Simon Dice*

**Autores:**
Laura Herrera Negrín
Dunia Suárez Rodríguez
Ayman Asbai Ghoudan

31 de diciembre de 2025

# Índice

# 1. Introducción

## 1.1. Descripción del proyecto

El presente proyecto, titulado «Simon Dice», consiste en el desarrollo de una versión moderna e interactiva del clásico juego de memoria «Simón Dice». La innovación principal radica en la interfaz de usuario: en lugar de utilizar botones físicos o periféricos convencionales (teclado/ratón), el juego se controla enteramente mediante **visión artificial (Computer Vision)**.

El sistema es capaz de reconocer gestos faciales y manuales del jugador en tiempo real a través de una webcam, interpretándolos como comandos de juego. Esto permite una experiencia inmersiva donde el jugador debe replicar secuencias de movimientos corporales.

Este enfoque se enmarca dentro de las **interfaces naturales de usuario (NUI)**, que buscan eliminar la barrera física entre la persona y la máquina, permitiendo una interacción más intuitiva y fluida basada en las capacidades motrices humanas naturales.

## 1.2. Objetivos

- Implementar un sistema de reconocimiento de gestos robusto utilizando técnicas de aprendizaje automático (Machine Learning).
- Desarrollar una lógica de juego interactiva que responda en tiempo real a las acciones del usuario.
- Integrar librerías de visión por computador como OpenCV y MediaPipe para la extracción de características.
- Entrenar clasificadores neuronales (MLP) para distinguir entre diferentes gestos (cara y manos).

# 2. Tecnologías y herramientas

Para el desarrollo de este proyecto se ha utilizado un stack tecnológico basado en Python, aprovechando su extenso ecosistema para ciencia de datos y visión artificial.

## 2.1. Lenguaje y entorno

- **Python 3.11**: Lenguaje principal del proyecto.
- **Conda**: Gestor de entornos utilizado para aislar las dependencias (entorno `VC_Trabajo` o `memo_env`).
- **Jupyter Notebooks**: Utilizados para el prototipado, generación de datasets y entrenamiento de modelos.

## 2.2. Librerías principales

- **OpenCV (cv2)**: Herramienta fundamental para la captura de vídeo desde la webcam, procesamiento de imágenes (cambios de espacio de color BGR a RGB) y visualización en tiempo real.
- **MediaPipe**: Framework desarrollado por Google, utilizado para la detección de hitos (landmarks) faciales y manuales. Específicamente se usan los módulos `FaceMesh` (malla facial) y `Hands` (detección de manos).
- **Scikit-Learn**: Utilizada para la implementación de algoritmos de Machine Learning. En concreto, se ha empleado el `MLPClassifier` (Perceptrón Multicapa) para la clasificación de los gestos basándose en los landmarks extraídos.
- **NumPy**: Para operaciones matemáticas eficientes y manipulación de arrays, crucial para la normalización de coordenadas.
- **Pickle**: Para la serialización y guardado de los modelos entrenados (`.pkl`).

# 3. Estructura del repositorio

El proyecto se organiza en la siguiente estructura de directorios, separando el código fuente (`src`), los datos (`dataset`) y la documentación:

```
/
|-- dataset/                # Imágenes para entrenamiento
|   |-- gestos_cara/        # Dataset de gestos faciales
|   |-- gestos_manos/       # Dataset de gestos manuales
|
|-- src/                    # Código fuente del proyecto
|   |-- game/               # Lógica del juego (Simon Says)
|   |-- generate_images/    # Scripts para captura de datos
|   |-- memory/             # Documentación LaTeX
|   |-- scripts/            # Utilidades auxiliares
|   |-- train/              # Notebooks de entrenamiento (MLP)
|
|-- README.md               # Documentación general y setup
```

# 4. Metodología y desarrollo

El proyecto se divide en tres fases principales: generación del dataset, entrenamiento de modelos y lógica del juego.

## 4.1. Fase 1: Generación del dataset

Para la construcción del *dataset*, inicialmente se realizó una búsqueda de imágenes en repositorios públicos como Kaggle. Se identificaron conjuntos de datos relevantes, tales como:

- CEW Dataset (Closed Eyes in the Wild), útil para la detección de ojos cerrados.
- HaGRID (HAnd Gesture Recognition Image Dataset), para el reconocimiento de posturas de la mano.

Sin embargo, para adaptar el sistema a las necesidades específicas del juego y mejorar la robustez ante el entorno real de uso, se desarrolló un código específico para enriquecer el conjunto de datos. Mediante los scripts `GuardarGestosCara.ipynb` y `GuardarGestosManoCuerpo.ipynb`, se capturaron imágenes propias utilizando la webcam, almacenándolas en carpetas etiquetadas para complementar las muestras obtenidas de fuentes externas.

Las clases definidas para el reconocimiento incluyen:

- **Gestos faciales**: Neutro, ojos cerrados, cabeza derecha y cabeza izquierda.
- **Gestos manuales**: Mano arriba (izquierda/derecha), puños cerrados, pulgar arriba, señal de victoria, gesto de rock, llamada y OK.

## 4.2. Fase 2: Procesamiento y entrenamiento

El entrenamiento se realiza en los notebooks `EntrenoCara.ipynb` y `EntrenoManoCuerpo.ipynb`. El flujo de trabajo es el siguiente:

En lugar de procesar la imagen cruda (píxeles), lo cual sería costoso computacionalmente y propenso a errores por cambios de iluminación, se utilizan los **landmarks** (puntos clave) de MediaPipe. Esta decisión permite una inferencia mucho más rápida, esencial para la fluidez del juego.

- **Malla facial (Face Mesh)**: Genera una malla densa de 468 puntos tridimensionales que mapean la superficie del rostro. Esto permite detectar sutilezas como el cierre de los ojos o la rotación de la cabeza con gran precisión.
- **Manos (Hands)**: Identifica 21 puntos clave por cada mano, incluyendo muñeca y falanges de cada dedo. La topología de esqueleto permite inferir gestos complejos independientemente del tamaño de la mano del usuario.

#### 4.2.1. Preprocesamiento y normalización

Los puntos extraídos $(x, y, z)$ se normalizan para que el modelo sea invariante a la posición del usuario en la pantalla y a la distancia a la cámara.

```python
def normalizar_puntos(landmarks):
    coords = np.array([[lm.x, lm.y] for lm in landmarks])
    centroid = np.mean(coords, axis=0) # Centrar
    centered = coords - centroid
    max_dist = np.max(np.abs(centered)) # Escalar
    return (centered / max_dist).flatten()
```

Listing 1: Función de normalización

#### 4.2.2. Data augmentation (aumento de datos)

Para mejorar la robustez del modelo con pocos datos, se generan variaciones sintéticas de cada muestra original:

- **Ruido Gaussiano**: Simula imperfecciones en la detección.
- **Escalado**: Simula ligeros acercamientos o alejamientos.
- **Rotación**: Simula inclinaciones de la cabeza o mano.

#### 4.2.3. Modelo de clasificación

Se utiliza una red neuronal artificial (MLP).

- **Arquitectura cara**: Capas ocultas de (128, 64) neuronas.
- **Arquitectura manos**: Capas ocultas de (64, 32) neuronas.
- **Validación**: Se utiliza Cross-Validation (K-Fold estratificado) para asegurar que el modelo generaliza correctamente, obteniendo altas tasas de precisión en las pruebas.

### 4.3. Fase 3: Lógica del juego (Simon Says)

El núcleo del juego se encuentra en `src/game/game.ipynb`. El sistema funciona como una máquina de estados:

1. **Generación de secuencia**: El sistema elige una serie de gestos aleatorios.
2. **Muestra**: Se indica al usuario qué gestos debe realizar (mediante texto o iconos en pantalla).
3. **Escucha (inferencia)**: Se activa la cámara y el modelo predice en tiempo real qué gesto está haciendo el usuario.
4. **Validación**: Si el gesto coincide con el esperado en la secuencia, se avanza. Si falla o tarda demasiado, pierde.

**Mecánicas adicionales:**

- **Trampa «Simón Dice»**: El juego puede intentar engañar al usuario mostrando instrucciones con un nombre diferente (ej. «Modesto dice»). Si el usuario realiza el gesto cuando no lo dijo «Simón», pierde.
- **Dificultad incremental**: Aumenta la longitud de la secuencia y disminuye el tiempo permitido por ronda.

# 5. Desafíos técnicos y soluciones

Durante el desarrollo del proyecto surgieron varios retos inherentes a la aplicación de visión por computador en entornos no controlados:

## 5.1. Variabilidad en la iluminación

Los algoritmos de detección visual son sensibles a las condiciones de luz extremas (contraluz o oscuridad).
**Solución**: La normalización de las coordenadas de los landmarks hace que el modelo dependa de la geometría relativa de los puntos y no de los valores de intensidad de los píxeles, mitigando parcialmente este problema.

## 5.2. Oclusiones parciales

En ocasiones, al realizar gestos cerca de la cara, las manos pueden ocultar partes del rostro, confundiendo al detector facial.
**Solución**: Se implementó una lógica de priorización en el bucle principal del juego y se ajustaron los umbrales de confianza (`min_detection_confidence`) para reducir falsos positivos.

## 5.3. Latencia en tiempo real

El procesamiento de vídeo requiere un alto rendimiento para no afectar a la jugabilidad.
**Solución**: El uso de clasificadores MLP (redes neuronales simples) sobre vectores de características ligeros, en lugar de redes convolucionales profundas (CNN) sobre imágenes completas, garantiza una tasa de fotogramas (FPS) alta incluso en equipos con hardware modesto.

# 6. Manual de usuario

## 6.1. Requisitos previos

Es necesario disponer de una webcam funcional y un entorno con las librerías instaladas:

```
pip install opencv-python numpy mediapipe scikit-learn
```

## 6.2. Ejecución

1. Abrir el proyecto en un editor compatible con Jupyter Notebooks (VS Code o Jupyter Lab). 2. Ejecutar el archivo principal del juego: `src/game/game.ipynb`. 3. Seguir las instrucciones en pantalla.

## 6.3. Cómo jugar

1. El juego mostrará una secuencia de acciones (ej. «Cerrar Ojos», «Mano Arriba»).
2. Memoriza la secuencia.
3. Cuando sea tu turno, repite los gestos frente a la cámara en el mismo orden.
4. ¡Cuidado! Si la instrucción no empieza por «Simón dice» (o la indicación visual correspondiente), no debes moverte.

# 7. Conclusiones

Este proyecto demuestra la viabilidad de crear interfaces hombre-máquina naturales y accesibles utilizando hardware de consumo estándar (webcam). La combinación de la extracción de características geométricas (mediante MediaPipe) con clasificadores ligeros (MLP) permite una inferencia en tiempo real extremadamente rápida, adecuada para videojuegos.

El uso de técnicas como la normalización de coordenadas y el aumento de datos ha sido crucial para obtener un sistema robusto que funcione con diferentes usuarios y condiciones de iluminación.

# 8. Bibliografía


- Documentación de OpenCV: `https://docs.opencv.org/`
- MediaPipe Solutions: `https://google.github.io/mediapipe/`
- Scikit-Learn Documentation: `https://scikit-learn.org/stable/`
- Ultralytics YOLO (referencia conceptual): `https://github.com/ultralytics/ultralytics`