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

4 de enero de 2026

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 o mandos), 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 (gesticular, mover la cabeza, cerrar ojos) para avanzar.

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

Los objetivos principales del proyecto son:

1. **Implementar un sistema de reconocimiento de gestos robusto**: Utilizando técnicas de aprendizaje automático (Machine Learning) para identificar patrones en tiempo real.

2. **Desarrollar una lógica de juego interactiva**: Crear una máquina de estados que gestione la fluidez del juego, desde el menú hasta la detección de movimientos y la respuesta auditiva/visual.

3. **Integración tecnológica eficiente**: Combinar librerías de visión por computador (OpenCV, MediaPipe) con síntesis de voz y audio para una experiencia multimedia completa.

4. **Entrenamiento de modelos personalizados**: Crear y entrenar clasificadores neuronales (MLP) capaces de distinguir entre diferentes gestos faciales (ojos cerrados, movimientos de cabeza) y manuales (puño, mano abierta, pulgar arriba, etc.).

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, visión artificial y desarrollo de videojuegos.

2.1. Lenguaje y entorno

- **Python 3.11**: Lenguaje principal del proyecto.
- **Conda**: Gestor de entornos utilizado para aislar las dependencias.
- **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, preprocesamiento de imágenes (flip, conversión de color) y visualización de la interfaz básica.
- **MediaPipe**: Framework desarrollado por Google. Se utilizan los módulos:
  - `FaceMesh`: Para generar una malla facial de 468 puntos y detectar gestos de la cabeza y ojos.
  - `Hands`: Para detectar 21 puntos clave en las manos y reconocer posturas complejas.
- **Scikit-Learn**: Librería de ML utilizada para implementar el `MLPClassifier` (Perceptrón Multicapa), responsable de clasificar los vectores de características en gestos concretos.
- **NumPy**: Esencial para operaciones matemáticas vectoriales, normalización de coordenadas y manipulación de arrays.
- **Pickle**: Para la serialización (guardado) y carga de los modelos entrenados (`.pkl`).
- **Pygame**: Utilizada para la gestión de efectos de sonido (feedback de acierto/error) en el juego.
- **Pyttsx3**: Librería de síntesis de voz (Text-to-Speech) que permite al juego "hablar"las instrucciones al jugador (ej. "Simón dice: Levanta la mano").
- **Pillow (PIL)**: Utilizada para renderizar texto con caracteres UTF-8 (como tildes o la 'ñ') en los frames de OpenCV, ya que la fuente por defecto de OpenCV tiene limitaciones con estos caracteres.

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 desarrollo se ha estructurado en tres fases secuenciales: generación del dataset, entrenamiento de los modelos clasificadores y desarrollo de la lógica del juego.

4.1. Fase 1: Recolección del dataset

Para entrenar un modelo robusto, es esencial contar con datos de calidad. Por ello, para construir el textitdataset, 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 desarrollaron dos scripts en Python que permiten capturar imágenes de la webcam sistemáticamente para enriquecer el conjunto de datos.

- El sistema captura ráfagas de imágenes mientras el usuario realiza un gesto específico.
- Las imágenes se guardan automáticamente en carpetas etiquetadas con el nombre del gesto (ej. `dataset/gestos_manos/4_Pulgar_Arriba`).
- Se capturaron múltiples variaciones de cada gesto (diferentes distancias, manos izquierda/derecha, ligeras rotaciones) para mejorar la generalización.

Las clases definidas incluyen:

- **Faciales**: Neutro, Ojos Cerrados, Cabeza Derecha, Cabeza Izquierda.
- **Manuales**: Mano Arriba, Puños Cerrados, Pulgar Arriba, Victoria, Rock, Llamada, OK.

4.2. Fase 2: Entrenamiento (método y cómo)

El entrenamiento se llevó a cabo en los notebooks:

- `EntrenoCara.ipynb`
- `EntrenoManoCuerpo.ipynb`

La metodología empleada se basa en el aprendizaje supervisado sobre características geométricas.

4.2.1. Extracción de características

En lugar de utilizar redes neuronales convolucionales (CNN) que procesan la imagen completa (pixel a pixel), se optó por un enfoque basado en **landmarks** (puntos clave).

1. Se procesa cada imagen del dataset con MediaPipe.
2. Se extraen las coordenadas (x, y, z) de los puntos de interés (468 para cara, 21 para mano).
3. **Normalización**: Crucial para que el modelo funcione independientemente de la posición del usuario en la pantalla. Se centra el gesto restando el centroide y se escala dividiendo por la máxima distancia absoluta desde el centro.

4.2.2. Aumento de datos (data augmentation)

Dado que el dataset propio es limitado en tamaño, se aplicaron técnicas de aumento de datos sintético sobre los vectores de características (no sobre las imágenes), lo cual es muy eficiente:

- **Ruido**: Se añade ruido gaussiano aleatorio a las coordenadas para simular el "jitter"de la cámara.
- **Escalado y rotación**: Se aplican transformaciones matriciales 2D para simular que el usuario está más cerca/lejos o inclina la mano.

4.2.3. Clasificación con MLP

Se entrenaron dos modelos `MLPClassifier` (Multi-Layer Perceptron) independientes:

- **Modelo facial**: Arquitectura (128, 64). Capaz de detectar sutilezas como el parpadeo.
- **Modelo manual**: Arquitectura (64, 32). Suficiente para distinguir posturas de dedos.

Los modelos alcanzaron una precisión superior al 95 % en el conjunto de validación y se exportaron como archivos `.pkl` para su uso en producción.

4.3. Fase 3: Desarrollo del juego (`src/game/codigo.py`)

La lógica principal del juego reside en el archivo `src/game/codigo.py`. El juego está implementado como una clase `SimonGame` que orquesta la captura de vídeo, la inferencia de modelos y la interacción con el usuario.

4.3.1. Máquina de estados

Para gestionar el flujo del juego, se implementó una máquina de estados finitos:

1. **MENU**: Pantalla de inicio esperando que el jugador pulse espacio.

2. **SHOW_NEW_STEP**: El turno de la CPU. El sistema:
   - Añade un nuevo paso a la secuencia.
   - Utiliza **Pyttsx3** para verbalizar el comando (ej. "Simón dice: Victoria").
   - Muestra texto en pantalla usando **Pillow** para renderizar una tipografía personalizada.
3. **WAIT_NEUTRAL**: Estado intermedio que obliga al jugador a volver a una posición neutra. Esto evita que el sistema detecte erróneamente el gesto anterior como el inicio del nuevo.
4. **PLAYER_TURN**: El jugador debe repetir la secuencia. El sistema valida en tiempo real si el gesto detectado coincide con el esperado.
5. **GAMEOVER / SUCCESS**: Estados finales de ronda.

4.3.2. Niveles de dificultad y trampas

Una característica clave implementada en el código es la lógica de "trampas", inspirada en el juego real:

- **Nivel básico**: El juego siempre dice "Simón dice...".
- **Nivel intermedio**: Aparecen comandos como "Modesto dice..." El jugador **no** debe realizar el gesto. Si se mueve, pierde.
- **Nivel avanzado**: Se omiten palabras clave ("Simón...", "Dice...").

Esta lógica se gestiona mediante la función `add_sequence_step`, que genera aleatoriamente instrucciones válidas o trampas según la puntuación actual.

4.3.3. Feedback Multimedia

El juego no solo es visual, sino que integra audio para mejorar la experiencia:

- **Voz**: El motor TTS narra las acciones.
- **Efectos**: Se utiliza `pygame.mixer` para reproducir sonidos de éxito o error inmediatamente tras la acción del jugador.

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. Renderizado de Texto

OpenCV no soporta caracteres UTF-8 nativamente, lo que impedía escribir "Simón" o "Puño" correctamente.
Solución: Se creó la función auxiliar `poner_texto_utf8` que convierte el frame a formato PIL Image, dibuja el texto con una fuente TrueType (.otf) personalizada y lo reconvierte a OpenCV.

5.4. 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 pygame pyttsx3 pillow
```

6.2. Ejecución

1. Abrir la terminal en la carpeta `src/game/`.
2. Ejecutar el archivo principal del juego: `python codigo.py`.
3. Seguir las instrucciones en pantalla y audio.

6.3. Cómo jugar

1. **Atención**: El juego te hablará y mostrará texto.
2. **Ordenes Válidas**: Si dice "Simón dice: [Acción]", debes imitar el gesto.
3. **Trampas**:
   - Si dice "Modesto dice..." o no menciona a "Simón", ¡No te muevas!
   - Mantén la posición neutra hasta que pase la ronda.
4. **Secuencia**: Debes memorizar y repetir todos los pasos acumulados.

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.

La integración de feedback multimedia (voz y sonido) enriquece significativamente la experiencia de usuario, haciendo el juego más dinámico y accesible. El uso de técnicas como la normalización de coordenadas y el aumento de datos ha sido crucial para obtener un sistema robusto.

8. Bibliografía

- Documentación de OpenCV: `https://docs.opencv.org/`
- MediaPipe Solutions: `https://google.github.io/mediapipe/`
- Scikit-Learn Documentation: `https://scikit-learn.org/stable/`
- Pygame Documentation: `https://www.pygame.org/docs/`
- Pyttsx3 Project: `https://pypi.org/project/pyttsx3/`