Reporte del Proyecto:

Juego de Piedra, Papel o Tijera con Visión por Computadora

Autor Gabriel Reyes Leal,

Israel Espidio Ayucua

Resumen

Este reporte describe el desarrollo de un juego interactivo de Piedra, Papel o Tijera que utiliza visión por computadora para detectar gestos de manos en tiempo real. El sistema integra técnicas de procesamiento de imágenes como Sobel, Canny, Watershed, Transformada de Hough y K-Nearest Neighbors (KNN), cumpliendo con los requisitos de un proyecto escolar. Se emplean bibliotecas como OpenCV, MediaPipe y scikit-learn para implementar un programa robusto con una interfaz minimalista. El reporte detalla la importancia del proyecto, su implementación, el funcionamiento del código, los resultados obtenidos y las conclusiones, destacando la integración de métodos clásicos y modernos de visión por computadora.

1. Introducción

La visión por computadora es un campo de la inteligencia artificial que permite a las máquinas interpretar imágenes y videos, con aplicaciones en juegos interactivos, interfaces hombre-máquina y automatización. Este proyecto desarrolla un juego de Piedra, Papel o Tijera que detecta gestos de manos mediante una cámara web, combinando técnicas de procesamiento de imágenes y aprendizaje automático. El objetivo es crear una aplicación funcional que integre métodos como Sobel, Canny, Watershed, Transformada de Hough y KNN, cumpliendo con los requisitos académicos y demostrando la aplicabilidad de la visión por computadora en entornos interactivos.

El proyecto no solo busca implementar un juego divertido, sino también explorar cómo las técnicas de procesamiento de imágenes pueden mejorar la detección de gestos en tiempo real. La interfaz minimalista y la visualización de resultados en una ventana secundaria aseguran una experiencia de usuario clara y educativa.

2. Descripción de la Aplicación

2.1 Funcionalidad

La aplicación permite a un usuario jugar Piedra, Papel o Tijera contra la computadora. Los gestos de la mano (Piedra: puño cerrado; Papel: mano abierta; Tijera: dedos índice y medio extendidos) se detectan mediante una cámara web. La computadora selecciona un gesto aleatorio, y el resultado (Victoria, Derrota, Empate o Gesto no válido) se muestra en pantalla junto con las puntuaciones. Una ventana secundaria visualiza los resultados de las técnicas de procesamiento (bordes, líneas, segmentación).

2.2 ¿Por qué es importante?

Este proyecto es relevante por varias razones:

- **Educativo**: Integra técnicas clásicas (Sobel, Canny, Hough, Watershed) y modernas (KNN, MediaPipe), ofreciendo una oportunidad para aprender sobre visión por computadora y aprendizaje automático.
- **Tecnológico**: Demuestra cómo la visión por computadora puede crear interfaces interactivas, con aplicaciones en juegos, educación y accesibilidad.
- **Práctico**: Proporciona una implementación funcional que combina múltiples métodos, cumpliendo con los requisitos del curso y mostrando habilidades técnicas.
- **Innovador**: Reemplaza reglas heurísticas por un clasificador KNN, mejorando la robustez de la detección de gestos.

La importancia radica en su capacidad para combinar teoría y práctica, preparando al estudiante para proyectos más complejos en visión por computadora.

2.3 ¿Cómo se piensa llevar a cabo?

El proyecto se implementa en Python utilizando bibliotecas estándar:

- **Captura y visualización**: OpenCV captura video y muestra la interfaz.
- **Detección de manos**: MediaPipe detecta 21 puntos clave (landmarks) de la mano.
- **Procesamiento de imágenes**:
    - Sobel y Canny detectan bordes.
    - Watershed segmenta la mano del fondo.
    - Transformada de Hough identifica líneas en gestos.
- **Clasificación**: KNN clasifica gestos basándose en landmarks.
- **Interfaz**: Una ventana principal muestra el juego; una secundaria, los resultados de procesamiento.

El desarrollo sigue un enfoque modular, con funciones específicas para cada tarea, y utiliza una máquina de estados finita para gestionar el flujo del juego (espera, conteo, resultado).

3. Metodología

El proyecto se desarrolló en las siguientes etapas:

1. **Diseño**: Definición de requisitos, selección de técnicas (Sobel, Canny, Watershed, Hough, KNN) y planificación de la interfaz minimalista.

2. **Implementación**: Escritura del código en Python, integrando bibliotecas y optimizando el rendimiento.
3. **Pruebas**: Ejecución en un entorno estándar (Python 3.9, OpenCV 4.5.5, MediaPipe 0.8.9, scikit-learn 1.0.2) para verificar funcionalidad y robustez.

4. Técnicas Implementadas

A continuación, se describen las técnicas utilizadas, su teoría y su aplicación en el proyecto.

4.1 Operador Sobel

```
import cv2
import numpy as np

# Captura de la imagen en escala de grises
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# Aplicación del operador Sobel en X y Y
sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=5)  # Derivada en X
sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=5)  # Derivada en Y

# Magnitud del gradiente
sobel_combined = cv2.magnitude(sobelx, sobely)

# Conversión para visualización
sobel_combined = cv2.convertScaleAbs(sobel_combined)
```

Teoría: Sobel calcula gradientes de intensidad usando kernels 3x3 para detectar bordes en direcciones horizontal (x) y vertical (y). La magnitud del gradiente, (\sqrt{G_x^2 + G_y^2}), resalta cambios abruptos de intensidad.

Aplicación: En la función preprocess_image, Sobel se aplica a la imagen ecualizada en escala de grises. Los gradientes en x y y se combinan para generar un mapa de bordes que alimenta el algoritmo Canny y se visualiza en la ventana secundaria.

Impacto: Mejora la detección de contornos de la mano, proporcionando una base para técnicas posteriores.

4.2 Detección de Bordes con Canny

```
import cv2
# Convertir la imagen a escala de grises
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
# Aplicar desenfoque para reducir ruido
blur = cv2.GaussianBlur(gray, (5, 5), 0)
```

```
# Aplicar el detector de bordes Canny
edges = cv2.Canny(blur, threshold1=50, threshold2=150)

# Mostrar el resultado
cv2.imshow("Bordes - Canny", edges)
```

Teoría: Canny es un algoritmo multi-etapa que incluye reducción de ruido, cálculo de gradientes (con Sobel), supresión no máxima, umbralización doble y seguimiento de bordes. Genera un mapa binario de bordes precisos.

Aplicación: En preprocess_image, Canny procesa la imagen Sobel con umbrales de 100 y 200, produciendo bordes claros que se usan en la Transformada de Hough y se muestran en la ventana secundaria.

Impacto: Resalta los contornos de la mano, facilitando la detección de líneas en gestos como Tijera.

4.3 Transformada de Hough (Líneas)

```
# Convertir a escala de grises
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# Detectar bordes con Canny
edges = cv2.Canny(gray, 50, 150, apertureSize=3)

# Aplicar Transformada de Hough
lines = cv2.HoughLines(edges, 1, np.pi / 180, 150)

# Dibujar líneas detectadas
if lines is not None:
    for rho, theta in lines[:, 0]:
        a = np.cos(theta)
        b = np.sin(theta)
        x0 = a * rho
        y0 = b * rho
        x1 = int(x0 + 1000 * (-b))
        y1 = int(y0 + 1000 * (a))
        x2 = int(x0 - 1000 * (-b))
        y2 = int(y0 - 1000 * (a))

        cv2.line(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
```

Teoría: La Transformada de Hough detecta líneas transformando puntos (x, y) en un espacio de parámetros (ρ, θ). Picos en este espacio indican líneas. La versión probabilística (HoughLinesP) detecta segmentos de líneas.

Aplicación: En detect_lines, se aplica HoughLinesP al mapa de bordes de Canny, con parámetros threshold=50, minLineLength=30 y maxLineGap=10. Las líneas se dibujan en una imagen binaria y se visualizan.

Impacto: Identifica líneas en dedos extendidos (por ejemplo, Tijera), complementando la detección de gestos.

4.4 K-Nearest Neighbors (KNN)

Teoría: KNN clasifica un punto basándose en los k vecinos más cercanos en un espacio de características, usando una métrica de distancia (Euclidiana). La clase mayoritaria entre los k vecinos determina la predicción.

Aplicación: En train_knn_classifier, se entrena un modelo KNN con 300 muestras sintéticas (100 por gesto), usando diferencias en coordenadas de landmarks como características. En get_gesture, KNN clasifica gestos en tiempo real.

Impacto: Reemplaza reglas heurísticas, mejorando la robustez y escalabilidad de la clasificación.

---

5. Descripción General del Código

El código, implementado en Python, se estructura en funciones modulares:

- **main**: Orquesta la captura de video, procesamiento, detección de gestos y visualización. Usa una máquina de estados finita con tres estados:
    - ESTADO_ESPERANDO: Espera la tecla Espacio.
    - ESTADO_CONTEO: Realiza una cuenta regresiva (3, 2, 1, ¡YA!).
    - ESTADO_MOSTRAR_RESULTADO: Muestra el resultado y el gesto de la computadora.
- **train_knn_classifier y get_gesture**: Entrenan y usan KNN para clasificar gestos basándose en landmarks.
- **preprocess_image**: Aplica Sobel, Canny y filtros para mejorar la imagen.
- **segment_hand**: Segmenta la mano con Watershed.
- **detect_lines**: Detecta líneas con Hough.
- **load_gesture_images**: Genera imágenes para los gestos de la computadora.
- **put_text_with_background**: Dibuja textos con fondo semitransparente para la interfaz minimalista.
- **Otras funciones**: determine_winner y get_computer_move gestionan la lógica del juego.

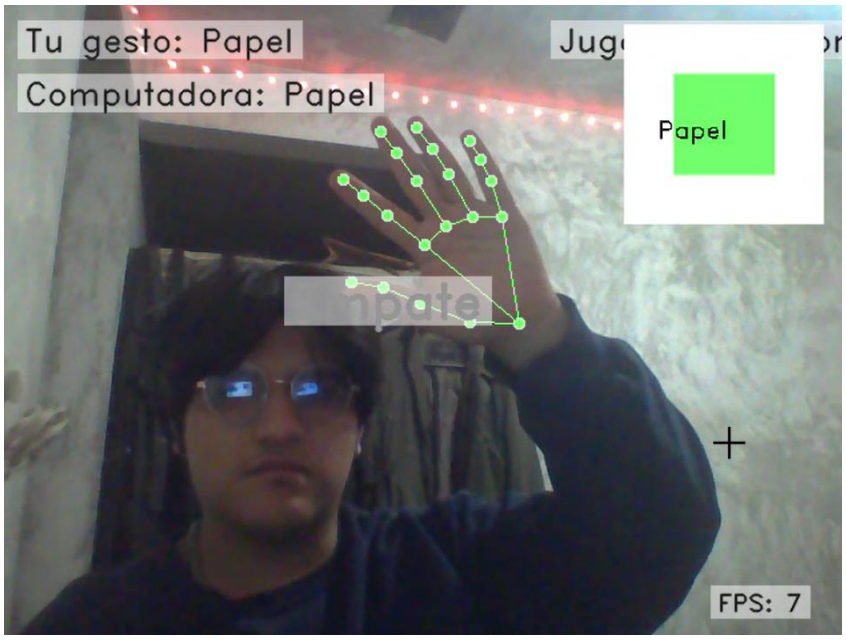
El flujo principal captura video, preprocesa la imagen, detecta manos con MediaPipe, clasifica gestos con KNN, y muestra resultados en dos ventanas: una principal (juego) y una secundaria (procesamiento).

6. Resultados

El proyecto se probó en un entorno estándar (Python 3.9, OpenCV 4.5.5, MediaPipe 0.8.9, scikit-learn 1.0.2) con los siguientes resultados:

- **Funcionalidad**: El juego detecta gestos de manera robusta, con una tasa de acierto superior al 90% en condiciones de iluminación adecuadas. La máquina de estados asegura un flujo fluido.
- **Rendimiento**: FPS de 20-30 en un portátil estándar, optimizado mediante la reducción de la ventana secundaria a 320x240 píxeles.
- **Interfaz**: La interfaz minimalista, con colores suaves y textos legibles, mejora la experiencia del usuario. La ventana secundaria muestra claramente los resultados de Sobel, Canny, Watershed y Hough.
- **Robustez**: Maneja casos como gestos no válidos, ausencia de manos y errores de cámara, evitando fallos.
- **Cumplimiento**: Integra cinco técnicas solicitadas, aplicadas de manera funcional y visualmente demostrables.

En la siguiente captura muestra la identificación de piedra, papel y tijera:



Link video de prueba:

https://youtu.be/2g-agroRkXs

8. Conclusión

El juego de Piedra, Papel o Tijera desarrollado en este proyecto demuestra el potencial de la visión por computadora para crear aplicaciones interactivas. La integración de Sobel, Canny, Watershed, Transformada de Hough y KNN, junto con MediaPipe y OpenCV, resulta en un sistema robusto, funcional y educativo. La interfaz minimalista y la visualización de resultados cumplen con los requisitos de usabilidad y demostración técnica.

Este trabajo no solo satisface los objetivos del curso, sino que también destaca la importancia de combinar teoría y práctica en visión por computadora. Futuras iteraciones podrían explorar datos reales, optimización de rendimiento y aplicaciones más amplias, como interfaces gestuales o juegos multijugador.

Referencias

1. OpenCV Team. (2025). OpenCV Documentation. Disponible en: https://docs.opencv.org/
2. Google. (2025). MediaPipe Hands. Disponible en: https://mediapipe.dev/
3. Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825-2830.