Detección de jugadores de tenis y verificación de zona de servicio

In [ ]:
```
!pip install -r requirements.txt
```

Descripción de las librerías utilizadas

Este bloque de código importa diversas bibliotecas estándar y especializadas para el procesamiento de videos, detección de objetos y análisis de imágenes, utilizando herramientas de aprendizaje profundo.

1. **Librerías estándar de Python**

- **`os` y `sys`**: Estas bibliotecas proporcionan herramientas para interactuar con el sistema operativo y manejar variables del entorno, como rutas de archivos y configuración del sistema.
- **`time`**: Se utiliza para añadir retrasos en la ejecución del código, lo cual puede ser útil para sincronizar procesos o medir tiempos de ejecución.
- **`collections.defaultdict`**: Una estructura de datos que proporciona un valor predeterminado para las claves que no existen en un diccionario.
- **`math`**: Biblioteca estándar para realizar cálculos matemáticos avanzados, como trigonometría y funciones algebraicas.

2. **Librerías de procesamiento de video e imágenes**

- **`cv2 (OpenCV)`**: Biblioteca de procesamiento de imágenes y videos ampliamente utilizada. Se usa para capturar, modificar, y analizar frames de video, como la detección de bordes, conversión a escala de grises, y la visualización de resultados.
- **`numpy (np)`**: Biblioteca para operaciones matemáticas y manipulación de arrays multidimensionales. Es fundamental para manejar datos de imágenes y videos en forma de matrices.
- **`moviepy.editor`**: Herramienta para la edición y manipulación de videos. Facilita la combinación, edición y exportación de videos.

3. **Librerías de PyTorch**

- **`torch`**: PyTorch es una biblioteca de aprendizaje profundo que se usa para construir y ejecutar modelos de redes neuronales. En este caso, se utiliza para la ejecución de modelos preentrenados.
- **`torchvision.transforms`**: Proporciona transformaciones para preprocesar imágenes, como redimensionar, recortar y normalizar las imágenes antes de

que sean ingresadas a un modelo.
- **`torchvision.models`**: Ofrece varios modelos preentrenados para la visión por computadora, como ResNet, que se utiliza en este proyecto para la detección de puntos clave.

4. **Librerías de YOLO y Ultralytics**

- **`ultralytics.YOLO`**: YOLO (You Only Look Once) es un modelo de detección de objetos en tiempo real. La implementación de Ultralytics facilita la carga y uso de modelos YOLO preentrenados para detectar jugadores y objetos en la cancha de tenis.
- **`ultralytics.utils.plotting`**: Incluye herramientas para anotar (dibujar) en las imágenes, como los cuadros delimitadores y la coloración de los objetos detectados.

5. **Librerías de SORT para el seguimiento de objetos**

- **`sort.Sort`**: SORT (Simple Online and Realtime Tracking) es un algoritmo utilizado para hacer seguimiento de objetos en videos. En este proyecto, se utiliza para rastrear a los jugadores a lo largo de los frames del video, manteniendo el mismo ID de jugador incluso si la detección es momentáneamente perdida.

In [1]:
```
# Librerías estándar de Python
import os
import sys
import time  # Para añadir retrasos
from collections import defaultdict
import math

# Librerías de procesamiento de video e imágenes
import cv2
import numpy as np
import moviepy.editor as mpe

# Librerías de PyTorch
import torch
import torchvision.transforms as transforms
from torchvision import models

# Librerías de YOLO y ultralytics
from ultralytics import YOLO
from ultralytics.utils.plotting import Annotator, colors
import ultralytics

# Librerías de SORT para seguimiento de objetos
from sort import Sort

# Librerías de TensorFlow (aunque no está claro si se está utilizando en est
import tensorflow as tf
```

In [ ]:
```
ultralytics.checks()
```

In [ ]:
```
import torch
print(torch.__version__)
```

Descripción paso a paso de la clase `PlayerDetection`

1. **Constructor (`__init__`)**

- **Objetivo**: Inicializa la clase con los parámetros necesarios para procesar un video, detectar jugadores y analizar si están en la zona de saque.
- **Parámetros**:
  - `video_path` : La ruta del archivo de video que se va a procesar.
  - `yolo_model_path` : Ruta del modelo YOLO utilizado para la detección de jugadores.
  - `pose_model_path` : Ruta del modelo YOLO utilizado para la detección de poses.
  - `output_path` : La ruta donde se guardará el video procesado (opcional).
- **Variables**:
  - `self.model` : Carga el modelo YOLO para detectar a los jugadores.
  - `self.pose_model` : Carga un modelo YOLO especializado en detectar poses.
  - `self.tracker` : Inicializa el algoritmo SORT para hacer seguimiento de los jugadores.
  - `self.player_mapping` : Un diccionario para almacenar el ID de cada jugador.
  - `self.threshold_y` : Umbral que se calcula para identificar la zona de saque.

2. **Método `process_video`**

- **Objetivo**: Procesar el video, detectar jugadores y analizar su relación con las zonas de saque.
- **Funcionamiento**:
  - Carga el video y establece sus dimensiones.
  - Actualiza los puntos clave de la cancha con el método `update_court_keypoints` .
  - Utiliza YOLO para detectar jugadores en cada frame.
  - Almacena las detecciones y usa el algoritmo SORT para hacer el seguimiento continuo de los jugadores.
  - Si se detectan dos jugadores, se les asignan IDs y se analiza si están en la zona de saque usando los puntos clave detectados por el modelo de pose.

3. **Método `update_court_keypoints`**

- **Objetivo**: Actualiza los puntos clave de la cancha (keypoints) y calcula el umbral Y (`self.threshold_y`) que se utiliza para verificar si un jugador está en la zona de saque.
- **Funcionamiento**:
  - Se predicen los puntos clave de la cancha usando `line_detection` .
  - Calcula la angulación de la cancha y ajusta el umbral Y dependiendo de la diferencia angular entre las líneas detectadas.

4. **Método `calculate_angle`**

- **Objetivo**: Calcula el ángulo entre dos puntos de la cancha, lo que permite analizar la inclinación de las líneas laterales.
- **Parámetros**:
  - `x1, y1` : Coordenadas del primer punto.
  - `x2, y2` : Coordenadas del segundo punto.
- **Funcionamiento**:
  - Calcula la diferencia en las coordenadas X e Y y utiliza `atan2` para calcular el ángulo entre los dos puntos.

5. **Método `detect_pose`**

- **Objetivo**: Detectar la pose del jugador en el frame actual y devolver las coordenadas de los tobillos izquierdo y derecho.
- **Funcionamiento**:
  - Recorta el área del jugador en el frame.
  - Usa el modelo de poses para predecir las coordenadas de los puntos clave del cuerpo.
  - Ajusta las posiciones de los tobillos con un factor de corrección y devuelve las coordenadas de los tobillos y los puntos clave.

6. **Método `draw_keypoints`**

- **Objetivo**: Dibuja los puntos clave detectados sobre el frame actual.
- **Funcionamiento**:
  - Recorre los puntos clave y los dibuja sobre el frame usando círculos y etiquetas.

7. **Método `detect_serve_zone`**

- **Objetivo**: Verificar si los tobillos del jugador están dentro de las zonas de saque.
- **Funcionamiento**:

- Define los polígonos que representan las zonas de saque para ambos jugadores.
- Verifica si los tobillos del jugador están dentro de los polígonos usando el método `is_inside_polygon`.
- Si un tobillo está dentro de la zona de saque, la zona se resalta en rojo.

8. **Método `highlight_polygon`**

- **Objetivo**: Resalta una zona en la cancha dibujando un polígono sobre el video.
- **Parámetros**:
  - `polygon`: Array de puntos que define el polígono.
  - `color`: El color que se utilizará para resaltar la zona (por defecto verde).

9. **Método `is_inside_polygon`**

- **Objetivo**: Verificar si un punto está dentro de un polígono.
- **Funcionamiento**:
  - Usa `cv2.pointPolygonTest` para determinar si un punto está dentro del área definida por el polígono.

10. **Método `assign_player_ids`**

- **Objetivo**: Asignar IDs a los jugadores en función de su posición en la cancha.
- **Funcionamiento**:
  - Si un jugador está por encima del umbral Y calculado, se le asigna como jugador 1; si está por debajo, como jugador 2.

In [17]:
```
keypoint_names = [
    "Left-shoulder", "Right-shoulder", "Left-elbow", "Right-elbow",
    "Left-hip", "Right-hip", "Left-knee", "Right-knee", "Left-ankle", "Right
]

class PlayerDetection:
    def __init__(self, video_path, yolo_model_path="bestv3.pt", pose_model_p
        self.video_path = video_path
        self.output_path = output_path
        self.model = YOLO(yolo_model_path)  # Modelo para detectar jugadores
        self.pose_model = YOLO(pose_model_path)  # Modelo para detectar pose
        self.tracker = Sort()
        self.player_mapping = {}
        self.threshold_y = None  # Umbral Y calculado
        self.angle_left = None  # Angulación de la cancha lado izquierdo
        self.angle_right = None  # Angulación de la cancha lado derecho

    def process_video(self, line_detection):
```

```
        cap = cv2.VideoCapture(self.video_path)
        if not cap.isOpened():
            print("Error al abrir el video.")
            return

        width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = int(cap.get(cv2.CAP_PROP_FPS))
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out = cv2.VideoWriter(self.output_path, fourcc, fps, (width, height)

        ret, frame = cap.read()

        # Actualiza puntos de la cancha
        line_detection.predict_keypoints(frame)
        self.update_court_keypoints(line_detection, frame)

        while ret:
            self.frame = frame

            # Realiza predicciones con el modelo YOLO
            results = self.model.predict(frame, conf=0.4, iou=0.3)  # Puede

            detections = []
            for result in results:  # Procesamos cada predicción
                boxes = result.boxes  # Extraemos el objeto Boxes
                for box in boxes:
                    # Extraemos las coordenadas xyxy, la clase y la confianz
                    x1, y1, x2, y2 = box.xyxy[0].cpu().numpy().astype(int)
                    class_id = int(box.cls[0].cpu().numpy())  # Clase de la
                    confidence = float(box.conf[0].cpu().numpy())  # Confian

                    if class_id == 3:  # Filtrar la clase 'player' con class
                        print(f"Jugador detectado con confianza {confidence}
                        detections.append([x1, y1, x2, y2, confidence])

            detections = np.array(detections)

            if len(detections) == 0:
                print("No se detectaron jugadores en este frame.")
            else:
                print(f"{len(detections)} jugadores detectados en este frame

            tracked_objects = self.tracker.update(detections)

            player_positions = []
            for track in tracked_objects:
                x1, y1, x2, y2, track_id = map(int, track[:5])
                player_positions.append((x2, y2))
                # Dibuja las cajas delimitadoras
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                cv2.putText(frame, f'ID: {track_id}', (x1, y1 - 10), cv2.FON

            if self.threshold_y is not None:
                cv2.line(frame, (0, int(self.threshold_y)), (width, int(self
```

```
            if len(player_positions) == 2:
                player_1, player_2 = self.assign_player_ids(player_positions

                for i, (x, y) in enumerate(player_positions):
                    player_label = "jugador 1" if (x, y) == player_1 else "j
                    x1, y1, x2, y2, track_id = map(int, tracked_objects[i][:

                    left_ankle, right_ankle, keypoints = self.detect_pose((x

                    if keypoints:
                        self.detect_serve_zone(player_positions, left_ankle,
                        self.draw_keypoints(frame, keypoints, (x1, y1, x2, y
                        print(f"Tobillo izquierdo: {left_ankle}, Tobillo der

                    # Dibuja caja delimitadora y etiqueta del jugador
                    cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 2)
                    cv2.putText(frame, player_label, (x1, y1 - 10), cv2.FONT

            out.write(frame)  # Escribe el frame con las detecciones en el v

            ret, frame = cap.read()

        cap.release()
        out.release()
        print(f"Video con jugadores guardado como {self.output_path}")

    def update_court_keypoints(self, line_detection, frame):
        keypoints = line_detection.predict_keypoints(frame)

        # Coordenadas de los puntos (0, 2) y (1, 3)
        x0, y0 = keypoints[0], keypoints[1]
        x2, y2 = keypoints[4], keypoints[5]
        x1, y1 = keypoints[2], keypoints[3]
        x3, y3 = keypoints[6], keypoints[7]

        self.angle_left = self.calculate_angle(x0, y0, x2, y2)
        self.angle_right = self.calculate_angle(x1, y1, x3, y3)

        angle_difference = abs(self.angle_left - self.angle_right)
        cateto_x_left = abs(x2 - x0)
        cateto_x_right = abs(x3 - x1)
        avg_cateto_x = (cateto_x_left + cateto_x_right) / 2
        self.threshold_y = y0 + (avg_cateto_x / 2)

        # Ajustes adicionales basados en la diferencia angular
        if angle_difference < 10:
            self.threshold_y += 275
        elif 10 <= angle_difference < 20:
            self.threshold_y += 230
        elif 20 <= angle_difference < 30:
            self.threshold_y += 185
        elif 30 <= angle_difference < 40:
            self.threshold_y += 140
        elif 40 <= angle_difference < 50:
            self.threshold_y += 95
        elif 50 <= angle_difference < 60:
```

```
            self.threshold_y += 50
        else:
            self.threshold_y += 45

    def calculate_angle(self, x1, y1, x2, y2):
        delta_y = y2 - y1
        delta_x = x2 - x1
        angle = math.degrees(math.atan2(delta_y, delta_x))
        return angle

    def detect_pose(self, player_box, frame, scale_factor=1.5, ankle_adjustm
        x1, y1, x2, y2 = map(int, player_box)

        width = x2 - x1
        height = y2 - y1
        new_width = int(width * scale_factor)
        new_height = int(height * scale_factor)

        x1 = max(0, x1 - (new_width - width) // 2)
        y1 = max(0, y1 - (new_height - height) // 2)
        x2 = min(frame.shape[1], x1 + new_width)
        y2 = min(frame.shape[0], y1 + new_height)

        cropped_frame = frame[y1:y2, x1:x2]

        results = self.pose_model(cropped_frame)

        if hasattr(results[0], 'keypoints'):
            keypoints = results[0].keypoints.xyn.cpu().numpy()[0]

            keypoints_scaled = []
            for i, keypoint in enumerate(keypoints[5:]):  # Solo desde el cu
                if i not in [4, 5]:  # Ignorar las muñecas
                    x_norm, y_norm = keypoint
                    x_scaled = int(x_norm * cropped_frame.shape[1]) + x1
                    y_scaled = int(y_norm * cropped_frame.shape[0]) + y1

                    if x1 <= x_scaled <= x2 and y1 <= y2:
                        keypoints_scaled.append((x_scaled, y_scaled))

            if len(keypoints_scaled) >= 10:
                left_ankle = keypoints_scaled[8]
                right_ankle = keypoints_scaled[9]
                height_player = y2 - y1

                left_ankle = (left_ankle[0], left_ankle[1] + int(height_play
                right_ankle = (right_ankle[0], right_ankle[1] + int(height_p

                return left_ankle, right_ankle, keypoints_scaled
            else:
                print(f"No se detectaron suficientes keypoints: {len(keypoir
                return None, None, None
        else:
            print("No se encontraron keypoints en los resultados.")
            return None, None, None
```

```
    def draw_keypoints(self, frame, keypoints, player_box):
        """
        Dibuja los keypoints en el frame solo si están dentro de la caja del
        """
        x1, y1, x2, y2 = map(int, player_box)

        for i, (x, y) in enumerate(keypoints):
            if x1 <= x <= x2 and y1 <= y2:
                if i < len(keypoint_names):
                    cv2.circle(frame, (x, y), 5, (0, 0, 255), -1)
                    cv2.putText(frame, keypoint_names[i], (x, y), cv2.FONT_H
            else:
                print(f"El keypoint '{keypoint_names[i]}' está fuera del áre

    def detect_serve_zone(self, player_positions, left_ankle, right_ankle, k
        keypoints_zona_jugador_1 = [
            (keypoints[8], keypoints[9]), (keypoints[12], keypoints[13]),
            (keypoints[18], keypoints[19]), (keypoints[16], keypoints[17])
        ]
        keypoints_zona_jugador_2 = [
            (keypoints[20], keypoints[21]), (keypoints[22], keypoints[23]),
            (keypoints[14], keypoints[15]), (keypoints[10], keypoints[11])
        ]

        zona_jugador_1 = np.array(keypoints_zona_jugador_1, np.int32).reshap
        zona_jugador_2 = np.array(keypoints_zona_jugador_2, np.int32).reshap

        if left_ankle or right_ankle:
            if self.is_inside_polygon(left_ankle, zona_jugador_1) or self.is
                self.highlight_polygon(zona_jugador_1, color=(0, 0, 255))
                print(f"Tobillo izquierdo: {left_ankle}, Tobillo derecho: {r

            if self.is_inside_polygon(left_ankle, zona_jugador_2) or self.is
                self.highlight_polygon(zona_jugador_2, color=(0, 0, 255))
                print(f"Tobillo izquierdo: {left_ankle}, Tobillo derecho: {r

    def highlight_polygon(self, polygon, color=(0, 255, 0)):
        cv2.fillPoly(self.frame, [polygon], color)

    def is_inside_polygon(self, point, polygon):
        return cv2.pointPolygonTest(polygon, point, False) >= 0

    def assign_player_ids(self, player_positions):
        player_1 = None
        player_2 = None

        for i, (x, y) in enumerate(player_positions):
            if y < self.threshold_y:
                player_1 = (x, y)
            else:
                player_2 = (x, y)

        return player_1, player_2
```

Descripción paso a paso de la clase `LineDetection`

1. **Constructor (`__init__`)**

- **Objetivo**: Inicializar la clase para procesar un video y detectar las líneas de la cancha utilizando puntos clave (keypoints).
- **Parámetros**:
  - `video_path` : La ruta del video a procesar.
  - `model_path` : La ruta del modelo preentrenado que se usará para detectar los puntos clave.
  - `output_path` : La ruta donde se guardará el video procesado (opcional, por defecto "resultado_con_lineas.mp4").
- **Variables**:
  - `self.device` : Detecta si hay disponible una GPU (CUDA) y, en caso contrario, usa la CPU.
  - `self.model` : Carga un modelo ResNet101 preentrenado y lo ajusta para predecir 14 puntos clave (28 coordenadas).
  - `self.transform` : Define las transformaciones necesarias para redimensionar y normalizar las imágenes antes de pasarlas al modelo.
  - `self.keypoints` : Almacena los puntos clave predichos para su posterior uso.

2. **Método `predict_keypoints`**

- **Objetivo**: Predecir los puntos clave (keypoints) en un frame del video utilizando el modelo entrenado.
- **Funcionamiento**:
  - Convierte la imagen de BGR (formato de OpenCV) a RGB.
  - Aplica las transformaciones necesarias (redimensionar y normalizar).
  - Utiliza el modelo para predecir los puntos clave en la imagen y los escala a la resolución original del video.

3. **Método `get_central_keypoints`**

- **Objetivo**: Devuelve los puntos centrales superior e inferior de la cancha.
- **Funcionamiento**:
  - Extrae el punto superior central (índice 8) y el punto inferior central (índice 12) de los puntos clave predichos.

4. **Método `draw_keypoint_lines`**

- **Objetivo**: Dibujar líneas de conexión entre los puntos clave sobre el frame del video.
- **Funcionamiento**:
  - Se define una lista de pares de puntos que deben conectarse entre sí mediante líneas.

- Para cada par de puntos, se dibuja una línea azul entre ellos si están dentro de los límites de la imagen.

5. **Método `draw_keypoints`**

- **Objetivo**: Dibujar los puntos clave detectados sobre el frame del video.
- **Funcionamiento**:
  - Por cada par de coordenadas `x, y`, se dibuja un círculo verde en la imagen para marcar el punto clave.
  - Cada punto también se etiqueta con su número de identificación (ID).

6. **Método `process_video`**

- **Objetivo**: Procesar el video cuadro por cuadro, detectar los puntos clave y dibujar las líneas y puntos detectados.
- **Funcionamiento**:
  - Carga el video y configura las propiedades como la resolución y el número de frames por segundo.
  - Para cada frame:
    1. Convierte el frame a escala de grises.
    2. Aplica un suavizado y binarización.
    3. Realiza la detección de bordes.
    4. Utiliza el modelo para predecir los puntos clave y los dibuja sobre el frame.
  - Guarda el video procesado con las líneas y puntos detectados resaltados.

In [5]:
```
class LineDetection:
    def __init__(self, video_path, model_path, output_path="resultado_con_li
        self.video_path = video_path
        self.output_path = output_path
        self.model_path = model_path

        # Definir el dispositivo (GPU si está disponible, de lo contrario CP
        self.device = torch.device("cuda" if torch.cuda.is_available() else
        if torch.cuda.is_available():
            print("GPU detectada. Se utilizará CUDA para la aceleración.")
        else:
            print("No se detectó GPU. Se utilizará CPU.")

        # Cargar modelo preentrenado y mover al dispositivo
        self.model = models.resnet101(pretrained=True)
        self.model.fc = torch.nn.Linear(self.model.fc.in_features, 14 * 2)
        self.model.load_state_dict(torch.load(self.model_path, map_location=
        self.model.to(self.device)  # Mover el modelo al dispositivo
        self.model.eval()  # Coloca el modelo en modo evaluación (no entrena

        # Transformaciones necesarias para el modelo
        self.transform = transforms.Compose([
            transforms.ToPILImage(),
```

```
            transforms.Resize((224, 224)),  # El modelo espera una imagen de
            transforms.ToTensor(),
            transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.2
        ])

        # Atributo para almacenar los keypoints
        self.keypoints = None

    def predict_keypoints(self, image):
        # Convertir la imagen a RGB (cv2 usa BGR)
        image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # Aplicar las transformaciones y agregar una dimensión (batch size)
        image_tensor = self.transform(image_rgb).unsqueeze(0).to(self.device

        # Desactivar la retropropagación y obtener predicciones
        with torch.no_grad():
            outputs = self.model(image_tensor)

        # Convertir las predicciones a un formato usable (numpy array)
        keypoints = outputs.squeeze().cpu().numpy()  # Mover a CPU antes de

        # Obtener el tamaño original del video
        original_h, original_w = image.shape[:2]
        # Escalar las coordenadas predichas desde 224x224 (modelo) a la resc
        keypoints[0::2] = keypoints[0::2] * original_w / 224  # Escalar las
        keypoints[1::2] = keypoints[1::2] * original_h / 224  # Escalar las
        self.keypoints = keypoints
        return keypoints

    def get_central_keypoints(self, keypoints):
        """
        Devuelve los puntos centrales superiores e inferiores.
        Supongamos que los puntos clave están en el orden [x1, y1, x2, y2, .
        """
        central_top = (keypoints[8], keypoints[9])     # Ejemplo de punto ce
        central_bottom = (keypoints[12], keypoints[13])  # Ejemplo de punto
        return central_top, central_bottom

    def draw_keypoint_lines(self, image, keypoints):
        # Dibujar líneas de conexión entre los puntos clave
        connections = [
            (0, 1), (2, 3), (4, 6), (5, 7), (8, 9), (10, 11), (12, 13),
            (0, 2), (1, 3), (4, 5), (6, 7), (8, 10), (9, 11)
        ]

        for (i, j) in connections:
            x1, y1 = int(keypoints[i * 2]), int(keypoints[i * 2 + 1])
            x2, y2 = int(keypoints[j * 2]), int(keypoints[j * 2 + 1])

            if (0 <= x1 < image.shape[1] and 0 <= y1 < image.shape[0] and
                0 <= x2 < image.shape[1] and 0 <= y2 < image.shape[0]):
                cv2.line(image, (x1, y1), (x2, y2), (255, 0, 0), 2)  # Línea
        return image

    def draw_keypoints(self, image, keypoints):
```

```
        # Dibujar puntos clave y su ID
        for i in range(0, len(keypoints), 2):
            x = int(keypoints[i])
            y = int(keypoints[i + 1])

            if 0 <= x < image.shape[1] and 0 <= y < image.shape[0]:
                cv2.circle(image, (x, y), 10, (0, 255, 0), -1)  # Dibujar pu
                cv2.putText(image, str(i // 2), (x, y - 10), cv2.FONT_HERSHE
        return image

    def process_video(self, kernel_size_dilate=7, dilate_iterations=1, kerne
        cap = cv2.VideoCapture(self.video_path)
        if not cap.isOpened():
            print(f"No se pudo abrir el video: {self.video_path}")
            return

        width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = int(cap.get(cv2.CAP_PROP_FPS))
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out = cv2.VideoWriter(self.output_path, fourcc, fps, (width, height)

        ret, frame = cap.read()
        while ret:
            gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            # Suavizado y binarización
            smoothed = cv2.GaussianBlur(gray, (kernel_size_blur, kernel_size
            _, binary = cv2.threshold(smoothed, 200, 255, cv2.THRESH_BINARY)

            # Dilatación y detección de bordes
            kernel = np.ones((kernel_size_dilate, kernel_size_dilate), np.ui
            dilated = cv2.dilate(binary, kernel, iterations=dilate_iteratior
            edges = cv2.Canny(dilated, 50, 150)

            # Detección de puntos clave y dibujo de líneas y puntos
            keypoints = self.predict_keypoints(frame)
            frame = self.draw_keypoints(frame, keypoints)
            frame = self.draw_keypoint_lines(frame, keypoints)

            out.write(frame)
            ret, frame = cap.read()

        cap.release()
        out.release()
        print(f"Video procesado y guardado en {self.output_path}")
```

Descripción paso a paso de la clase `VideoMerger`

1. **Constructor (`__init__`)**

- **Objetivo**: Inicializar la clase con las rutas de los videos y definir las rutas de salida.
- **Parámetros**:

- `player_video_path`: Ruta del video que contiene la detección de jugadores.
- `line_video_path`: Ruta del video que contiene la detección de las líneas.
- `output_path`: Ruta donde se guardará el video final (opcional, por defecto "video_final.mp4").
- `original_video_path`: Ruta del video original con audio que se utilizará para añadir el audio al video final.

2. **Método `merge_videos`**

- **Objetivo**: Combinar los videos de la detección de jugadores y líneas, y luego añadir el audio del video original.
- **Funcionamiento**:
  - Abre los videos de detección de jugadores y líneas con OpenCV.
  - Verifica que ambos videos se hayan abierto correctamente.
  - Crea un archivo de video temporal donde se combinarán ambos videos.
  - Superpone los frames de los dos videos utilizando la función `cv2.addWeighted` con un peso de 0.5 para cada video.
  - Guarda el video combinado en un archivo temporal.
  - Llama al método `add_audio_to_video` para añadir el audio del video original al video combinado.

3. **Método `add_audio_to_video`**

- **Objetivo**: Añadir el audio del video original al video final que se generó al combinar los videos.
- **Funcionamiento**:
  - Utiliza la biblioteca `moviepy` para cargar el video combinado y el video original.
  - Extrae el audio del video original y lo añade al video combinado.
  - Guarda el video final con el audio en el archivo de salida definido.

Descripción de la función `process_videos`

Objetivo: Procesar una lista de videos, realizar detección de líneas y jugadores, combinarlos y luego añadir el audio del video original.

Funcionamiento:

1. Para cada video en la lista `video_names`, se imprime un mensaje indicando que se está procesando.
2. **Generación de nombres de salida**:

- Se crea un nombre único para cada archivo de salida basado en el nombre del video de entrada:
  - `player_output`: Video con la detección de jugadores.
  - `line_output`: Video con la detección de líneas.
  - `merged_output`: Video final que combina ambos resultados.

3. **Detección de líneas**:
   - Se inicializa un objeto `LineDetection` y se llama a su método `process_video()` para procesar la detección de líneas en la cancha de tenis.
4. **Detección de jugadores**:
   - Se inicializa un objeto `PlayerDetection` y se pasa el detector de líneas para procesar la detección de jugadores y su relación con las líneas de la cancha.
5. **Combinación de videos**:
   - Se utiliza la clase `VideoMerger` para combinar el video de la detección de jugadores con el de las líneas y generar un video final.
6. **Añadir audio**:
   - El método `add_audio_to_video` añade el audio del video original al video final combinado.

Lista de videos a procesar

- La lista `video_names` contiene los nombres de los videos a procesar. El código procesará cada uno de los videos, generando un video final con audio para cada entrada.

Ejecución del código

- El último bloque del código llama a la función `process_videos()` con la lista `video_names` especificada, procesando todos los videos indicados en esa lista.

In [ ]:
```
class VideoMerger:
    def __init__(self, player_video_path, line_video_path, output_path="vide
        self.player_video_path = player_video_path
        self.line_video_path = line_video_path
        self.output_path = output_path
        self.original_video_path = original_video_path  # El video original

    def merge_videos(self):
        cap1 = cv2.VideoCapture(self.player_video_path)
        cap2 = cv2.VideoCapture(self.line_video_path)

        if not cap1.isOpened() or not cap2.isOpened():
            print(f"Error al abrir los videos: {self.player_video_path} o {s
            return

        width = int(cap1.get(cv2.CAP_PROP_FRAME_WIDTH))
```

```
        height = int(cap1.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = int(cap1.get(cv2.CAP_PROP_FPS))

        # Utilizamos 'XVID' para mayor compatibilidad
        fourcc = cv2.VideoWriter_fourcc(*'XVID')
        temp_video_path = "temp_video_without_audio.mp4"
        out = cv2.VideoWriter(temp_video_path, fourcc, fps, (width, height))

        ret1, frame1 = cap1.read()
        ret2, frame2 = cap2.read()

        while ret1 and ret2:
            combined_frame = cv2.addWeighted(frame1, 0.5, frame2, 0.5, 0)
            out.write(combined_frame)

            ret1, frame1 = cap1.read()
            ret2, frame2 = cap2.read()

        cap1.release()
        cap2.release()
        out.release()

        print(f"Video combinado guardado como {temp_video_path}, ahora agreg

        # Usar moviepy para agregar el audio del video original
        self.add_audio_to_video(temp_video_path, self.original_video_path, s

    def add_audio_to_video(self, video_path, audio_video_path, output_path):
        """
        Agrega el audio del video original al video final renderizado.
        """
        video_clip = mpe.VideoFileClip(video_path)
        original_video = mpe.VideoFileClip(audio_video_path)

        # Combinar el video renderizado con el audio del video original
        final_video = video_clip.set_audio(original_video.audio)

        # Guardar el video final con audio
        final_video.write_videofile(output_path, codec="libx264", audio_code

        print(f"Video final con audio guardado como {output_path}")

def process_videos(video_names):
    for video_name in video_names:
        print(f"Procesando el video: {video_name}")

        # Remover la extensión del archivo para crear nombres únicos
        base_name = os.path.splitext(video_name)[0]

        # Crear nombres únicos para los archivos de salida para cada video
        player_output = f"resultado_con_jugadores_{base_name}.mp4"
        line_output = f"resultado_con_lineas_{base_name}.mp4"
        merged_output = f"final_{base_name}.mp4"

        # Procesar detección de líneas blancas
        detector = LineDetection(video_name, 'keypoints_model_v2.pth', outpu
```

```
        detector.process_video()

        # Procesar detección de jugadores usando el detector de líneas
        player_detection = PlayerDetection(video_name, output_path=player_ou
        player_detection.process_video(detector)

        # Combinar ambos resultados y añadir el audio del video original
        video_merger = VideoMerger(player_output, line_output, merged_output
        video_merger.merge_videos()


# Lista de los videos a procesar
video_names = ["input_video.mp4","Untitled.mp4","Untitled2.mp4","Untitled32.
#video_names = ["Untitled32.mp4"]
# Procesar todos los videos
process_videos(video_names)
```