Universidad Nacional Autónoma de México
Facultad de Ingeniería


# Proyecto Final

## Procesamiento Digital de Imagenes

## Imágenes de energía de movimiento MEI

Grupo

Profesor:

Dr. Boris Escalante Ramírez

Diaz Guerrero Alan Mauricio 316166496

García V. Karla Paulina 316173670

Zamora Morales Ángel Adrián 316252272

Manuscrito recibido el 04 de diciembre de 2023; revisado el 04 de diciembre de 2023. Primera publicación el 04 de diciembre de 2023. Versión actual publicada el 04 de diciembre de 2023. Esta investigación se presentó parcialmente en la Facultad de Ingeniería, UNAM, en la Clase de Análisis Inteligente de Textos.

Ciudad de México 05 de noviembre, 2023

# Índice

# 1. Planteamiento del problema

## 1.1. Abstract

Las Imágenes de Energía de Movimiento (MEI) son elementos cruciales para comprender la dinámica del movimiento en secuencias de vídeo, ofreciendo una representación condensada mediante plantillas binarias que resaltan la presencia o ausencia de movimiento en cada cuadro. Su eficacia se manifiesta especialmente en escenarios con baja resolución, convirtiéndo se en herramientas valiosas para aplicaciones como el reconocimiento de patrones y el análisis de video.

En el contexto de un proyecto específico, se tiene la intención de desarrollar un código especializado para la identificación de MEI, aprovechando sus ventajas en diversas aplicaciones prácticas. Este enfoque implica simplificar el proceso de reconocimiento mediante un algoritmo eficiente y adaptable, centrándose en la esencia visual del movimiento capturada por las MEI.

La implementación de esta solución contribuirá significativamente a la capacidad de analizar y comprender el movimiento en entornos con información limitada o condiciones adversas de calidad de imagen. El proyecto busca aprovechar la versatilidad de las MEI como herramienta de procesamiento de video, con el objetivo de facilitar la identificación y análisis eficientes de acciones humanas y patrones de movimiento en diversas aplicaciones prácticas.

## 1.2. Objetivos

- El objetivo principal de este proyecto es diseñar y desarrollar un código especializado capaz de identificar Imágenes de Energía de Movimiento (MEI) de manera eficiente en secuencias de vídeo.

- Este sistema se enfocará en aprovechar las ventajas de las MEI, que ofrecen representaciones condensadas del movimiento en entornos con baja resolución, y aplicarlas en diversas situaciones prácticas.

# 2. Análisis del problema

## 2.1. Introducción

Las Imágenes de Energía de Movimiento (MEI) se han consolidado como herramientas visuales esenciales que capturan de manera intrínseca y dinámica el movimiento presente en secuencias de video. Estas imágenes se distinguen por su enfoque en plantillas binarias, asignando a cada píxel valores discretos de 0 o 1 para resaltar la existencia o ausencia de movimiento en cada posición del cuadro. Su diseño está orientado a la aprehensión de la "forma"de actividades específicas a lo largo de un intervalo definido por el número de cuadros de video considerados, proporcionando así una representación visual condensada y expresiva.

Particularmente eficaces en escenarios con información de baja resolución de movimiento, las MEI ofrecen una representación eficiente de acciones humanas, destacándose en aplicaciones como el reconocimiento de patrones, visión por computadora y análisis de video. Su utilidad se manifiesta especialmente en entornos donde la calidad de la imagen es limitada, como en sistemas de vigilancia por cámaras.

La obtención de información tridimensional del movimiento sigue una técnica común que implica la determinación de la posición tridimensional de un objeto o persona en cada instante temporal mediante un modelo tridimensional. Este ajuste se realiza al minimizar una medida residual entre el modelo proyectado y los contornos del objeto en la imagen, enfocándose especialmente en los bordes del cuerpo. La efectividad de este proceso se maximiza a través de una segmentación sólida entre el primer plano y el fondo, así como entre las distintas partes del cuerpo, facilitando la alineación precisa del modelo tridimensional con la información visual del objeto en movimiento.

En este marco, las Imágenes de Energía de Movimiento (MEI) y las Imágenes de Historia de Movimiento (MHI) desempeñan roles clave como componentes de una imagen vectorial. Esta representación visual busca codificar diversas propiedades de movimiento de manera espacialmente indexada. Aunque existen otros posibles componentes, como la potencia en el movimiento direccional integrado a lo largo del tiempo o la periodicidad espacialmente localizada del movimiento, la simplicidad y eficacia destacadas en los resultados presentados en el artículo se logran utilizando únicamente los componentes MEI y MHI para la representación y el reconocimiento.

$$H_\tau(x,y,t) = \begin{cases} \tau & \text{if } D(x,y,t) = 1 \\ \max(0, H_\tau(x,y,t-1) - 1) & \text{otherwise.} \end{cases}$$

Figura 1: Formula

La fase final del proceso de reconocimiento se centra en la segmentación y el emparejamiento temporal. Durante la etapa de entrenamiento, se establecen parámetros cruciales, como la duración mínima y máxima que puede tener un movimiento. Para abordar variaciones en la velocidad de movimientos de prueba, se emplea una ventana de tiempo variable que retrocede en el tiempo. La implementación de este enfoque se traduce en un algoritmo altamente eficiente que aproxima la búsqueda en un amplio rango temporal, calculando configuraciones de Imágenes de Historia de Movimiento (MHI) en cada paso temporal. La elección de la ventana de tiempo máxima (máx) y la cantidad de ventanas de integración temporal (norte) proporciona flexibilidad en la adaptación del sistema a diferentes velocidades de movimiento. La eficiencia computacional de este operador de reemplazo permite una implementación factible en tiempo real.

En términos de la metodología de coincidencia para las plantillas temporales, se opta por un enfoque basado en la apariencia, requiriendo la definición de invariantes deseadas para la técnica de comparación. Dada la naturaleza sensible a la vista de este enfoque, se busca una técnica de coincidencia que sea invariable de escala y traducción. Durante la fase de entrenamiento, se recopilan ejemplos de cada movimiento desde diversos ángulos de visión, generando conjuntos de MEI y MHI para cada combinación de vista y movimiento. A través del cálculo de descripciones estadísticas utilizando características basadas en momentos, se seleccionan los 7 momentos Hu, conocidos por proporcionar discriminación de forma razonable e invarianza en términos de traducción y escala.

La creación de modelos estadísticos de los momentos (matrices de media y covarianza) tanto para MEI como para MHI facilita el reconocimiento de movimientos de entrada. La distancia de Mahalanobis entre la descripción del momento de la entrada y cada uno de los movimientos conocidos se utiliza como métrica de coincidencia. Aunque los momentos Hu pueden carecer de interpretación intuitiva, su ventaja computacional al ser poco exigentes permite la implementación en tiempo real. Sin embargo, se destaca que los métodos de comparación para MEI y MHI no necesariamente tienen que ser los mismos, ya que capturan aspectos diferentes del movimiento.

# 3. Desarrollo

## 3.1. Estimación del Flujo Óptico

- La estimación del flujo óptico es esencial para comprender el movimiento en secuencias de vídeo. Se propone calcular el flujo óptico durante dos subsecuencias de video ïnteresantesütilizando tanto un método implementado (a elección) como funciones incorporadas de bibliotecas como OpenCV o MATLAB.

- Método Implementado: [Especificar el método implementado y su breve descripción]. Se seleccionará un valor apropiado para el parámetro *w* (por ejemplo, 10 o 20 frames) para capturar una parte característica de la actividad. Este método se explicará detalladamente en términos de su funcionamiento.

- OpenCV/MATLAB: Se emplearán las funciones de flujo óptico proporcionadas por OpenCV o MATLAB, seleccionando también un valor apropiado para el parámetro w. Estas bibliotecas implementan métodos establecidos para la estimación del flujo óptico, como el método de Lucas-Kanade o el Farnebäck. La comparación de resultados entre el método implementado y las funciones incorporadas ofrecerá una evaluación completa

## 3.2. Generación de MEI

- Después de calcular el flujo óptico para las subsecuencias seleccionadas, se procederá a generar las Imágenes de Energía de Movimiento (MEI). Se elegirá un valor adecuado para *w* que capture de manera efectiva la dinámica de la actividad.

## 3.3. Limpieza de MEI

- Las MEI resultantes, siendo imágenes binarias, se someterán a operaciones morfológicas para eliminar el ruido. Las operaciones morfológicas, como la apertura o el cierre, se seleccionarán y justificarán según sus capacidades para preservar las características esenciales mientras eliminan artefactos no deseados.

## 3.4. Contorno del MEI

- Se identificarán los contornos en las MEI utilizando un método adecuado, como el algoritmo de detección de contornos de Canny o algoritmos de contornos de

OpenCV o MATLAB. La visualización de los contornos destacarán las regiones de interés y permitirá la comparación entre las dos subsecuencias.

## 3.5. Opcional (Puntos Extra)

- Para obtener puntos extra, se puede extraer el descriptor de forma de los contornos MEI utilizando descriptores de Fourier, Hu o momentos de Zernike. Este paso adicional permitirá una comparación cuantitativa de las características de forma, proporcionando información adicional sobre la variabilidad en las acciones capturadas.

## 3.6. Comparación de Descriptores de Forma

Como paso adicional en la investigación, se llevará a cabo una comparación simple de los descriptores de forma extraídos de los contornos MEI correspondientes a las tres acciones seleccionadas. Se utilizarán descriptores de forma específicos, como los momentos Hu, para cuantificar las diferencias entre las acciones.

- Extracción de Descriptores de Forma

  1. Se utilizarán descriptores de forma como los momentos Hu para cada uno de los contornos MEI generados.

  2. La extracción de estos descriptores se realizará mediante funciones especializadas de bibliotecas como OpenCV o MATLAB.

- Comparación de Descriptores

  1. Se realizará una comparación simple entre los descriptores de forma de las tres acciones.

  2. Una medida común podría ser el cálculo del error cuadrático medio (ECM) entre los descriptores Hu de los tres MEI.

  3. Otras medidas de diferencia también pueden ser consideradas según la naturaleza específica de los descriptores utilizados.

- Resultados y Discusión

  1. Los resultados se presentarán visualmente, mostrando las diferencias en los descriptores de forma entre las acciones.

  2. Se discutirá la interpretación de estas diferencias y su relevancia para la discriminación entre las acciones observadas.

  3. Se analizarán posibles patrones o tendencias que puedan surgir de la comparación de descriptores.

# 4. Solución

## 4.1. Requisitos previos

El código presentado tiene como objetivo principal la estimación del flujo óptico en secuencias de video y el procesamiento de las imágenes resultantes para la generación de Imágenes de Energía de Movimiento (MEI) y el cómputo del Mapa de Historia de Movimiento (MHI).

Figura 2: Video original con el que se trabajó

Se importarán las librerías correspondientes para trabajar, de igual manera el video se importa en un formato .mp4:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Importando el video
vc = cv2.VideoCapture("dance2.mp4")
```

## 4.2. 1. Código Generado

```
mport cv2
import numpy as np
import matplotlib.pyplot as plt
```

```
        # Importando el video
        vc = cv2.VideoCapture("dance2.mp4")

        #Crear medios para guardar el video del Optic Flow RGB
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        # Generando el archivo donde se guardara el video
        out = cv2.VideoWriter('NuestroOFResultados/NuestroOpticalFlow
.mp4', fourcc, 20.0, (1000, 562))

        """Leyendo el primer frame y sacando medidas"""
        _, first_frame = vc.read() # Leer el primer frame

        # Reescalando imagen y obteniendo el primer frame
        resize_dim = 1000 #Ancho total
        max_dim = max(first_frame.shape)
        scale = resize_dim/max_dim
        first_frame = cv2.resize(first_frame, None, fx=scale, fy=
scale)
        # Convierte el primer cuadro a escala de grises
        prev_gray = cv2.cvtColor(first_frame, cv2.COLOR_BGR2GRAY)

        """Declaracion de variables varias"""
        umbral=50 #Umbral de deteccion de movimiento
        mask = np.zeros_like(first_frame) #Matriz de zeros para
alojar O.F.
        mask[..., 1] = 255 # Nos deshacemos de la saturacion de la
imagen poniendo al maximo
        output_folder2="NuestroOFResultados/"
        i=0
        flow2 = np.zeros((562, 1000))
        flow3 = np.zeros((562, 1000))
        kernel=np.ones((11,11),np.uint8)

        while(vc.isOpened() and _==True):
            # Read a frame from video
            _, frame = vc.read()
            if frame is None:
                print("Fin del video.")
                break

            # Convert new frame format`s to gray scale and resize
```

```
gray frame obtained
            gray = cv2.resize(frame, None, fx=scale, fy=scale)
            gray = cv2.cvtColor(gray, cv2.COLOR_BGR2GRAY)
            gray = cv2.normalize(gray,None, 0, 1.0, cv2.NORM_MINMAX,
dtype=cv2.CV_32F)
            #gray = cv2.bilateralFilter(gray,9,75,75) #Filtrado de
fotograma para eliminar ruido
            gray = cv2.GaussianBlur(gray,(3,3),0)

            """Calculo del Optic Flow"""
            diff = abs(prev_gray-gray)
            #np.savetxt('mi_arreglo.csv', diff, delimiter=',')
            cv2.imshow('GRIS-ORIGINAL', gray)
            condicion= diff> umbral/255
            condicion2= diff<= umbral/255
            diff[condicion] = 255
            diff[condicion2] = 0
            diff = diff.astype(np.uint8)
            cv2.imshow('OF+Original', diff)
            #np.savetxt('mi_arreglo.csv', diff, delimiter=',')
            prev_gray=gray

            """Despliegue Optic Flow"""
            factor_atenuacion = 0.1  # Ajusta este valor segun tus
preferencias
            frame_oscuro = frame.astype(float) * factor_atenuacion
            frame = cv2.resize(frame_oscuro.astype(np.uint8), None,
fx=scale, fy=scale) # Redimension
            diff2=diff
            diff = cv2.cvtColor(diff, cv2.COLOR_GRAY2BGR)
            # Anadido y despliegue
            dense_flow = cv2.addWeighted(frame, 1,diff, 2, 0)
            cv2.imshow("Dense optical flow", dense_flow)

            """Generacion MEI imagen"""
            if 50 <= i <= 80:
                flow2=diff2+flow2
                #imagen = flow2[:, :, 1]
                imagen_escalada = cv2.normalize(flow2, None, 0, 255,
cv2.NORM_MINMAX)
                # Convertir la imagen a tipo uint8
                imagen_uint8 = imagen_escalada.astype(np.uint8)
                # Mostrar la imagen con cv2
```

```
                cv2.imshow('MEI', imagen_uint8)
                cv2.imwrite('NuestroOFResultados/MEI.png',
imagen_uint8)

            """Generacion MHI"""
            if 50 <= i <= 80:
                magnitude2=cv2.normalize(diff2, None, 0, 255, cv2.
NORM_MINMAX)

                # Aplica la condicion a cada elemento
                condicion = magnitude2 >= 18
                condicion2 = magnitude2 < 18
                # Establece a 255 solo los elementos que cumplen la
condicion
                magnitude2[condicion] = 255
                magnitude2[condicion2] = 0

                flow3=magnitude2+flow3*.8
                #imagen = flow2[:, :, 1]
                imagen_escalada2 = cv2.normalize(flow3, None, 0, 255,
 cv2.NORM_MINMAX)
                # Convertir la imagen a tipo uint8
                imagen_uint82 = imagen_escalada2.astype(np.uint8)
                # Mostrar la imagen con cv2
                cv2.imshow('MHI', imagen_uint82)
                cv2.imwrite('NuestroOFResultados/MHI.png',
imagen_uint82)

            out.write(dense_flow) #Creacion de video de imagen
original + optic flow rgb
            i=i+1
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        for x in range(562):
            for y in range(1000):
                if imagen_escalada[x,y] > 0:
                    imagen_escalada[x,y] = 255

                else:
                    imagen_escalada[x,y] = 0

        cv2.imwrite('NuestroOFResultados/MEI binaria2.png',
```

```
imagen_escalada)

        """Contorno del MEI/SOBEL"""
        sobel=cv2.Sobel(imagen_escalada, ddepth=cv2.CV_64F,dx=1,dy=1,
ksize=5)
        cv2.imwrite('NuestroOFResultados/ContornoDelMEI.png',sobel)

        vc.release()
        cv2.destroyAllWindows()
```

## 4.3. 2. Código Generado

```
        import cv2
        import numpy as np
        import matplotlib.pyplot as plt

        # Importando el video
        vc = cv2.VideoCapture("dance2.mp4")

        #Crear medios para guardar el video del Optic Flow RGB
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Generando el
archivo donde se guardara el video
        out = cv2.VideoWriter('NuestroOFResultados/NuestroOpticalFlow
.mp4', fourcc, 20.0, (1000, 562))

        """Leyendo el primer frame y sacando medidas"""
        _, first_frame = vc.read() # Leer el primer frame

        # Reescalando imagen y obteniendo el primer frame
        resize_dim = 1000 #Ancho total
        max_dim = max(first_frame.shape)
        scale = resize_dim/max_dim
        first_frame = cv2.resize(first_frame, None, fx=scale, fy=
scale)
        # Convierte el primer cuadro a escala de grises
        prev_gray = cv2.cvtColor(first_frame, cv2.COLOR_BGR2GRAY)

        """Declaracion de variables varias"""
        umbral=50 #Umbral de deteccion de movimiento
        mask = np.zeros_like(first_frame) #Matriz de zeros para
alojar O.F.
```

```
        mask[..., 1] = 255 # Nos deshacemos de la saturacion de la
imagen poniendo al maximo
        output_folder2="NuestroOFResultados/"
        i=0
        flow2 = np.zeros((562, 1000))
        flow3 = np.zeros((562, 1000))
        kernel=np.ones((11,11),np.uint8)

        while(vc.isOpened() and _==True):
            # Read a frame from video
            _, frame = vc.read()
            if frame is None:
                print("Fin del video.")
                break

            # Convert new frame format`s to gray scale and resize
gray frame obtained
            gray = cv2.resize(frame, None, fx=scale, fy=scale)
            gray = cv2.cvtColor(gray, cv2.COLOR_BGR2GRAY)
            gray = cv2.normalize(gray,None, 0, 1.0, cv2.NORM_MINMAX,
dtype=cv2.CV_32F)
            #gray = cv2.bilateralFilter(gray,9,75,75) #Filtrado de
fotograma para eliminar ruido
            gray = cv2.GaussianBlur(gray,(3,3),0)

            """Calculo del Optic Flow"""
            diff = abs(prev_gray-gray)
            #np.savetxt('mi_arreglo.csv', diff, delimiter=',')
            cv2.imshow('GRIS-ORIGINAL', gray)
            condicion= diff> umbral/255
            condicion2= diff<= umbral/255
            diff[condicion] = 255
            diff[condicion2] = 0
            diff = diff.astype(np.uint8)
            cv2.imshow('OF+Original', diff)
            #np.savetxt('mi_arreglo.csv', diff, delimiter=',')
            prev_gray=gray

            """Despliegue Optic Flow"""
            factor_atenuacion = 0.1  # Ajusta este valor segun tus
preferencias
            frame_oscuro = frame.astype(float) * factor_atenuacion
            frame = cv2.resize(frame_oscuro.astype(np.uint8), None,
```

```
fx=scale, fy=scale) # Redimension
                diff2=diff
                diff = cv2.cvtColor(diff, cv2.COLOR_GRAY2BGR)
                # Anadido y despliegue
                dense_flow = cv2.addWeighted(frame, 1,diff, 2, 0)
                cv2.imshow("Dense optical flow", dense_flow)

                """Generacion MEI imagen"""
                if 50 <= i <= 80:
                    flow2=diff2+flow2
                    #imagen = flow2[:, :, 1]
                    imagen_escalada = cv2.normalize(flow2, None, 0, 255,
cv2.NORM_MINMAX)
                    # Convertir la imagen a tipo uint8
                    imagen_uint8 = imagen_escalada.astype(np.uint8)
                    # Mostrar la imagen con cv2
                    cv2.imshow('MEI', imagen_uint8)
                    cv2.imwrite('NuestroOFResultados/MEI.png',
imagen_uint8)

                """Generacion MHI"""
                if 50 <= i <= 80:
                    magnitude2=cv2.normalize(diff2, None, 0, 255, cv2.
NORM_MINMAX)
                    # Aplica la condicion a cada elemento
                    condicion = magnitude2 >= 18
                    condicion2 = magnitude2 < 18
                    # Establece a 255 solo los elementos que cumplen la
condicion
                    magnitude2[condicion] = 255
                    magnitude2[condicion2] = 0
                    flow3=magnitude2+flow3*.8
                    #imagen = flow2[:, :, 1]
                    imagen_escalada2 = cv2.normalize(flow3, None, 0, 255,
 cv2.NORM_MINMAX)
                    # Convertir la imagen a tipo uint8
                    imagen_uint82 = imagen_escalada2.astype(np.uint8)
                    # Mostrar la imagen con cv2
                    cv2.imshow('MHI', imagen_uint82)
                    cv2.imwrite('NuestroOFResultados/MHI.png',
imagen_uint82)

                out.write(dense_flow) #Creacion de video de imagen
```

```
original + optic flow rgb
            i=i+1
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        for x in range(562):
            for y in range(1000):
                if imagen_escalada[x,y] > 0:
                    imagen_escalada[x,y] = 255
                else:
                    imagen_escalada[x,y] = 0

        cv2.imwrite('NuestroOFResultados/MEI binaria2.png',
imagen_escalada)

        """Contorno del MEI/SOBEL"""
        sobel=cv2.Sobel(imagen_escalada, ddepth=cv2.CV_64F,dx=1,dy=1,
ksize=5)
        cv2.imwrite('NuestroOFResultados/ContornoDelMEI.png',sobel)

        vc.release()
        cv2.destroyAllWindows()
```

## 4.4. Explicación específica de los programas

### 4.4.1. Funciones Codigo 1

- Estimación de Flujo Óptico
  Se emplea una técnica de diferenciación de frames para detectar cambios significativos, marcando las regiones de movimiento en blanco. El flujo óptico se calcula mediante la detección de cambios de intensidad entre frames consecutivos:

```
        diff = abs(prev_gray-gray)
        #np.savetxt('mi_arreglo.csv', diff, delimiter=',')
        cv2.imshow('GRIS-ORIGINAL', gray)
        condicion= diff> umbral/255
        condicion2= diff<= umbral/255
        diff[condicion] = 255
        diff[condicion2] = 0
```

- Preprocesamiento
  Se aplica un filtrado bilateral y un desenfoque gaussiano para reducir el ruido y mejorar la robustez del flujo óptico.

```
        #gray = cv2.bilateralFilter(gray,9,75,75)
        #Filtrado de fotograma para eliminar ruido
            gray = cv2.GaussianBlur(gray,(3,3),0)
```

- Visualización
  La representación visual del flujo óptico se superpone en la imagen original, destacando las áreas de movimiento.

```
        #Anadido y despliegue
        dense_flow = cv2.addWeighted(frame, 1,diff, 2, 0)
        cv2.imshow("Dense optical flow", dense_flow)
```

- Generación de Imágenes de Energía de Movimiento (MEI)
  Proceso de Generación: Se acumulan las regiones de movimiento detectadas durante un intervalo específico para obtener una Imagen de Energía de Movimiento (MEI).

```
        """Generacion MEI imagen"""
        if 50 <= i <= 80:
            flow2=diff2+flow2
            #imagen = flow2[:, :, 1]
            imagen_escalada = cv2.normalize(flow2, None, 0, 255, cv2.
NORM_MINMAX)
            # Convertir la imagen a tipo uint8
            imagen_uint8 = imagen_escalada.astype(np.uint8)
            # Mostrar la imagen con cv2
            cv2.imshow('MEI', imagen_uint8)
            cv2.imwrite('NuestroOFResultados/MEI.png',imagen_uint8)
```

- Limpieza de Ruido
  Se aplican operaciones morfológicas, como la apertura y cierre, para eliminar posibles ruidos en la MEI.

- Mapa de Historia de Movimiento (MHI)
  Proceso de Generación: Similar a MEI, se acumulan las magnitudes de movimiento, generando el Mapa de Historia de Movimiento (MHI).

```
            if 50 <= i <= 80:
            magnitude2=cv2.normalize(diff2, None, 0, 255, cv2.NORM_MINMAX
)


            # Aplica la condicion a cada elemento
            condicion = magnitude2 >= 18
            condicion2 = magnitude2 < 18
            # Establece a 255 solo los elementos que cumplen la condicion
            magnitude2[condicion] = 255
            magnitude2[condicion2] = 0



            flow3=magnitude2+flow3*.8
            #imagen = flow2[:, :, 1]
            imagen_escalada2 = cv2.normalize(flow3, None, 0, 255, cv2.
NORM_MINMAX)
            # Convertir la imagen a tipo uint8
            imagen_uint82 = imagen_escalada2.astype(np.uint8)
            # Mostrar la imagen con cv2
            cv2.imshow('MHI', imagen_uint82)
            cv2.imwrite('NuestroOFResultados/MHI.png',imagen_uint82)
```

- Binarización de la Imagen de Energía de Movimiento (MEI)

  Esta parte del código se encarga de realizar una binarización de la imagen ‘imagen-escalada‘ y luego guardarla en un archivo llamado ’MEI binaria2.png’. La binarización implica convertir la imagen en una imagen binaria, donde todos los píxeles se establecen en blanco (255 en escala de grises) si su valor original es mayor que 0, y en negro (0 en escala de grises) si su valor original es igual a 0.

  Recorrido de píxeles:
  - El bucle ‘for x in range(562)‘ itera sobre las filas de la imagen.
  - El bucle ‘for y in range(1000)‘ itera sobre las columnas de la imagen.

  Binarización:

- Para cada píxel en la posición ‘(x, y)‘ de la imagen ‘imagen-escalada‘, se verifica si el valor original (‘imagen-escalada[x, y]‘) es mayor que 0.
- Si el valor es mayor que 0, se establece el píxel en blanco (255).
- Si el valor es igual a 0, se establece el píxel en negro (0).

Guardado de la imagen binaria
- Finalmente, después de recorrer todos los píxeles y realizar la binarización, se guarda la imagen resultante en el archivo ’MEI binaria2.png’ utilizando la función ‘cv2.imwrite()‘.

```
        for x in range(562):
        for y in range(1000):
            if imagen_escalada[x,y] > 0:
                imagen_escalada[x,y] = 255
            else:
                imagen_escalada[x,y] = 0
    cv2.imwrite('NuestroOFResultados/MEI binaria2.png',
imagen_escalada)
```

- Contorno del MEI y Sobel
  Generación del Contorno: Se aplica el operador Sobel al MEI binarizado para resaltar los contornos de las regiones de movimiento.

```
        sobel=cv2.Sobel(imagen_escalada, ddepth=cv2.CV_64F,dx=1,dy=1,
ksize=5)
        cv2.imwrite('NuestroOFResultados/ContornoDelMEI.png',sobel)
```

- Video de Resultados
  Generación del Video: Se crea un video que combina la imagen original con la representación visual del flujo óptico.

```
        out.write(dense_flow) #Creacion de video de imagen original +
 optic flow rgb
```

### 4.4.2. Funciones Codigo 2

El flujo óptico denso es una técnica crucial en visión por computadora que permite entender el movimiento en un video al analizar el desplazamiento de los píxeles entre cuadros consecutivos.

- Funciones de Procesamiento
  El código implementa dos funciones para el procesamiento de imágenes: ‘flujo-to-image‘ y ‘high-pass-filter‘.

- 1.flujo-to-image‘
  Esta función toma un flujo óptico (un arreglo bidimensional que representa el desplazamiento de píxeles entre dos cuadros consecutivos) y devuelve una imagen en tonos de gris que representa la magnitud del flujo óptico normalizada entre 0 y 255.

  Entrada:
  flujo: Un arreglo bidimensional que representa el flujo óptico.

  Proceso:
  Calcula la magnitud del flujo óptico utilizando la fórmula de la norma euclidiana (‘np.sqrt(np.sum(flujo**2, axis=-1))‘). Normaliza la magnitud del flujo a valores entre 0 y 255 utilizando la función ‘cv2.normalize‘. Crea una imagen en tonos de gris utilizando el arreglo de magnitudes normalizadas.

  Salida:
  ‘magnitud-normalizada‘: Una imagen en tonos de gris que representa la magnitud del flujo óptico normalizada.

```
        def flujo_to_image(flujo):
        # Convertir el flujo a valores entre 0 y 255 para su
visualizacion
        magnitud = np.sqrt(np.sum(flujo**2, axis=-1))
        magnitud_normalizada = cv2.normalize(magnitud, None, 0, 255,
cv2.NORM_MINMAX)
        # Crear una imagen en tonos de gris
        imagen_flujo = np.zeros_like(flujo)
        imagen_flujo[..., 0] = magnitud_normalizada
        return magnitud_normalizada
```

- 2.high-pass-filter

  Esta función implementa un filtro pasa altas en el dominio de la frecuencia utilizando la transformada de Fourier. El filtro pasa altas se utiliza para resaltar las frecuencias de alta magnitud en la imagen.

  Entrada
  - image: Una imagen de entrada.
  - cutoff-frequency: La frecuencia de corte que determina qué tan .ªltas"deben ser las frecuencias para ser conservadas.

  Proceso
  - Aplica la transformada de Fourier 2D a la imagen de entrada.
  - Desplaza el componente de baja frecuencia al centro de la imagen utilizando np.fft.fftshift.
  - Crea un filtro pasa altas en forma de un círculo con valores 0 en el centro y 1 en las regiones externas.
  - Multiplica el filtro pasa altas con la transformada de Fourier desplazada.
  - Deshace el desplazamiento y aplica la transformada inversa para obtener la imagen filtrada.

  Salida
  - image-filtered‘: Una imagen filtrada que resalta las frecuencias de alta magnitud

.

```
def high_pass_filter(image, cutoff_frequency):
# Transformada de Fourier 2D
f_transform = np.fft.fft2(image)

# Desplazamos el componente de baja frecuencia al centro
f_transform_shifted = np.fft.fftshift(f_transform)

# Dimensiones de la imagen
rows, cols = image.shape
crow, ccol = rows // 2 , cols // 2

# Creamos un filtro pasa altas
mask = np.ones((rows, cols), np.uint8)
```

```
        r = cutoff_frequency
        center = [crow, ccol]
        x, y = np.ogrid[:rows, :cols]
        mask_area = (x - center[0])**2 + (y - center[1])**2 <= r*r
        mask[mask_area] = 0

        # Aplicamos el filtro pasa altas multiplicando con la
transformada de Fourier
        f_transform_shifted_filtered = f_transform_shifted * mask

        # Deshacemos el desplazamiento
        f_transform_filtered = np.fft.ifftshift(
f_transform_shifted_filtered)

        # Aplicamos la transformada inversa para obtener la imagen
filtrada
        image_filtered = np.fft.ifft2(f_transform_filtered)
        image_filtered = np.abs(image_filtered)

        return image_filtered
```

- Configuración Inicial

  Comienza abriendo un archivo de video llamado "dance2.mp4z configura un objeto ‘cv2.VideoWriter‘ para almacenar el resultado del procesamiento. El primer cuadro se escala y convierte a escala de grises para establecer una referencia inicial.

```
        vc = cv2.VideoCapture("dance2.mp4")
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Generando el
archivo donde se guardara el video
        out = cv2.VideoWriter('dense_optical_flow_output.mp4', fourcc
, 20.0, (1000, 562))
        # Read first frame
        _, first_frame = vc.read()
        # Scale and resize image
        resize_dim = 1000
        max_dim = max(first_frame.shape)
        scale = resize_dim/max_dim
        first_frame = cv2.resize(first_frame, None, fx=scale, fy=
scale)
        # Convierte el primer cuadro a escala de grises
        prev_gray = cv2.cvtColor(first_frame, cv2.COLOR_BGR2GRAY)
```

```
        # Create mask
        mask = np.zeros_like(first_frame)
        # Set image saturation to maximum value as we do not need it
        mask[..., 1] = 255
        i=0
        output_folder="frames_opticflow/"
        kernel=np.ones((11,11),np.uint8)
        mei_accumulator = np.zeros_like(prev_gray, dtype=np.float32)
        mei_accumulator2 = np.zeros_like(prev_gray, dtype=np.float32)
        flow2 = np.zeros((562, 1000))
        flow3 = np.zeros((562, 1000))
        threshold=30
```

- Proceso Principal

  Cálculo del Flujo Óptico: Dentro de un bucle principal, cada cuadro del video se procesa mediante el método de Farneback para calcular el flujo óptico denso. Se visualiza y guarda la magnitud del flujo óptico como imágenes individuales en la carpeta 'frames-opticflow'.

```
        while(vc.isOpened() and _==True):
        # Read a frame from video
        _, frame = vc.read()

        # Convert new frame format's to gray scale and resize gray
frame obtained
        gray = cv2.resize(frame, None, fx=scale, fy=scale)
        gray = cv2.cvtColor(gray, cv2.COLOR_BGR2GRAY)

        cv2.imshow('gray',gray)
        # Calculate dense optical flow by Farneback method
        flow = cv2.calcOpticalFlowFarneback(prev_gray, gray, None,
pyr_scale = 0.5, levels = 3, winsize = 15, iterations = 3, poly_n = 5,
 poly_sigma = 1.2, flags = 0)

        filename = f'{output_folder}opticflow_{i}.png'
        cv2.imwrite(filename, flujo_to_image(flow))
```

- Generación del MEI RGB

  Se genera una representación visual del flujo óptico utilizando el modelo de color

HSV. La dirección del flujo determina el tono, mientras que la magnitud influye en la intensidad y saturación de la imagen. La imagen resultante se muestra junto con la imagen original.

```
                """Calculo del MEI RGB"""
                # Compute the magnitude and angle of the 2D vectors
                magnitude, angle = cv2.cartToPolar(flow[..., 0], flow
[..., 1])
                # Set image hue according to the optical flow direction
                mask[..., 0] = angle * 180 / np.pi / 2
                # Set image value according to the optical flow magnitude
 (normalized)
                mask[..., 2] = cv2.normalize(magnitude, None, 0, 255, cv2
.NORM_MINMAX)
                # Convert HSV to RGB (BGR) color representation
                rgb = cv2.cvtColor(mask, cv2.COLOR_HSV2BGR)
                # Ajustar la intensidad de la imagen original para
hacerla mas oscura
                factor_atenuacion = 0.1  # Ajusta este valor segun tus
preferencias
                frame_oscuro = frame.astype(float) * factor_atenuacion
                # Resize frame size to match dimensions
                frame = cv2.resize(frame_oscuro.astype(np.uint8), None,
fx=scale, fy=scale)
                # Open a new window and displays the output frame
                dense_flow = cv2.addWeighted(frame, 1,rgb, 2, 0)
                cv2.imshow("Dense optical flow", dense_flow)
```

- Generación del MEI y MHI
  Durante un intervalo específico del video (cuadros 50 a 80), se acumulan las magnitudes del flujo óptico para generar el MEI y se calcula la magnitud del hueco de información (MHI). Ambas imágenes se escalan y normalizan para su visualización.

```
            """Generacion MHI"""
                if 50 <= i <= 80:
                    magnitude2=cv2.normalize(magnitude, None, 0, 255, cv2
.NORM_MINMAX)

                    # Aplica la condicion a cada elemento
```

```
                    condicion = magnitude2 >= 18
                    condicion2 = magnitude2 < 18
                    # Establece a 255 solo los elementos que cumplen la
condicion
                    magnitude2[condicion] = 255
                    magnitude2[condicion2] = 0


                    flow3=magnitude2+flow3*.8
                    #imagen = flow2[:, :, 1]
                    imagen_escalada2 = cv2.normalize(flow3, None, 0, 255,
 cv2.NORM_MINMAX)
                    # Convertir la imagen a tipo uint8
                    imagen_uint82 = imagen_escalada2.astype(np.uint8)
                    # Mostrar la imagen con cv2
                    cv2.imshow('Imagen2', imagen_uint82)
                    cv2.imwrite('MHI.png',imagen_uint82)


                out.write(dense_flow) #Creacion de video de imagen
original + optic flow rgb
                prev_gray = gray # Update previous frame
                i=i+1
                # Frame are read by intervals of 10 millisecond. The
programs breaks out of the while loop when the user presses the q key
                if cv2.waitKey(1) & 0xFF == ord('q'):
                    break
```

- Escritura de Resultados

  Cada cuadro procesado se agrega al video de salida, que combina la imagen original con la representación visual del flujo óptico. Se actualiza el cuadro anterior y se verifica si el usuario presiona la tecla 'q' para salir del bucle.

- Postprocesamiento

  Limpieza del MEI: Se aplica erosión y dilatación al MEI para mejorar su representación y se binariza para resaltar las áreas de interés.

```
                """Limpieza del MEI"""
            imagen_erodida = cv2.erode(imagen_escalada, kernel,
iterations=2)
```

```
        imagen_escalada = cv2.dilate(imagen_erodida, kernel,
iterations=2)
        for x in range(562):
            for y in range(1000):
                if imagen_escalada[x,y] >= 23:
                    imagen_escalada[x,y] = 255

                else:
                    imagen_escalada[x,y] = 0


        cv2.imwrite('MEI binaria2.png',imagen_escalada)
```

- Contorno del MEI
  Se utiliza el operador de Sobel para obtener el contorno del MEI y se guarda como una imagen separada.

```
        """Contorno del MEI/SOBEL"""
    sobel=cv2.Sobel(imagen_escalada, ddepth=cv2.CV_64F,dx=1,dy=1,
ksize=5)
    cv2.imwrite('ContornoDelMEI.png',sobel)
```

## 4.5. Resultados

### 4.5.1. Resultados: 1. Código

- Generación de Imágenes de Energía de Movimiento (MEI)

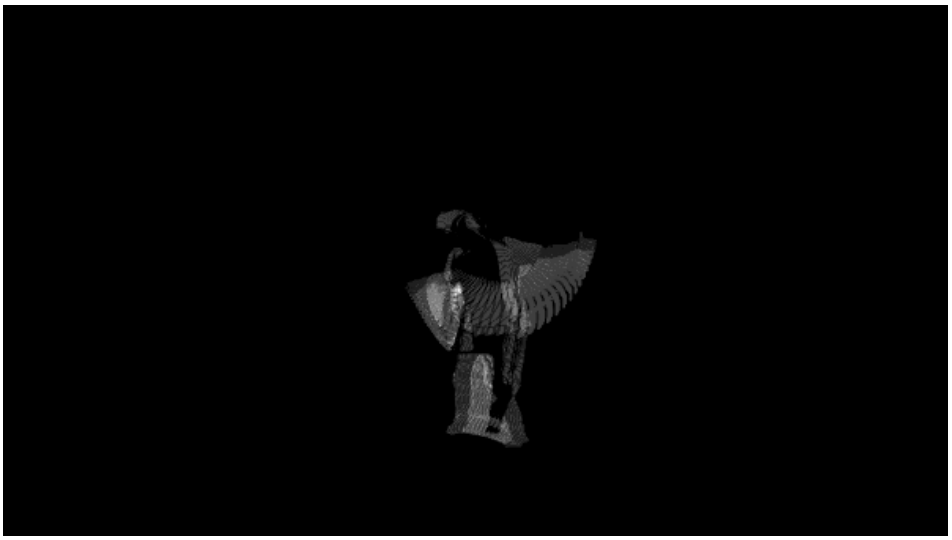

Figura 3: Imagen MEI de un frame del video

- Mapa de Historia de Movimiento (MHI)

Figura 4: Imagen MHI de un frame del video.

- Binarización de la Imagen de Energía de Movimiento (MEI)

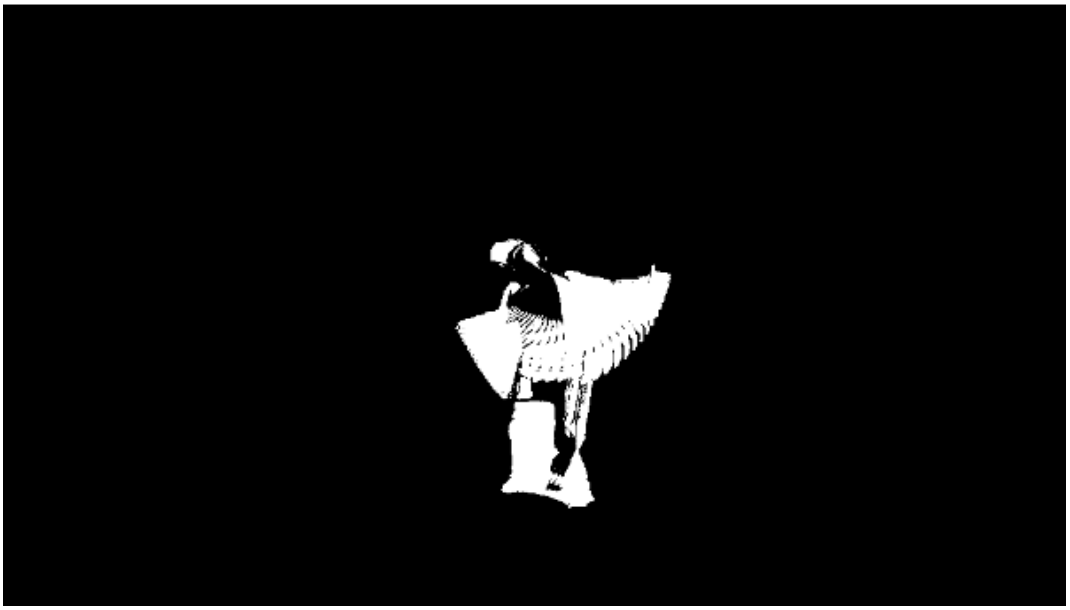

Figura 5: Binarización de la imagen.

- Contorno del MEI y Sobel

Figura 6: Contorno del MEI.

- Video de Resultados

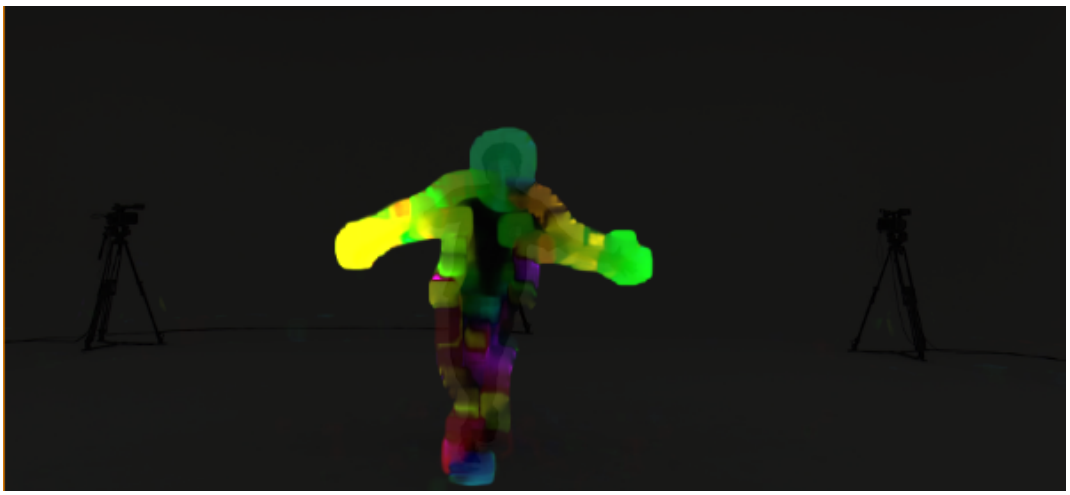

Figura 7: Vídeo creado.

### 4.5.2. Resultados: 2. Código

- Generación de Imágenes de Energía de Movimiento (MEI)

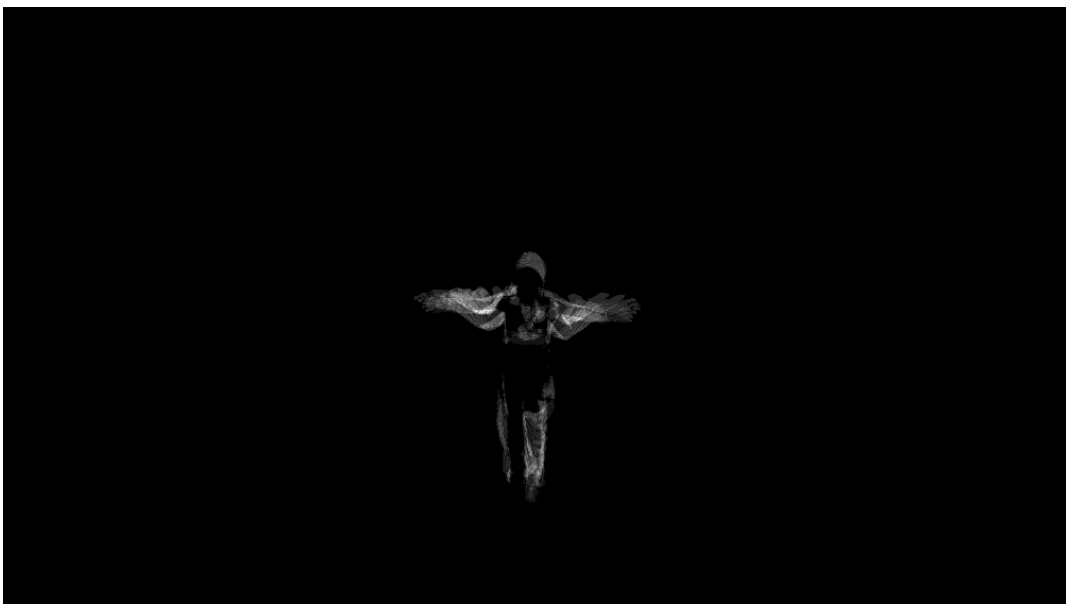

Figura 8: Imagen MEI de un frame del video

- Mapa de Historia de Movimiento (MHI)

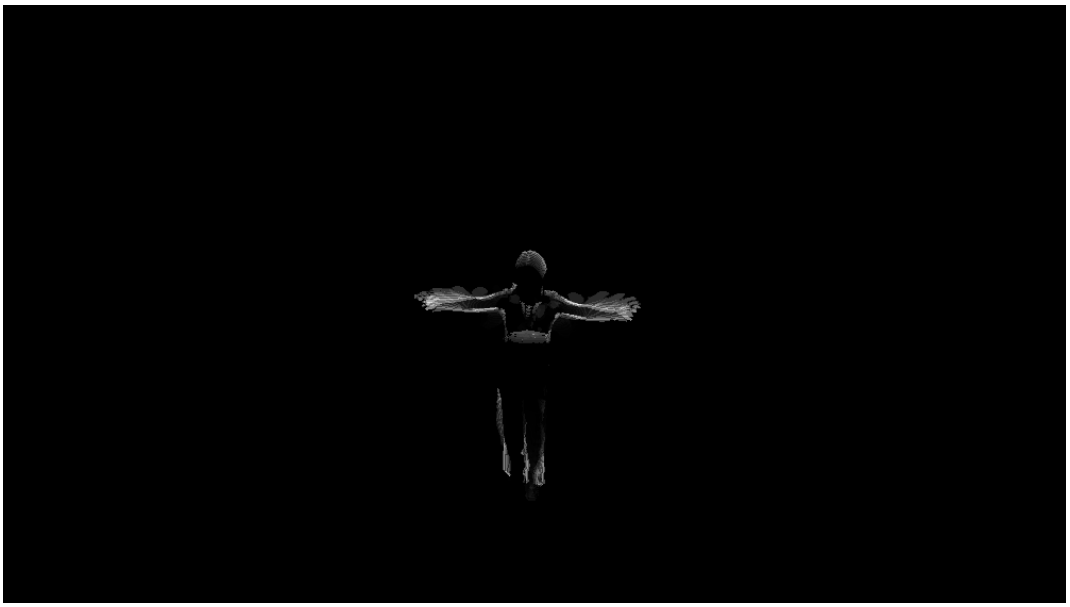

Figura 9: Imagen MHI de un frame del video.

- Binarización de la Imagen de Energía de Movimiento (MEI)

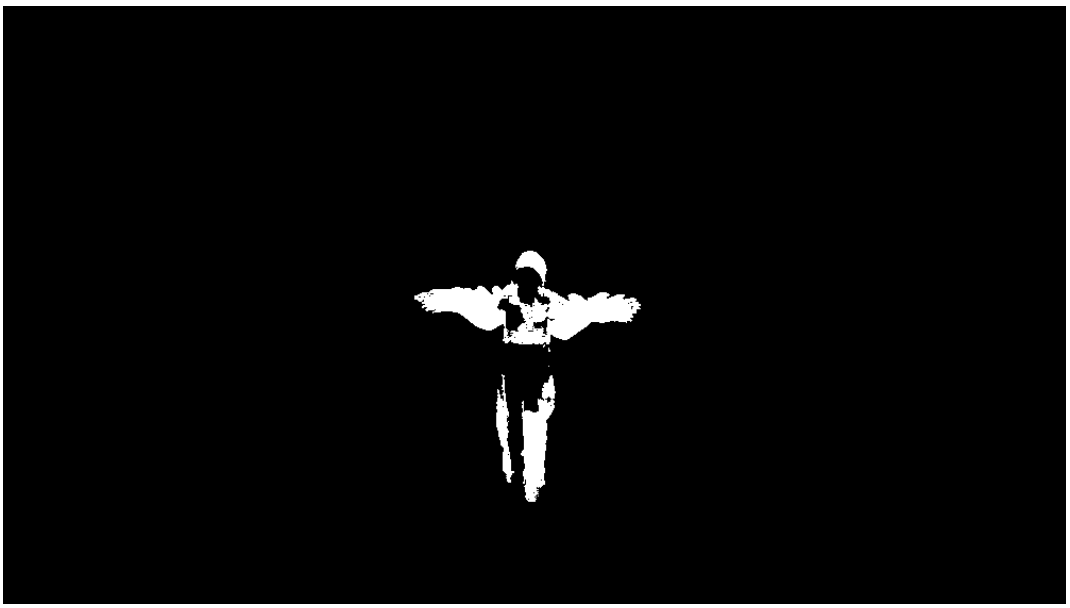

Figura 10: Binarización de la imagen.

- Contorno del MEI

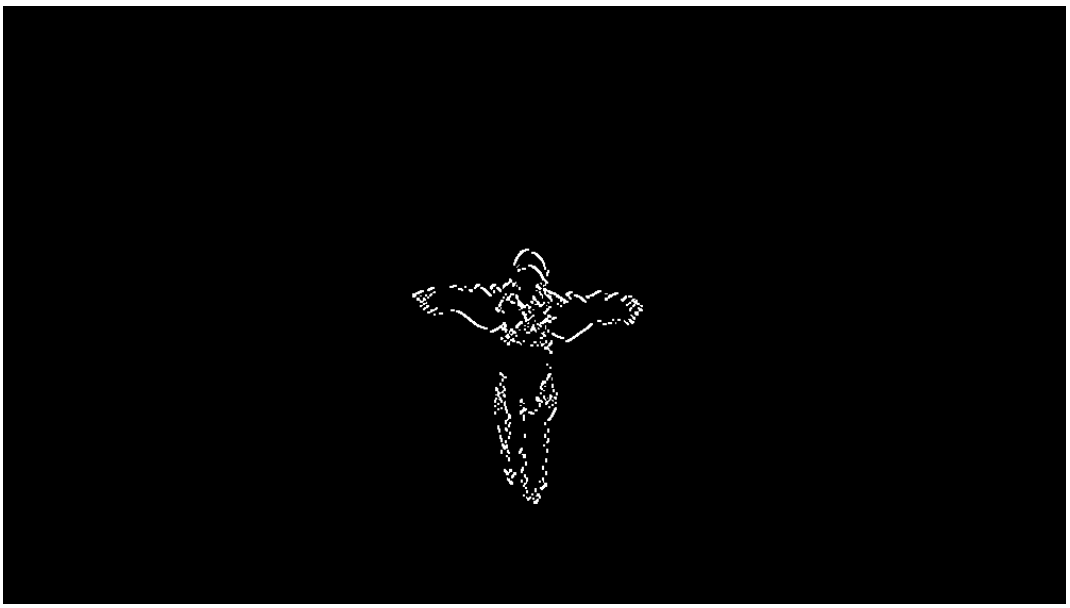

Figura 11: Contorno del MEI.

- Video de Resultados

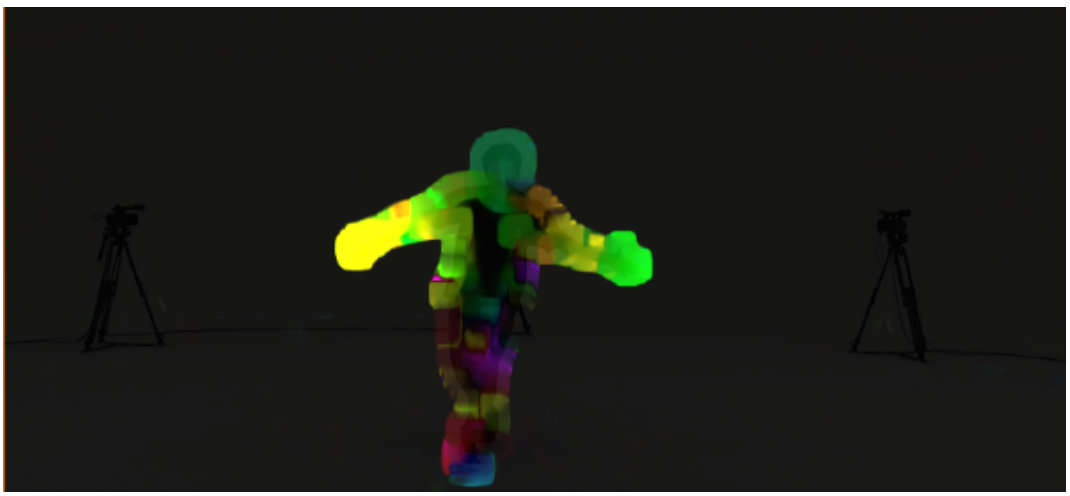

Figura 12: Vídeo creado.

## 4.6. Tabla de subsecuencias

| Video | SUBSECUENCIA 1/Video 1 | | | | SUBSECUENCIA 2/Video 2 | | | |
|---|---|---|---|---|---|---|---|---|
| Metodo | NUESTRO METODO | | METODO CV2 | | NUESTRO METODO | | METODO CV2 | |
| #frames | 20 fps | 30 fps | 20 fps | 30 fps | 20 fps | 30 fps | 20 fps | 30 fps |
| Flujo Óptico | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| MEI | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| MHI | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Limpieza del MEI | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Contorno del MEI | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Figura 13: Tabla de subsecuencias

# 5. Conclusiones

## 5.1. Conclusión General

En términos de eficiencia, el código presenta una implementación robusta y eficaz para la estimación de flujo óptico y la generación de imágenes de movimiento. Aunque se logra un rendimiento en tiempo real, existe margen para explorar optimizaciones adicionales que puedan mejorar aún más la velocidad de procesamiento, asegurando una ejecución más fluida en diversas configuraciones.

En cuanto a la adaptabilidad, es importante destacar que el código está actualmente diseñado para trabajar con un video específico ("dance2.mp4"). Sería altamente beneficioso avanzar hacia una implementación más general que permita trabajar con diferentes secuencias de video. Esto proporcionaría una mayor versatilidad y utilidad del código en diversos contextos.

Una mejora significativa podría introducirse mediante la interactividad del programa. La capacidad de seleccionar intervalos de interés para la generación de Imágenes de Energía de Movimiento (MEI) y Mapas de Historia de Movimiento (MHI) a través de parámetros de entrada sería valiosa. Esto haría que la aplicación sea más flexible y adaptable a las necesidades específicas del usuario, permitiendo un análisis más detallado y personalizado de las secuencias de video.

Adicionalmente, una característica sugerida para una mayor interactividad sería la capacidad de utilizar la cámara en tiempo real en lugar de depender de un archivo de video pregrabado. Esto convertiría la aplicación en una herramienta aún más dinámica

y práctica, abriendo posibilidades para aplicaciones en tiempo real y escenarios donde la interacción en vivo sea esencial.

# 6. Referencias


- Bobick, A. F., Davis, J. W. (2001). The recognition of human movement using temporal templates. IEEE Transactions on Pattern Analysis and Machine Intelligence, 23(3), 257-267.
  https://doi.org/10.1109/34.910878

- Nicolai Nielsen. (2021, 29 septiembre). Motion Detection made easy: Optical flow in OpenCV Python [Vídeo]. YouTube.
  https://www.youtube.com/watch?v=WrlH5hHv0gE

- Tsuchida, S. (s. f.-a). AIST Dance Video Database (AIST Dance DB).
  https://aistdancedb.ongaaccel.jp/

- Tsuchida, S. (s. f.-b). Database download - AIST Dance Video Database (AIST Dance DB). https://aistdancedb.ongaaccel.jp/database$_d$*ownload*/