UNIVERSIDAD DE LAS FUERZAS ARMADAS ESPE

APLICACIONES BASADAS EN EL CONOCIMIENTO

TEMA: MOVIMIENTO DEL MOUSE POR MEDIO DEL RECONOCIMIENTO DE LAS PUPILAS DE LOS OJOS

INTEGRANTES

RICHARD ALBAN
CHRISTIAN ANDRANGO
AYME ESCOBAR
CARLOS IPIALES
ANGELO YANACALLO

DOCENTE: ING. DIEGO LEONARDO GAMBOA SAFLA

NRC: 15039

SANGOLQUI

Noviembre 2023 - Marzo 2024

ÍNDICE

Índice de Tablas

Índice de Ilustraciones

1. Objetivo General

Desarrollar un sistema de control de ratón basado en el movimiento de los ojos utilizando técnicas de procesamiento de imágenes y detección facial en tiempo real.

1.1. Objetivos Específicos

- Implementar un algoritmo de detección facial utilizando la biblioteca MediaPipe para identificar los puntos clave del rostro, especialmente los relacionados con los ojos.
- Utilizar la información de los puntos clave detectados para calcular la posición del cursor del ratón en la pantalla en función del movimiento de los ojos.
- Integrar la funcionalidad de control de clics del ratón basada en el parpadeo o la posición relativa de los puntos clave de los ojos, con el objetivo de permitir al usuario interactuar con la computadora sin necesidad de un dispositivo de entrada físico.

2. Introducción

En la actualidad, con el vertiginoso desarrollo de nuevas tecnologías, se ha evidenciado la necesidad apremiante de explorar soluciones innovadoras que no solo simplifiquen nuestras interacciones con la tecnología, sino que también abran nuevas fronteras de accesibilidad para todos los usuarios. En este contexto, el equipo de desarrollo de la aplicación se propuso enfrentar el desafío de redefinir la experiencia de usuario, especialmente para aquellos que enfrentan limitaciones en la movilidad convencional.

El rápido avance tecnológico ha proporcionado herramientas increíbles, pero también ha dejado al descubierto las brechas en la accesibilidad, particularmente para aquellos con discapacidades motoras. Inspirados por la posibilidad de hacer una contribución significativa en este espacio, el equipo se sumergió en la creación de una solución que aprovecha tecnologías ya existentes como OpenCV y Mediapipe que cuentan con un buen grado de maduración y un entorno de desarrollo que está a la vanguardia en los tiempos que vivimos como lo es Python, lo que nos para proporciona una gran entrono de desarrollo que es flexible ante las necesidades para la implementación de la aplicación ayudando a crear una aplicación de interacción más intuitiva y accesible entre los usuarios y los medios digitales.

La aplicación desarrollada surge como una respuesta a la creciente necesidad de ofrecer opciones inclusivas que trasciendan los límites convencionales de control de dispositivos. La premisa fundamental de este desarrollo es proporcionar una herramienta que permita a los usuarios controlar su entorno digital mediante el seguimiento ocular, superando las barreras físicas que podrían dificultar la experiencia de uso.

La aplicación permite controlar el puntero del ratón y realizar clics mediante el seguimiento de los movimientos oculares del usuario. Utilizando técnicas de detección de malla facial, la aplicación identifica los puntos clave en el rostro para traducir el

movimiento de los ojos en acciones de control del ratón. Además, incorpora la capacidad de grabar estas interacciones, lo que puede ser útil para análisis de usabilidad o para documentar las acciones realizadas.

Lo innovador de esta aplicación radica en su enfoque no invasivo y en su capacidad para ofrecer una solución de control del ratón basada en movimientos oculares. Esto podría ser particularmente beneficioso en entornos donde el uso de extremidades superiores está limitado o no es posible. Además, la capacidad de grabación permite un análisis detallado de las interacciones, lo que podría ser valioso en entornos de investigación y desarrollo de interfaces de usuario.

Dado este preámbulo, exploraremos cómo la aplicación no solo busca simplificar las interacciones diarias, sino también redefinir el paradigma de accesibilidad tecnológica, proveyendo una solución vanguardista que tiene el potencial de mejorar significativamente la calidad de vida de aquellos que enfrentan desafíos de movilidad.

3. Desarrollo

Con respecto a la implementación del sistema de control del ratón basado en el movimiento de los ojos, se detallará la información que se obtuvo en los siguientes subtítulos:

3.1. Lenguaje de programación:

Python como lenguaje de programación, el cual es reconocido por su sintaxis clara y legible, así como por su versatilidad y amplia gama de bibliotecas disponibles para el desarrollo (Martín, 2024).

3.2. Bibliotecas Utilizadas:

En el desarrollo del proyecto, se han empleado diversas bibliotecas con el propósito de habilitar funcionalidades específicas y facilitar el procesamiento de datos. A continuación, en la Tabla 1 se detallan las bibliotecas utilizadas:

Biblioteca	Descripción
OpenCV	Utilizada para el procesamiento de imágenes, incluida la captura de video, manipulación de imágenes y detección de objetos.
Mediapipe	Utilizada para detectar y procesar puntos de referencia faciales en el video.
PyAutoGUI	Utilizada para controlar el mouse y realizar acciones como hacer clic en la pantalla.
Tkinter	Utilizada para crear la interfaz gráfica de usuario (GUI) en la aplicación.
Threading	Utilizada para ejecutar el procesamiento de video en hilos separados.

Tabla 1 Bibliotecas Utilizadas

3.3. IDEs

Para el desarrollo, se puede implementar en los diferentes entornos de desarrollo integrado (IDEs) presentados en la Tabla 2.

IDEs	Descripción
Visual Studio Code	Editor de código fuente desarrollado por Microsoft compatible con varios lenguajes de programación (Flores, 2022).
PyCharm	Entorno de desarrollo integrado (IDE) para Python desarrollado por JetBrains (JetBrains, 2021).
Colab (Google Colaboratory)	Entorno de cuadernos colaborativos basado en la nube proporcionado por Google (Google, s.f.).

Tabla 2 IDEs Utilizados

3.4. Modelo de Detección de Objetos Pre-Entrenados:

Se emplea la detección de puntos de referencia faciales mediante la biblioteca Mediapipe. Esta biblioteca ofrece un conjunto de modelos pre-entrenados que permiten detectar y procesar de manera eficiente los puntos de referencia faciales en el flujo de video (MediaPipe Solutions, s.f.). La utilización de Mediapipe proporciona una solución robusta y precisa para la detección de objetos en tiempo real, lo cual es fundamental para las funcionalidades de la aplicación en desarrollo.

3.5. Sistema de Seguimiento de Objetos:

En esta sección, se ha implementado un sistema de seguimiento de los puntos de referencia faciales, especialmente focalizado en el seguimiento de los ojos, a través de la detección de puntos clave y la aplicación de círculos circundantes en cada fotograma del video. Esta técnica se ha empleado con el propósito de monitorear y determinar la posición de los ojos en cada instancia del video.

El procedimiento consiste en la identificación meticulosa de los puntos característicos en la región facial, seguido por la colocación precisa de círculos en torno a estos puntos en cada fotograma del video. Este enfoque posibilita la visualización efectiva de la posición ocular y, potencialmente, la observación de su trayectoria a medida que varía a lo largo del transcurso del video.

4. Implementación

4.1. Función Alternar Detección

En esta función se encarga de alternar el estado de la detección y actualizar la apariencia del botón correspondiente en la interfaz gráfica. Cuando se llama, cambia la variable ***detection_active*** de True a False o viceversa, y actualiza el texto, color de fondo y color de texto del botón ***btn_toggle_detection*** en consecuencia para reflejar el nuevo estado de la detección.

```
def toggle_detection(event=None):
    global detection_active
    detection_active = not detection_active
    if detection_active:
        btn_toggle_detection.config(text="Detener Detección (D)",
bg="#FF6347", fg="white", font=("Arial", 12, "bold"))
    else:
        btn_toggle_detection.config(text="Iniciar Detección (D)",
bg="#32CD32", fg="white", font=("Arial", 12))
```

Tabla 3. Función para Alternar la Detección.

4.2. Función para Alternar Grabación

La función ***toggle_recording*** se utiliza para alternar entre iniciar y detener la grabación de video, en donde vemos apartados como:

- La variable global ***recording_active*** para determinar si la grabación está actualmente activa o no.
- Si la grabación está activa, la función la detiene y actualiza la interfaz gráfica y la apariencia del botón correspondiente.
- Si la grabación no está activa, la función la inicia y configura el objeto de escritura de video para comenzar a grabar el video. También actualiza la interfaz gráfica y la apariencia del botón correspondiente para reflejar el estado de la grabación.
- Además, se accede a otras variables globales como ***start_time***, ***fps***, ***out*** para controlar el tiempo de inicio, la tasa de fotogramas y el objeto de salida de video respectivamente.

```
def toggle_recording(event=None):
# Variables globales para controlar el estado de la grabación, el
objeto de salida de video, la tasa de fotogramas y el tiempo de inicio
de la grabación
    global recording_active, out, fps, start_time
    if recording_active:
        # Si la grabación está activa, detenerla
        recording_active = False
        lbl_recording_status.config(text="Grabación terminada",
# Actualiza el estado de grabación en la interfaz gráfica
bg="#228B22", fg="white", font=("Arial", 12))
        btn_toggle_recording.config(text="Iniciar Grabación (R)",
# Actualiza la apariencia del botón para iniciar grabación
bg="#32CD32", fg="white", font=("Arial", 12))
    else:
        # Si la grabación no está activa, iniciarla
        # Establece el estado de grabación como activo
        recording_active = True
        # Guarda el tiempo de inicio de la grabación
        start_time = time.time()
```

```
# Crea un objeto codec de video
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')
        out = cv2.VideoWriter('output.mp4', fourcc, fps,
(int(cam.get(cv2.CAP_PROP_FRAME_WIDTH)),
# Inicia la grabación de video con el nombre 'output.mp4', el codec, la
tasa de fotogramas y las dimensiones del fotograma obtenidas de la
cámara
int(cam.get(cv2.CAP_PROP_FRAME_HEIGHT))))
        lbl_recording_status.config(text="Grabando...", bg="#DC143C",
# Actualiza el estado de grabación en la interfaz gráfica
fg="white", font=("Arial", 12))
        btn_toggle_recording.config(text="Detener Grabación (R)",
# Actualiza la apariencia del botón para detener grabación
bg="#FF6347", fg="white", font=("Arial", 12, "bold"))
```

Tabla 4. Función para alternar la grabación.

4.3. Función Salir del Programa

Esta función se encarga de salir del programa cuando se invoca, realizando las siguientes acciones:

- Verifica si la grabación está activa ***(recording_active)***, si lo está, llama a la función ***toggle_recording()*** para detener la grabación antes de salir del programa.

- Destruye la ventana principal de ***(root.destroy())***, lo que finaliza la aplicación visual.

- Verifica si existe un objeto de salida de video ***(out)***, si existe, libera los recursos asociados a él mediante el método ***release()***, lo que podría cerrar el archivo de video que se estaba grabando.

```
def quit_program(event=None):
    global root, out
    if recording_active:
        toggle_recording()
    root.destroy()
    if out is not None:
        out.release()
```

Tabla 5. Función para salir del Programa.

4.4. Procesamiento de Entrada de la Cámara

La función ***opencv_loop*** es la función principal encargada de procesar la entrada de la cámara en un bucle continuo, así como:

- Utilizar un bucle ***while True*** para continuar procesando los fotogramas de la cámara de forma indefinida.
- Lee un fotograma de la cámara utilizando ***cam.read()***, donde ***cam*** es el objeto de captura de video.
- Comprueba si la lectura del fotograma fue exitosa ***(ret es True)***.
  Si no se puede leer la cámara (por ejemplo, si se desconecta), la función sale del bucle.
- Voltea horizontalmente el fotograma utilizando ***cv2.flip()*** para asegurarse de que la imagen refleje correctamente los movimientos.
- Convierte el fotograma de BGR a RGB utilizando ***cv2.cvtColor()*** para asegurar que esté en el formato correcto para su posterior procesamiento.

```
def opencv_loop():
    global frame, detection_active, recording_active, out, fps,
start_time
    while True:
        ret, frame = cam.read()
        if not ret:
            break  # Si no se puede leer la cámara, salir del bucle
        frame = cv2.flip(frame, 1)
        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
```

Tabla 6. Procesamiento entrada de Cámara.

4.5. Activación de Detección

En esta sección de código es parte de un bucle que procesa cada fotograma de la cámara en tiempo real en el que es crucial para la funcionalidad del control ocular del programa y demuestra cómo se detectan y utilizan los puntos de referencia faciales para interactuar con el entorno de la computadora. El código contiene:

- ***if detection_active:*** Verifica si la detección está activada antes de procesar el fotograma.

- ***output = face_mesh.process(rgb_frame):*** Utiliza el modelo de detección facial (face_mesh) para procesar el fotograma convertido a formato RGB. Esto detectará los puntos de referencia faciales en la imagen.

- ***landmark_points = output.multi_face_landmarks:*** Extrae los puntos de referencia faciales detectados del resultado del procesamiento. Estos puntos de referencia incluyen información sobre la posición de varios puntos clave en la cara, como los ojos, la nariz y la boca.

- ***frame_h, frame_w, _ = frame.shape:*** Obtiene las dimensiones del fotograma actual para calcular las coordenadas de los puntos de referencia detectados.

- ***if landmark_points:*** Verifica si se detectaron puntos de referencia faciales en el fotograma actual.

- ***landmarks = landmark_points[0].landmark:*** Extrae los puntos de referencia específicos de la primera cara detectada en el fotograma.

- ***for id, landmark in enumerate(landmarks[474:478]):*** Itera sobre una selección específica de puntos de referencia faciales que corresponden a los ojos.

- ***x = int(landmark.x * frame_w), y = int(landmark.y * frame_h):*** Calcula las coordenadas (x, y) de cada punto de referencia facial en función de las dimensiones del fotograma.

- ***cv2.circle(frame, (x, y), 3, (0, 255, 0***)): Dibuja un círculo en el fotograma en la posición del punto de referencia facial, lo que ayuda a visualizar la detección de los ojos.

- ***if id == 1:*** Identifica el segundo punto de referencia facial (correspondiente a un ojo).

- ***screen_x = (screen_w * landmark.x), screen_y = (screen_h * landmark.y):*** Calcula las coordenadas en la pantalla en función de la posición del punto de referencia facial en el fotograma.

- ***pyautogui.moveTo(screen_x, screen_y):*** Mueve el cursor del mouse a la posición calculada en la pantalla. Esto se utiliza típicamente para simular el movimiento del mouse en función de los movimientos de los ojos detectados.

```
    if detection_active:
        output = face_mesh.process(rgb_frame)
        landmark_points = output.multi_face_landmarks
        frame_h, frame_w, _ = frame.shape
        if landmark_points:
            landmarks = landmark_points[0].landmark
            for id, landmark in enumerate(landmarks[474:478]):
                x = int(landmark.x * frame_w)
                y = int(landmark.y * frame_h)
                cv2.circle(frame, (x, y), 3, (0, 255, 0))
                if id == 1:
                    screen_x = (screen_w * landmark.x)
                    screen_y = (screen_h * landmark.y)
                    pyautogui.moveTo(screen_x, screen_y)
```

Tabla 7. Activación de Detección.

4.6. Movimiento de Detección de Ojos

Este bloque de código es fundamental para la funcionalidad de control ocular del programa, ya que detecta y responde a los movimientos mínimos del ojo izquierdo para simular un clic del mouse, tenemos secciones como:

```
left = [landmarks[145], landmarks[159]]
for landmark in left:
    x = int(landmark.x * frame_w)
    y = int(landmark.y * frame_h)
    cv2.circle(frame, (x, y), 3, (0, 255, 255))
if (left[0].y - left[1].y) < 0.005:
    pyautogui.click()
    pyautogui.sleep(1)
```

Tabla 8. Detección de Ojos.

- ***left = [landmarks[145], landmarks[159]]:*** Selecciona dos puntos de referencia faciales que representan los puntos extremos izquierdo y derecho del ojo izquierdo. Estos puntos se identifican mediante sus índices en la lista de puntos de referencia landmarks.

- ***for landmark in left:*** Itera sobre los puntos de referencia seleccionados (que representan el ojo izquierdo).

- ***x = int(landmark.x * frame_w), y = int(landmark.y * frame_h):*** Calcula las coordenadas (x, y) de cada punto de referencia en el fotograma actual en función de las dimensiones del fotograma.

- ***cv2.circle(frame, (x, y), 3, (0, 255, 255)):*** Dibuja un círculo en el fotograma en la posición del punto de referencia facial del ojo izquierdo. Esto ayuda a visualizar la detección de los puntos de referencia del ojo.

- ***if (left[0].y - left[1].y) < 0.005:*** Calcula la diferencia en las coordenadas y de los dos puntos de referencia del ojo izquierdo. Si esta diferencia es menor que un umbral específico (0.005 en este caso), se considera que el ojo ha realizado un movimiento mínimo.

- ***pyautogui.click():*** Simula un clic del mouse utilizando la biblioteca pyautogui. Esto se realiza cuando se detecta un movimiento mínimo del ojo izquierdo.

- ***pyautogui.sleep(1):*** Se introduce un breve retraso de 1 segundo después de realizar el clic del mouse. Esto puede ser útil para evitar clics múltiples no deseados si el movimiento del ojo es continuo durante un corto período de tiempo.

4.7. Activación de Grabación

En esta implementación de código nos sirve para mantener actualizado el estado de la grabación y para agregar los fotogramas grabados al archivo de video en tiempo real mientras la grabación está activa, tenemos los siguientes aparatados:

- ***current_time = time.time():*** Obtiene el tiempo actual en segundos desde el inicio del programa.

- ***elapsed_time = current_time - start_time:*** Calcula el tiempo transcurrido desde el inicio de la grabación, restando el tiempo de inicio *(start_time)* del tiempo actual *(current_time).*

- ***formatted_time = time.strftime("%H:%M:%S", time.gmtime(elapsed_time)):*** Formatea el tiempo transcurrido en horas:minutos:segundos utilizando la función strftime de la biblioteca time.

- ***lbl_recording_status.config(text=f"Grabando... Tiempo: {formatted_time}", bg="#DC143C", fg="white", font=("Arial", 12)):*** Actualiza la etiqueta de estado de grabación (lbl_recording_status) para mostrar el estado de la grabación junto con el tiempo transcurrido formateado. Cambia el texto de la etiqueta para indicar que se está grabando y muestra el tiempo transcurrido.

- ***out.write(frame):*** Escribe el fotograma actual (frame) en el objeto de salida de video (out). Esto agrega el fotograma actual al archivo de video que se está grabando.

```
# Grabación activa
        if recording_active:
            current_time = time.time()
            elapsed_time = current_time - start_time  # Calcular el
tiempo transcurrido
            formatted_time = time.strftime("%H:%M:%S",
time.gmtime(elapsed_time))  # Formatear el tiempo transcurrido
            lbl_recording_status.config(text=f"Grabando... Tiempo:
{formatted_time}", bg="#DC143C", fg="white", font=("Arial", 12))
            out.write(frame)
```

Tabla 9. Activación de Grabación.

4.8. Ventana de Actualización de Frames

En esta implementación nos ayudara a garantizar que la ventana de la interfaz gráfica se actualice con el fotograma actual de la cámara en tiempo real, proporcionando una vista en vivo de la entrada de video, los métodos que hemos utilizado son:

- ***if frame is not None:*** Verifica si se ha capturado un fotograma de la cámara. Si el fotograma es None, no se realiza ninguna actualización de la ventana.

- ***img = Image.fromarray(cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)):*** Convierte el fotograma del formato BGR de OpenCV al formato RGB compatible con la biblioteca Pillow. Se utiliza la clase Image de la biblioteca Pillow para crear un objeto de imagen a partir del fotograma.

- ***img = ImageTk.PhotoImage(image=img):*** Convierte el objeto de imagen de Pillow en un objeto de imagen de Tkinter (PhotoImage). Esto permite que la imagen sea utilizada por los widgets de la interfaz gráfica de Tkinter.

- ***label.configure(image=img):*** Configura el widget de etiqueta (label) para mostrar la imagen actualizada. El parámetro image se establece en el objeto de imagen de Tkinter (img), lo que hace que la etiqueta muestre el fotograma de la cámara.

- ***label.image = img:*** Actualiza el atributo image del widget de etiqueta (label) con el objeto de imagen de Tkinter (img). Esto asegura que la referencia al objeto de imagen se mantenga, evitando que sea eliminado por el recolector de basura de Python.

- ***label.update_idletasks():*** Actualiza la interfaz gráfica de forma inmediata, permitiendo que se muestre el fotograma actualizado en la ventana sin esperar a que se complete el ciclo de eventos de Tkinter.

```
# Actualizar la ventana con el frame actual
        if frame is not None:
            img = Image.fromarray(cv2.cvtColor(frame,
cv2.COLOR_BGR2RGB))
            img = ImageTk.PhotoImage(image=img)
            label.configure(image=img)
            label.image = img
            label.update_idletasks()
```

Tabla 10. Actualización de Frames.

4.9. Metodos de MediaPipe y FaceMesh

Preparáramos los recursos necesarios para la detección de puntos de referencia faciales para obtener las dimensiones de la pantalla del sistema, lo que facilita la interacción con el entorno visual del usuario.

- ***cam = cv2.VideoCapture(0):*** Inicializa la cámara del sistema para capturar video. El argumento 0 indica que se usará la cámara predeterminada del sistema. Si hay múltiples cámaras conectadas al sistema, se puede especificar el índice de la cámara deseada (por ejemplo, 1 para la segunda cámara).

- ***face_mesh = mp.solutions.face_mesh.FaceMesh(refine_landmarks=True):*** Inicializa la instancia de detección de puntos de referencia faciales utilizando MediaPipe Face Mesh. Esta instancia se utiliza posteriormente para detectar y seguir los puntos de referencia faciales en el fotograma de la cámara. El parámetro *refine_landmarks=True* indica que se aplicará un refinamiento adicional a los puntos de referencia detectados.

- ***screen_w, screen_h = pyautogui.size():*** Obtiene las dimensiones de la pantalla del sistema utilizando la función size() de la biblioteca pyautogui. Estas dimensiones se utilizan más adelante en el código para realizar acciones basadas en la posición en pantalla, como mover el cursor del mouse.

```
# Inicializar la cámara y MediaPipe Face Mesh
cam = cv2.VideoCapture(0)
face_mesh = mp.solutions.face_mesh.FaceMesh(refine_landmarks=True)
screen_w, screen_h = pyautogui.size()
```

Tabla 11. Método MediaPipe y Face Mesh.

4.10. Variables de Control

En estas variables de control son esenciales para el funcionamiento del programa, ya que controlan el estado de la detección de puntos de referencia faciales, la grabación de video y almacenan información importante como el objeto VideoWriter y el tiempo de inicio de la grabación.

```
# Variables de control
detection_active = False  # Iniciar con la detección desactivada
recording_active = False  # No grabar hasta que se indique
out = None  # Inicializar el objeto VideoWriter como None
start_time = 0  # Inicializar el tiempo de inicio de grabación
```

Tabla 12. Variables de Control.

4.11. Obtención de Fotogramas de Cámara

Verificamos la configuración de la cámara y obtenemos la información sobre la tasa de fotogramas que está proporcionando. Es útil para ajustar los parámetros del programa o para fines de diagnóstico si hay problemas con la captura de video.

- ***fps = cam.get(cv2.CAP_PROP_FPS):*** Se utiliza el método get() del objeto VideoCapture (cam) para obtener la propiedad CAP_PROP_FPS, que representa la tasa de fotogramas por segundo de la cámara. Esta propiedad devuelve la tasa de fotogramas actual configurada en la cámara.

- ***print(f"Tasa de fotogramas de la cámara: {fps}"):*** Se imprime la tasa de fotogramas de la cámara en la consola utilizando una cadena formateada (f-string). Esta línea muestra la tasa de fotogramas obtenida en la consola para que el usuario pueda conocerla.

```
# Obtener la tasa de fotogramas de la cámara
fps = cam.get(cv2.CAP_PROP_FPS)
print(f"Tasa de fotogramas de la cámara: {fps}")
```

Tabla 13. Tasa de Fotogramas.

4.12. Título de la Ventana de la Interfaz Gráfica

Es el punto de partida para construir la interfaz gráfica de la aplicación. Tenemos lo siguiente:

- ***root = tk.Tk():*** Crea una nueva instancia de la clase Tk() de la biblioteca Tkinter y la asigna a la variable root. Esta instancia representa la ventana principal de la aplicación GUI. Todas las demás ventanas, marcos y widgets se agregan a esta ventana principal.

- ***root.title("Eye Controlled Mouse"):*** Establece el título de la ventana principal como "Eye Controlled Mouse". Este título se mostrará en la barra de título de la ventana GUI y proporciona una etiqueta descriptiva para la aplicación.

```
# Crear la ventana de la interfaz gráfica
root = tk.Tk()
root.title("Eye Controlled Mouse")
```

Tabla 14. Ventana de Interfaz Gráfica.

4.13. Dimensión de los Botones

En esta línea de código es proporcionar una manera fácil de ajustar y mantener la consistencia en el tamaño de los botones en la interfaz gráfica de usuario.

```
button_width = 20
```

Tabla 15. Dimensión de los Botones.

4.14. Configuración de Estilo de Letra

El propósito de esta linea de código es proporcionar una manera conveniente de definir y aplicar un estilo uniforme a todos los botones en la interfaz gráfica de usuario.

```
btn_style = {"font": ("Arial", 12), "width": button_width}
```

Tabla 16. Estilo de Letra de los Botones.

4.15. Botones de Detección – Grabacion y Salidad del Programa

Implementamos la sección de botones en donde se empaquetan en la ventana principal *(root)* utilizando el método *pack()* con diferentes opciones de diseño (*side=tk.TOP* para alinearlos en la parte superior y *pady=5* para agregar un espacio de relleno vertical). Esto los coloca correctamente en la interfaz gráfica para que el usuario pueda interactuar con ellos fácilmente.

```
# Botones para alternar detección, grabación y salir del programa
btn_toggle_detection = tk.Button(root, text="Iniciar Detección (D)",
command=toggle_detection, bg="#32CD32", fg="white", **btn_style)
btn_toggle_detection.pack(side=tk.TOP, pady=5)

btn_toggle_recording = tk.Button(root, text="Iniciar Grabación (R)",
command=toggle_recording, bg="#32CD32", fg="white", **btn_style)
btn_toggle_recording.pack(side=tk.TOP, pady=5)

btn_quit = tk.Button(root, text="Salir (Q)", command=quit_program,
bg="#FF6347", fg="white", **btn_style)
btn_quit.pack(side=tk.TOP, pady=5)
```

Tabla 17. Botones de la Interfaz Gráfica.

4.16. Etiqueta de la Cámara

En esta sección de código nos ayuda a mostrar información actual de la aplicación como mensajes de error o resultados de operaciones.

```
label = tk.Label(root)
label.pack()
```

Tabla 18. Etiqueta de la Cámara.

4.17. Etiqueta del estado de la Grabación

Creamos una etiqueta en la interfaz gráfica que mostrará el estado de la grabación, en el que la etiqueta inicialmente está vacía (text=""), pero al actualizarse nos mostrara el mensaje sobre el estado de la grabación el cual será el de "Grabando..." y "Grabación terminada". La etiqueta se coloca en la parte inferior de la ventana para que sea fácilmente visible y no interfiera con otros elementos de la interfaz.

- ***lbl_recording_status = tk.Label(root, text="", bg="white", font=("Arial", 12)):*** Crea un objeto de etiqueta utilizando la clase Label de Tkinter. La etiqueta se creará dentro de la ventana principal (root) de la aplicación GUI. Se especifican los siguientes parámetros:
- ***text="":*** Define el texto inicial de la etiqueta como una cadena vacía. Esto significa que inicialmente no se mostrará ningún texto en la etiqueta.
- ***bg="white":*** Establece el color de fondo de la etiqueta en blanco.
- ***font=("Arial", 12):*** Especifica el tipo de fuente y el tamaño del texto de la etiqueta. En este caso, se usa la fuente Arial con un tamaño de 12 puntos.
- ***lbl_recording_status.pack(side=tk.BOTTOM, pady=5):*** Coloca la etiqueta en la ventana principal utilizando el método pack(). La etiqueta se empaqueta en la parte inferior de la ventana (side=tk.BOTTOM) con un espacio de relleno vertical de 5 píxeles (pady=5).

```
lbl_recording_status = tk.Label(root, text="", bg="white",
font=("Arial", 12))
```

```
lbl_recording_status.pack(side=tk.BOTTOM, pady=5)
```

Tabla 19. Estado de la Grabación.

4.18. Teclas de los Eventos

En esta sección se implementa enlaces de teclas proporcionan una forma conveniente para que el usuario interactúe con la aplicación mediante el teclado.

El usuario puede iniciar o detener la detección con la tecla “D” , iniciar o detener la grabación con solo presionar la tecla “R” y salir del programa rápidamente presionando la tecla “Q”.

```
root.bind('<KeyPress-d>', toggle_detection)
root.bind('<KeyPress-r>', toggle_recording)
root.bind('<KeyPress-q>', quit_program)
```

Tabla 20. Eventos por Teclado.

4.19. Bucle para la Detección de la Cámara

Ejecutamos la función opencv_loop() en un hilo separado para evitar bloquear el hilo principal de la GUI mientras se procesa la entrada de la cámara, permitiéndonos que la aplicación continúe siendo receptiva y que el usuario pueda interactuar con la interfaz mientras se procesa la entrada de la cámara en segundo plano.

- ***target=opencv_loop:*** Este argumento especifica la función que se ejecutará en el nuevo hilo. En este caso, la función opencv_loop() será la tarea principal que se realizará en el hilo separado.
- ***daemon=True:*** Este argumento especifica que el hilo creado será un hilo que se ejecutan en segundo plano y se detendra automáticamente cuando todos los demás hilos no han terminado.

```
thread = threading.Thread(target=opencv_loop, daemon=True)
thread.start()
```

Tabla 21. Bucle para la Entrada de la Cámara.

4.20. Loop de la Interfaz Grafica

Al crear el objeto root del método mainloop nos ayudara a iniciar la interfaz gráfica de la aplicación y mantenerla en funcionamiento, permitiendo la interacción del usuario y la actualización de la interfaz según los eventos que ocurren.

```
root.mainloop()
```

Tabla 22. Loop de la Interfaz Gráfica.

5. Conclusiones

- En el desarrollo del aplicativo se puede concluir que en la implementación del programa hacemos uso de la biblioteca Tkinter para crear una interfaz gráfica de usuario (GUI) y OpenCV junto con MediaPipe para el procesamiento de entrada de imágenes detectadas por la cámara, consiguiendo una interacción intuitiva con la aplicación ejecutando en su desarrollo la detección de los movimientos faciales y los parámetros del ojo, como el parpadeo o el movimiento permitiendo al usuario poder controlar el programa de manera natural y sin la necesidad de dispositivos adicionales.

- La creación de un sistema de control de mouse con seguimiento ocular exitoso es un gran avance en cómo interactuamos con las computadoras. Básicamente, funciona usando cámaras para seguir nuestros ojos, lo que nos permite mover el cursor y hacer clic solo con movimientos oculares simples, como parpadear o mirar en diferentes direcciones.

- La implementación exitosa de un sistema de reconocimiento en tiempo real implica superar desafíos técnicos, en nuestro caso fue el de las versiones tanto del entorno de desarrollo de python como de la librería de mediapipe, ya que esta librería no funciona en versiones actuales de python(la última versión actual de python se encuentra en la 3.12.2), además se debe tener en cuenta la gestión eficiente de datos en tiempo real, la optimización de algoritmos y la integración con hardware específico, en este último especialmente ya que a mejor hardware de captura video, mejor será la detección a tiempo real.

- La aplicación de reconocimiento en tiempo real tiene aplicaciones prácticas en áreas como vigilancia y seguridad, análisis de comportamiento humano, interacción natural en interfaces de usuario, entretenimiento y más. Estas aplicaciones pueden generar un impacto tangible en la eficiencia y la toma de decisiones.

- El desarrollo de esta aplicación demuestra un avance en la integración de tecnologías para mejorar la accesibilidad y la experiencia del usuario. Donde al aprovechar la detención de pupilas y el seguimiento ocular con OpenCV y MediaPie, permite el control preciso del cursor del ratón y la realización de hacer acciones sin la necesidad de dispositivos físicos adicionales. Esta iniciativa destaca la importancia de la invocación tecnológica en la mejora de la calidad de vida y la inclusión digital de todos los usuarios.

6. Recomendaciones

- Antes de comenzar el desarrollo, es crucial realizar una investigación exhaustiva sobre las tecnologías de reconocimiento facial y ocular disponibles, así como sobre las bibliotecas y frameworks que mejor se adapten a las necesidades del

proyecto con ello podremos entender las capacidades, limitaciones y mejores prácticas asociadas con estas tecnologías.

- Con el pasar del tiempo y el avance de las nuevas tecnologías, es de suma importancia optimizar el programa constantemente, tanto en velocidad de ejecución como en la precisión del reconocimiento ocular. Además, se necesita probarlo en diferentes situaciones de luz y con diferentes personas para asegurarse de que funcione bien en todas partes.

- Dado que el reconocimiento en tiempo real implica el procesamiento de datos en tiempo real, es fundamental implementar medidas sólidas de seguridad y privacidad. Garantizar el manejo adecuado de datos sensibles y el cumplimiento de regulaciones de privacidad es esencial.

- Si el proyecto tiene la posibilidad de expandirse a diferentes plataformas o entornos, es importante diseñar la arquitectura con la escalabilidad en mente. Considerar el rendimiento y los requisitos de recursos a medida que el sistema crece.

- Algo importante en este proyecto es que se deben realizar pruebas exhaustivas de usabilidad y accesibilidad, lo que ayudaría a identificar posibles áreas de mejora y garantizar que la aplicación sea verdaderamente más fácil de usar para el público objetivo.

7. Bibliografía

Flores, F. (22 de Julio de 2022). *Qué es Visual Studio Code y qué ventajas ofrece*. Obtenido de OpenWebinars: https://openwebinars.net/blog/que-es-visual-studio-code-y-que-ventajas-ofrece/#:~:text=Visual%20Studio%20Code%20(VS%20Code,%2C%20GNU%2FLinux%20y%20macOS.

Google. (s.f.). *Colaboratory*. Obtenido de Preguntas Frecuentes: https://research.google.com/colaboratory/intl/es/faq.html

JetBrains. (2 de Junio de 2021). *PyCharm*. Obtenido de https://www.jetbrains.com/es-es/pycharm/

Martín, M. (2024). *¿Cuáles son las ventajas de utilizar Python como lenguaje de programación?* Obtenido de Domestika: https://www.domestika.org/es/blog/12499-cuales-son-las-ventajas-de-utilizar-python-como-lenguaje-de-programacion

MediaPipe Solutions. (s.f.). Obtenido de Google for Developers. : https://developers.google.com/mediapipe/solutions/guide

Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., ... & Grundmann, M. (2019, June). *Mediapipe: A framework for perceiving and processing reality. In Third workshop on computer vision for AR/VR at IEEE computer vision and pattern recognition* (CVPR) (Vol. 2019).Obtenido de Google: https://github.com/google/mediapipe

Caballero Barriga, E. R. (2017). *Aplicación práctica de la visión artificial para el reconocimiento de rostros en una imagen, utilizando redes neuronales y algoritmos de reconocimiento de objetos de la biblioteca opencv*.Obtenido de: https://repository.udistrital.edu.co/bitstream/handle/11349/6104/CaballeroBarrigaEdisonRene2017.pdf;jsessionid=2E0ADEC500E835B467DD197155F7C589?sequence=1

8. Anexos

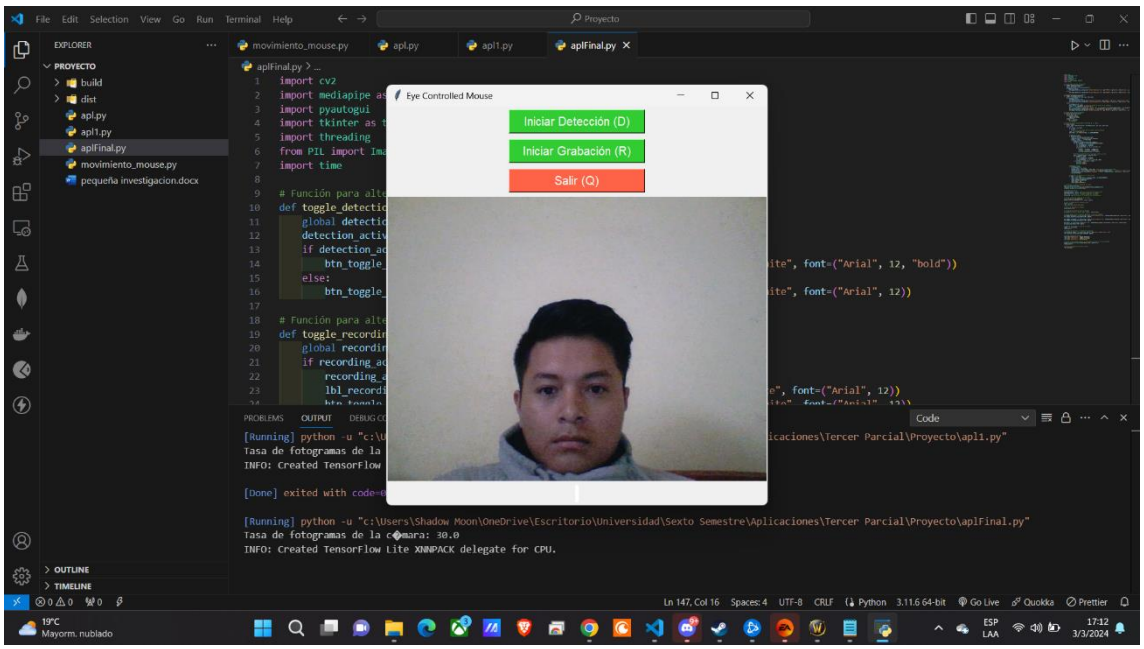

Ilustración 1 Ejecución del Programa.

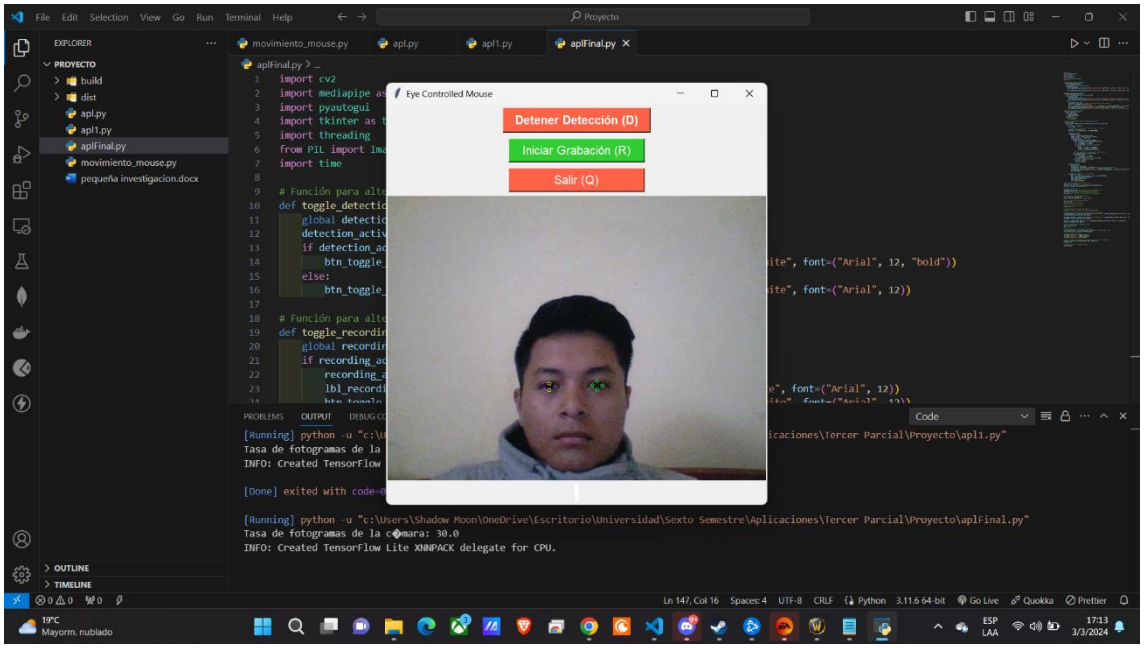

Ilustración 2. Detección de Rostro y Ojos.

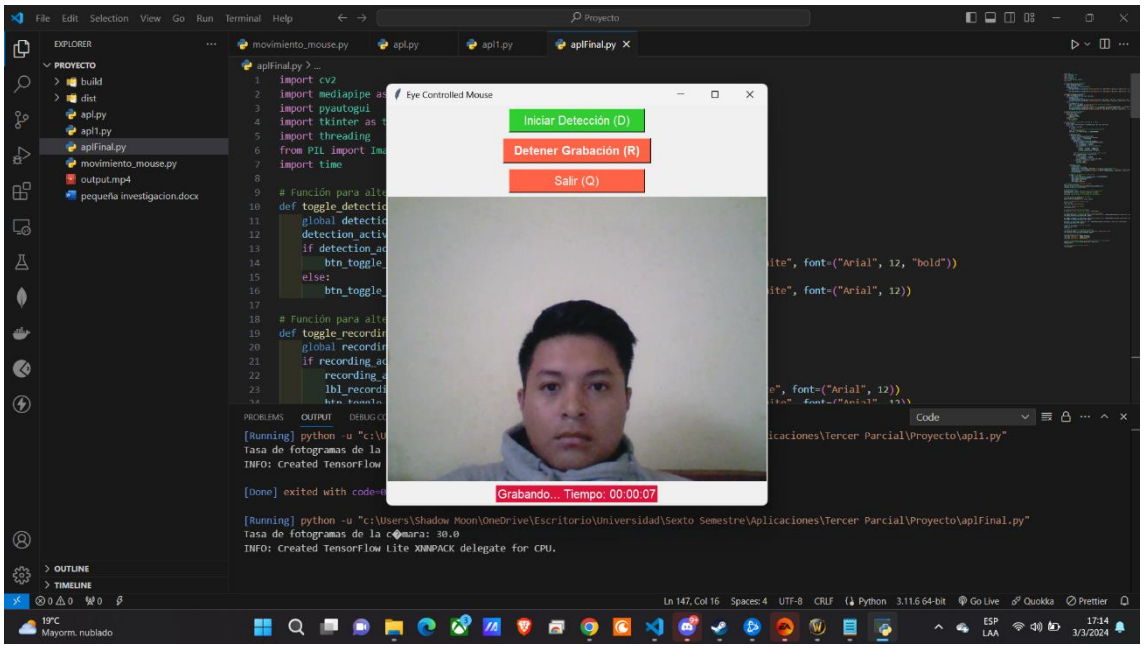

Ilustración 3. Inicio de Grabación.

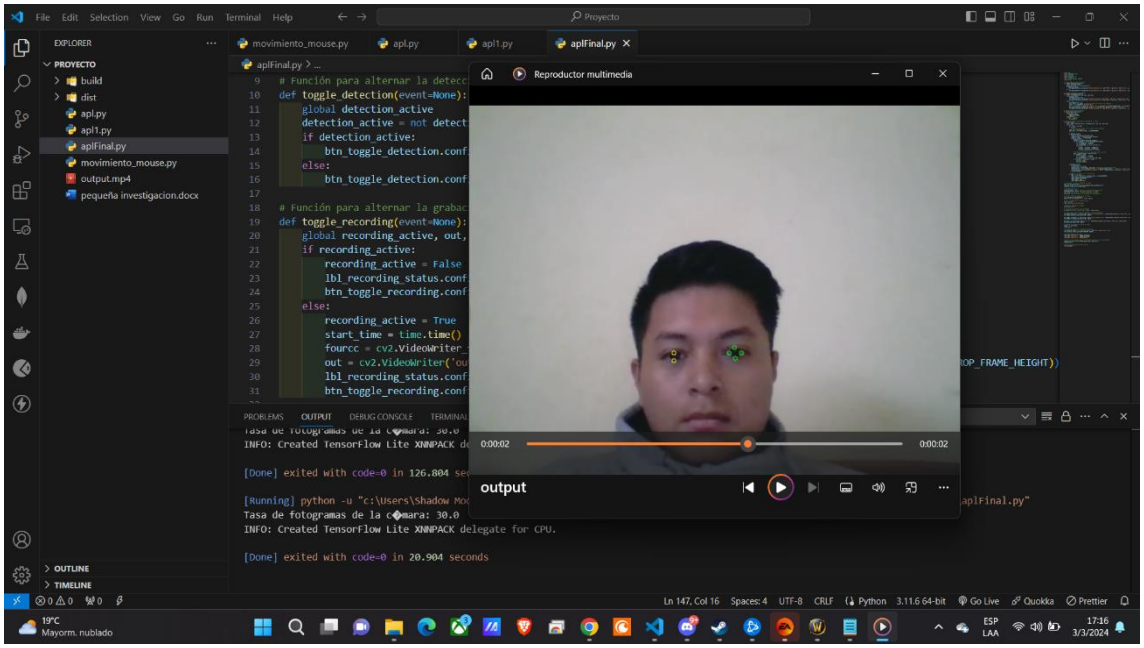

Ilustración 4. Archivo mp4 del video filmado.