**UNIVERSIDAD TECNOLÓGICA BOLIVIANA**

**INGENIERÍA DE SISTEMAS**



**Propuesta de un Sistema de Reconocimiento Facial para la Validación de Usuarios de Tarjetas Preferenciales en el Transporte Público**

**Estudiantes:**

* Fabio Camacho Encinas
* Juan Carlos Marca Alvarez
* Fátima Lucia Rocha Rospigliossi

**Materia:** Inteligencia Artificial

**LA PAZ – BOLIVIA**

**2024**

[1. Introducción 1](#_Toc184195574)

[2. Planteamiento del problema 2](#_Toc184195575)

[3. Objetivo general 2](#_Toc184195576)

[4. Objetivos específicos 2](#_Toc184195577)

[5. Alcance 2](#_Toc184195578)

[6. Límites 2](#_Toc184195579)

[7. Marco teórico 3](#_Toc184195580)

[7.1. Visión artificial 3](#_Toc184195581)

[7.2. Conceptos Básicos en Visión Artificial 3](#_Toc184195582)

[7.3. Componentes de un Sistema de Visión Artificia 4](#_Toc184195583)

[7.4. Aplicaciones de la Visión Artificial 5](#_Toc184195584)

[8. Desarrollo 6](#_Toc184195585)

[8.1. Librerías 6](#_Toc184195586)

[8.2. Funciones 7](#_Toc184195587)

[8.2.1. Función encender\_webcam 7](#_Toc184195588)

[8.2.2. Función apagar\_webcam 8](#_Toc184195589)

[8.2.3. Función guardar\_captura 10](#_Toc184195590)

[8.2.4. Función comparar\_captura 11](#_Toc184195591)

[8.2.5. Función iniciar\_reconocimiento 14](#_Toc184195592)

[8.2.6. Función comparar\_en\_tiempo\_real 15](#_Toc184195593)

[8.2.7. Función detener\_reconocimiento 17](#_Toc184195594)

[8.2.8. Función actualizar\_frame: 19](#_Toc184195595)

[9. Recomendaciones 21](#_Toc184195596)

[10. Conclusiones 22](#_Toc184195597)

Propuesta de un Sistema de Reconocimiento Facial para la Validación de Usuarios de Tarjetas Preferenciales en el Transporte Público

# Introducción

En un mundo donde la movilidad urbana se ha convertido en un factor esencial para la calidad de vida, el transporte público debe ser accesible y equitativo para todos. Las tarifas preferenciales son un claro ejemplo de políticas inclusivas que buscan beneficiar a sectores vulnerables como las personas mayores, estudiantes y personas con discapacidad. Sin embargo, estas iniciativas, diseñadas con un propósito noble, se enfrentan a desafíos significativos derivados del abuso y la suplantación de identidad. Este problema no solo impacta las finanzas de los sistemas de transporte, sino que también erosiona la confianza de los usuarios legítimos en el sistema.

El avance de la tecnología ha abierto puertas a soluciones innovadoras que podrían transformar la gestión de estos beneficios. Entre ellas, el reconocimiento facial destaca como una herramienta poderosa y confiable. Esta tecnología biométrica, que permite identificar de manera única a las personas mediante el análisis de sus rasgos faciales, representa un cambio paradigmático en la forma en que se autentica la identidad en contextos cotidianos. Su aplicación en el transporte público no solo garantiza la seguridad y la eficiencia, sino que también redefine la experiencia del usuario, eliminando la necesidad de controles manuales engorrosos.

El presente proyecto propone un sistema de reconocimiento facial como solución a la problemática de la suplantación de identidad en el uso de tarjetas preferenciales. Mediante el desarrollo de un software accesible y funcional, basado en tecnologías de código abierto como Python, OpenCV y Mediapipe, se busca implementar un mecanismo que valide automáticamente la identidad del usuario en tiempo real. Este sistema no solo refuerza la integridad del programa de tarifas preferenciales, sino que también contribuye a la optimización operativa de los servicios de transporte.

En esencia, este proyecto se erige como un puente entre la innovación tecnológica y la responsabilidad social, demostrando cómo el reconocimiento facial puede ser una herramienta clave para garantizar la equidad en el acceso a recursos públicos. Más allá de un simple sistema, esta iniciativa plantea un modelo replicable para la modernización de políticas inclusivas, posicionándose como un ejemplo de cómo la tecnología puede ser un motor para el cambio positivo.

# Planteamiento del problema

Los sistemas de transporte público con tarifas preferenciales enfrentan problemas de suplantación de identidad y uso indebido de las tarjetas destinadas a grupos específicos de usuarios. Esto resulta en pérdidas económicas y una desventaja para los usuarios que realmente necesitan el servicio. Los métodos tradicionales de control, como revisiones manuales, son ineficaces y poco prácticos en escenarios de alta demanda. Por lo tanto, es necesario un sistema automatizado que autentique la identidad de los usuarios preferenciales de manera rápida y precisa para evitar abusos.

# Objetivo general

Desarrollar un sistema de reconocimiento facial para validar la identidad de los usuarios preferenciales en los servicios de transporte público, garantizando que solo las personas autorizadas utilicen las tarjetas preferenciales.

# Objetivos específicos

* Diseñar un módulo de reconocimiento facial en tiempo real que valide la identidad del usuario.
* Proveer una interfaz gráfica que facilite el uso del sistema para operadores y administradores del transporte.
* Garantizar la precisión del sistema mediante técnicas de procesamiento de imágenes que minimicen errores en la autenticación.
* Demostrar la capacidad del sistema para prevenir suplantaciones mediante pruebas en tiempo real.

# Alcance

* Implementará un reconocimiento facial en tiempo real utilizando técnicas accesibles con Python y bibliotecas como OpenCV y Mediapipe.
* Funcionará como una herramienta demostrativa sin depender de bases de datos para almacenamiento o recuperación de registros.
* Proveerá alertas en caso de detección de discrepancias, permitiendo a los operadores tomar medidas inmediatas.

# Límites

* El sistema no integrará un manejo de bases de datos para almacenamiento masivo o gestión de múltiples usuarios.
* La precisión del reconocimiento puede verse afectada en condiciones adversas, como iluminación insuficiente o ángulos de captura desfavorables.
* No se considerarán métodos avanzados de aprendizaje profundo para el reconocimiento facial; se utilizarán técnicas básicas accesibles con herramientas de Python.
* No es adecuado para aplicaciones que requieran alta precisión o tolerancia a condiciones adversas de iluminación y ángulos.
* El módulo está diseñado únicamente como una demostración funcional y no es apto para su implementación directa en redes de transporte a gran escala.
* El módulo está diseñado únicamente como una demostración funcional y no es apto para su implementación directa en redes de transporte a gran escala.

# Marco teórico

# Visión artificial

La visión artificial o visión por computadora es un campo de la inteligencia artificial que se encarga de desarrollar sistemas capaces de interpretar y comprender imágenes y videos para realizar tareas que, hasta ahora, solo los humanos podían llevar a cabo con su visión. La visión artificial permite que las computadoras "vean" y "entiendan" el mundo visual que las rodea, utilizando datos visuales provenientes de imágenes o secuencias de video. Esta disciplina abarca un conjunto de métodos y tecnologías que buscan emular o incluso superar las capacidades visuales humanas en ciertas tareas específicas, como la clasificación, detección y segmentación de objetos.

El objetivo principal de la visión artificial es permitir que las máquinas puedan interpretar y extraer información significativa de las imágenes, realizar tareas como el reconocimiento de patrones y tomar decisiones en función de esos datos visuales. En este marco, se combinan conocimientos de diferentes áreas, como la computación, el procesamiento de señales, la teoría de imágenes y el aprendizaje automático.

# Conceptos Básicos en Visión Artificial

Para comprender la visión artificial, es necesario familiarizarse con algunos conceptos clave:

* **Imagen Digital**: Una imagen digital es una representación bidimensional de información visual compuesta por píxeles, que son unidades discretas de información visual (cada píxel tiene un valor de color o intensidad).
* **Procesamiento de Imágenes**: Consiste en aplicar una serie de técnicas matemáticas y computacionales para transformar, analizar y extraer información útil de una imagen digital. Algunos procesos comunes incluyen el filtrado, la mejora de contrastes, la detección de bordes y la segmentación.
* **Segmentación**: La segmentación de una imagen es el proceso de dividir una imagen en regiones o partes de interés. En visión artificial, se busca separar los objetos dentro de una escena para facilitar su análisis.
* **Características Visuales**: Las características visuales son atributos importantes de una imagen que permiten identificar patrones o objetos. Algunas características comunes incluyen el color, la textura, la forma y el tamaño.
* **Reconocimiento de Patrones**: Es el proceso mediante el cual un sistema de visión artificial identifica patrones o estructuras en las imágenes para clasificarlas, reconocer objetos, personas o acciones específicas.
* **Modelos de Aprendizaje Automático**: En la visión artificial moderna, se utilizan modelos de machine learning (aprendizaje automático) y Deep learning (aprendizaje profundo), particularmente redes neuronales convolucionales (CNNs), para enseñar a las máquinas a reconocer patrones y realizar tareas complejas basadas en grandes volúmenes de datos visuales.

# Componentes de un Sistema de Visión Artificia

Un sistema de visión artificial consta de varios componentes fundamentales que interactúan entre sí para lograr una interpretación efectiva de las imágenes:

* **Captura de Imágenes**: El primer paso en cualquier sistema de visión artificial es la adquisición de imágenes. Esto se realiza mediante cámaras o sensores que convierten la información visual en datos digitales que pueden ser procesados por el sistema.
* **Preprocesamiento de Imágenes**: Las imágenes adquiridas suelen estar afectadas por ruido, distorsiones o malas condiciones de iluminación. El preprocesamiento mejora la calidad de las imágenes mediante técnicas como el filtrado, la corrección de color y la normalización.
* **Segmentación de Imágenes**: Una vez preprocesada la imagen, se realiza la segmentación, es decir, el proceso de dividir la imagen en regiones más pequeñas y manejables. Esto facilita la identificación de objetos o áreas de interés, como el fondo y los elementos que se desean analizar.
* **Extracción de Características**: La extracción de características es el proceso de identificar elementos clave dentro de una imagen (bordes, texturas, formas, etc.). Las características extraídas sirven como la base para la clasificación de objetos, el reconocimiento de patrones y otras tareas.
* **Reconocimiento y Clasificación**: Una vez que las características relevantes se han extraído de la imagen, los algoritmos de clasificación, como las redes neuronales profundas, se utilizan para identificar o clasificar los objetos presentes en la imagen. Esto permite asignar una etiqueta o categoría a los objetos identificados.
* **Toma de Decisiones y Acción**: En algunos casos, el sistema de visión artificial no solo tiene que identificar los objetos, sino también tomar decisiones basadas en esa información. Por ejemplo, en sistemas de conducción autónoma, el sistema debe identificar obstáculos y decidir cómo actuar (frenar, desviar, etc.).

# Aplicaciones de la Visión Artificial

La visión artificial tiene un amplio espectro de aplicaciones en diversos campos. Algunas de las más destacadas son:

* **Reconocimiento Facial**: Se utiliza en sistemas de seguridad y verificación de identidad, como en el desbloqueo de dispositivos móviles y la autenticación en entornos bancarios.
* **Vehículos Autónomos**: Los coches autónomos utilizan visión artificial para reconocer señales de tráfico, detectar obstáculos y entender el entorno de conducción.
* **Medicina**: En la medicina, la visión artificial se emplea para analizar imágenes médicas, como radiografías y resonancias magnéticas, para ayudar en el diagnóstico de enfermedades y la planificación de tratamientos.
* **Robótica**: Los robots utilizan visión artificial para percibir su entorno y realizar tareas como la manipulación de objetos, la navegación y la interacción con seres humanos.
* **Inspección Industrial**: En el ámbito industrial, los sistemas de visión artificial se emplean para detectar defectos en productos fabricados, asegurando la calidad en la producción.
* **Reconocimiento de Texto**: La visión artificial también se usa en el reconocimiento óptico de sus principales caracteres para convertir texto impreso en texto digital, lo que facilita la digitalización de documentos.

# Desarrollo

# Librerías



* **Cv2 (OpenCV):**

Cv2 es el módulo de *OpenCV* para Python. *OpenCV* es una librería de computación visual para el procesamiento de imágenes en Python. Esta biblioteca proporciona herramientas para realizar operaciones de procesamiento de imágenes, como el filtrado, la detección de bordes, el reconocimiento de características, el seguimiento de objetos, etc. Estas herramientas nos permiten desarrollar aplicaciones de visión artificial, como el reconocimiento facial, el seguimiento de objetos, etc.

* **Mediapipe:**

Es un entorno de código abierto para construir aplicaciones que realicen inferencia de visión por computadora sobre datos sensoriales como video, imágenes o audio.

Fue creado para equipos de inteligencia artificial, aprendizaje automático y desarrolladores de software con aplicaciones listas para producción o prototipos como parte de trabajos de investigación.

El entorno *Mediapipe* permite implementar soluciones para detección facial, manos, gestos, iris, etc.

* **TKinter:**

Es una biblioteca de lenguaje de programación Python que se utiliza para la creación y desarrollo de aplicaciones de escritorio. Esta librería proporciona soluciones para el posicionamiento de interfaces graficas de usuario en aplicaciones de escritorio.

* **PIL (*Pillow*):**

Es una biblioteca de procesamiento de imágenes en Python que proporciona funcionalidades para abrir, manipular y guardar imágenes en varios formatos. Permite realizar tareas como:

* Cargar imágenes desde archivos.
* Modificar el tamaño y la resolución de una imagen.
* Aplicar filtros y efectos a las imágenes.
* Realizar operaciones de recorte y rotación.
* Guardar imágenes en diferentes formatos, como JPEG, PNG, BMP, etc.
* **Image:**

El módulo de *Image* proporciona una clase con el mismo nombre que se utiliza para representar una imagen PIL. El módulo también proporciona una serie de funciones de fábrica, incluidas funciones para cargar imágenes desde archivos y crear nuevas imágenes.

* **Numpy:**

Es una librería de Python especializada en el cálculo numérico y el análisis de datos, especialmente para un gran volumen de datos.

Incorpora una nueva clase de objetos llamados *arrays* que permite representar colecciones de datos de un mismo tipo en varias dimensiones, y funciones muy eficientes para su manipulación.

La ventaja de *Numpy* frente a las listas predefinidas en Python es que el procesamiento de los *arrays* se realiza mucho más rápido (hasta 50 veces más) que las listas, lo cual la hace ideal para el procesamiento de vectores y matrices de grandes dimensiones.

# Variables



**MediaPipe Face Mesh y utilidades**

1. **mp\_face\_mesh**
   * **Tipo:** Objeto de *MediaPipe* *Face* *Mesh*.
   * **Descripción:**  
     Inicializa el modelo de *MediaPipe* para la detección y seguimiento de puntos faciales.  
     Configuración de parámetros:
     + *min\_detection\_confidence*: Nivel mínimo de confianza para detectar un rostro (0.2 significa que al menos el 20% de probabilidad es necesario para considerarlo válido).
     + *min\_tracking\_confidence*: Nivel mínimo de confianza para rastrear puntos faciales tras la detección inicial.
     + El objeto permite procesar imágenes y extraer coordenadas faciales en tiempo real.
2. **mp\_drawing**
   * **Tipo:** Utilidad de *MediaPipe* *Drawing*.
   * **Descripción:**  
     Herramienta que facilita la visualización de puntos clave y conexiones faciales en imágenes o videos procesados.
     + Proporciona funciones predefinidas para dibujar puntos y mallas en imágenes, lo que es útil para depuración o demostración.

**Variables globales**

1. **cap**
   * **Tipo:** Objeto de *OpenCV* *VideoCapture*.
   * **Descripción:**  
     Representa la conexión con la cámara web o dispositivo de captura de video.
     + Inicia con un valor *None* hasta que se llame a la función *encender*\_*webcam* para inicializarlo.
     + Se usa para capturar cuadros (fotogramas) del flujo de video en tiempo real.
2. **captura\_guardada**
   * **Tipo:** Variable de imagen (*ndarray* de *NumPy* o similar).
   * **Descripción:**  
     Almacena una captura de pantalla o un cuadro procesado del flujo de video para referencia posterior.
     + Se utiliza como punto de comparación en el proceso de reconocimiento facial.
3. **reconocimiento\_activo**
   * **Tipo:** Booleano (bool).
   * **Descripción:**  
     Indica si el reconocimiento facial en tiempo real está activado o no.
     + Valor inicial: False.
     + Cambia a True cuando se llama a la función *iniciar*\_*reconocimiento* y a False al detener el proceso mediante la función *detener*\_*reconocimiento*.
4. **puntos\_clave\_guardados**
   * **Tipo:** Lista de tuplas (*list* *de* *tuple*).
   * **Descripción:**  
     Almacena las coordenadas tridimensionales (x, y, z) de los puntos clave faciales extraídos de la captura guardada.
     + Estos puntos clave se usan para comparar similitudes con otros cuadros en el flujo de video.
     + Se obtiene mediante la función *obtener*\_*puntos*\_*clave*.

**Puntos clave críticos**

1. **PUNTOS\_CRITICOS**
   * **Tipo:** Lista de enteros (*list* de *int*).
   * **Descripción:**  
     Conjunto de índices que representan puntos clave específicos de la malla facial generada por *MediaPipe*.
     + Incluye puntos importantes de las siguientes áreas:
       - **Ojo izquierdo:** Índices del 33 al 42.
       - **Ojo derecho:** Índices del 43 al 52.
       - **Nariz:** Índices del 1 al 5.
       - **Boca:** Índices del 61 al 72.
       - **Contorno del rostro:** Índices del 0, 17 al 23.
     + Estos puntos son cruciales para reducir la complejidad del procesamiento al enfocar solo en áreas relevantes del rostro.

# Interfaz gráfica de usuario

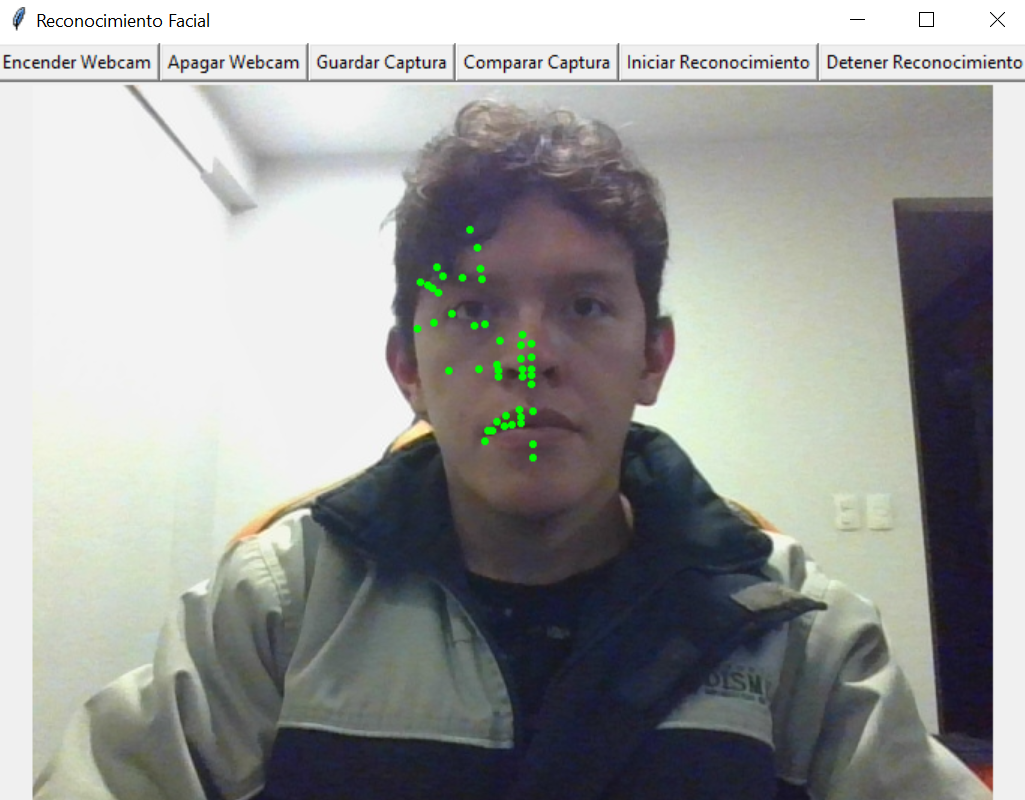


1. **Ventana principal**

* **Título:**
  + La ventana principal lleva el título "Reconocimiento Facial" por medio al método *root.title().*
* **Tamaño:**
  + No se especifica un tamaño fijo, pero los elementos de la ventana (botones y video) se ajustan automáticamente al tamaño del contenido.

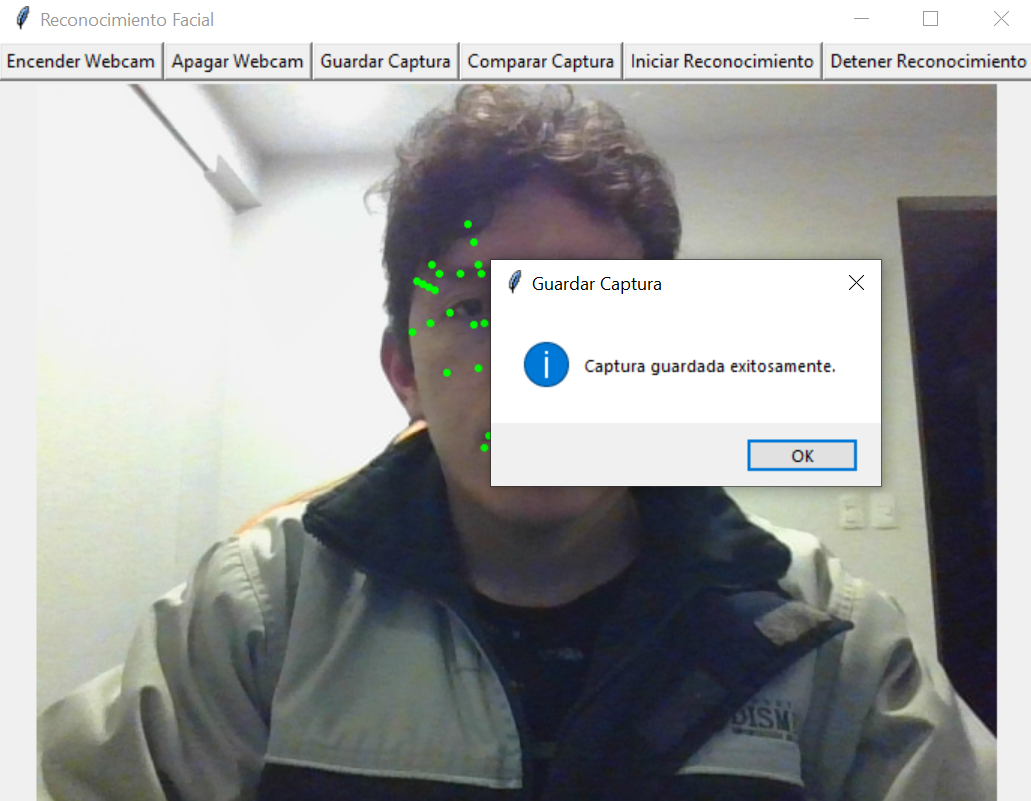
1. **Sección de botones**

* **Frame de botones (*frame*\_*botones*):**
  + Ubicado en la parte superior de la ventana mediante *pack(side=tk.TOP, fill=tk.X).*
  + Contiene una fila horizontal de botones, cada uno con una funcionalidad específica.
* **Botones dentro del frame:**
  + Los botones están dispuestos horizontalmente con *pack(side=tk.LEFT, fill=tk.X, expand=True),* lo que permite que cada botón se ajuste proporcionalmente al ancho del frame.
  + **Lista de botones y sus funciones:**
    - **Encender Webcam:**
      * Texto: "Encender Webcam".
      * Acción: Llama a la función *encender*\_*webcam* para activar la cámara (Figura 1).



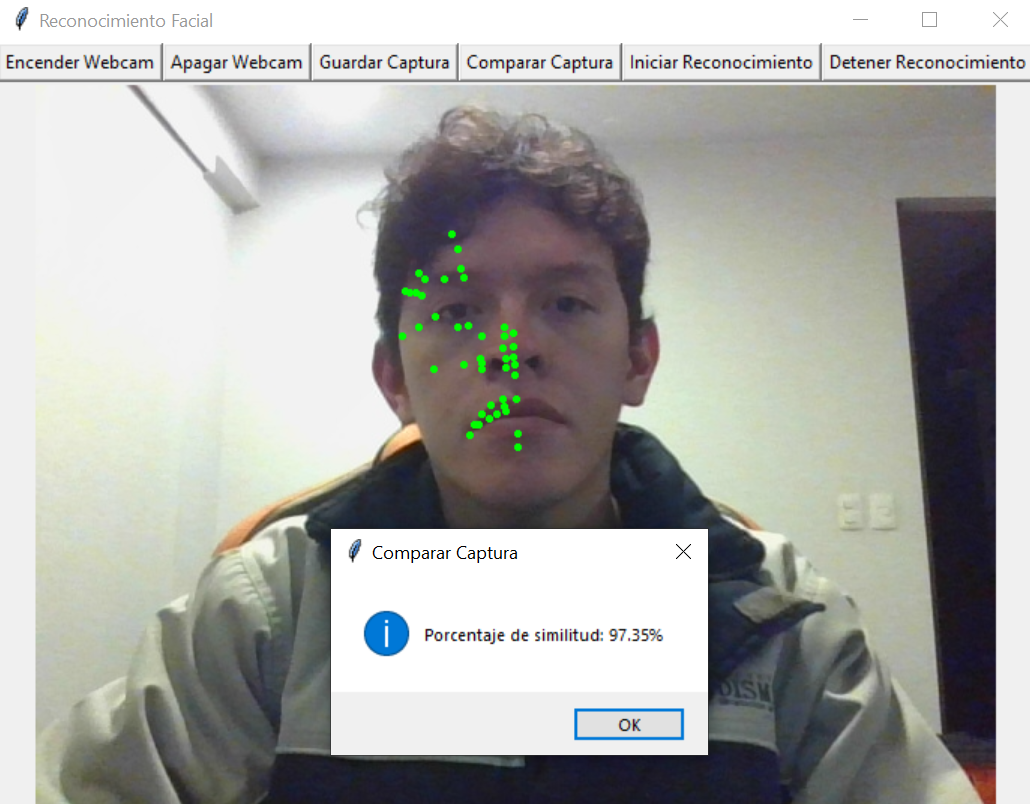
*Figura 1. Encender webcam*

* + - **Apagar Webcam:**
      * Texto: "Apagar Webcam".
      * Acción: Llama a la función *apagar*\_*webcam* para detener la cámara.
    - **Guardar Captura:**
      * Texto: "Guardar Captura".
      * Acción: Llama a la función *guardar*\_*captura* para capturar y guardar la imagen actual (Figura 2).



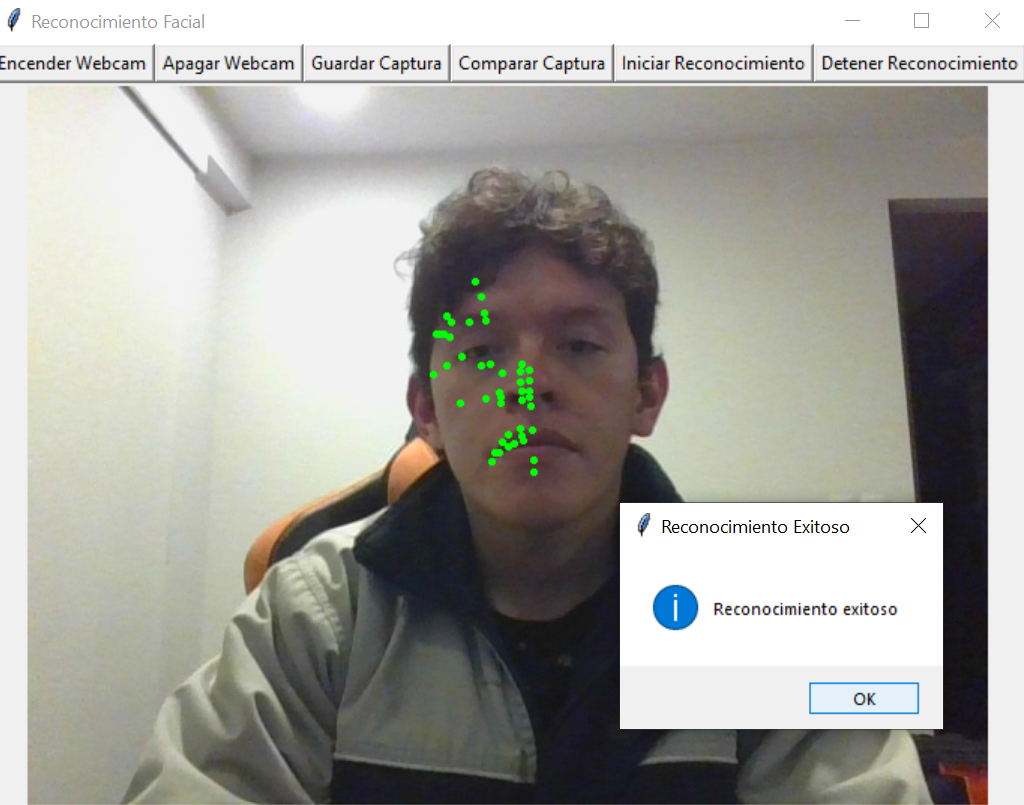
*Figura 2. Guardar captura*

* + - **Comparar Captura:**
      * Texto: "Comparar Captura".
      * Acción: Llama a la función *comparar\_captura* para comparar la imagen guardada con el cuadro actual (Figura 3).



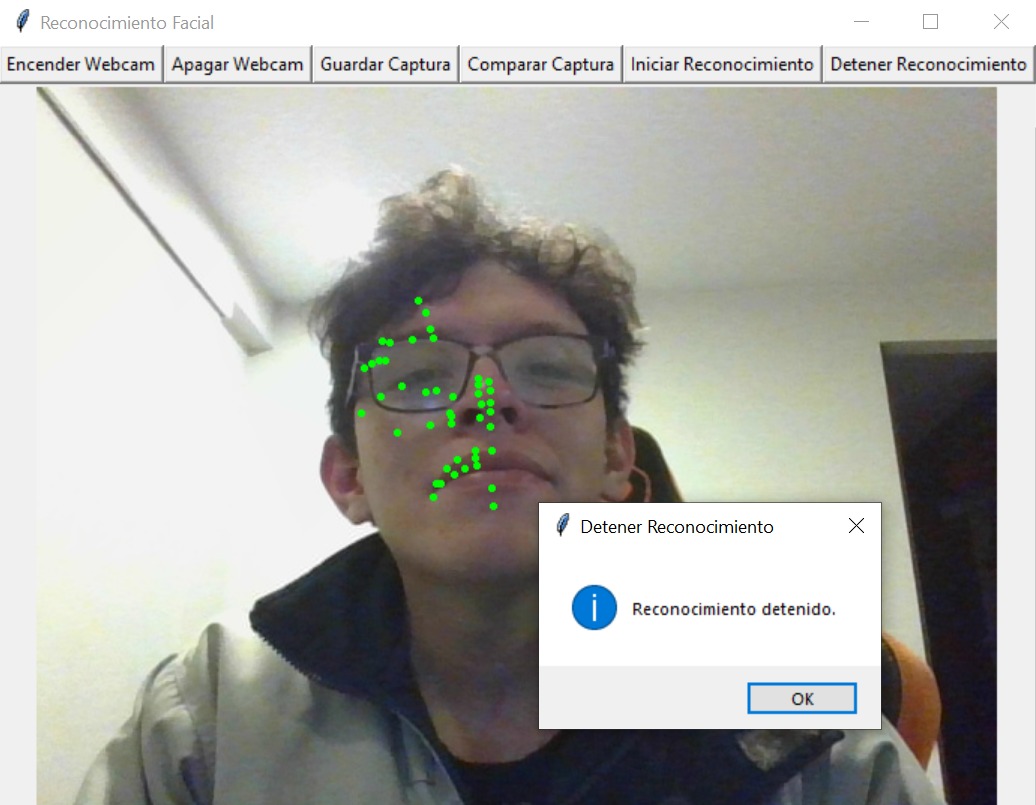
*Figura 3. Comparar captura*

* + - **Iniciar Reconocimiento:**
      * Texto: "Iniciar Reconocimiento".
      * Acción: Llama a la función *iniciar*\_*reconocimiento* para comenzar a analizar el video en tiempo real.
      * Si la identidad de la persona transmitida por la cámara corresponde a la captura guardada mientras el reconocimiento esta activo, entonces se mostrara una ventana de dialogo con el texto “Reconocimiento exitoso” (Figura 4).



*Figura 4. Comparación exitosa en tiempo real*

* + - **Detener Reconocimiento:**
      * Texto: "Detener Reconocimiento".
      * Acción: Llama a la función *detener*\_*reconocimiento* para finalizar el análisis (Figura 4).



*Figura 4. Detener reconocimiento*

1. **Visualización de video**

* **Etiqueta para video (*lbl*\_*video*):**
  + Ubicada debajo del *frame* de botones mediante *lbl\_video.pack().*
  + Sirve como contenedor para mostrar los cuadros de video procesados en tiempo real.
  + Se actualizará continuamente con la función *actualizar*\_*frame* durante el reconocimiento facial o al encender la cámara.

1. **Bucle principal**

* ***root*.*mainloop*():**
  + Mantiene la interfaz gráfica activa y en constante escucha de eventos.
  + Gestiona las interacciones del usuario, como clics en los botones.

# Funciones

# Función encender\_webcam



**Propósito general:**

La función *encender\_webcam* tiene como objetivo inicializar la captura de video desde la cámara web del sistema y comenzar el flujo de procesamiento de los fotogramas en tiempo real.

**Declaración de variable global:**

La línea global *cap* indica que la variable *cap,* utilizada para manejar la captura de video, será accedida y modificada a nivel global. Esto permite que otras funciones del programa interactúen con la misma instancia de la cámara web.

**Inicialización de la captura de video:**

La línea *cap = cv2.VideoCapture(0)* crea un objeto de captura de video usando *OpenCV (cv2),* asignado a la variable global cap.

El parámetro 0 indica que se utiliza la cámara web predeterminada del sistema. Si existieran múltiples cámaras conectadas, podrían utilizarse otros índices como 1, 2, etc., para seleccionar una cámara específica.

**Inicio del procesamiento de fotogramas:**

La línea *actualizar\_frame()* llama a la función *actualizar\_frame,* lo que inicia el procesamiento continuo de los fotogramas capturados por la cámara. Esta función se encargará de realizar tareas como la detección de rostros y la actualización de la interfaz gráfica con los datos de video.

**Comportamiento esperado:**

* Una vez ejecutada, la función activa la cámara web y pone en marcha el flujo de procesamiento en tiempo real.
* Si la cámara ya estaba activa o en uso, el código no incluye una verificación explícita para manejar esa condición, por lo que podría ser necesario agregar validaciones adicionales para evitar conflictos.

# Función apagar\_webcam



**Propósito general:**

La función *apagar\_webcam* tiene como objetivo detener la captura de video desde la cámara web y liberar los recursos asociados, además de limpiar el contenido visual de la ventana que muestra el video en la interfaz gráfica.

**Declaración de variable global:**

* La línea *global cap* declara que la variable *cap* es global, permitiendo que la función acceda y modifique la instancia de captura de video inicializada previamente en *encender\_webcam.*

**Liberación de recursos de la cámara:**

La condición *if cap:* verifica si *cap* ha sido inicializada, es decir, si la cámara está activa. Si esta condición es verdadera:

* La línea *cap.release()* libera los recursos asociados a la cámara web, deteniendo efectivamente la captura de video.
* La línea *cap = None* asigna *None* a la variable *global cap*, indicando que la cámara ya no está en uso y evitando futuras operaciones no deseadas con esta variable.

**Limpieza de la ventana de video:**

* La línea *lbl\_video.imgtk = None* elimina la referencia a cualquier imagen previa asignada a la ventana *lbl\_video*, asegurándose de que no se muestre un fotograma anterior o residual.
* La línea *lbl\_video.configure(image='')* establece la propiedad de imagen de la ventana a un valor vacío, limpiando visualmente el área donde se mostraba el video en la interfaz gráfica.

**Comportamiento esperado:**

Cuando se ejecuta esta función, la cámara web se detiene y su recurso es liberado para que pueda ser utilizado por otros programas. Además, la interfaz gráfica deja de mostrar el contenido capturado por la cámara, proporcionando un estado limpio para la ventana de video.

**Relación con otras funciones:**

Esta función complementa a *encender\_webcam*, ya que asegura que los recursos de la cámara se manejen correctamente cuando la cámara ya no es necesaria. Es especialmente útil para evitar problemas como conflictos en el acceso a la cámara por parte de otros programas o errores al intentar reiniciar la captura de video.

# Función guardar\_captura

  
**Propósito general:**

La función *guardar\_captura* permite capturar un fotograma de video desde la cámara activa, guardarlo como imagen en el disco y extraer los puntos clave del rostro para su posterior comparación.

**Revisión del estado de la cámara:**

La función verifica si la cámara está activa utilizando la variable global cap. Si *cap* es válida, significa que la cámara está lista para capturar un fotograma.

**Captura del fotograma:**

Se utiliza el método *read()* para obtener el fotograma actual de la cámara.

* Si la captura es exitosa, el fotograma se almacena en la variable *captura\_guardada*.
* La imagen capturada se guarda en el archivo captura.png en el directorio de trabajo utilizando *OpenCV*.

**Extracción de puntos clave:**

Se llama a la función *obtener\_puntos\_clave*, que procesa el fotograma capturado y extrae los puntos clave específicos del rostro definidos previamente en PUNTOS\_CRITICOS. Estos puntos clave se guardan en la variable global *puntos\_clave\_guardados* para su uso en funciones de comparación.

**Notificación al usuario:**

Tras guardar la captura y los puntos clave, se muestra un cuadro de diálogo informativo para notificar al usuario que la captura se ha realizado correctamente. Esto se realiza mediante la función *messagebox.showinfo* de *Tkinter*.

**Relación con otras funciones:**

* Los datos generados por esta función son fundamentales para las operaciones de reconocimiento facial, ya que proporcionan los puntos clave almacenados en *puntos\_clave\_guardados*, que se comparan con los puntos de fotogramas en tiempo real.

**Comportamiento esperado:**

Si la cámara está activa y se captura correctamente un fotograma, se guarda la imagen y se registran los puntos clave del rostro. Si la cámara no está activa, la función no realiza ninguna acción.

# Función obtener\_puntos\_clave

**Propósito general:**

La función *obtener\_puntos\_clave* procesa una imagen para detectar un rostro y extraer puntos clave específicos, utilizando la tecnología *FaceMesh* de *MediaPipe*. Estos puntos clave son necesarios para el reconocimiento facial y su comparación.

**Entrada:**

La función recibe un único parámetro llamado *frame*, que representa el fotograma capturado por la cámara. Este fotograma es un arreglo de tipo *NumPy* que contiene los datos de imagen en formato BGR (azul, verde, rojo).

**Conversión de formato de color:**

Antes de procesar la imagen, esta se convierte del formato BGR (usado por *OpenCV*) al formato RGB (requerido por *MediaPipe*). Esto se logra mediante la función *cv2.cvtColor*.

**Procesamiento con MediaPipe FaceMesh:**

El fotograma convertido se pasa al modelo *mp\_face\_mesh* para detectar rostros y obtener sus características faciales.

El método *process* de *FaceMesh* devuelve un objeto *results* que contiene los puntos de referencia (*landmarks*) de los rostros detectados, si los hay.

**Extracción de puntos clave:**

La función verifica si se detectaron puntos de referencia en el rostro mediante *results.multi\_face\_landmarks.*  
 **Si se detectan rostros:**

* Se inicializa una lista vacía llamada *puntos\_clave.*
* Para cada conjunto de puntos de referencia detectados, se seleccionan los índices especificados en *PUNTOS\_CRITICOS*.
* Los puntos seleccionados se agregan a la lista *puntos\_clave* en formato tridimensional (coordenadas X, Y, Z), donde las coordenadas están normalizadas entre 0 y 1.

**Retorno de datos:**

* Si se detectan y extraen puntos clave, la función devuelve una lista de tuplas con las coordenadas tridimensionales de los puntos clave seleccionados.
* Si no se detectan rostros, la función devuelve *None.*

**Relación con otras funciones:**

Esta función es esencial para *guardar\_captura*, *comparar\_captura*, y *comparar\_en\_tiempo\_real*, ya que proporciona los puntos clave del rostro necesarios para el reconocimiento facial y la comparación.

**Comportamiento esperado:**

* Si el fotograma contiene un rostro visible y detectable, los puntos clave del rostro se extraen correctamente.
* Si no se detecta ningún rostro o el fotograma está vacío, la función no genera puntos clave, retornando *None*.

# Función comparar\_captura



**Propósito general:**

La función *comparar\_captura* permite comparar el rostro capturado previamente (y guardado junto con sus puntos clave) con el rostro detectado en el fotograma actual. Su objetivo es calcular un porcentaje de similitud entre ambos conjuntos de puntos clave.

**Condiciones previas:**

* **Requisitos:** Para que esta función se ejecute correctamente, deben existir una captura previa y puntos clave asociados, almacenados en las variables globales *captura\_guardada* y *puntos\_clave\_guardados*.
* **Manejo de errores:** Si no hay una captura previa o puntos clave disponibles, muestra un mensaje de advertencia indicando que no hay datos con los cuales comparar.

**Flujo de ejecución:**

1. **Validación inicial:**

Verifica si las variables *captura\_guardada* y *puntos\_clave\_guardados* contienen datos válidos.

* + Si no están inicializadas, se muestra un cuadro de advertencia con el mensaje:  
    *"No hay captura guardada para comparar."*
  + En este caso, la función finaliza sin realizar ninguna acción adicional.

1. **Captura del fotograma actual:**
   * Verifica si la cámara está activa (*cap* no es *None*).
   * Si la cámara está activa, lee un nuevo fotograma de video con el método *cap.read().*
   * Si la lectura del fotograma es exitosa, este se pasa a la función *obtener\_puntos\_clave* para extraer los puntos clave del rostro en el fotograma actual.
2. **Extracción de puntos clave:**
   * Si *obtener\_puntos\_clave* devuelve un conjunto de puntos clave válidos (*puntos\_clave\_frame*), se procede a comparar estos puntos con los almacenados en *puntos\_clave\_guardados*.
3. **Cálculo de similitud:**
   * Llama a la función *comparar\_puntos\_clave*, pasando como argumentos los puntos clave guardados y los del fotograma actual.
   * La función *comparar\_puntos\_clave* calcula el porcentaje de similitud entre los dos conjuntos de puntos clave.
4. **Resultado:**
   * Muestra un cuadro de diálogo con el porcentaje de similitud calculado, utilizando el mensaje:  
     *"Porcentaje de similitud: XX.XX%"*, donde XX.XX representa el valor numérico obtenido.

**Relación con otras funciones:**

* Utiliza *obtener\_puntos\_clave* para extraer puntos clave del rostro en el fotograma actual.
* Se apoya en *comparar\_puntos\_clave* para calcular la similitud entre los puntos clave guardados y los detectados.

**Comportamiento esperado:**

* Si ambos conjuntos de puntos clave (captura guardada y fotograma actual) están disponibles, la función calcula y muestra un porcentaje de similitud.
* Si no hay captura guardada, muestra un mensaje de advertencia y termina sin errores.
* Si no se detecta un rostro en el fotograma actual, no realiza ninguna comparación ni muestra resultados.

# Función comparar\_puntos\_clave



**Propósito general:**

La función *comparar\_puntos\_clave* calcula la similitud entre dos conjuntos de puntos clave faciales. Esta comparación se basa en la distancia euclidiana promedio entre los puntos correspondientes en los dos conjuntos, y la similitud resultante se expresa como un porcentaje.

**Parámetros de entrada:**

1. **puntos\_guardados** - Lista de coordenadas tridimensionales (x, y, z) correspondientes a los puntos clave del rostro guardados previamente.
2. **puntos\_frame** - Lista de coordenadas tridimensionales (x, y, z) de los puntos clave extraídos del fotograma actual.

**Flujo de ejecución:**

1. **Validación inicial:**
   * Comprueba si el número de puntos clave en *puntos\_guardados* es diferente del número de puntos clave en *puntos\_frame*.
   * Si las listas no tienen la misma longitud, devuelve inmediatamente un valor de similitud del 0.0%, ya que no es posible realizar una comparación válida.
2. **Cálculo de la distancia promedio:**
   * Itera sobre ambos conjuntos de puntos clave en paralelo utilizando la función zip.
   * Para cada par de puntos clave (p1 y p2), calcula la distancia euclidiana entre ellos utilizando la fórmula:
   * Suma todas las distancias obtenidas y las almacena en una variable llamada *distancia\_total*.
3. **Normalización de la similitud:**
   * Divide la *distancia\_total* entre el número total de puntos clave para obtener la distancia promedio.
   * Calcula la similitud como un porcentaje inversamente proporcional a la distancia promedio, utilizando la fórmula:
   * Asegura que el valor de similitud nunca sea negativo utilizando *max(0, similitud)*.
4. **Retorno del resultado:**
   * Devuelve el porcentaje de similitud calculado.

**Relación con otras funciones:**

* Es llamada por la función *comparar\_captura* para comparar los puntos clave de una captura guardada con los del fotograma actual.
* También es utilizada en *comparar\_en\_tiempo\_real* para evaluar similitudes continuamente mientras el reconocimiento facial está activo.

**Comportamiento esperado:**

* Si ambos conjuntos de puntos clave tienen el mismo tamaño, devuelve un porcentaje de similitud basado en la distancia promedio entre los puntos.
* Si los conjuntos tienen tamaños diferentes, devuelve una similitud del 0.0%.
* Es robusta para manejar casos en los que la similitud pueda ser mínima (distancias grandes) o máxima (distancias pequeñas).

# Función iniciar\_reconocimiento



**Propósito general:**

La función *iniciar\_reconocimiento* activa un proceso de comparación continua entre los puntos clave de una captura facial previamente guardada y los puntos clave extraídos en tiempo real de la transmisión de la cámara. Es una preparación para realizar el reconocimiento facial continuo.

**Flujo de ejecución:**

1. **Validación inicial:**
   * Comprueba si la captura guardada (*captura\_guardada*) y sus puntos clave (*puntos\_clave\_guardados*) están disponibles.
   * Si cualquiera de estos elementos está ausente, muestra un mensaje de advertencia al usuario indicando que no hay datos disponibles para realizar la comparación, utilizando *messagebox.showwarning*. La ejecución se detiene en este punto.
2. **Activación del reconocimiento:**
   * Si la validación inicial es exitosa, establece la variable global *reconocimiento\_activo* como True. Esto indica que el sistema debe comenzar a comparar puntos clave en tiempo real.
3. **Llamada a la comparación en tiempo real:**
   * Llama a la función *comparar\_en\_tiempo\_real* para iniciar el proceso de comparación continua mientras *reconocimiento\_activo* esté habilitado.

**Parámetros de entrada:**

La función no recibe parámetros externos, pero utiliza las siguientes variables globales:

* *captura\_guardada*: Imagen guardada previamente desde la webcam.
* *puntos\_clave\_guardados*: Lista de puntos clave faciales extraídos de la captura guardada.
* *reconocimiento\_activo*: Indicador booleano que controla si el reconocimiento en tiempo real está habilitado.

**Relación con otras funciones:**

* Se complementa con *comparar\_en\_tiempo\_real*, que ejecuta el proceso de comparación continua en fotogramas sucesivos de la cámara.
* Requiere que las funciones *guardar\_captura* y *obtener\_puntos\_clave* hayan guardado correctamente los datos de referencia.

**Comportamiento esperado:**

* Si los datos necesarios (captura guardada y puntos clave) están disponibles, el sistema activa el reconocimiento facial en tiempo real.
* Si faltan datos, el usuario es informado mediante una advertencia y no se activa el reconocimiento.

**Errores manejados:**

* Muestra un mensaje claro al usuario si no se ha guardado previamente una captura o los puntos clave asociados.
* Evita iniciar procesos innecesarios cuando no hay datos válidos para comparar.

Esta función es el punto de entrada para iniciar el reconocimiento facial continuo del sistema.

# Función comparar\_en\_tiempo\_real



**Propósito general:**

La función *comparar\_en\_tiempo\_real* realiza una comparación continua de los puntos clave faciales extraídos de la transmisión de la cámara en tiempo real contra los puntos clave almacenados previamente de una captura guardada. Su objetivo es detectar si la similitud entre las dos caras supera un umbral definido.

**Flujo de ejecución:**

1. **Condiciones iniciales:**
   * Verifica que el reconocimiento está activo (*reconocimiento\_activo* es True) y que existen una captura guardada (*captura\_guardada*) y sus puntos clave asociados (*puntos\_clave\_guardados*).
   * Si alguna de estas condiciones no se cumple, la función no continúa ejecutándose.
2. **Captura de un nuevo fotograma:**
   * Comprueba si la cámara está encendida (*cap* es válido).
   * Lee un nuevo fotograma de la transmisión en vivo usando *cap.read*. Si no es posible capturar un fotograma válido, la función no procede con la comparación.
3. **Extracción de puntos clave:**
   * Llama a la función *obtener\_puntos\_clave* para obtener los puntos clave faciales del fotograma actual.
   * Si no se detectan puntos clave, la función salta la comparación y espera al siguiente fotograma.
4. **Comparación de puntos clave:**
   * Llama a la función *comparar\_puntos\_clave* para calcular la similitud entre los puntos clave del fotograma actual y los puntos clave de la captura guardada.
   * La similitud se calcula como un porcentaje, donde valores más altos indican mayor coincidencia.
5. **Verificación del umbral de similitud:**
   * Si la similitud calculada es igual o superior al 95%, se considera que el reconocimiento ha sido exitoso:
     + Se muestra un mensaje informativo al usuario mediante *messagebox.showinfo* indicando que el reconocimiento facial fue exitoso.
     + Detiene el proceso de reconocimiento estableciendo *reconocimiento\_activo* como False.
6. **Continuación del proceso:**
   * Si el reconocimiento no ha alcanzado el umbral, la función programa otra ejecución tras un intervalo de 500 ms utilizando *root.after(500, comparar\_en\_tiempo\_real)*. Esto permite la comparación continua en tiempo real mientras el reconocimiento está activo.

**Parámetros de entrada:**

La función no recibe parámetros, pero utiliza las siguientes variables globales:

* **reconocimiento\_activo:** Determina si el reconocimiento en tiempo real debe continuar.
* **captura\_guardada:** Imagen guardada previamente desde la webcam.
* **puntos\_clave\_guardados:** Lista de puntos clave faciales extraídos de la captura guardada.
* **cap**: Objeto que representa la conexión con la cámara web.

**Relación con otras funciones:**

* Depende de *obtener\_puntos\_clave* para extraer puntos clave faciales del fotograma actual.
* Utiliza *comparar\_puntos\_clave* para calcular la similitud entre los puntos clave de la captura guardada y los del fotograma actual.
* Complementa a *iniciar\_reconocimiento*, que activa este proceso.

**Comportamiento esperado:**

* Compara continuamente los puntos clave extraídos de la cámara en tiempo real con los puntos clave de una captura guardada.
* Detiene el proceso si se alcanza un umbral de similitud del 95% o más.
* Si no se cumple el umbral, sigue ejecutándose a intervalos regulares de 500 ms.

**Errores manejados:**

* Evita intentos de comparación si la cámara no está encendida o si no se detectan puntos clave en el fotograma actual.
* Detiene la comparación en caso de reconocimiento exitoso para evitar procesamiento innecesario.

Esta función garantiza un reconocimiento facial continuo y eficiente mientras el sistema está en funcionamiento.

# Función detener\_reconocimiento



**Propósito general:**

La función *detener\_reconocimiento* permite al usuario detener el proceso de reconocimiento facial en tiempo real, interrumpiendo la ejecución continua de la función *comparar\_en\_tiempo\_real*.

**Desactivación del reconocimiento:**

La función modifica la variable global *reconocimiento\_activo* para detener el proceso de comparación en tiempo real:

* *global reconocimiento\_activo*
* *reconocimiento\_activo = False*
* *reconocimiento\_activo*: Se asigna el valor False, indicando que el reconocimiento facial ya no está activo.

**Notificación al usuario:**

Una vez que se ha desactivado el reconocimiento, se muestra un mensaje al usuario para confirmar la acción:

* *messagebox.showinfo("Detener Reconocimiento", "Reconocimiento detenido.")*
* *messagebox.showinfo*: Utiliza un cuadro de diálogo informativo de Tkinter para comunicar que el reconocimiento ha sido detenido correctamente.

**Relación con otras funciones:**

* Se utiliza para finalizar el proceso iniciado por *iniciar\_reconocimiento*.
* Detiene la ejecución cíclica de *comparar\_en\_tiempo\_real* al cambiar el estado de *reconocimiento\_activo*.

**Comportamiento esperado:**

* Interrumpe inmediatamente cualquier comparación en curso.
* Garantiza que no se realicen nuevas evaluaciones de similitud hasta que el reconocimiento sea activado nuevamente.

# Función actualizar\_frame



**Propósito general:**

La función *actualizar\_frame* gestiona la visualización en tiempo real del flujo de video capturado por la cámara web en la interfaz gráfica. Además, aplica un procesamiento facial mediante *MediaPipe* para detectar puntos clave en el rostro y dibujarlos directamente sobre el fotograma.

**Flujo de ejecución:**

1. **Captura de un nuevo fotograma:**
   * Comprueba si la cámara está encendida (*cap* no es *None*).
   * Usa *cap.read()* para capturar un fotograma de la transmisión en vivo.
   * Si la captura no es válida (ret es False), la función termina su ejecución para ese ciclo.
2. **Procesamiento facial:**
   * Convierte el fotograma capturado de BGR (formato predeterminado de *OpenCV*) a RGB, ya que *MediaPipe* requiere imágenes en formato RGB.
   * Procesa el fotograma con *MediaPipe Face Mesh* (*mp\_face\_mesh.process*), que detecta puntos clave en los rostros presentes en la imagen.
3. **Dibujar puntos clave:**
   * Si *MediaPipe* detecta rostros en el fotograma, itera a través de los puntos clave definidos en *PUNTOS\_CRITICOS*.
   * Para cada punto, calcula las coordenadas en píxeles utilizando el ancho y alto del fotograma y las proporciones proporcionadas por *MediaPipe*.
   * Dibuja un pequeño círculo en la posición de cada punto clave usando cv2.circle. Los círculos son de color verde (RGB: (0, 255, 0)) y tienen un tamaño de 2 píxeles.
4. **Conversión para la interfaz gráfica:**
   * Convierte el fotograma procesado de nuevo a formato RGB usando *PIL* (*Python Imaging Library*), para que sea compatible con la visualización en *Tkinter*.
   * Convierte la imagen a un objeto *ImageTk.PhotoImage*, que puede usarse como imagen en una ventana de *Tkinter*.
5. **Actualización de la interfaz:**
   * Asigna la imagen procesada a la ventana *lbl\_video* que se encarga de mostrar el video en la ventana principal de la aplicación.
   * Programa la ejecución de la función *actualizar\_frame* nuevamente después de un breve intervalo de 10 milisegundos, utilizando *lbl\_video.after(10, actualizar\_frame)*. Esto crea un bucle de actualización constante para la transmisión de video.

**Parámetros de entrada:**

La función no recibe parámetros, pero utiliza las siguientes variables globales:

* **cap:** Objeto que representa la conexión con la cámara web.
* **lbl\_video:** Ventana de etiqueta (*label*) en el cual se muestra el flujo de video procesado.
* **PUNTOS\_CRITICOS:** Lista de índices que representan los puntos clave específicos que deben destacarse en el rostro.

**Relación con otras funciones:**

* Complementa a *encender\_webcam*, que inicia la captura de video al encender la cámara.
* Utiliza la funcionalidad de *MediaPipe* (instanciada como *mp\_face\_mesh*) para detectar puntos faciales en tiempo real.

**Comportamiento esperado:**

* Muestra el flujo de video en tiempo real dentro de la ventana principal de la aplicación.
* Dibuja puntos clave críticos en los rostros detectados, proporcionando una visualización clara de las áreas de interés procesadas por el sistema.
* Continúa actualizándose constantemente mientras la cámara esté encendida.

**Errores manejados:**

* Si no se puede capturar un fotograma (por ejemplo, si la cámara está desconectada o apagada), no se genera ningún error visible y la función simplemente no actualiza la ventana.
* Si no se detectan rostros o puntos clave en el fotograma, no intenta dibujar los puntos y sigue mostrando el video original.

Esta función es esencial para proporcionar retroalimentación visual al usuario durante el uso de la aplicación, mostrando tanto el flujo de video en vivo como los resultados de las detecciones faciales en tiempo real.

# Recomendaciones

* Se recomienda explorar métodos más avanzados, como el uso de modelos de aprendizaje profundo y redes neuronales, para incrementar la precisión y robustez del sistema en condiciones adversas.
* Para ampliar la funcionalidad y la capacidad del sistema, sería beneficioso incorporar un manejo de bases de datos que permita almacenar, consultar y gestionar múltiples registros faciales. Esto facilitaría su aplicación a gran escala.
* Considerar el uso de cámaras con mejores especificaciones técnicas.

# Conclusiones

El desarrollo de un sistema de reconocimiento facial para la validación de usuarios preferenciales en el transporte público responde a una necesidad apremiante: garantizar el uso adecuado de las tarjetas preferenciales y prevenir la suplantación de identidad. Este proyecto ha demostrado cómo las tecnologías accesibles y las herramientas de código abierto, como Python, OpenCV y Mediapipe, pueden ser utilizadas para implementar soluciones funcionales y eficientes.