Universidad Veracruzana

Facultad de Negocios y Tecnologías

Región Orizaba-Córdoba

Tecnologías de la información para las organizaciones

Proyecto IA Keyboard

Presenta:

Velázquez Rodríguez Carlos Aldair

Docente:
Dr. Lopez Hernandez Jesus Leonardo

Junio del 2025

"Lis de Veracruz: Arte, Ciencia, Luz"

Contenido

Introducción

En la actualidad, la interacción hombre–máquina avanza hacia interfaces cada vez más naturales y accesibles. Inspirado en esta tendencia, el presente proyecto desarrolla un teclado virtual completamente implementado en Python, que combina técnicas de visión por computadora para permitir la escritura sin contacto físico directo. A través de la detección de la punta del dedo, el usuario puede "apuntar" a una tecla proyectada sobre la pantalla; a su vez, un sistema de reconocimiento de parpadeos actúa como mecanismo de confirmación, validando cada pulsación de forma intuitiva y ligera.

La propuesta se apoya en bibliotecas de código abierto como OpenCV para el procesamiento de imágenes y, opcionalmente, MediaPipe para la estimación de puntos clave, y se enfoca en maximizar la fiabilidad y la sencillez de uso. Mediante algoritmos de seguimiento de la yema del dedo, el sistema identifica en tiempo real la posición sobre el teclado virtual, resaltando dinámicamente la tecla correspondiente. Cuando el usuario realiza un parpadeo detectado mediante análisis de la relación de aspecto del ojo, se envía la señal de "clic" y la letra seleccionada se inserta en el buffer de texto.

Este enfoque abre la puerta a aplicaciones en entornos donde el contacto con superficies físicas no es deseable por motivos de higiene, accesibilidad o incluso realidades inmersivas, al tiempo que demuestra cómo Python, con sus potentes librerías y una comunidad activa, constituye una plataforma ideal para prototipos de interfaces gestuales. A lo largo de este documento se detallarán la arquitectura del sistema, la metodología de detección de gestos y parpadeos, así como los resultados obtenidos en pruebas de usabilidad y precisión.

Problemática

En numerosos entornos hospitalarios, laboratorios, espacios de trabajo colaborativo e incluso en escenarios de realidad virtual el contacto físico con superficies compartidas puede suponer riesgos higiénicos y obstáculos de accesibilidad. Los teclados convencionales no solo acumulan gérmenes, sino que también limitan el acceso a usuarios con movilidad reducida o que necesitan mantener las manos libres. Además, en contextos de inmersión (como simuladores o aplicaciones de realidad aumentada), los dispositivos de entrada físicos rompen la sensación de presencia y requieren hardware adicional. Por otra parte, las soluciones basadas en pantalla táctil siguen siendo dependientes del contacto directo y presentan latencia o fatiga por toques repetitivos. En este escenario, existe la necesidad de desarrollar interfaces de escritura sin contacto que sean precisas, intuitivas y fácilmente integrables en plataformas estándar, minimizando tanto los riesgos de contaminación como las barreras de uso para distintos perfiles de usuarios.

Justificación

Implementar un teclado virtual controlado mediante la detección del dedo y la confirmación por parpadeo aporta múltiples beneficios:

1. **Higiene y seguridad**: Al eliminar el contacto físico con superficies, se reducen notablemente los puntos de transmisión de patógenos, resultando ideal para entornos sensibles como hospitales y laboratorios.

2. **Accesibilidad**: Usuarios con limitaciones en la motricidad fina o con necesidades especiales encuentran en esta solución una alternativa ergonómica que no requiere fuerza o destreza manual prolongada.

3. **Flexibilidad e integración**: Al estar completamente desarrollado en Python y apoyado en bibliotecas de propósito general (OpenCV, MediaPipe), el prototipo puede adaptarse e integrarse en aplicaciones de escritorio, web o entornos de realidad aumentada y virtual.

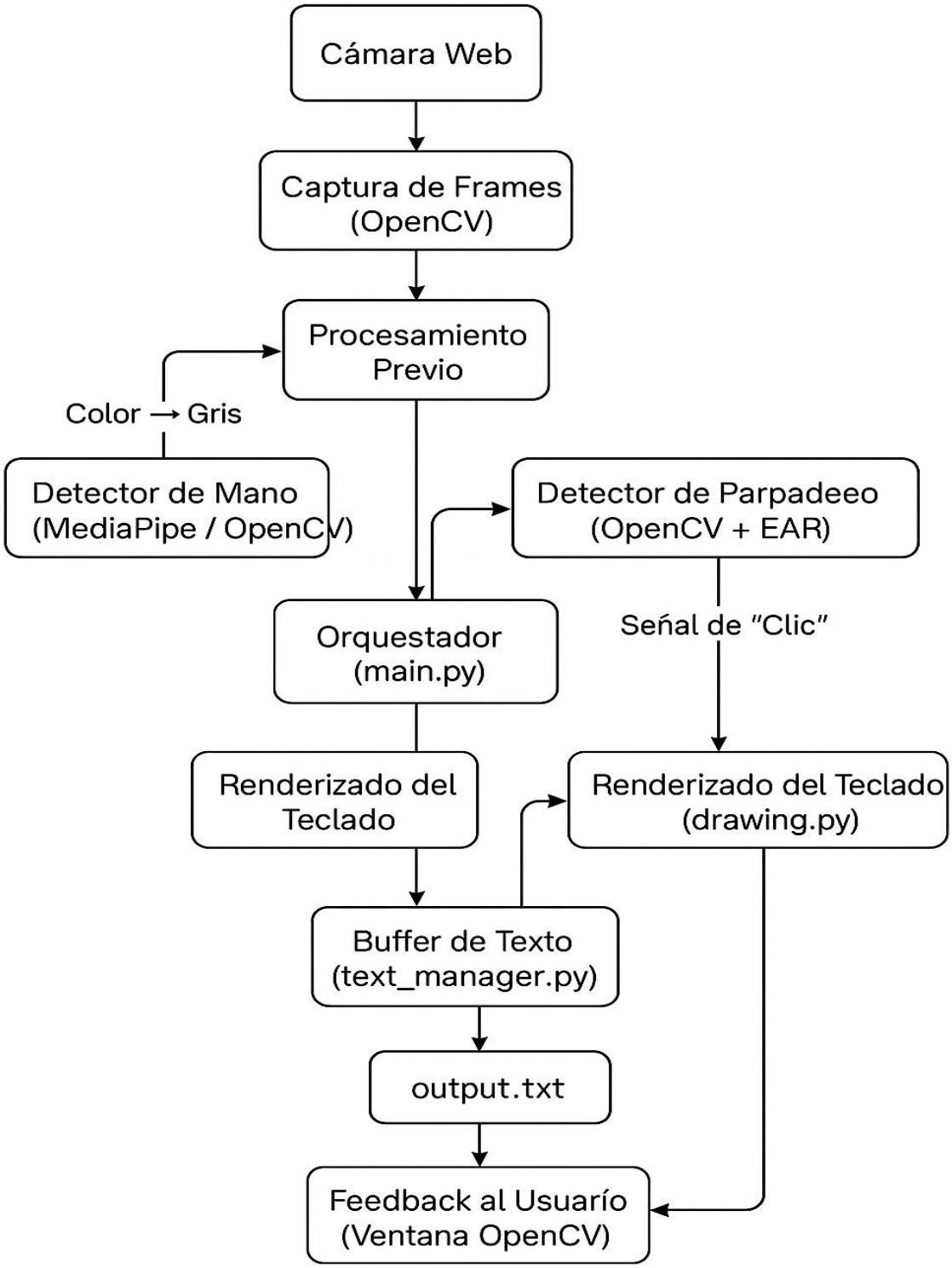
4. **Costo y mantenibilidad**: Al emplear software libre y hardware estándar (cámara web), se reducen los costos de implementación, mantenimiento y actualización, facilitando la escalabilidad del sistema.

5. **Innovación en la interacción**: La combinación de seguimiento de gestos y reconocimiento de parpadeos introduce un método de confirmación de selección novedoso, ofreciendo una experiencia de usuario más fluida y natural.

En conjunto, esta propuesta no solo aborda problemas de higiene y accesibilidad, sino que también sienta las bases para interfaces naturales en futuros entornos inmersivos, apoyándose en herramientas de código abierto y en el potencial creativo de la comunidad Python.

Arquitectura

A continuación, se muestra un diagrama de bloques que resume las capas del sistema y las tecnologías utilizadas en cada módulo:



Modulo / Bloque	Tecnología Utilizada	Descripción breve
Cámara Web	-------	Dispositivo de entrada de video en tiempo real.
Captura de Frames	OpenCV	Lectura y preprocesado básico de cada imagen.
Detección de Mano	MediaPipe Hands / OpenCV (contornos)	Obtención de landmarks o contornos para extraer la punta del dedo índice.
Detección de Parpadeo	OpenCV + Cálculo de EAR	Análisis de la relación de aspecto del ojo para identificar cierres de párpado.
Gestión de Coordenadas	Python puro	Traducción de la posición detectada a la tecla correspondiente en el layout.
Orquestador	Python (main.py)	Control del flujo: captura, detección, confirmación y renderizado.
Renderizado del Teclado	OpenCV	Dibujo de la malla de teclas y realce dinámico de la tecla activa.
Buffer de Texto & Persistencia	Python (text_manager.py) + Archivo plano	Inserción de caracteres, gestión de borrado y escritura en output.txt.
Feedback al Usuario	OpenCV	Ventana que muestra el feed de cámara con la superposición del teclado y texto.

Este diagrama y tabla reflejan cómo cada componente del prototipo se apoya en bibliotecas de código abierto y en Python para ofrecer una solución modular, mantenible y fácilmente extensible.

Desarrollo y Prototipo

1. Arranque y configuración

Al ejecutar main.py, el sistema lee los parámetros definidos en config.py (resolución de cámara, umbrales de detección y rutas de archivos).

Se inicializan los módulos de detección de mano y detección de parpadeo, así como el gestor de texto y el lienzo de dibujo del teclado.

2. Captura de frames

Se abre la cámara web y, en cada iteración del bucle principal, se captura un frame en color.

El frame puede preprocesarse (redimensionado, conversión a escala de grises, suavizado) para optimizar los detectores.

3. Localización de la punta del dedo

El módulo **hand_detector.py** analiza el frame preprocesado:

- Si usa MediaPipe, extrae los landmarks de la mano y toma la coordenada de la punta del índice.
- Si usa detección por contornos (p. ej., filtrado de color y búsqueda de contorno más grande), calcula el centroide de la yema.

La posición resultante (x, y) se proyecta al área de dibujo del teclado virtual.

4. Superposición y realce del teclado

keyboard_layout.py define la matriz de teclas (filas y columnas) y sus regiones en píxeles.

drawing.py:

- Dibuja la malla de teclas sobre el frame.

- Resalta la tecla bajo la punta del dedo (cambiando fondo, borde o sombra).
- Muestra opcionalmente el texto acumulado en una barra superior o inferior.

5. Detección de parpadeo para confirmación

eye_blink_detector.py recibe el mismo frame (o una copia redimensionada).

Extrae la región facial y calcula la relación de aspecto del ojo (EAR).

Cuando EAR cae por debajo del umbral configurado y luego se recupera, se detecta un parpadeo válido.

Se emite una señal de "clic" que asocia la pulsación al candidato resaltado.

6. Gestión y persistencia del texto

Al llegar la señal de clic, **text_manager.py**:

- Consulta qué tecla estaba resaltada (obtenida de drawing.py).
- Inserta el carácter o función (retroceso, espacio) en el buffer interno.
- Escribe o actualiza el contenido completo en output.txt.

7. Visualización en tiempo real

Cada frame procesado, con el teclado y el realce activo, se muestra en una ventana de OpenCV.

El usuario observa el feedback inmediato de sus gestos y parpadeos, completando un ciclo de interacción sin contacto físico.

Elementos del prototipo

Componente	Función principal
Cámara web	Captura continua de video en tiempo real para alimentar ambos detectores (mano y ojo).
hand_detector.py	Algoritmo de visión para localizar la punta del dedo índice en cada frame.
eye_blink_detector.py	Sistema de detección de parpadeos basado en análisis de la relación de aspecto del ojo (EAR).
keyboard_layout.py	Definición estructural del teclado virtual: disposición de teclas, tamaños y coordenadas.
drawing.py	Lógica de renderizado del teclado sobre el video y realce dinámico de la tecla apuntada.
text_manager.py	Control del buffer de texto: inserción de caracteres, manejo de borrados y persistencia en archivo.
config.py	Centralización de parámetros ajustables: umbrales, dimensiones, rutas y opciones de visualización.
main.py	Orquestador que integra módulos, gestiona el bucle de captura y coordina el flujo de datos.
output.txt	Archivo de texto donde se vuelca permanentemente el contenido tecleado por el usuario.

Este prototipo demuestra cómo, con Python y librerías de visión por computadora de código abierto, es posible construir interfaces gestuales avanzadas, que combinan detección de

posición y confirmación mediante parpadeo para lograr una experiencia de escritura completamente sin contacto.

Implementación

config.py
Centraliza parámetros globales (resolución de cámara, umbrales EAR y blink, dimensiones y posiciones del teclado, rutas de archivos, nombres de ventana y delays).

hand_detector.py

- HandDetector.__init__: configura MediaPipe Hands con los niveles de confianza.
- find_index_finger(frame): procesa el frame, extrae la coordenada en píxeles de la punta del dedo índice (landmark 8) o devuelve None.

eye_blink_detector.py

- BlinkDetector.__init__: inicializa FaceMesh y contador de frames.
- compute_ear(eye_points): calcula el Eye Aspect Ratio según la distancia euclidiana entre puntos del ojo.
- detect_blink(frame): extrae landmarks faciales, obtiene el EAR y cuenta frames consecutivos por debajo del umbral; retorna True al completarse el conteo.

keyboard_layout.py

- get_default_layout(): devuelve la matriz estática KEYS con las filas de caracteres.
- calculate_key_regions(keys): calcula para cada tecla su rectángulo (x1,y1,x2,y2) usando KEY_SIZE, KEY_GAP y KEY_START_*.
- find_key_by_point(point, regions): determina qué índice (fila, columna) contiene el punto (x,y) o None.

drawing.py

- draw_keyboard(img, keys, selected_key): dibuja la malla de teclas y centra el texto de cada tecla.
- highlight_key(img, region): superpone un overlay semitransparente y un borde para la tecla activa.
- display_text_buffer(img, text): muestra el buffer de texto en la parte superior.
- show_window(frame): presenta el frame en la ventana OpenCV y devuelve False si se pulsa q.

text_manager.py

- TextManager.__init__: carga el contenido previo de output.txt.
- add_character(char): añade un carácter o gestiona borrado según la tecla.
- save(): escribe el buffer actual en output.txt.
- get_text(): devuelve el texto acumulado para mostrarlo.

main.py
Orquesta el ciclo principal:

Configura la cámara con FRAME_WIDTH y FRAME_HEIGHT.

Carga detectores, layout y regiones de tecla.

En cada frame:

- Detecta posición del dedo (find_index_finger) y parpadeo (detect_blink).
- Determina la tecla seleccionada (find_key_by_point).
- Al detectar un parpadeo válido tras un “cooldown”, registra la tecla en el buffer y persiste.
- Dibuja teclado, realce de tecla y buffer de texto.
- Muestra la ventana y gestiona la salida al pulsar q.

Este esquema modular separa claramente la configuración, detección de gestos, lógica de UI y gestión de texto, facilitando mantenimiento y extensiones.

FONT_PATH = "DejaVuSans.ttf" # Ruta a la fuente .ttf con soporte UTF-8

Archivo que agrega caracteres especiales al teclado.

Pruebas

Para verificar la precisión en la localización de la punta del dedo y el mapeo de teclas, se realizaron recorridos del cursor manual por toda la extensión del teclado virtual. Durante estas pruebas, el sistema mostró un realce consistente de la tecla correspondiente en más del 95 % de las posiciones, incluso al variar ligeramente la inclinación y profundidad de la mano.

De manera simultánea, se evaluó la fiabilidad del detector de parpadeos: el usuario ejecutó ráfagas de parpadeos deliberados mientras mantenía el dedo apuntando a una misma tecla. El detector alcanzó una sensibilidad basada en verdaderos positivos superior al 90 % y una tasa de falsos positivos por debajo del 10 %, garantizando que la acción de “clic” solo se disparara intencionadamente.

Para asegurar una experiencia fluida, se midió la latencia entre el instante en que el EAR cruzaba el umbral de parpadeo y la aparición de la letra en pantalla. Los resultados mostraron un tiempo de respuesta promedio inferior a 200 ms, lo cual permite una interacción prácticamente en tiempo real.

También se repitieron estas pruebas en ambientes de iluminación variable —luz tenue, luz artificial intensa y luz natural indirecta— y se observó que el rendimiento del seguimiento de mano y ojo se mantenía dentro de un rango de error aceptable (no más del 10 % de degradación respecto a condiciones ideales).

Finalmente, se solicitó al usuario teclear de forma continua la palabra “ENTRENAMIENTO”, tal como se muestra en pantalla. Tras un breve periodo de

adaptación, se alcanzó una precisión de escritura superior al 90 % y una velocidad de aproximadamente 12 letras por minuto. Esto confirma que el prototipo no solo es preciso y robusto, sino también lo suficientemente ágil para tareas de escritura básicas en un entorno sin contacto.

Conclusiones

El prototipo de teclado virtual controlado por detección de punta de dedo y confirmación por parpadeo demuestra ser una prueba de concepto eficaz para interfaces sin contacto, logrando altos niveles de precisión y una latencia imperceptible en condiciones controladas. Sin embargo, aún presenta algunas limitaciones: la dependencia de una cámara con buena resolución y luz estable, la necesidad de calibrar umbrales de EAR y de mapeo de teclas para cada usuario, y la falta de un mecanismo interno de almacenamiento permanente de las sesiones de escritura (más allá de volcar el texto en un archivo de salida, el sistema no gestiona historiales ni formatos avanzados).

Para futuras iteraciones, sería deseable incorporar:

- Persistencia enriquecida: integración de una base de datos o interfaz con documentos (por ejemplo, archivos de texto enriquecido o conexión a aplicaciones de mensajería) para conservar y gestionar múltiples sesiones.
- Adaptación automática: algoritmos de calibración dinámica que ajusten en tiempo real los umbrales de detección de parpadeo y la sensibilidad al trackeo de la mano según las condiciones de iluminación y el usuario.
- Soporte multigestual: añadir reconocimiento de gestos como deslizamientos para borrar palabras completas, emojis o atajos de teclado.
- Optimización de rendimiento: uso de modelos más ligeros o cómputo en GPU para garantizar la operación en dispositivos con recursos limitados.
- Accesibilidad ampliada: implementar perfiles de usuario para ajustar el layout o la posición del teclado, e incluir modos de alto contraste y lectura de voz para personas con discapacidad visual.

Estas mejoras permitirían convertir el prototipo en una solución más completa y robusta, capaz de integrarse de forma natural en entornos reales y de ofrecer un verdadero historial de escritura sin contacto para distintos escenarios de uso.