Universidad Veracruzana

Facultad de Negocios y Tecnologías

Región Orizaba-Córdoba

Tecnologías de la información para las organizaciones

Sistema de teclado virtual con confirmación por parpadeo, seguimiento de manos y autocompletado LSTM

Inteligencia Artificial

Presenta:
CARRILLO RINCON ARMANDO

Docente:
LOPEZ HERNANDEZ JESUS LEONARDO

Junio de 2025

"Lis de Veracruz: Arte, Ciencia, Luz"

Contenido

1.- Introducción

1.1.- Problemática

En entornos donde la interacción tradicional con teclado y ratón resulta poco ergonómica o inaccesible, por ejemplo, para personas con movilidad reducida, condiciones neuromusculares o fatiga repetitiva, la escritura de texto puede convertirse en un proceso lento, cansado e incluso frustrante.

Los principales retos identificados son:

- **Limitaciones de accesibilidad**: Usuarios con discapacidad motriz carecen de alternativas eficientes al teclado convencional.
- **Velocidad de entrada reducida**: La selección de caracteres uno a uno, especialmente mediante interfaces adaptativas, suele ser muy lenta y propensa a errores.
- **Carga cognitiva y fatiga**: La necesidad de fijar la mirada prolongadamente o usar dispositivos de apuntado manual genera cansancio ocular y muscular.
- **Escalabilidad funcional**: Muchas soluciones se centran únicamente en un modo de interacción (p. ej., ojo o mano), sin aprovechar la complementariedad de ambas para optimizar la experiencia

1.2.- Justificación

El sistema propuesto aborda estas limitaciones combinando cuatro tecnologías clave, tal como se refleja en los módulos del código:

1. **Seguimiento ocular (Gaze Tracking)**: Utiliza MediaPipe Face Mesh para estimar de forma continua el punto de mirada, permitiendo una navegación por el teclado sin contacto físico directo.
2. **Confirmación por parpadeo (Blink Confirmation)**: Detecta parpadeos voluntarios para aceptar la selección de un carácter, minimizando falsos positivos y reduciendo la saturación de la interfaz.
3. **Seguimiento de manos (Hand Tracking)**: Emplea MediaPipe Hands para detectar gestos complementarios (por ejemplo, señalar “espacio” o “borrar”), incrementando la versatilidad y rapidez de entrada.
4. **Autocompletado predictivo con LSTM**: Un modelo entrenado con un corpus en español sugiere continuaciones de texto tras cada secuencia de caracteres, acelerando la escritura y disminuyendo la carga cognitiva.

Al integrar estos componentes, el sistema no solo mejora la accesibilidad, sino que también optimiza la velocidad y precisión en la entrada de texto. La fusión de métodos oculares y manuales ofrece redundancia y flexibilidad. Asimismo, el autocompletado basado en LSTM eleva la eficiencia, anticipando palabras y reduciendo pulsaciones.

2.- Arquitectura

2.1.- Diagrama del sistema (tecnologías implementadas).

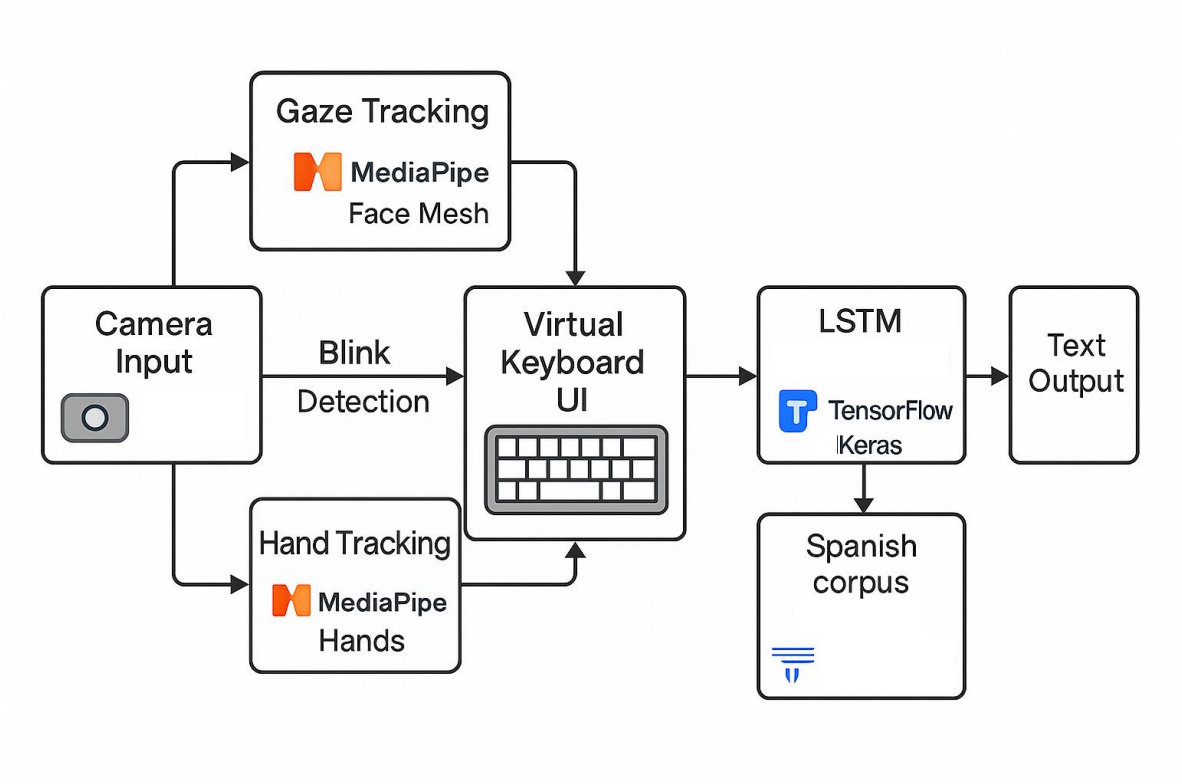

Ilustración 1.- El diagrama muestra de manera clara y detallada cómo fluye la información desde la captura de la cámara, pasando por los módulos de seguimiento ocular (MediaPipe Face Mesh), confirmación por parpadeo, seguimiento de manos (MediaPipe Hands) y autocompletado LSTM (TensorFlow/Keras), hasta llegar a la interfaz de teclado virtual y la salida de texto.

3.- Desarrollo y Prototipo

3.1.- Flujo de Diseño e Implementación

Definición de Requisitos

- Identificación de las necesidades de accesibilidad y eficiencia: usuarios con movilidad reducida, escenarios manos libres.
- Selección de tecnologías: MediaPipe para detección facial y de manos, TensorFlow/Keras para LSTM, OpenCV para captura de video.

Arquitectura Modular

- **capturing_camera**: Inicializa la captura de video con OpenCV, ajusta resolución (240×180) y tasa de cuadros.
- **detecting_eye_blink_module.py**: Usa MediaPipe Face Mesh para extraer landmarks oculares; calcula la razón de parpadeo (EAR) para detectar blinks voluntarios.
- **tracking_hand_module.py**: Emplea MediaPipe Hands para identificar gestos y posiciones de dedos; mapea índices de landmark a acciones ('espacio', 'borrar', 'tabulador').
- **autocompleter_module.py**: Carga el modelo LSTM (autocomplete_es.h5) y el tokenizer.pkl; define la función suggest(texto) que retorna top-3 predicciones.
- **vir_keyboard_eye_det.py**: Integra módulos anteriores; dibuja la interfaz de teclado virtual con teclas de tamaño 100px y margen de 20px.

Entrenamiento del Modelo LSTM

- Preparación del corpus: spanish_corpus.txt, limpieza de caracteres especiales y normalización a minúsculas.
- Tokenización: construcción de secuencias de tamaño fijo (p. ej., 20 tokens) y generación de pares (secuencia, siguiente palabra).
- Arquitectura LSTM: 2 capas LSTM de 256 unidades, dropout 0.2, embedding de dimensión 100.
- Proceso de entrenamiento: optimizador Adam, tasa de aprendizaje 0.001, 50 épocas, batch size 64.
- Exportación: guardado de pesos en autocomplete_es.h5 y tokenizador en tokenizer.pkl.

3.2.- Componentes del Prototipo

Hardware

- Cámara web USB de 30 fps: captura simultánea para tracking facial y manual.
- Computadora estándar: CPU Ryzen 5, 8 GB RAM.

Software y Librerías

- Python 3.10
- OpenCV 4.x
- MediaPipe
- TensorFlow 2.x / Keras
- NumPy

Interfaz de Usuario

- Ventana OpenCV de 1280×720 px: renderizado de teclado virtual en zona inferior (altura 300 px).
- Feedback visual: resaltado de tecla activa al fijar la mirada durante DWELL_TIME = 1.0 s.
- Confirmación por parpadeo: escritura al cerrar ojos intencionalmente (umbral EAR < 0.20).
- Botones adicionales: "Tab", "Espacio", "Borrar", "Guardar"; dispuestos en módulo lateral para evitar superposición.
- Sugerencias predictivas: cuadro emergente con hasta 3 palabras sugeridas tras cada selección.

3.3.- Descripción del Proceso de Uso

Inicio

- Ejecutar vir_keyboard_eye_det.py. Se cargan los modelos y se abre la captura de cámara.

Selección de Teclas

- El usuario navega por la interfaz desplazando su mano detectada por MediaPipe Hands; al moverla sobre una tecla, esta se resalta (resaltado azul).
- Un parpadeo voluntario (EAR < 0.20) confirma la selección de la tecla resaltada y la envía al buffer de texto.
- La acción de pulsación actualiza el buffer de texto y activa las sugerencias de autocompletado LSTM.

Autocompletado y Guardado

- Tras cada secuencia de 5–7 caracteres, el módulo LSTM sugiere hasta 3 palabras.
- El texto final acumulado puede guardarse en un archivo .txt pulsando el botón "Guardar".

4.- Implementación: Explicación del Código (Funciones Clave)

4.1.- Capturing_camera

- init_camera(resolution: tuple, fps: int) -> cv2.VideoCapture: Configura y retorna el objeto de captura de OpenCV con resolución 240×180 y 30 fps.
- read_frame(cap: cv2.VideoCapture) -> numpy.ndarray: Lee un frame de la cámara, lo convierte a RGB y lo retorna para su procesamiento.

4.2.- Detección de Parpadeo (detecting_eye_blink_module)

- calculate_EAR(landmarks: List[float]) -> float: Calcula el Eye Aspect Ratio (EAR) a partir de los landmarks oculares para medir apertura de ojos.
- detect_blink(ear: float, threshold: float = 0.20, consecutive_frames: int = 2) -> bool: Determina si ha ocurrido un parpadeo voluntario cuando EAR cae por debajo del umbral durante los cuadros requeridos.

4.3.- Seguimiento de Manos (tracking_hand_module.py)

- process_hand_landmarks(landmarks) -> Dict[str, bool]: Interpreta la posición de dedos para generar acciones de gestos (p. ej., para "Tab" o para "Borrar").
- map_gesture_to_key(gestures: Dict[str, bool]) -> str: Traduce la posición de la mano detectado a la tecla correspondiente.

4.4.- Autocompletado LSTM (vir_keyboard_eye_det.py)

- load_model_and_tokenizer(model_path: str, tokenizer_path: str): Carga el modelo autocomplete_es.h5 y el tokenizer.pkl para predicciones.
- text_to_sequence(text: str) -> List[int]: Convierte el buffer de texto actual en secuencia de enteros según el tokenizador.
- predict_next_words(sequence: List[int], top_k: int = 3) -> List[str]: Realiza inferencia con el modelo LSTM y devuelve las top_k palabras más probables.

4.5.- Integración y UI (vir_keyboard_eye_det.py)

- draw_keyboard(keys: List[List[str]], size: int, margin: int) -> None: Dibuja el teclado virtual en la ventana de OpenCV.
- update_highlight(cursor_pos: tuple, dwell_time: float) -> Optional[str]: Resalta la tecla bajo el cursor de la mano; mantiene un temporizador para la preselección.
- confirm_selection() -> bool: Verifica la detección de parpadeo para confirmar la tecla preseleccionada.
- main() -> None: Ciclo principal que lee frames, invoca módulos de detección, actualiza UI, gestiona buffer de texto, sugerencias de autocompletado y guarda el archivo.

5.- Pruebas: Casos de prueba

5.1.- Selección de Caracteres con Blink

- **Objetivo**: Verificar que el parpadeo confirme correctamente la tecla resaltada.
- **Procedimiento**:
    1. Ejecutar el sistema y esperar el renderizado del teclado virtual.
    2. Mover la mano para resaltar la tecla "A".
    3. Realizar un parpadeo voluntario intencional (< 0.20 EAR) durante al menos 2 frames.
    4. Comprobar que en el buffer de texto aparece la letra "A".
- **Criterio de Éxito**: La letra seleccionada coincide con la tecla resaltada en ≥ 95 % de los intentos.

5.2.- Navegación Manual y Resaltado

- **Objetivo**: Asegurar el mapeo preciso del cursor de la mano a las teclas.
- **Procedimiento**:
    1. Con la cámara activa, desplazar la mano lentamente sobre varias teclas.
    2. Verificar visualmente que el resaltado azul sigue correctamente la posición de la mano.
    3. Repetir el desplazamiento a diferentes alturas y ángulos.
- **Criterio de Éxito**: Resaltado estable y sin saltos erráticos en ≥ 90 % de los movimientos.

5.3.- Autocompletado de Palabras

- **Objetivo**: Validar las sugerencias predictivas tras ingreso de secuencias de texto.
- **Procedimiento**:
    1. Teclear manualmente (o simular con script) una secuencia conocida: "hol".
    2. Esperar a que el módulo LSTM genere sugerencias.
    3. Seleccionar una sugerencia con parpadeo.
- **Criterio de Éxito**: La sugerencia correcta aparece en top-3 y se inserta en el buffer sin errores.

5.4.- Guardado de Texto

- **Objetivo**: Comprobar funcionalidad del botón "Guardar".
- **Procedimiento**:
    1. Escribir la frase de prueba: "prueba guardado".
    2. Resaltar el botón "Guardar" con la mano.
    3. Confirmar con parpadeo.
    4. Verificar la existencia de un archivo .txt con el nombre y contenido correcto.
- **Criterio de Éxito**: El archivo se crea en la ruta esperada con contenido idéntico al buffer.

6.- Conclusiones: Limitaciones y Mejoras Futuras

6.1.- Limitaciones

- **Dependencia de Condiciones de Luz**: El rendimiento de los módulos de visión (gaze y hand tracking) puede degradarse significativamente en condiciones de iluminación extrema o con sombras intensas.
- **Latencia de Procesamiento**: En hardware con capacidades limitadas, el procesamiento en tiempo real de detección facial, manos y predicción LSTM introduce retrasos perceptibles (>200 ms), afectando la fluidez de la interacción.
- **Falsos Positivos en Parpadeo**: Movimientos involuntarios o parpadeos rápidos pueden generar selecciones no deseadas si no se ajustan correctamente los umbrales de EAR y el conteo de frames.
- **Complejidad de Uso Inicial**: Usuarios nuevos requieren un periodo de adaptación para calibrar el sistema y aprender el flujo de navegación con la mano y confirmación por parpadeo.

- **Cobertura Lingüística**: El modelo LSTM está entrenado únicamente en español y su corpus está limitado a vocabulario cotidiano; no cubre terminología técnica o jerga específica.

6.2.- Mejoras Futuras

- **Adaptación de Iluminación**: Implementar técnicas de normalización de imagen y ajuste dinámico de brillo/contraste para mejorar robustez en distintas condiciones lumínicas.
- **Optimización de Rendimiento**: Migrar procesamiento a GPU o emplear modelos más ligeros (e.g., modelos quantizados o redes neuronales eficientes) para reducir latencia.
- **Mejora de la Detección de Parpadeos**: Incorporar aprendizaje supervisado para diferenciar parpadeos voluntarios de involuntarios y ajustar dinámicamente los thresholds.
- **Calibración Automática**: Desarrollar un módulo de calibración guiada que adapte parámetros de seguimiento ocular y manual según características individuales del usuario.
- **Expansión del Corpus LSTM**: Ampliar el dataset de entrenamiento con textos técnicos, jerga especializada y neologismos, así como permitir entrenamiento incremental en línea.
- **Interfaz Personalizable**: Permitir al usuario configurar tamaños de teclas, posiciones de botones y activar/desactivar módulos (e.g., solo gaze o solo hand tracking) mediante un panel de ajustes.
- **Multi-idioma**: Integrar modelos LSTM para otros idiomas y permitir cambio dinámico de idioma en la interfaz.