**Universidad Veracruzana**

**Tecnologías de la información en las organizaciones**

**Inteligencia artificial**

**Teclado manejado con la vista**

**(EyeTacking)**

**Armando Mencias Gonzalez**

**13 de junio del 2025**

Contenido

[introducción 3](#_Toc200648712)

[Problemática 3](#_Toc200648713)

[Justificación 4](#_Toc200648714)

[Arquitectura 5](#_Toc200648715)

[Desarrollo y Prototipo 6](#_Toc200648716)

[Desarrollo 6](#_Toc200648717)

[Componentes Principales del Prototipo 7](#_Toc200648718)

[Capturas del Prototipo 8](#_Toc200648719)

[implementación 11](#_Toc200648720)

[Pruebas 17](#_Toc200648721)

[Conclusiones 20](#_Toc200648722)

# introducción

Las personas con discapacidad motriz parcial o severa a menudo enfrentan barreras significativas para interactuar con dispositivos electrónicos y comunicarse eficazmente con su entorno. La dependencia de interfaces tradicionales como el teclado y el mouse les impide acceder a información, expresar sus necesidades y participar plenamente en la sociedad digital. Si bien existen diferntes soluciones, estas suelen ser costosas, complejas o limitadas en su funcionalidad.

La visión por computadora, en particular el seguimiento ocular (eye tracking), ha demostrado ser una vía prometedora para la comunicación alternativa. Al traducir los movimientos oculares y el parpadeo en comandos de interfaz, se puede ofrecer a estas personas una forma intuitiva y natural de interactuar con la tecnología. El desafío radica en desarrollar un sistema accesible, preciso y robusto que permita la entrada de texto de manera eficiente, abriendo así nuevas vías de comunicación y autonomía.

## Problemática

**Objetivos del Proyecto**

**Objetivo General**

* Desarrollar un prototipo accesible, preciso y robusto que permita la entrada de texto de manera eficiente mediante una interfaz visual que se gestione por medio del seguimiento ocular.

**Objetivos Específicos**

1. Aplicar los fundamentos de la visión por computadora para el seguimiento ocular y la detección de parpadeo.
2. Implementar modelos de aprendizaje automático (Machine Learning) para la interpretación de movimientos oculares, la selección de caracteres y el completado de palabras.
3. Integrar diferentes módulos de IA (visión por computadora, procesamiento de lenguaje natural básico para predicción) en un prototipo funcional.
4. Evaluar la usabilidad y precisión de un prototipo de interfaz humano-computadora para personas con discapacidad.

## Justificación

Desarrollar un prototipo de interfaz visual que se utilice por medio del seguimiento ocular (eye tracking) usando técnicas de IA.

# Arquitectura

**Tabla

El contenido generado por IA puede ser incorrecto.tecnologías implementadas**

**Diagrama

El contenido generado por IA puede ser incorrecto.**

**Diagrama de secuencia**

# Desarrollo y Prototipo

## Desarrollo

El desarrollo del proyecto se organizó en etapas claramente definidas y secuenciales, comenzando con la identificación detallada de los requisitos del sistema. Este proceso inicial implicó reconocer la necesidad de crear una herramienta accesible que permita a usuarios con movilidad reducida interactuar con una computadora mediante el seguimiento ocular (eye tracking), complementado con funciones avanzadas de autocompletado predictivo. Tras establecer estos objetivos, se procedió al diseño del sistema, definiendo un esquema interactivo que consiste en una interfaz gráfica intuitiva elaborada con la biblioteca OpenCV. Este diseño se enfocó en organizar el teclado virtual de manera clara, facilitando al usuario la selección precisa de letras y otros caracteres mediante la mirada.

Interfaz de usuario gráfica, Diagrama, Aplicación, Teams

El contenido generado por IA puede ser incorrecto.

Una vez realizado el diseño preliminar, se procedió a la selección de tecnologías adecuadas para garantizar la eficiencia, precisión y accesibilidad del sistema. Se escogió Python debido a su versatilidad y soporte robusto para tareas relacionadas con visión artificial y procesamiento del lenguaje natural. OpenCV fue utilizado como núcleo gráfico y visual, proporcionando la capacidad necesaria para generar interfaces dinámicas e interactivas. Además, MediaPipe se seleccionó por su excelente rendimiento en detección ocular y reconocimiento facial, y se optó por un modelo predictivo basado en bigramas para proporcionar sugerencias contextuales que faciliten la escritura rápida y precisa.

Finalmente, se llevó a cabo una fase intensiva de integración y optimización donde se combinaron cuidadosamente el seguimiento ocular con el sistema predictivo textual. Durante esta etapa, se implementaron técnicas avanzadas de suavizado para mejorar la precisión del cursor visual y se optimizó la lógica del tiempo de selección (dwell-time) para garantizar una experiencia fluida y eficiente. Estas mejoras técnicas fueron cruciales para asegurar que el sistema pudiera adaptarse efectivamente a una variedad de usuarios, ofreciendo comodidad y precisión en tiempo real.

## Componentes Principales del Prototipo

El prototipo desarrollado integra múltiples componentes esenciales, cada uno diseñado meticulosamente para cumplir una función específica dentro del sistema general. El teclado virtual, construido con OpenCV, es uno de los elementos más visibles y fundamentales, caracterizado por una distribución clara y accesible de teclas que facilita notablemente la interacción del usuario mediante seguimiento ocular. Esta interfaz está diseñada para maximizar la precisión en la selección ocular, lo que resulta especialmente importante para usuarios con dificultades motrices.



La detección ocular (Eye Tracker), implementada a través de MediaPipe, constituye otro componente central del sistema. Este módulo emplea algoritmos avanzados para identificar puntos faciales clave y localizar con precisión la posición del iris del usuario. Esto permite una interacción exacta y dinámica, adaptándose eficazmente a diferentes condiciones de iluminación y variabilidad en los rasgos faciales de distintos usuarios. Complementando esta función, se desarrolló un sistema de calibración automática basado en cuatro puntos específicos que aseguran que cada usuario pueda ajustar rápida y fácilmente el sistema a sus características individuales.



Otro aspecto clave del prototipo es el sistema predictivo de texto, conocido como WordPredictor, que emplea técnicas de bigramas para proporcionar sugerencias contextuales relevantes mientras se está escribiendo. Este sistema mejora significativamente la rapidez y precisión del proceso de escritura, facilitando a los usuarios la composición de textos extensos con menos esfuerzo y en menos tiempo. Asimismo, el teclado cuenta con una función de síntesis de voz integrada mediante pyttsx3, proporcionando retroalimentación auditiva en español para cada tecla seleccionada, mejorando así la accesibilidad y la experiencia de usuario de manera significativa.

## Capturas del Prototipo

Las capturas del prototipo presentan visualmente aspectos fundamentales del sistema en funcionamiento real. La primera captura muestra claramente la interfaz principal del teclado virtual, destacando la organización visual y la disposición lógica de las teclas, que permite al usuario seleccionar fácilmente las letras utilizando únicamente el seguimiento ocular. La segunda captura enfatiza la precisión del seguimiento ocular, mostrando visualmente cómo el cursor responde de forma dinámica y precisa a los movimientos oculares del usuario.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Finalmente, una tercera captura ilustra claramente el funcionamiento del sistema de autocompletado predictivo, donde las sugerencias contextuales aparecen claramente durante el proceso de escritura, facilitando la composición rápida y fluida de textos. Estas imágenes combinadas ofrecen una representación visual convincente del potencial del prototipo para mejorar significativamente la accesibilidad informática para usuarios con limitaciones físicas o movilidad reducida.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

más del 90% de los casos. La función de síntesis de voz también fue evaluada, demostrando claridad y comprensión adecuada de las pronunciaciones, contribuyendo así significativamente a la accesibilidad del sistema.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

Estas pruebas confirman que el prototipo cumple adecuadamente con los requisitos planteados inicialmente, ofreciendo una interacción fluida y precisa que potencialmente puede mejorar significativamente la calidad de vida de usuarios con movilidad limitada.

# implementación

La implementación del proyecto se realizó con una estructura modular clara, donde cada función clave cumple un propósito específico y esencial dentro del sistema:

* Clase EyeTracker: Gestiona el seguimiento ocular mediante la detección del iris con MediaPipe, calibración inicial automática y determinación precisa del cursor.

*class* EyeTracker:

*def* \_\_init\_\_(*self*):

*self*.face\_mesh = mp\_face\_mesh.FaceMesh(

*max\_num\_faces*=1, *refine\_landmarks*=True,

*min\_detection\_confidence*=0.5, *min\_tracking\_confidence*=0.5)

*self*.calibration\_stage = 0

*self*.calibration\_start = 0

*self*.calibration\_points = []

*self*.calibration\_complete = False

*self*.cal\_min = (0,0)

*self*.cal\_max = (1,1)

*self*.last\_cursor = (SCREEN\_WIDTH//2, SCREEN\_HEIGHT//2)

*self*.smoothing\_window\_size = 10

*self*.cursor\_history = [*self*.last\_cursor]\**self*.smoothing\_window\_size

*def* get\_iris\_center(*self*, *lm*, *idx*):

        pts = np.array([[*lm*[i].x, *lm*[i].y] for i in *idx*])

        return pts.mean(*axis*=0)

*def* calibrate(*self*, *frame*, *gv*):

        t = time.time()

        if *self*.calibration\_stage == 0:

*self*.calibration\_stage = 1

*self*.calibration\_start = t

        targets = [

            (SCREEN\_WIDTH//4, SCREEN\_HEIGHT//4),

            (3\*SCREEN\_WIDTH//4, SCREEN\_HEIGHT//4),

            (SCREEN\_WIDTH//4, 3\*SCREEN\_HEIGHT//4),

            (3\*SCREEN\_WIDTH//4, 3\*SCREEN\_HEIGHT//4)

        ]

        x,y = targets[*self*.calibration\_stage-1]

        cv2.circle(*frame*, (x,y), 20, (0,0,255), -1)

        if t - *self*.calibration\_start > DWELL\_TIME:

*self*.calibration\_points.append(tuple(*gv*))

*self*.calibration\_stage += 1

*self*.calibration\_start = t

            winsound.Beep(500,100)

            if *self*.calibration\_stage > 4:

                xs = [p[0] for p in *self*.calibration\_points]

                ys = [p[1] for p in *self*.calibration\_points]

*self*.cal\_min = (min(xs), min(ys))

*self*.cal\_max = (max(xs), max(ys))

*self*.calibration\_complete = True

                winsound.Beep(1000,300)

*def* detect\_gaze(*self*, *frame*):

        rgb = cv2.cvtColor(*frame*, cv2.COLOR\_BGR2RGB)

        res = *self*.face\_mesh.process(rgb)

        cursor = *self*.last\_cursor

        if res.multi\_face\_landmarks:

            lm = res.multi\_face\_landmarks[0].landmark

            ir\_l = *self*.get\_iris\_center(lm, LEFT\_IRIS\_INDICES)

            ir\_r = *self*.get\_iris\_center(lm, RIGHT\_IRIS\_INDICES)

            gv = (ir\_l + ir\_r) / 2

            if not *self*.calibration\_complete:

*self*.calibrate(*frame*, gv)

            else:

                nx = (gv[0] - *self*.cal\_min[0])/(*self*.cal\_max[0]-*self*.cal\_min[0])

                ny = (gv[1] - *self*.cal\_min[1])/(*self*.cal\_max[1]-*self*.cal\_min[1])

                x = int((1-nx)\*SCREEN\_WIDTH)

                y = int(ny\*SCREEN\_HEIGHT)

*self*.cursor\_history.append((x,y))

                if len(*self*.cursor\_history) > *self*.smoothing\_window\_size:

*self*.cursor\_history.pop(0)

                sm = np.mean(*self*.cursor\_history, *axis*=0).astype(int)

                cursor = (max(0,min(sm[0],SCREEN\_WIDTH-1)),

                          max(0,min(sm[1],SCREEN\_HEIGHT-1)))

*self*.last\_cursor = cursor

        return cursor

* Clase VirtualKeyboard: Maneja la creación dinámica de teclas, la lógica para la detección y gestión del tiempo de selección (dwell-time), así como la integración de audio mediante síntesis de voz.

*class* VirtualKeyboard:

*def* \_\_init\_\_(*self*):

*self*.text = ""

*self*.keys = *self*.\_create\_keys()

*self*.shift = False

*self*.hover = None

*self*.hover\_t = 0

*self*.hover\_sugg = None

*self*.hover\_sugg\_t = 0

*self*.block = False

*self*.last\_press = 0

*self*.last\_key = None               # <--- guardamos la última tecla

*self*.suggestions = []

        if not os.path.exists('palabras.txt'):

            with open('palabras.txt','w',*encoding*='utf-8') as f:

                f.write('HOLA\nMUNDO\nPYTHON\n')

*self*.words = [l.strip().upper() for l in open('palabras.txt',*encoding*='utf-8') if l.strip()]

*self*.predictor = WordPredictor('corpus.txt', *ngram*=2)

*self*.engine = pyttsx3.init()

*self*.engine.setProperty('rate',150)

*self*.engine.setProperty('volume',1)

        for v in *self*.engine.getProperty('voices'):

            if 'spanish' in v.name.lower():

*self*.engine.setProperty('voice', v.id)

                break

*def* \_create\_keys(*self*):

        ks = []

        oy = (SCREEN\_HEIGHT - KEYBOARD\_AREA\_HEIGHT)//2

        for r,row in enumerate(KEYBOARD):

            tw = len(row)\*KEY\_SIZE + (len(row)-1)\*KEY\_MARGIN

            ox = (SCREEN\_WIDTH - tw)//2

            y  = oy + r\*(KEY\_SIZE+KEY\_MARGIN)

            for c,ch in enumerate(row):

                ks.append({'char':ch, 'pos':(ox+c\*(KEY\_SIZE+KEY\_MARGIN), y),

                           'size':(KEY\_SIZE,KEY\_SIZE), 'cd':0})

        return ks

*def* update\_cd(*self*):

        for k in *self*.keys:

            if k['cd']>0: k['cd']-=1

        if *self*.block and time.time()-*self*.last\_press > 0.2:

*self*.block = False

*def* get\_key(*self*, *cur*):

        if *cur* is None: return None

        for k in *self*.keys:

            x,y = k['pos']; w,h = k['size']

            if x<=*cur*[0]<=x+w and y<=*cur*[1]<=y+h:

                return k

        return None

*def* press(*self*, *k*):

        if not *k* or *k*['cd']>0 or *self*.block: return

        winsound.Beep(440,100)

        ch = *k*['char']

        if ch == 'Shift':

*self*.shift = not *self*.shift

        elif ch == 'SPACE':

*self*.text += ' '

        elif ch == 'Borrar':

*self*.text = *self*.text[:-1]

        else:

*self*.text += ch.upper() if *self*.shift else ch.lower()

*self*.shift = False

            # decir en voz alta

*self*.engine.say(ch)

*self*.engine.runAndWait()

        # marcar última tecla y cooldown

*self*.last\_key   = *k*

*self*.block      = True

*self*.last\_press = time.time()

*k*['cd']        = 15

        # actualizar sugerencias

        cands = *self*.predictor.predict(*self*.text.lower(), *top\_k*=3)

*self*.suggestions = [w.upper() for w in cands]

*def* update\_hover(*self*, *cur*):

        now = time.time()

        # dwell en teclas

        k = *self*.get\_key(*cur*)

        if k != *self*.hover:

*self*.hover   = k

*self*.hover\_t = now

        elif k and now - *self*.hover\_t > DWELL\_TIME:

*self*.press(k)

*self*.hover = None

        # dwell en sugerencias (igual coords que en draw\_keyboard)

        keyboard\_top = (SCREEN\_HEIGHT - KEYBOARD\_AREA\_HEIGHT)//2

        suggestion\_y = keyboard\_top - 60

        sugg = None

        for i,s in enumerate(*self*.suggestions):

            x = 50 + i\*200; y = suggestion\_y; w,h = 180,40

            if x<=*cur*[0]<=x+w and y<=*cur*[1]<=y+h:

                sugg = s

                break

        if sugg != *self*.hover\_sugg:

*self*.hover\_sugg   = sugg

*self*.hover\_sugg\_t = now

        elif sugg and now - *self*.hover\_sugg\_t > DWELL\_TIME:

            winsound.Beep(880,100)

            parts = *self*.text.rstrip().split(' ')

            parts[-1] = sugg.lower()

*self*.text = ' '.join(parts) + ' '

*self*.suggestions = []

*self*.hover\_sugg  = None

*self*.block        = True

*self*.last\_press   = now

* Clase WordPredictor: Implementa la predicción textual usando bigramas basados en un corpus específico, proporcionando sugerencias contextuales en tiempo real.

*class* WordPredictor:

*def* \_\_init\_\_(*self*, *path\_txt*, *ngram*=2):

*self*.ngram = *ngram*

*self*.model = defaultdict(Counter)

*self*.unigram = Counter()

*self*.\_build\_model(*path\_txt*)

*def* \_tokenize(*self*, *text*):

        return re.findall(*r*'\b\w+\b', *text*.lower(), *flags*=re.UNICODE)

*def* \_build\_model(*self*, *path\_txt*):

        if not os.path.exists(*path\_txt*):

            return

        text = open(*path\_txt*, *encoding*='utf-8').read()

        tokens = *self*.\_tokenize(text)

*self*.unigram.update(tokens)

        for i in range(len(tokens)-(*self*.ngram-1)):

            key = tuple(tokens[i:i+*self*.ngram-1])

*self*.model[key][tokens[i+*self*.ngram-1]] += 1

*def* predict(*self*, *prefix*, *top\_k*=3):

        toks = *self*.\_tokenize(*prefix*)

        key = tuple(toks[-(*self*.ngram-1):]) if len(toks)>=*self*.ngram-1 else tuple(toks)

        counter = *self*.model[key]

        if counter:

            return [w for w,\_ in counter.most\_common(*top\_k*)]

        last = toks[-1] if toks else ''

        candidates = [(w,c) for w,c in *self*.unigram.items() if w.startswith(last)]

        candidates.sort(*key*=*lambda* *x*: *x*[1], *reverse*=True)

        return [w for w,\_ in candidates[:*top\_k*]]

Estas funciones se integran eficientemente para ofrecer un sistema completo, interactivo y accesible, optimizado para una amplia gama de usuarios.

# Pruebas

Para validar la eficacia del sistema, se llevaron a cabo diversas pruebas enfocadas en la precisión del seguimiento ocular (y antes de iniciar el teclado se te solocita una “calibración”), la interacción con el teclado virtual y el sistema de autocompletado predictivo.

Hombre con barba y bigote

El contenido generado por IA puede ser incorrecto.

Se desarrollaron casos de prueba específicos como la selección de caracteres individuales mediante la mirada, midiendo el tiempo requerido para una selección efectiva y validando la precisión del cursor respecto a la posición ocular real del usuario.

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.

Se determinó que el tiempo óptimo de selección (dwell-time) fue efectivamente de un segundo (puede llegar a variar por la maquina), proporcionando un equilibrio entre rapidez y precisión.

*def* update\_hover(*self*, *cur*):

        now = time.time()

        # dwell en teclas

        k = *self*.get\_key(*cur*)

        if k != *self*.hover:

*self*.hover   = k

*self*.hover\_t = now

        elif k and now - *self*.hover\_t > DWELL\_TIME:

*self*.press(k)

*self*.hover = None

        # dwell en sugerencias (igual coords que en draw\_keyboard)

        keyboard\_top = (SCREEN\_HEIGHT - KEYBOARD\_AREA\_HEIGHT)//2

        suggestion\_y = keyboard\_top - 60

        sugg = None

        for i,s in enumerate(*self*.suggestions):

            x = 50 + i\*200; y = suggestion\_y; w,h = 180,40

            if x<=*cur*[0]<=x+w and y<=*cur*[1]<=y+h:

                sugg = s

                break

        if sugg != *self*.hover\_sugg:

*self*.hover\_sugg   = sugg

*self*.hover\_sugg\_t = now

        elif sugg and now - *self*.hover\_sugg\_t > DWELL\_TIME:

            winsound.Beep(880,100)

            parts = *self*.text.rstrip().split(' ')

            parts[-1] = sugg.lower()

*self*.text = ' '.join(parts) + ' '

*self*.suggestions = []

*self*.hover\_sugg  = None

*self*.block        = True

*self*.last\_press   = now

Interfaz de usuario gráfica

El contenido generado por IA puede ser incorrecto.Adicionalmente, se realizaron pruebas exhaustivas del autocompletado predictivo usando textos previamente definidos y corpus lingüísticos específicos para evaluar la precisión y relevancia de las sugerencias generadas.

Durante estas pruebas, se comprobó la capacidad del sistema para ofrecer sugerencias contextuales acertadas en Para evaluar el funcionamiento del sistema, se realizaron pruebas específicas relacionadas con la selección de caracteres y el autocompletado predictivo.

En cuanto al autocompletado de palabras, se realizaron pruebas utilizando palabras predeterminadas y un corpus amplio en español para evaluar la precisión de las sugerencias. También se probó la rapidez con la cual las sugerencias aparecen en pantalla tras escribir parcialmente una palabra, observándose una latencia mínima, lo cual mejora significativamente la fluidez del proceso de escritura.

Estas pruebas confirman la efectividad del prototipo, evidenciando que cumple satisfactoriamente con los objetivos planteados, proporcionando una solución eficiente y accesible para usuarios con limitaciones de movilidad.

# Conclusiones

El prototipo desarrollado ha demostrado ser una herramienta efectiva, precisa y accesible, ofreciendo una alternativa viable para usuarios con movilidad limitada. Sin embargo, se identificaron diversas limitaciones durante el proceso de desarrollo y pruebas que pueden ser abordadas en futuras mejoras. Una de estas limitaciones radica en la sensibilidad del sistema a variaciones ambientales, particularmente en condiciones extremas de iluminación, que ocasionalmente afectan la precisión del seguimiento ocular. Para mitigar este problema, futuras actualizaciones podrían integrar algoritmos más robustos de adaptación a cambios lumínicos o utilizar sensores adicionales.

Otra limitación observada fue la relativa dependencia del sistema en la calidad del corpus lingüístico utilizado para el autocompletado predictivo. Aunque el modelo de bigramas mostró un desempeño bueno, existe un margen significativo para mejorar la precisión y relevancia de las sugerencias mediante el uso de modelos más avanzados de procesamiento del lenguaje natural como redes neuronales recurrentes (LSTM) o transformers.

Finalmente, se recomienda ampliar las opciones de accesibilidad, explorando la integración con otros métodos de interacción como reconocimiento de gestos faciales o comandos de voz avanzados. Estas mejoras no solo aumentarían la autonomía del usuario sino que también facilitarían una experiencia más personalizada y satisfactoria. En resumen, aunque el prototipo actual cumple satisfactoriamente con los objetivos iniciales, estas limitaciones y oportunidades de mejora delinean claramente las direcciones futuras para continuar perfeccionando el sistema.