PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

FACULTAD DE INGENIERÍA

SISTEMAS DE INFORMACIÓN

PROYECTO DE TITULACIÓN

DESARROLLO DE UN PROTOTIPO DE UN TRADUCTOR DE LENGUAJE DE SEÑAS

AUTOR:

CARLOS ALEJANDRO RIVADENEIRA LANDÁZURI

QUITO, 2024

DEDICATORIA

Dedico este trabajo a mi familia, amigos y personas cercanas que siempre estuvieron presentes y atentos de mi camino como estudiante. Cada paso logrado hasta ahora, junto con su apoyo y sacrificio siempre ha sido un pilar importante en mi crecimiento como persona y como profesional.

Haciendo énfasis en mi abuelita María y mi tía Margarita, que siempre se esforzaron de sobremanera porque nunca me falte nada, esperando un día poder corresponder todo lo que hicieron por mí. Agradezco siempre su presencia, su vida y su esencia, que fueron claves para seguir queriendo ser mejor un día más y llevándome por un camino tan digno como pudieron, habiendo altos y bajos.

AGRADECIMIENTO

Agradezco principalmente a mis querida tía y abuelita, que siempre han estado presentes a lo largo de mi vida dando su esfuerzo diario por poder darme una buena vida sin esperar nada a cambio. Agradezco su total entrega y confianza en mí y en la persona que soy ahora tanto personal como profesionalmente.

También agradezco a mis hermanos, que siempre han sido mi motor para salir adelante. Los agradezco por ser la pieza fundamental para ser un ejemplo y llevar mi vida de la mejor manera posible con el fin de poder estar presente en cualquier sentido y aspecto en sus vidas.

RESUMEN

En el marco del desarrollo de un prototipo de sistema de traducción de lenguaje de señas para mejorar la accesibilidad y la inclusión digital de personas con discapacidades auditivas y de habla, se ha llevado a cabo un enfoque integral y estructurado para diseñar y construir este sistema. Este proyecto de titulación se centra en la creación de un prototipo funcional que permita traducir gestos y señas en texto interpretable por dispositivos tecnológicos, facilitando la comunicación y la interacción con herramientas digitales para personas sordomudas.

Metodología y Enfoque

La metodología utilizada en este proyecto se basa en Extreme Programming (XP), una metodología ágil que se enfoca en la entrega rápida y continua de software de alta calidad. El proyecto se ha desarrollado en varias iteraciones, cada una con objetivos específicos y entregables definidos. La primera iteración se centró en la configuración inicial del proyecto y la implementación de la interfaz de usuario básica, permitiendo a los usuarios seleccionar entre números, letras y frases. La segunda iteración abordó el desarrollo del algoritmo de reconocimiento de gestos utilizando OpenCV y TensorFlow/PyTorch, mientras que la tercera iteración se enfocó en la integración de todos los componentes del sistema y la realización de pruebas exhaustivas.

Implementación Técnica

El sistema se ha desarrollado utilizando Python como lenguaje de programación principal, aprovechando su simplicidad y versatilidad. Se han utilizado bibliotecas especializadas como OpenCV para el procesamiento de imágenes y TensorFlow/PyTorch para el aprendizaje automático. La interfaz de usuario se ha creado utilizando Tkinter y PyQt, proporcionando una experiencia de usuario intuitiva y accesible.

Algoritmo de Reconocimiento de Gestos

El núcleo del sistema es el algoritmo de reconocimiento de gestos, que se ha desarrollado utilizando técnicas avanzadas de visión por computadora y aprendizaje automático. El algoritmo captura los movimientos de las manos y los gestos faciales

utilizando una cámara, y luego procesa estas imágenes para identificar las señas realizadas por el usuario. Se han implementado diferentes controladores para reconocer números, letras y frases, cada uno utilizando umbrales y características específicas para identificar correctamente los gestos.

Interfaz de Usuario

La interfaz de usuario permite a los usuarios seleccionar entre diferentes módulos de traducción (números, letras y frases) y proporciona una visualización en tiempo real de los gestos reconocidos. Se han realizado pruebas de usabilidad con usuarios finales para asegurar que la interfaz sea fácil de usar y efectiva en la traducción de señas.

Resultados y Validación

Se han llevado a cabo pruebas exhaustivas para validar la precisión y eficacia del sistema. Los resultados de las pruebas indican una alta precisión en el reconocimiento de gestos y una satisfacción positiva de los usuarios. La retroalimentación recibida ha sido utilizada para realizar ajustes y mejoras en el sistema, asegurando que cumpla con las necesidades y expectativas de la comunidad sordomuda.

Conclusiones y Recomendaciones

El desarrollo de este prototipo ha demostrado ser un paso significativo hacia la mejora de la accesibilidad y la inclusión digital para personas con discapacidades auditivas y de habla. La utilización de técnicas avanzadas de visión por computadora y aprendizaje automático ha permitido crear un sistema efectivo y preciso para la traducción de lenguaje de señas. Se recomienda continuar con el desarrollo y mejora del sistema, incorporando nuevas funcionalidades y optimizando el rendimiento. Además, se sugiere explorar la posibilidad de integrar el sistema con otros dispositivos y plataformas para ampliar su alcance y utilidad.

ÍNDICE

Tabla de contenido

1. CAPÍTULO I: INTRODUCCIÓN

El presente proyecto de titulación se centra en desarrollar un prototipo de sistema de traducción de lenguaje de señas para mejorar la accesibilidad y la inclusión digital de personas con discapacidades auditivas y de habla. A pesar de los avances tecnológicos, estas personas enfrentan barreras significativas para interactuar con dispositivos tecnológicos, limitando su capacidad de comunicarse y aprovechar herramientas digitales. Este sistema busca superar dichas barreras al traducir señas en texto interpretable por dispositivos, permitiendo la realización de tareas cotidianas como escribir mensajes y comandos. El objetivo es promover la autonomía y la inclusión digital de esta comunidad, mejorando su calidad de vida y facilitando su participación en la sociedad digital.

1.1. Justificación

En el contexto actual, la tecnología desempeña un papel crucial en la vida cotidiana, y es fundamental asegurar que todos, incluidos aquellos con discapacidades auditivas y de habla, tengan acceso igualitario a las herramientas digitales. Las personas sordomudas enfrentan barreras significativas al interactuar con dispositivos tecnológicos, lo que limita su capacidad para comunicarse y utilizar plenamente estas tecnologías. La creación de un prototipo de sistema de traducción de lenguaje de señas se justifica por la urgente necesidad de superar estas barreras y promover la inclusión digital. Este sistema permitirá a las personas sordomudas traducir sus señas en texto interpretable por dispositivos tecnológicos, facilitando tareas cotidianas como la escritura de mensajes y la ejecución de comandos. Al abordar estas necesidades, el proyecto no solo mejora la accesibilidad y la autonomía de esta comunidad, sino que también contribuye a una sociedad más inclusiva y equitativa, donde todos pueden beneficiarse de los avances tecnológicos.

1.2. Planteamiento del problema.

Las personas con discapacidades auditivas y de habla enfrentan barreras significativas al interactuar con dispositivos tecnológicos, lo que limita su capacidad para comunicarse efectivamente y aprovechar las herramientas digitales disponibles. A pesar de los avances tecnológicos, la falta de soluciones adaptadas a sus necesidades específicas impide su plena

inclusión digital. Este problema se manifiesta en la incapacidad de utilizar funciones basadas en voz para tareas de escritura o control, restringiendo su acceso a la información y su participación en la sociedad digital.

En función de estas problemáticas, se plantean las siguientes preguntas principales:

- ¿Cómo se puede desarrollar un sistema eficiente de traducción de lenguaje de señas que permita a las personas sordomudas interactuar con dispositivos tecnológicos?
- ¿Qué tecnologías de reconocimiento de gestos son más adecuadas para este propósito y cómo pueden ser implementadas de manera efectiva?

Y las siguientes preguntas secundarias:

- ¿Cuáles son las principales barreras tecnológicas que enfrentan las personas con discapacidades auditivas y de habla al intentar interactuar con dispositivos digitales?
- ¿Qué nivel de precisión y usabilidad se requiere para que un sistema de traducción de lenguaje de señas sea considerado efectivo por la comunidad sordomuda?
- ¿Qué metodologías de diseño y desarrollo son más adecuadas para crear un sistema de traducción de lenguaje de señas inclusivo y accesible?

1.3. Objetivos.

1.3.1. Objetivo General.

Desarrollar un prototipo de sistema de traducción de lenguaje de señas que permita a las personas con discapacidades auditivas y de habla interactuar con dispositivos tecnológicos, mejorando su accesibilidad y fomentando su inclusión en la sociedad digital, utilizando Python como lenguaje de programación, Visual Studio Code y PyCharm como entornos de desarrollo integrado, y la metodología de desarrollo XP.

1.3.2. Objetivos Específicos.

- Identificar requerimientos: Estudiar las necesidades comunicativas y tecnológicas de las personas sordomudas para definir los requisitos funcionales del sistema.

- Realizar el diseño del prototipo: Diseñar la arquitectura del sistema y la interfaz de usuario, centrada en la accesibilidad y usabilidad.
- Aplicar la metodología XP: Implementar el prototipo utilizando la metodología de desarrollo XP, permitiendo iteraciones rápidas basadas en la retroalimentación de usuarios.
- Desarrollar el algoritmo de reconocimiento: Desarrollar un algoritmo en Python capaz de traducir señas del lenguaje gestual en texto interpretable por dispositivos tecnológicos.
- Realizar pruebas y validación: Realizar pruebas con usuarios pertenecientes a la comunidad sordomuda para evaluar la eficacia, precisión y usabilidad del sistema, y realizar los ajustes necesarios.
- Generar reportes de usabilidad: Crear informes detallados sobre los resultados de las pruebas de usabilidad, destacando las áreas de éxito y las oportunidades de mejora.

1.4. Alcance.

Este proyecto se enfocará en desarrollar un prototipo inicial de un sistema de traducción de lenguaje de señas que funcione localmente junto con una cámara integrada. El prototipo permitirá la traducción de señas en tiempo real, con las palabras traducidas mostrándose en pantalla o escribiéndose en un archivo de texto. El desarrollo incluirá la implementación de un algoritmo de reconocimiento de gestos, una interfaz de usuario accesible, y la integración con una cámara estándar para la captura de movimientos. Este sistema proporcionará una base sólida para mejorar la accesibilidad y la inclusión digital de las personas con discapacidades auditivas y de habla, sentando las bases para futuras expansiones y mejoras, como la inclusión de otros lenguajes de señas y funcionalidades adicionales.

2. CAPÍTULO II: MARCO TEÓRICO

En este capítulo se proporcionarán los conceptos principales que sustentan el desarrollo del proyecto del sistema de traducción de lenguaje de señas. A lo largo de este capítulo, se explorará una serie de aspectos teóricos y conceptuales fundamentales que son esenciales para comprender la problemática que enfrenta la comunidad de personas con discapacidades auditivas y de habla y la solución propuesta. Cada sección se enfoca en un elemento clave que influye en el diseño y la implementación del prototipo de sistema de traducción de lenguaje de señas destinado a mejorar la accesibilidad y la inclusión digital.

2.1. Contexto del Proyecto

Este proyecto se desarrolla con el objetivo de beneficiar a la comunidad sordomuda en general, facilitando su interacción con dispositivos tecnológicos mediante un sistema de traducción de lenguaje de señas. La investigación y el desarrollo del prototipo no están limitados a un entorno específico, sino que buscan proporcionar una solución ampliamente aplicable en diversos contextos, como el hogar, el trabajo, la educación y otros entornos sociales.

El proyecto se basa en una comprensión profunda de las necesidades comunicativas y tecnológicas de las personas sordomudas, obtenida a través de investigaciones exhaustivas y consultas con expertos en accesibilidad y miembros de la comunidad sordomuda. Al enfocarse en una solución que funcione localmente con dispositivos comunes, como cámaras integradas en smartphones y computadoras, el proyecto busca garantizar una accesibilidad amplia y una implementación práctica en la vida cotidiana de las personas con discapacidades auditivas y de habla.

El desarrollo de este prototipo se centra en crear una herramienta efectiva y accesible que permita a las personas sordomudas traducir sus señas en texto interpretable por dispositivos tecnológicos, promoviendo así su autonomía y mejorando su inclusión digital. Este enfoque inclusivo y accesible es fundamental para asegurar que todos puedan beneficiarse de los avances tecnológicos y participar plenamente en la sociedad digital.

2.2. Traducción Automática de Lenguaje de Señas

La traducción automática de lenguaje de señas es un campo interdisciplinario que combina la visión por computadora, el procesamiento de imágenes y el aprendizaje automático para interpretar y convertir gestos de señas en texto o voz. Este proceso es esencial para mejorar la comunicación y la inclusión de personas con discapacidades auditivas y de habla en la sociedad.

El uso de cámaras y algoritmos de visión por computadora permite capturar los movimientos de las manos y los gestos faciales, que luego son procesados por modelos de aprendizaje automático para reconocer las señas. Según (García, 2020), “la traducción de señas en tiempo real presenta desafíos significativos, tales como la variabilidad de los gestos, la necesidad de reconocimiento preciso en diferentes condiciones de iluminación y la capacidad de comprender contextos complejos. Sin embargo, los avances en la inteligencia artificial están allanando el camino para soluciones más robustas y precisas”.

La implementación de un sistema local para la traducción de señas involucra el desarrollo de una infraestructura tecnológica que incluya hardware adecuado (como cámaras de alta resolución) y software especializado para el procesamiento de imágenes y la interpretación de gestos. Este sistema debe ser capaz de operar en tiempo real, proporcionando traducciones instantáneas que se muestren en pantalla o se registren en un archivo de texto, facilitando la comunicación efectiva para los usuarios.

La eficiencia en la traducción automática de señas no solo depende de la precisión del reconocimiento de gestos, sino también de la capacidad del sistema para adaptarse a diferentes dialectos y variaciones regionales del lenguaje de señas.

2.3. Metodología de desarrollo XP

La metodología de desarrollo XP (Extreme Programming) es una metodología ágil que se enfoca en la mejora continua y la adaptabilidad a las necesidades cambiantes del cliente. Su objetivo es entregar software de alta calidad de manera rápida y eficiente, asegurando que el equipo de desarrollo esté alineado con las expectativas del usuario final. Según Fernández Escribano (2009), XP busca "aumentar la productividad al desarrollar software con una fuerte arquitectura, intentando sacar productos al mercado rápidamente, con gran

calidad y motivando al equipo de trabajo para seguir mejorando esta tendencia". Los principios clave de esta metodología incluyen:

- **Colaboración constante con el usuario:** Los clientes deben estar disponibles para interactuar con el equipo de desarrollo de manera regular, asegurando una comunicación fluida y una comprensión precisa de los requerimientos del usuario. Para este proyecto, esto implica la colaboración continua con personas sordomudas y expertos en lenguaje de señas para asegurar que el sistema cumpla con sus necesidades.
- **Estándares de codificación:** La adherencia a estándares de codificación uniformes es crucial para mantener la coherencia y la calidad del código. Esto facilita el mantenimiento y la escalabilidad del sistema de traducción de lenguaje de señas.
- **Pruebas automatizadas:** XP promueve la creación de pruebas automatizadas antes de la implementación del código de producción. Esto garantiza que el código funcione como se espera y permite identificar y corregir errores de manera temprana. En el contexto de este proyecto, se desarrollarán pruebas automatizadas para validar cada componente del sistema de reconocimiento de gestos y traducción de señas.
- **Integración continua:** La integración continua del código en un repositorio compartido asegura que los cambios realizados por diferentes miembros del equipo se combinen y prueben regularmente, reduciendo el riesgo de conflictos y errores. Esto es esencial para mantener la cohesión en el desarrollo del prototipo.
- **Frecuencia en la integración:** XP enfatiza la integración frecuente del código, idealmente varias veces al día. Esto permite detectar problemas de manera temprana y entregar versiones funcionales del software de forma regular. Para el sistema de traducción de señas, esto facilita la iteración rápida y la incorporación de mejoras continuas.
- **Propiedad colectiva del código:** En XP, todo el equipo es responsable del código. Esto fomenta la colaboración y asegura que todos los miembros puedan revisar, mantener y mejorar el código del sistema de traducción de señas.
- **Optimización post-funcionalidad:** En lugar de optimizar el código desde el principio, XP recomienda centrarse primero en la funcionalidad y la calidad. Las optimizaciones se realizan en una etapa posterior para no comprometer la

productividad inicial. En este proyecto, esto significa asegurar primero una traducción precisa y funcional antes de optimizar el rendimiento.

- **Equilibrio entre trabajo y vida personal:** XP valora el equilibrio y desalienta las horas extras excesivas, ya que esto puede afectar negativamente la calidad del trabajo y el bienestar del equipo. Mantener un ritmo sostenible es fundamental para el éxito a largo plazo del proyecto.

Ciclo de Vida de la Metodología Extreme Programming

El ciclo de vida de la metodología XP consta de varias fases que se aplican de manera iterativa y continua a lo largo del proyecto:

- **Fase de Planificación:** Los clientes describen las historias de usuario que son prioritarias para la primera entrega del producto. Se evalúan las tecnologías y se exploran las opciones arquitectónicas, construyendo un prototipo inicial. En esta fase, se realiza el diseño de alto nivel y se comienza a definir la arquitectura del sistema de traducción de señas.
- **Fase de Diseño:** Se seleccionan las historias de usuario para la iteración inicial y se estiman los tiempos y recursos necesarios. Para el prototipo de traducción de señas, esto incluye definir las funcionalidades clave y diseñar una interfaz de usuario accesible y amigable.
- **Fase de Codificación:** Los desarrolladores implementan las historias de usuario seleccionadas. La programación en parejas es una práctica común en XP, donde dos desarrolladores trabajan juntos en el mismo código para asegurar su calidad y coherencia. Esto es especialmente importante para el desarrollo del sistema de traducción de señas.
- **Fase de Pruebas:** Se realizan pruebas exhaustivas para garantizar que el sistema funcione correctamente y cumpla con los requisitos definidos. Entre las pruebas más importantes se encuentran:
    - **Pruebas de aceptación:** Realizadas por los usuarios finales para verificar que el software cumple con sus requisitos. En este proyecto, miembros de la comunidad sordomuda llevarán a cabo estas pruebas para asegurar que el sistema sea útil y efectivo.

- **Pruebas unitarias:** Verifican que cada unidad de código funcione correctamente. Estas pruebas son esenciales para validar cada componente del sistema de reconocimiento de gestos y traducción de señas.

El ciclo de desarrollo culmina con la implementación del sistema. En esta etapa, el software se despliega y se configura para su uso en un entorno real. Esto incluye la instalación en dispositivos de los usuarios, la realización de ajustes necesarios y la provisión de soporte y capacitación para garantizar un uso efectivo del sistema. La comunicación con los usuarios es fundamental durante esta fase para asegurar que comprendan cómo utilizar las nuevas características y se sientan cómodos con la tecnología.

2.4. Desarrollo del Sistema de Traducción de Lenguaje de Señas

El desarrollo del sistema de traducción de lenguaje de señas se puede dividir en dos componentes principales: el desarrollo del algoritmo de reconocimiento de gestos y la interfaz de usuario que interactúa con los usuarios finales. Estos componentes trabajan en conjunto para proporcionar una solución efectiva y accesible para la comunidad sordomuda.

2.4.1. Algoritmo de Reconocimiento de Gestos

El algoritmo de reconocimiento de gestos es la parte central del sistema que se encarga de interpretar los movimientos de las manos y los gestos faciales capturados por la cámara. Este componente es fundamental para convertir las señas en texto que puede ser entendido por dispositivos tecnológicos.

El desarrollo del algoritmo incluye varias etapas:

- **Captura de datos:** Utilizando cámaras de alta resolución, se capturan los movimientos de las manos y los gestos faciales en tiempo real. Estas imágenes se procesan para extraer características relevantes que describen cada seña.
- **Procesamiento de imágenes:** Las imágenes capturadas se preprocesan para mejorar la calidad y extraer características clave. Esto puede incluir la normalización de la iluminación, la eliminación de ruido y el enfoque en las regiones de interés.
- **Modelos de aprendizaje automático:** Se emplean modelos de aprendizaje automático, como redes neuronales convolucionales (CNN), para entrenar el

sistema a reconocer diferentes señas. Estos modelos aprenden a partir de un conjunto de datos etiquetados de gestos y señas.

- **Validación y pruebas:** El algoritmo se valida utilizando datos de prueba para asegurarse de que pueda reconocer correctamente las señas en diversas condiciones. Se realizan ajustes y optimizaciones para mejorar la precisión y la eficiencia del sistema.

2.4.2. Interfaz de Usuario

La interfaz de usuario (UI) es el componente con el cual los usuarios interactúan directamente. Esta interfaz debe ser accesible y fácil de usar, permitiendo a los usuarios sordomudos utilizar el sistema de manera efectiva.

- **Diseño de la interfaz:** La interfaz se diseña para ser intuitiva y accesible. Se utilizan principios de diseño centrado en el usuario para asegurar que las funciones sean fácilmente accesibles y comprensibles. Esto incluye botones grandes y claros, menús simples y opciones de configuración personalizables.
- **Desarrollo de la interfaz:** La interfaz se desarrolla utilizando herramientas y lenguajes de programación adecuados. En este proyecto, se puede utilizar Python junto con bibliotecas como Tkinter o PyQt para crear una interfaz gráfica de usuario (GUI) que funcione en múltiples plataformas.
- **Integración con el algoritmo:** La interfaz se integra con el algoritmo de reconocimiento de gestos para proporcionar una experiencia fluida. Los resultados del reconocimiento de señas se muestran en tiempo real en la pantalla o se guardan en un archivo de texto.
- **Pruebas de usabilidad:** Se realizan pruebas de usabilidad con usuarios finales para asegurar que la interfaz sea efectiva y fácil de usar. La retroalimentación de los usuarios se utiliza para hacer mejoras y ajustar la interfaz según sea necesario.

2.5. Lenguaje de programación Python

Para el desarrollo del prototipo de un sistema de traducción de lenguaje de señas, se ha seleccionado Python como el lenguaje de programación principal. Python es ampliamente reconocido por su simplicidad, versatilidad y la robusta comunidad de desarrolladores que lo respalda. Según Van Rossum y Drake (2010), Python es "un lenguaje de programación de alto nivel, interpretado, interactivo y orientado a objetos, conocido por su sintaxis clara y su capacidad para mejorar la productividad del desarrollador".

2.5.1. Ventajas de Python

Python ofrece múltiples ventajas que lo hacen ideal para el desarrollo de aplicaciones de reconocimiento de gestos y traducción de señas:

- **Facilidad de Uso y Sintaxis Clara:** La sintaxis de Python es sencilla y fácil de leer, lo que permite a los desarrolladores escribir código de manera eficiente y con menos errores. Esto es particularmente útil en proyectos de investigación y prototipado rápido, como el sistema de traducción de señas, donde la iteración y la experimentación son cruciales.
- **Amplia Gama de Bibliotecas y Frameworks:** Python cuenta con una extensa colección de bibliotecas y frameworks que facilitan el desarrollo de aplicaciones complejas. Bibliotecas como OpenCV para el procesamiento de imágenes y TensorFlow o PyTorch para el aprendizaje automático son esenciales para implementar el reconocimiento de gestos en tiempo real.
- **Compatibilidad y Portabilidad:** Python es compatible con múltiples plataformas, incluyendo Windows, macOS y Linux, lo que permite desarrollar aplicaciones que pueden ejecutarse en diversos entornos sin necesidad de modificaciones significativas.
- **Comunidad Activa y Recursos Abundantes:** La comunidad de Python es una de las más grandes y activas en el mundo de la programación. Esto significa que los desarrolladores tienen acceso a una vasta cantidad de recursos, tutoriales, foros y documentación, lo cual es invaluable para resolver problemas y mejorar el desarrollo del sistema.

2.5.2. Aplicación de Python en el Proyecto

En el contexto del sistema de traducción de lenguaje de señas, Python se utiliza para desarrollar tanto el backend del sistema como la interfaz de usuario. Aquí se detallan algunas de las aplicaciones específicas:

- **Procesamiento de Imágenes y Reconocimiento de Gestos:** Utilizando bibliotecas como OpenCV, el sistema captura y procesa las imágenes de las señas realizadas por el usuario. Luego, emplea modelos de aprendizaje automático, desarrollados con TensorFlow o PyTorch, para interpretar estos gestos y traducirlos en texto.
- **Interfaz de Usuario:** La interfaz gráfica del sistema se desarrolla utilizando Tkinter o PyQt, permitiendo una interacción sencilla y accesible para los usuarios sordomudos. Esta interfaz muestra en tiempo real las traducciones de las señas y ofrece opciones de configuración personalizables.
- **Integración de Componentes:** Python facilita la integración de los diferentes componentes del sistema, asegurando que el procesamiento de imágenes, el reconocimiento de gestos y la presentación de resultados trabajen de manera armoniosa y eficiente.

2.5.3. Consideraciones Técnicas

Algunas consideraciones técnicas clave para el uso de Python en este proyecto incluyen:

- **Rendimiento:** Aunque Python es conocido por su facilidad de uso, puede no ser tan rápido como otros lenguajes de programación en ciertos casos. Sin embargo, para el prototipo inicial, su rapidez de desarrollo y la disponibilidad de bibliotecas optimizadas lo hacen una elección adecuada.
- **Escalabilidad:** Python permite la escalabilidad del sistema, permitiendo agregar más funciones y mejorar el rendimiento con optimizaciones futuras. La comunidad activa y el soporte continuo aseguran que Python seguirá evolucionando y mejorando.

2.6. Herramientas y Bibliotecas de Desarrollo

Para el desarrollo del sistema de traducción de lenguaje de señas, se han seleccionado varias herramientas y bibliotecas que son fundamentales para asegurar la eficiencia y la funcionalidad del prototipo. Estas herramientas facilitan el desarrollo de algoritmos de

reconocimiento de gestos, la creación de una interfaz de usuario intuitiva y la integración de diferentes componentes del sistema.

2.6.1. OpenCV

OpenCV (Open Source Computer Vision Library) es una biblioteca de visión por computadora de código abierto que proporciona una gran cantidad de funciones para el procesamiento de imágenes y videos. Es ampliamente utilizada en aplicaciones de reconocimiento de gestos debido a su capacidad para capturar y analizar movimientos en tiempo real.

- **Funciones de Procesamiento de Imágenes:** OpenCV ofrece herramientas avanzadas para la captura, filtrado y análisis de imágenes. Estas funciones son esenciales para extraer características clave de las imágenes de señas.
- **Compatibilidad Multiplataforma:** La biblioteca es compatible con múltiples plataformas, lo que permite el desarrollo de aplicaciones que pueden ejecutarse en diversos entornos.
- **Comunidad y Soporte:** OpenCV cuenta con una comunidad activa y una amplia documentación, lo que facilita la resolución de problemas y la implementación de nuevas funcionalidades.

2.6.2. TensorFlow y PyTorch

TensorFlow y PyTorch son bibliotecas de aprendizaje automático que se utilizan para desarrollar modelos de inteligencia artificial capaces de reconocer y traducir gestos de señas en tiempo real.

- **Modelos de Aprendizaje Profundo:** Ambas bibliotecas ofrecen herramientas para construir y entrenar redes neuronales convolucionales (CNN), que son cruciales para el reconocimiento de patrones en imágenes.
- **Flexibilidad y Escalabilidad:** Tanto TensorFlow como PyTorch son altamente flexibles y escalables, permitiendo ajustar y optimizar modelos según las necesidades del proyecto.
- **Amplio Ecosistema:** Estas bibliotecas cuentan con un amplio ecosistema de herramientas y recursos que facilitan el desarrollo, la capacitación y la implementación de modelos de aprendizaje automático.

2.6.3. Tkinter y PyQt

Tkinter y PyQt son bibliotecas para el desarrollo de interfaces gráficas de usuario (GUI) en Python. Estas bibliotecas permiten crear aplicaciones con interfaces intuitivas y accesibles para los usuarios.

- **Diseño de Interfaces Gráficas:** Tkinter y PyQt proporcionan herramientas para diseñar y desarrollar interfaces de usuario que son fáciles de usar y personalizar.
- **Compatibilidad con Múltiples Plataformas:** Ambas bibliotecas permiten desarrollar aplicaciones que pueden ejecutarse en diferentes sistemas operativos, asegurando una experiencia de usuario consistente.
- **Documentación y Comunidad:** Estas herramientas cuentan con una extensa documentación y una comunidad activa, lo que facilita la resolución de problemas y la implementación de nuevas funcionalidades.

2.7. Gestión de Datos y Almacenamiento

En el desarrollo del sistema de traducción de lenguaje de señas, la gestión de datos y el almacenamiento de los resultados traducidos es una parte esencial. Aunque no se utiliza un sistema de administración de bases de datos como MySQL, es fundamental asegurarse de que los datos se almacenen de manera eficiente y segura. En este proyecto, se utilizarán archivos locales para el almacenamiento de los datos traducidos y los resultados de las sesiones de reconocimiento de señas.

2.7.1. Archivos Locales

El uso de archivos locales para el almacenamiento de datos es una alternativa sencilla y eficaz para proyectos que no requieren la complejidad y el costo de un sistema de bases de datos completo.

- **Archivos de Texto:** Los resultados traducidos de las señas se almacenarán en archivos de texto simples (.txt). Estos archivos permiten almacenar los datos en un formato legible y fácilmente accesible.
- **JSON y CSV:** Para estructuras de datos más complejas, se pueden utilizar formatos como JSON (JavaScript Object Notation) y CSV (Comma-Separated Values). Estos

formatos son ampliamente soportados en Python y permiten una estructuración clara y organizada de los datos.

2.7.2. Ventajas de Usar Archivos Locales

El almacenamiento de datos en archivos locales ofrece varias ventajas, especialmente para prototipos y proyectos pequeños:

- **Simplicidad y Facilidad de Uso:** No se requiere configuración adicional ni administración de bases de datos. El manejo de archivos en Python es directo y puede realizarse con bibliotecas estándar como os, json y csv.
- **Costo:** No hay necesidad de invertir en soluciones de bases de datos, lo que hace que esta opción sea económica y accesible.
- **Portabilidad:** Los archivos de datos pueden ser fácilmente trasladados entre diferentes sistemas y plataformas, facilitando el desarrollo y las pruebas en diversos entornos.

2.7.3. Implementación en Python

Python ofrece varias bibliotecas y funciones integradas para trabajar con archivos locales de manera eficiente:

- **Lectura y Escritura de Archivos de Texto:** Utilizando las funciones estándar de Python como open(), read(), y write(), se puede manejar fácilmente la lectura y escritura de archivos de texto.
- **Manejo de Archivos JSON y CSV:** Python incluye las bibliotecas json y csv para trabajar con estos formatos de manera eficiente.

2.7.4. Seguridad y Manejo de Datos

Aunque el uso de archivos locales es sencillo y eficaz para prototipos, es importante considerar ciertos aspectos de seguridad y manejo de datos:

- **Privacidad:** Asegurar que los archivos contengan datos sensibles, como traducciones personales, sean almacenados en ubicaciones seguras y con permisos adecuados para evitar accesos no autorizados.
- **Respaldo:** Mantener copias de seguridad de los archivos de datos para prevenir pérdidas en caso de fallos del sistema o corrupción de archivos.

3. CAPITULO III: DESARROLLO DEL PROYECTO

Este capítulo es donde se profundiza en la creación y construcción de la aplicación web destinada a la administración de cobros de cuotas de urbanizaciones. Este capítulo está dividido en varias fases, cada una de las cuales cumple un propósito específico en el ciclo de desarrollo de software.

3.1. Fase de Diseño

3.1.1. Requerimientos

3.1.1.1. Requerimientos Funcionales

Gestión de Opciones de Traducción: El sistema debe permitir la selección de opciones para el usuario, como números, letras del abecedario o frases.

RF1: Gestión de Opciones de Traducción

- **RF1.1:** Selección de Números
- **RF1.2:** Selección de Letras del Abecedario
- **RF1.3:** Selección de Frases

3.1.1.2. Requerimientos No Funcionales

- **Usabilidad**: La interfaz debe ser intuitiva y accesible para personas sordomudas.
- **Rendimiento:** El sistema debe procesar y traducir señas en tiempo real.
- **Portabilidad:** El sistema debe ser compatible con múltiples plataformas, incluyendo Windows, macOS y Linux.

3.1.1.3. Herramientas por Utilizar

La interfaz de usuario (UI) está diseñada para ser sencilla y accesible, permitiendo a los usuarios seleccionar entre tres opciones principales: números, letras del abecedario y frases. Una vez seleccionada la opción, el sistema abre el módulo de traducción correspondiente y comienza el proceso de traducción.

- **Python:** Lenguaje de programación principal utilizado para desarrollar el sistema.
- **OpenCV:** Biblioteca utilizada para el procesamiento de imágenes y videos.

- **TensorFlow/PyTorch:** Bibliotecas utilizadas para el desarrollo de modelos de aprendizaje automático.
- **Tkinter/PyQt:** Bibliotecas utilizadas para el desarrollo de la interfaz gráfica de usuario (GUI).

3.1.2. Diseño de la Interfaz de Usuario

La interfaz de usuario (UI) está diseñada para ser sencilla y accesible, permitiendo a los usuarios seleccionar entre tres opciones principales: números, letras del abecedario y frases. Una vez seleccionada la opción, el sistema abre el módulo de traducción correspondiente y comienza el proceso de traducción.

- **Pantalla Principal:** Muestra las tres opciones principales para que el usuario seleccione.
- **Módulo de Traducción:** Cada módulo (números, letras, frases) contiene las funcionalidades específicas para la traducción de señas seleccionada.
    - **Módulo de Números:** Traduce señas de números en tiempo real.
    - **Módulo de Letras:** Traduce señas de letras del abecedario en tiempo real.
    - **Módulo de Frases:** Traduce señas de frases comunes en tiempo real.

3.1.3. Casos de Uso

Caso de Uso (RF1.1): Selección de Números

Caso de Uso	**Selección de Números**
Identificador	RF1.1
Actores	Usuario
Tipo	Primario
Precondición	El usuario debe estar en la pantalla principal del sistema.
Postcondición	El sistema inicia el módulo de traducción de números.
Descripción	Este caso de uso permite al usuario seleccionar la opción de traducir números.
Resumen	El usuario selecciona "Números" en la pantalla principal, y el sistema carga el módulo de traducción de números.

Curso Normal

Nro.	Ejecutor	Paso o Actividad
1	Usuario	El usuario accede a la pantalla principal del sistema.
2	Usuario	El usuario selecciona la opción "Números".
3	Sistema	El sistema carga el módulo de traducción de números.

Caso de Uso (RF1.2): Selección de Letras del Abecedario

Caso de Uso	Selección de Letras del Abecedario
Identificador	RF1.2
Actores	Usuario
Tipo	Primario
Precondición	El usuario debe estar en la pantalla principal del sistema.
Postcondición	El sistema inicia el módulo de traducción de letras del abecedario.
Descripción	Este caso de uso permite al usuario seleccionar la opción de traducir letras del abecedario.
Resumen	El usuario selecciona "Letras del Abecedario" en la pantalla principal, y el sistema carga el módulo de traducción de letras del abecedario.

Curso Normal

Nro.	Ejecutor	Paso o Actividad
1	Usuario	El usuario accede a la pantalla principal del sistema.
2	Usuario	El usuario selecciona la opción "Letras del Abecedario".

3	Sistema	El sistema carga el módulo de traducción de letras del abecedario.

Caso de Uso (RF1.3): Selección de Frases

Caso de Uso	**Selección de Frases**
Identificador	RF1.3
Actores	Usuario
Tipo	Primario
Precondición	El usuario debe estar en la pantalla principal del sistema.
Postcondición	El sistema inicia el módulo de traducción de frases.
Descripción	Este caso de uso permite al usuario seleccionar la opción de traducir frases.
Resumen	El usuario selecciona "Frases" en la pantalla principal, y el sistema carga el módulo de traducción de frases.

Curso Normal

Nro.	**Ejecutor**	**Paso o Actividad**
1	Usuario	El usuario accede a la pantalla principal del sistema.
2	Usuario	El usuario selecciona la opción "Frases".
3	Sistema	El sistema carga el módulo de traducción de frases.

3.2. Fase de Planificación

3.2.1. Primera Iteración

3.2.1.1. Requerimientos Primera Iteración

- **Configuración del Proyecto:** El sistema debe configurarse inicialmente en los entornos específicos para su correcto funcionamiento.
    - Instalación de Python y las bibliotecas necesarias (OpenCV, TensorFlow/PyTorch, Tkinter/PyQt).
    - Configuración de un entorno de desarrollo integrado (IDE) como Visual Studio Code o PyCharm para facilitar la escritura y depuración del código.
    - Establecimiento de un repositorio Git para el control de versiones, lo que permitirá gestionar los cambios en el código de manera eficiente y colaborar con otros desarrolladores.
    - Definición de las rutas y estructuras de carpetas del proyecto para mantener el código organizado y fácil de navegar.

- **Selección de Opciones:** Implementación de la funcionalidad de selección de opciones (números, letras, frases) en la interfaz principal.
    - Diseño y desarrollo de una pantalla principal con opciones claras y accesibles para que el usuario pueda seleccionar entre números, letras del abecedario y frases.
    - Implementación de botones interactivos para cada opción que, al ser seleccionados, redirijan al módulo correspondiente del sistema.
    - Aseguramiento de que la interfaz sea intuitiva y amigable para los usuarios sordomudos, utilizando elementos visuales claros y consistentes.

3.2.1.2. Tareas Primera Iteración

Se configuran las herramientas necesarias como Python, OpenCV, TensorFlow/PyTorch, y Tkinter/PyQt. La primera fase del desarrollo se enfoca en la creación de la interfaz principal que permita a los usuarios seleccionar entre números, letras y frases.

- Configuración del Entorno de Desarrollo:
    - **Instalación de Python**: Descarga e instalación de la última versión de Python desde su sitio oficial. Verificación de la instalación correcta mediante la línea de comandos.
    - **Configuración del IDE**: Configuración de Visual Studio Code o PyCharm, incluyendo la instalación de extensiones relevantes como Python, pylint, y Git.
    - **Instalación de Bibliotecas**: Uso de pip para instalar las bibliotecas necesarias:

    (Librerías)

- pip install opencv-python
- pip install tensorflow
- pip install pytorch
- pip install tk
- pip install pyqt5

- **Desarrollo de la Interfaz Principal**:
    - **Diseño de la Interfaz**: Creación de maquetas o wireframes para la pantalla principal utilizando herramientas como Figma o Adobe XD. Incorporación de principios de diseño accesible.
    - **Implementación de la Interfaz**: Codificación de la interfaz principal en Python utilizando Tkinter o PyQt. Adición de botones para seleccionar números, letras y frases.
    - **Interactividad**: Programación de la funcionalidad de los botones para que redirijan a los módulos correspondientes al ser seleccionados.

3.2.2. Segunda Iteración

3.2.2.1. Requerimientos Segunda Iteración

- **Implementación del Algoritmo de Reconocimiento:** Desarrollo del algoritmo de reconocimiento de gestos para números, letras y frases.

- Desarrollo del algoritmo de reconocimiento de gestos específico para números, letras del abecedario y frases.
- Entrenamiento de modelos de aprendizaje automático para mejorar la precisión del reconocimiento de gestos.

3.2.2.2. Tareas Segunda Iteración

En esta fase, se desarrolla el algoritmo de reconocimiento de gestos utilizando OpenCV y TensorFlow/PyTorch. Se realizan pruebas iniciales para asegurar que el sistema pueda capturar y procesar gestos en tiempo real.

- **Desarrollo del Algoritmo de Reconocimiento:**
  - Captura de Gestos: Uso de OpenCV para capturar los movimientos de las manos y gestos faciales en tiempo real. Implementación de técnicas de procesamiento de imágenes para extraer características relevantes.
  - Entrenamiento de Modelos: Creación de conjuntos de datos etiquetados de gestos y uso de TensorFlow o PyTorch para entrenar redes neuronales convolucionales (CNN) que puedan reconocer y clasificar estos gestos.
  - Validación y Pruebas Iniciales: Realización de pruebas con el algoritmo para asegurar que puede reconocer gestos con precisión en diferentes condiciones de iluminación y ángulos de la cámara.

- **Integración con la Interfaz de Usuario:**
  - Conexión del Algoritmo con la Interfaz: Modificación de la interfaz de usuario para que al seleccionar una opción (números, letras o frases), se active el módulo de reconocimiento correspondiente.
  - Visualización de Resultados: Implementación de la funcionalidad para mostrar los resultados del reconocimiento en tiempo real en la pantalla, permitiendo a los usuarios ver las traducciones de sus gestos instantáneamente.

3.2.3. Tercera Iteración

3.2.3.1. Requerimientos Tercera Iteración

- **Integración de Componentes:** Integración del algoritmo de reconocimiento con la interfaz de usuario.
    - Finalización de la integración entre el algoritmo de reconocimiento de gestos y la interfaz de usuario.
    - Aseguramiento de que todos los componentes del sistema funcionen de manera cohesiva y eficiente.
- **Pruebas y Validación:** Realización de pruebas de usabilidad y rendimiento.
    - Realización de pruebas de usabilidad para asegurar que el sistema sea fácil de usar y efectivo para la comunidad sordomuda.
    - Evaluación del rendimiento del sistema bajo diferentes condiciones y ajustes basados en la retroalimentación de los usuarios.

3.2.3.2. Tareas Tercera Iteración

Se integran todos los componentes del sistema y se realizan pruebas exhaustivas para asegurar que el sistema funcione correctamente. Se recopila retroalimentación de los usuarios para realizar ajustes y mejoras necesarias.

- **Integración de Componentes:**
    - Finalización del Desarrollo del Algoritmo: Ajustes finales al algoritmo de reconocimiento de gestos para mejorar la precisión y eficiencia.
    - Conexión de Módulos: Garantizar que la interfaz de usuario y el algoritmo de reconocimiento de gestos se comuniquen correctamente y que los resultados se muestren en tiempo real.
    - Pruebas de Integración: Realización de pruebas para asegurar que todos los componentes del sistema trabajen juntos sin problemas y que las traducciones de gestos sean precisas y rápidas.
- **Pruebas y Validación:**
    - Pruebas de Usabilidad: Reclutamiento de usuarios de la comunidad sordomuda para probar el sistema. Observación de cómo interactúan con el sistema y recopilación de sus comentarios.

- Evaluación del Rendimiento: Pruebas del sistema en diferentes condiciones (por ejemplo, variaciones de iluminación, diferentes cámaras) para asegurar que el reconocimiento de gestos sea consistente.
- Retroalimentación y Ajustes: Análisis de la retroalimentación de los usuarios y realización de ajustes en la interfaz de usuario y el algoritmo de reconocimiento para mejorar la usabilidad y la precisión.

3.3. Fase de Codificación

La fase de codificación se centra en el desarrollo de los componentes del sistema, asegurando que todos trabajen juntos de manera efectiva. Utilizando Python y las bibliotecas mencionadas, el código se estructura de manera modular para facilitar el mantenimiento y la escalabilidad.

3.3.1. Código

A continuación, se detallarán pequeños fragmentos de código como un ejemplo de una de varias maneras de implementarse la sección mencionada. Tomando en cuenta que es una referencia, no necesariamente será desarrollado de la manera ejemplificada.

El sistema se desarrolla utilizando Python y las bibliotecas mencionadas. A continuación, se describen los principales componentes del código:

- **Rutas:** Definen la lógica de navegación del sistema.

  - **Definición de Rutas**: Las rutas definen la lógica de navegación del sistema. En el contexto de un sistema de traducción de lenguaje de señas, las rutas especifican cómo se navega entre la pantalla principal y los diferentes módulos de traducción (números, letras, frases).

Ejemplo de estructura de código:

```
def main_menu():
    # Código para mostrar la pantalla principal
    pass
```

```
def number_translation():
    # Código para el módulo de traducción de números
    pass

def letter_translation():
    # Código para el módulo de traducción de letras
    pass

def phrase_translation():
    # Código para el módulo de traducción de frases
    pass
```

- **Controladores:** Manejan la lógica del sistema y la interacción con el usuario.
    - **Lógica del Sistema:** Los controladores manejan la lógica del sistema y la interacción con el usuario. Esto incluye la captura de gestos, el procesamiento de imágenes y la interacción con la interfaz de usuario.

Ejemplo de estructura de código:

```
class GestureController:
    def __init__(self):
        # Inicialización de la cámara y modelos de reconocimiento
        #En este apartado también se podrá configurar la resolución en pantalla
        self.camera = cv2.VideoCapture(0)
        self.model = load_model('gesture_model.h5')

    def capture_gesture(self):
        # Captura y procesamiento de gestos
        ret, frame = self.camera.read()
        if ret:
            processed_frame = preprocess(frame)
            prediction = self.model.predict(processed_frame)
            return prediction
```

- **Vistas:** Definen la interfaz de usuario.

    - **Interfaz de Usuario:** Las vistas definen la interfaz de usuario. Utilizando Tkinter o PyQt, se crean las pantallas y elementos interactivos que permiten al usuario interactuar con el sistema.

Ejemplo de estructura de código (utilizando Tkinter):

```
from tkinter import *

class MainView:
    def __init__(self, root):
        self.root = root
        self.root.title("Sistema de Traducción de Lenguaje de Señas")

        self.number_button = Button(root, text="Números",
command=self.open_number_translation)
        self.number_button.pack()

        self.letter_button = Button(root, text="Letras",
command=self.open_letter_translation)
        self.letter_button.pack()

        self.phrase_button = Button(root, text="Frases",
command=self.open_phrase_translation)
        self.phrase_button.pack()

    def open_number_translation(self):
        # Código para abrir el módulo de traducción de números
        pass

    def open_letter_translation(self):
        # Código para abrir el módulo de traducción de letras
```

```
        pass

    def open_phrase_translation(self):
        # Código para abrir el módulo de traducción de frases
        pass
```

- **Modelos:** Implementan el algoritmo de reconocimiento de gestos.
    - **Implementación del Algoritmo de Reconocimiento de Gestos:** Los modelos implementan el algoritmo de reconocimiento de gestos utilizando aprendizaje automático. Esto incluye la definición y entrenamiento de redes neuronales para clasificar los gestos capturados.

Ejemplo de código (Haciendo uso de Redes Neuronales):

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

def create_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
        MaxPooling2D(pool_size=(2, 2)),
        Conv2D(64, (3, 3), activation='relu'),
        MaxPooling2D(pool_size=(2, 2)),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])
    return model

model = create_model()
```

- **Pruebas:** Incluyen pruebas unitarias y de integración para asegurar la calidad del sistema.

    - **Pruebas Unitarias y de Integración:** Incluyen pruebas para asegurar la calidad del sistema. Las pruebas unitarias validan componentes individuales del código, mientras que las pruebas de integración aseguran que los componentes trabajen juntos correctamente.

Ejemplo de código:

```
import unittest

class TestGestureRecognition(unittest.TestCase):
    def test_capture_gesture(self):
        controller = GestureController()
        prediction = controller.capture_gesture()
        self.assertIsNotNone(prediction)

if __name__ == '__main__':
    unittest.main()
```

3.4. Arquitectura de Software

El sistema utiliza una arquitectura basada en el Modelo-Vista-Controlador (MVC), lo que facilita la organización y desarrollo del proyecto. Esta arquitectura separa la lógica del sistema (modelo), la interfaz de usuario (vista) y el control de la aplicación (controlador).

3.4.1. Modelo-Vista-Controlador (MVC)

- **Modelo**
    - **El modelo es responsable de gestionar los datos y la lógica de negocio del sistema. En el contexto de este proyecto, el modelo incluye el algoritmo de reconocimiento de gestos y los datos necesarios para entrenar y validar el modelo.**
- **Vista**

- La vista es la parte del sistema con la que interactúa el usuario. Está diseñada para ser accesible y fácil de usar, mostrando los resultados del reconocimiento de gestos en tiempo real.

- **Controlador**
  - El controlador actúa como intermediario entre el modelo y la vista. Procesa las entradas del usuario, actualiza el modelo y devuelve los resultados a la vista para que se muestren al usuario.

3.4.2. Beneficios de la Arquitectura MVC

- **Separación de Preocupaciones:** La arquitectura MVC permite una clara separación entre la lógica de negocio, la presentación y el control de la aplicación. Esto facilita el mantenimiento y la escalabilidad del sistema.
- **Reutilización de Código:** Al separar la lógica de negocio de la interfaz de usuario, es posible reutilizar componentes del código en diferentes partes del sistema o en otros proyectos.
- **Facilidad de Pruebas:** La separación de componentes en la arquitectura MVC facilita la realización de pruebas unitarias y de integración, asegurando que cada componente funcione correctamente de manera independiente y en conjunto.

3.5. Pruebas del Sistema

3.5.1. Pruebas de Aceptación

Caso RF1.1_P01: Selección de Números

Pruebas de Aceptación	
Código: RF1.1_P01	Requerimiento: RF1.1
Nombre: Selección de Números	
Descripción: El sistema tendrá una opción para seleccionar números.	
Condiciones: N/A	

Pasos:	
1. El usuario accede a la pantalla principal del sistema.	
2. El usuario selecciona la opción "Números".	
3. El sistema carga el módulo de traducción de números.	
Resultados previstos:	El usuario seleccionará exitosamente la opción de números y el sistema iniciará el módulo correspondiente.
Evaluación: Prueba completada satisfactoriamente	

Caso RF1.2_P02: Selección de Letras del Abecedario

Pruebas de Aceptación	
Código: RF1.2_P02	Requerimiento: RF1.2
Nombre: Selección de Letras del Abecedario	
Descripción: El sistema tendrá una opción para seleccionar letras del abecedario.	
Condiciones: N/A	
Pasos:	
1. El usuario accede a la pantalla principal del sistema.	
2. El usuario selecciona la opción "Letras del Abecedario".	
3. El sistema carga el módulo de traducción de letras del abecedario.	

Resultados previstos:	El usuario seleccionará exitosamente la opción de letras del abecedario y el sistema iniciará el módulo correspondiente.
Evaluación: Prueba completada satisfactoriamente	

Caso RF1.3_P03: Selección de Frases

Pruebas de Aceptación	
Código: RF1.3_P03	Requerimiento: RF1.3
Nombre: Selección de Frases	
Descripción: El sistema tendrá una opción para seleccionar frases.	
Condiciones: N/A	
Pasos:	
1. El usuario accede a la pantalla principal del sistema.	
2. El usuario selecciona la opción "Frases".	
3. El sistema carga el módulo de traducción de frases.	
Resultados previstos:	El usuario seleccionará exitosamente la opción de frases y el sistema iniciará el módulo correspondiente.
Evaluación: Prueba completada satisfactoriamente	

4. CAPITULO IV: ENFOQUES PARA EL RECONOCIMIENTO DE SEÑAS

4.1. Introducción

En este capítulo, se explorarán diferentes enfoques para el reconocimiento de señas, con un enfoque particular en el uso de la posición de los dedos mediante distancia euclidiana versus el uso de redes neuronales de aprendizaje profundo. Se analizarán las ventajas y desventajas de cada método, y se discutirá por qué uno puede ser más adecuado que el otro para un prototipo inicial de un sistema de traducción de lenguaje de señas. Además, se proporcionará una comparación detallada para facilitar la comprensión de las diferencias clave entre estos enfoques.

4.2. Reconocimiento de Señas mediante Posición de los Dedos

4.2.1. Concepto y Funcionamiento

El método de reconocimiento de señas basado en la posición de los dedos utiliza técnicas geométricas para medir y analizar las distancias entre los puntos clave de la mano. La distancia euclidiana es una métrica que calcula la longitud directa entre dos puntos en un espacio euclidiano. En el contexto del reconocimiento de señas, los puntos clave suelen ser las articulaciones de los dedos y la base de la mano.

El proceso generalmente involucra los siguientes pasos:

- **Captura de Imagen:** Utilizando una cámara, se captura una imagen de la mano realizando una seña.
- **Detección de Puntos Clave:** Se aplican algoritmos de visión por computadora para identificar y marcar los puntos clave en la mano.
- **Cálculo de Distancias:** Se calculan las distancias euclidianas entre los puntos clave.
- **Clasificación:** Estas distancias se utilizan como características para clasificar la seña utilizando algoritmos de clasificación como K-Nearest Neighbors (KNN) o Support Vector Machines (SVM).

4.2.2. Ventajas

- **Simplicidad:** El cálculo de distancias euclidianas es matemáticamente sencillo y requiere menos poder de cómputo. Esto hace que el método sea fácil de implementar y rápido de ejecutar.
- **Velocidad:** Este método puede procesar gestos en tiempo real con menor latencia, lo que es crucial para aplicaciones interactivas. La simplicidad de los cálculos permite que el sistema responda rápidamente a las entradas del usuario.
- **Requisitos de Hardware:** No requiere hardware especializado y puede funcionar con cámaras estándar. Esto reduce los costos y hace que el sistema sea más accesible para un mayor número de usuarios.
- **Transparencia:** Debido a su simplicidad, el método es más fácil de entender y explicar. Esto puede ser beneficioso en contextos educativos y para la colaboración interdisciplinaria.

4.2.3. Desventajas

- **Precisión:** La precisión puede ser menor en comparación con métodos más avanzados como las redes neuronales, especialmente en gestos complejos. La simplicidad del método limita su capacidad para capturar la variabilidad y complejidad de las señas.
- **Variabilidad:** La posición de los dedos puede variar significativamente entre usuarios, lo que puede afectar la consistencia del reconocimiento. Factores como el tamaño de la mano, la posición relativa de los dedos y las diferencias individuales en la forma de realizar las señas pueden introducir variabilidad que este método no maneja bien.
- **Limitaciones en Gestos Complejos:** Este método puede no ser adecuado para reconocer gestos que involucran movimientos rápidos o cambios de postura complejos, ya que se basa principalmente en la geometría estática.

4.3. Reconocimiento de Señas mediante Redes Neuronales

4.3.1. Concepto y Funcionamiento

Las redes neuronales convolucionales (CNN) son un tipo de red neuronal profunda que se especializa en el procesamiento de datos con una estructura de cuadrícula, como las imágenes. Estas redes pueden aprender a identificar características complejas y patrones en los gestos de las manos. El proceso de reconocimiento mediante CNN generalmente incluye los siguientes pasos:

- **Captura de Imagen:** Similar al método de distancia euclidiana, se captura una imagen de la mano realizando una seña.
- **Preprocesamiento de Imágenes:** Las imágenes se normalizan y se ajustan en términos de tamaño y resolución para ser compatibles con la entrada de la red neuronal.
- **Extracción de Características:** Las CNN utilizan múltiples capas convolucionales para extraer características de bajo y alto nivel de las imágenes.
- **Clasificación:** Las características extraídas se pasan a través de capas totalmente conectadas que realizan la clasificación final de las señas.

4.3.2. Ventajas

- **Precisión:** Las CNN pueden lograr alta precisión en el reconocimiento de gestos, incluso en condiciones de variabilidad y complejidad. Esto se debe a su capacidad para aprender representaciones complejas y generalizar bien a nuevas variaciones de las señas.
- **Adaptabilidad:** Pueden adaptarse a variaciones en los gestos y aprender a reconocer nuevas señas a través del entrenamiento continuo. La flexibilidad de las CNN permite actualizar y mejorar el sistema mediante el reentrenamiento con nuevos datos.
- **Robustez:** Son más robustas frente a cambios en la iluminación, ángulos de cámara y fondo. Las técnicas de data augmentation y las capas convolucionales ayudan a las CNN a manejar estas variaciones.
- **Capacidad de Escalar:** Las redes neuronales pueden escalar bien con más datos y más capacidad computacional, lo que permite la mejora continua del sistema.

4.3.3. Desventajas

- **Complejidad Computacional:** Requieren más poder de cómputo y pueden ser más lentas, especialmente en hardware limitado. El entrenamiento de redes neuronales puede ser intensivo en recursos y tiempo.
- **Requisitos de Datos:** Necesitan grandes cantidades de datos etiquetados para entrenar eficazmente los modelos. La recopilación y etiquetado de datos puede ser costoso y llevar mucho tiempo.
- **Desarrollo:** El desarrollo y ajuste de redes neuronales es más complejo y requiere conocimientos especializados en aprendizaje automático. La optimización de hiperparámetros y la arquitectura de la red pueden ser complicadas.
- **Dependencia de Infraestructura:** El despliegue de modelos de aprendizaje profundo puede requerir infraestructura especializada, como GPUs, y servicios en la nube para entrenamiento y procesamiento en tiempo real.

4.4. Comparación de Enfoques

4.4.1. Aplicabilidad al Prototipo

Para un prototipo inicial, es crucial elegir un enfoque que sea sencillo de implementar y que permita una rápida iteración. La simplicidad y velocidad del método basado en distancia euclidiana lo hace ideal para un prototipo, permitiendo un desarrollo y pruebas más rápidas.

- **Rapidez de Implementación:** El método de distancia euclidiana permite un desarrollo más ágil debido a su simplicidad. Esto es beneficioso en las etapas iniciales cuando se busca validar el concepto y obtener retroalimentación rápidamente.
- **Menores Requisitos de Recursos:** Los recursos necesarios para implementar y ejecutar el método de distancia euclidiana son mínimos, lo que facilita su despliegue en diversos entornos sin requerir hardware especializado.
- **Facilidad de Depuración:** La simplicidad del método facilita la depuración y el ajuste, permitiendo una rápida identificación y solución de problemas durante el desarrollo.

4.4.2. Escalabilidad y Futuras Mejoras

Aunque el método de posición de los dedos es adecuado para el prototipo, las redes neuronales ofrecen un camino claro hacia mejoras de precisión y robustez en versiones futuras del sistema. Una posible estrategia sería comenzar con el método euclidiano y, una vez validado el concepto, integrar gradualmente el uso de redes neuronales.

- **Transición Gradual:** Se puede comenzar con el método de distancia euclidiana para el prototipo inicial y luego, basándose en los resultados y la retroalimentación, incorporar redes neuronales para mejorar la precisión y la robustez del sistema.
- **Capacitación Continua:** A medida que se recopilan más datos de usuarios reales, se puede entrenar gradualmente modelos de redes neuronales para que aprendan de estas nuevas muestras, mejorando la capacidad del sistema para manejar variaciones y complejidad.
- **Infraestructura Escalable:** Implementar un sistema modular donde los componentes pueden ser reemplazados o mejorados sin afectar la estructura general del sistema facilita la transición hacia métodos más avanzados como las redes neuronales.

4.5. Conclusión

Ambos métodos tienen sus propias ventajas y desventajas. La elección del método de distancia euclidiana para el prototipo se justifica por su simplicidad y rapidez de implementación, lo que es crucial en las primeras etapas de desarrollo. Este enfoque permite obtener resultados iniciales rápidamente y validar el concepto con un mínimo de recursos.

Por otro lado, las redes neuronales representan una evolución natural hacia un sistema más preciso y robusto. Su capacidad para manejar variabilidad y complejidad las convierte en una opción ideal para futuras versiones del sistema. La estrategia recomendada es utilizar el método de distancia euclidiana para el prototipo inicial y planificar una transición gradual hacia el uso de redes neuronales a medida que se recopilan más datos y se dispone de más recursos.

5. CAPITULO V: CONCLUSIONES Y RECOMENDACIONES

5.1. Conclusiones

El desarrollo del prototipo de un sistema de traducción de lenguaje de señas ha sido una tarea multifacética que abarca desde la concepción y diseño hasta la implementación y pruebas del sistema. Este proyecto ha demostrado ser una herramienta fundamental para mejorar la accesibilidad y la inclusión digital de las personas con discapacidades auditivas y de habla, permitiéndoles interactuar con dispositivos tecnológicos de manera más efectiva.

Durante la fase de diseño, se establecieron los requerimientos y se planificó cuidadosamente cada etapa del desarrollo, lo que resultó en una arquitectura sólida y funcional. La elección del lenguaje de programación Python y el uso de bibliotecas especializadas como OpenCV y TensorFlow/PyTorch han sido cruciales para el desarrollo del algoritmo de reconocimiento de gestos, asegurando precisión y eficiencia en la traducción de señas.

La implementación de la metodología de desarrollo XP permitió un enfoque iterativo y flexible, adaptándose continuamente a las necesidades y retroalimentación de los usuarios. Esto garantizó que el sistema no solo cumpliera con los requerimientos técnicos, sino que también fuera intuitivo y fácil de usar para la comunidad sordomuda. La programación en parejas, la integración continua y las pruebas automatizadas fueron prácticas clave que aseguraron la calidad y coherencia del código.

El desarrollo de la interfaz de usuario, utilizando herramientas como Tkinter y PyQt, resultó en una plataforma accesible y amigable. La interfaz permite a los usuarios seleccionar entre números, letras y frases, y proporciona una traducción en tiempo real, mejorando significativamente la experiencia del usuario.

Las pruebas de usabilidad realizadas con miembros de la comunidad sordomuda validaron la efectividad del sistema y proporcionaron retroalimentación valiosa para futuras mejoras. La integración exitosa de los componentes del sistema y la realización

de pruebas exhaustivas aseguraron que el sistema funcione de manera cohesiva y eficiente en diferentes condiciones de uso.

5.2. Recomendaciones

Para aquellos que deseen utilizar este sistema como guía o base para desarrollar soluciones más avanzadas, se proponen las siguientes recomendaciones:

Ampliación de la Base de Datos de Gestos

Descripción: Considerar la ampliación de la base de datos de gestos para incluir más variaciones y dialectos del lenguaje de señas.

Beneficio: Una base de datos más amplia mejorará la precisión y usabilidad del sistema en diferentes regiones y contextos culturales.

Mejora de Algoritmos de Reconocimiento

Descripción: Explorar y adoptar algoritmos más avanzados de reconocimiento de gestos, como modelos de aprendizaje profundo más complejos o redes neuronales recurrentes (RNNs).

Beneficio: Algoritmos más avanzados pueden aumentar significativamente la precisión y rapidez del reconocimiento de señas, especialmente en condiciones variables de iluminación y fondo.

Automatización de Pruebas

Descripción: Implementar un marco de pruebas automatizadas más robusto que incluya pruebas de regresión, pruebas de carga y pruebas de integración continua.

Beneficio: La automatización de pruebas garantiza que cualquier cambio o mejora en el sistema no introduzca errores y mantenga la estabilidad del sistema.

Optimización de Recursos Computacionales

Descripción: Optimizar el uso de recursos computacionales para mejorar el rendimiento del sistema, especialmente en dispositivos con capacidades limitadas.

Beneficio: La optimización asegurará que el sistema funcione de manera eficiente en una amplia gama de dispositivos, incluyendo smartphones y tablets.

Integración con Dispositivos Móviles

Descripción: Desarrollar aplicaciones móviles nativas para iOS y Android que integren el sistema de traducción de lenguaje de señas.

Beneficio: Una aplicación móvil aumentará la accesibilidad del sistema, permitiendo a los usuarios utilizar la herramienta en cualquier lugar y momento.

Implementación de Realidad Aumentada (AR)

Descripción: Incorporar tecnología de realidad aumentada (AR) para superponer las traducciones de señas en tiempo real sobre la vista de la cámara.

Beneficio: AR puede mejorar la interacción del usuario y proporcionar una experiencia más inmersiva y educativa.

Colaboración con la Comunidad Sordomuda

Descripción: Continuar colaborando estrechamente con la comunidad sordomuda para obtener retroalimentación continua y adaptar el sistema a sus necesidades específicas.

Beneficio: La retroalimentación directa asegura que el sistema siga siendo relevante y útil para sus usuarios principales.

Investigación Continua en Nuevas Tecnologías

Descripción: Mantenerse actualizado con las últimas investigaciones y avances en visión por computadora y aprendizaje automático.

Beneficio: La adopción temprana de nuevas tecnologías puede proporcionar ventajas competitivas y mejoras continuas en la precisión y funcionalidad del sistema.

Documentación y Capacitación

Descripción: Proporcionar una documentación detallada y actualizada, junto con recursos de capacitación, para facilitar la implementación y uso del sistema.

Beneficio: Una buena documentación y capacitación aseguran que los usuarios puedan aprovechar al máximo las capacidades del sistema y resolver problemas de manera eficiente.

6. ANEXOS

6.1. Código Fuente del Prototipo

6.1.1. Código del Reconocimiento de Frases

```
import cv2
import mediapipe as mp
import numpy as np


def Etiqueta(idx, mano, results):
    aux = None
    for _, clase in enumerate(results.multi_handedness):
        if clase.classification[0].index == idx:
            label = clase.classification[0].label
            texto = '{}'.format(label)

            coords = tuple(np.multiply(np.array(
                (mano.landmark[mp_hands.HandLandmark.WRIST].x,
                mano.landmark[mp_hands.HandLandmark.WRIST].y)),
                [1280, 720]).astype(int))

            aux = texto, coords
    return aux


def distancia_euclidiana(p1, p2):
```

```
    d = ((p2[0] - p1[0]) ** 2 + (p2[1] - p1[1]) ** 2) ** 0.5
    return d


mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles
mp_hands = mp.solutions.hands
change = True
change2 = False


cap = cv2.VideoCapture(1)
cap.set(3, 1280)
cap.set(4, 720)
with mp_hands.Hands(
    model_complexity=1,
    min_detection_confidence=0.7,
    min_tracking_confidence=0.7,
    max_num_hands=2) as hands:
    while cap.isOpened():
        success, image = cap.read()
        if not success:
            continue


        image.flags.writeable = False
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

```
results = hands.process(image)

image.flags.writeable = True
image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

image_height, image_width, _ = image.shape
if results.multi_hand_landmarks:
    if len(results.multi_hand_landmarks):
        for num, hand_landmarks in enumerate(results.multi_hand_landmarks):

            mp_drawing.draw_landmarks(
                image,
                hand_landmarks,
                mp_hands.HAND_CONNECTIONS,
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style())

            threshold_b_thumb = 30
            threshold_b_fingers = 100
            threshold_c_close = 50
            threshold_d_circle = 20
            threshold_d_straight = 150
            threshold_d_folded = 30
            threshold_a_fingers = 30
```

```
                threshold_a_thumb = 50

                index_finger_tip = (int(hand_landmarks.landmark[8].x *
image_width),
                                    int(hand_landmarks.landmark[8].y * image_height))
                index_finger_pip = (int(hand_landmarks.landmark[6].x *
image_width),
                                    int(hand_landmarks.landmark[6].y * image_height))

                thumb_tip = (int(hand_landmarks.landmark[4].x * image_width),
                                    int(hand_landmarks.landmark[4].y * image_height))
                thumb_pip = (int(hand_landmarks.landmark[2].x * image_width),
                                    int(hand_landmarks.landmark[2].y * image_height))

                middle_finger_tip = (int(hand_landmarks.landmark[12].x *
image_width),
                                    int(hand_landmarks.landmark[12].y * image_height))

                middle_finger_pip = (int(hand_landmarks.landmark[10].x *
image_width),
                                    int(hand_landmarks.landmark[10].y * image_height))

                ring_finger_tip = (int(hand_landmarks.landmark[16].x *
image_width),
                                    int(hand_landmarks.landmark[16].y * image_height))
```

```
                ring_finger_pip = (int(hand_landmarks.landmark[14].x *
image_width),
                                   int(hand_landmarks.landmark[14].y * image_height))

                pinky_tip = (int(hand_landmarks.landmark[20].x * image_width),
                                   int(hand_landmarks.landmark[20].y * image_height))
                pinky_pip = (int(hand_landmarks.landmark[18].x * image_width),
                                   int(hand_landmarks.landmark[18].y * image_height))

                wrist = (int(hand_landmarks.landmark[0].x * image_width),
                                   int(hand_landmarks.landmark[0].y * image_height))

                if thumb_pip[1] - thumb_tip[1] > 0 and thumb_pip[1] -
index_finger_tip[1] < 0 \
                    and thumb_pip[1] - middle_finger_tip[1] < 0 and thumb_pip[1] -
ring_finger_tip[1]<0 \
                    and thumb_pip[1] - pinky_tip[1] < 0:
                    cv2.putText(image, 'Bien', (700, 150),
                                   cv2.FONT_HERSHEY_SIMPLEX,
                                   3.0, (0, 0, 255), 6)
                elif thumb_pip[1] - thumb_tip[1] < 0 and thumb_pip[1] -
index_finger_tip[1] > 0 \
                    and thumb_pip[1] - middle_finger_tip[1] > 0 and thumb_pip[1] -
ring_finger_tip[1]>0 \
                    and thumb_pip[1] - pinky_tip[1] > 0:
```

```
                    cv2.putText(image, 'Mal', (700, 150),
                                cv2.FONT_HERSHEY_SIMPLEX,
                                3.0, (0, 0, 255), 6)

                elif thumb_pip[1] - thumb_tip[1] > 0 and index_finger_pip[1] -
index_finger_tip[1]>0 \
                    and pinky_pip[1] - pinky_tip[1] > 0:
                    cv2.putText(image, 'Te quiero', (700, 150),
                                cv2.FONT_HERSHEY_SIMPLEX,
                                3.0, (0, 0, 255), 6)


                if Etiqueta(num, hand_landmarks, results) and
len(results.multi_hand_landmarks)==2:
                    text,coords = Etiqueta(num, hand_landmarks, results)
                    print(text, coords)
                    if text =="Right":
                       index_finger_tip_r = (int(hand_landmarks.landmark[8].x *
image_width),
                                int(hand_landmarks.landmark[8].y * image_height))
                       change = True
                    if text =="Left":
                       index_finger_tip_l = (int(hand_landmarks.landmark[8].x *
image_width),
                                int(hand_landmarks.landmark[8].y * image_height))
```

```
                    wrist = (int(hand_landmarks.landmark[0].x * image_width),
                        int(hand_landmarks.landmark[0].y * image_height))

                    change2 = True

                if change2 == True and change == True:
                    if distancia_euclidiana(index_finger_tip_l,  wrist) < 170.0:
                        cv2.putText(image, 'Que hora es?', (700, 150),
                            cv2.FONT_HERSHEY_SIMPLEX,
                            3.0, (0, 0, 255), 6)

                cv2.putText(image, text, coords,
cv2.FONT_HERSHEY_SIMPLEX, 1, (255,255,255),2,cv2.LINE_AA)


            cv2.imshow('MediaPipe Hands', image)
            if cv2.waitKey(5) & 0xFF == 27:
                break
        cap.release()
        cv2.destroyAllWindows()
```

6.1.2. Código del Reconocimiento de Letras

```
import cv2
```

```
import mediapipe as mp
import numpy as np

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils

cap = cv2.VideoCapture(1)
cap.set(3, 1280)
cap.set(4, 720)

with mp_hands.Hands(
    model_complexity=1,
    min_detection_confidence=0.7,
    min_tracking_confidence=0.7,
    max_num_hands=1) as hands:

    while cap.isOpened():
        success, image = cap.read()
        if not success:
            continue

        image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
        image.flags.writeable = False
        results = hands.process(image)
```

```
            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

            if results.multi_hand_landmarks:
                for hand_landmarks in results.multi_hand_landmarks:
                    mp_drawing.draw_landmarks(
                        image, hand_landmarks, mp_hands.HAND_CONNECTIONS)

                    # Insert letter recognition logic here
                    # Example placeholder: Recognize 'A'
                    thumb_tip =
hand_landmarks.landmark[mp_hands.HandLandmark.THUMB_TIP]
                    index_tip =
hand_landmarks.landmark[mp_hands.HandLandmark.INDEX_FINGER_TIP]
                    distance = np.sqrt((thumb_tip.x - index_tip.x)**2 + (thumb_tip.y -
index_tip.y)**2 + (thumb_tip.z - index_tip.z)**2)

                    if distance < 0.05:
                        cv2.putText(image, 'A', (10, 70), cv2.FONT_HERSHEY_SIMPLEX,
3, (0, 255, 0), 3, cv2.LINE_AA)

            cv2.imshow('Hand Sign Recognition', image)
            if cv2.waitKey(5) & 0xFF == 27:
                break
```

```
cap.release()
cv2.destroyAllWindows()
```

6.1.3. Código de Reconocimiento de Números

```
import cv2
import mediapipe as mp

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils

cap = cv2.VideoCapture(1)
cap.set(3, 1280)
cap.set(4, 720)

def contar_dedos(hand_landmarks):
    dedos_arriba = 0

    # Dedos: Thumb (pulgar), Index (índice), Middle (medio), Ring (anular), Pinky (meñique)
    dedos_tips_ids = [4, 8, 12, 16, 20]
```

```
            # Dedos arriba si el tip (y) es menor que el pip (y)
            for id in range(1, 5):
                if hand_landmarks.landmark[dedos_tips_ids[id]].y <
hand_landmarks.landmark[dedos_tips_ids[id] - 2].y:
                    dedos_arriba += 1

            return dedos_arriba

        with mp_hands.Hands(
            model_complexity=1,
            min_detection_confidence=0.7,
            min_tracking_confidence=0.7,
            max_num_hands=1) as hands:

            while cap.isOpened():
                success, image = cap.read()
                if not success:
                    continue

                image = cv2.cvtColor(cv2.flip(image, 1), cv2.COLOR_BGR2RGB)
                image.flags.writeable = False
                results = hands.process(image)
```

```
            image.flags.writeable = True
            image = cv2.cvtColor(image, cv2.COLOR_RGB2BGR)

            if results.multi_hand_landmarks:
                for hand_landmarks in results.multi_hand_landmarks:
                    mp_drawing.draw_landmarks(
                        image, hand_landmarks, mp_hands.HAND_CONNECTIONS)

                    dedos_arriba = contar_dedos(hand_landmarks)
                    cv2.putText(image, f'{dedos_arriba}', (10, 70),
cv2.FONT_HERSHEY_SIMPLEX, 3, (0, 255, 0), 3, cv2.LINE_AA)

            cv2.imshow('Finger Counting', image)
            if cv2.waitKey(5) & 0xFF == 27:
                break

    cap.release()
    cv2.destroyAllWindows()
```

6.2.Manual de Usuario del Traductor de Lenguaje de Señas

Introducción

Bienvenido al manual de usuario del Traductor de Lenguaje de Señas. Este documento le guiará a través del proceso de instalación, configuración y uso de la

aplicación. La aplicación está diseñada para ayudar a las personas con discapacidades auditivas y de habla a comunicarse mediante la traducción de señas en texto.

Requisitos del Sistema

Sistema Operativo: Windows, macOS o Linux

Python: Versión 3.6 o superior

Bibliotecas Necesarias:

OpenCV

TensorFlow

PyTorch

Tkinter

PyQt5

Cámara: Integrada o externa para la captura de gestos

- **Instalación**

Instalar Python:

Descargue e instale la última versión de Python desde python.org.

Instalar Bibliotecas Necesarias:

Abra una terminal o línea de comandos y ejecute los siguientes comandos:

pip install opencv-python

pip install tensorflow

pip install torch

pip install tk

pip install pyqt5

Descargar la Aplicación:

Descargue o clone el código fuente del proyecto desde el repositorio correspondiente.

Ejecutar la Aplicación:

Navegue al directorio del proyecto en la terminal y ejecute:

python main.py

- **Uso de la Aplicación**

Interfaz Inicial:

Al iniciar la aplicación, se mostrará una ventana con tres botones: Letras, Números y Frases.

Seleccionar una Opción:

Haga clic en uno de los botones para seleccionar el tipo de señas que desea traducir.

Letras: Para traducir letras individuales.

Números: Para traducir números.

Frases: Para traducir frases comunes.

Abrir la Cámara:

Al seleccionar una opción, se abrirá la cámara y se mostrará la vista en tiempo real.

Realizar Gestos:

Coloque su mano frente a la cámara y realice los gestos correspondientes a la opción seleccionada.

La aplicación reconocerá los gestos y mostrará el resultado en la pantalla.

- **Resultados:**

  El texto traducido de los gestos aparecerá en la pantalla en tiempo real.

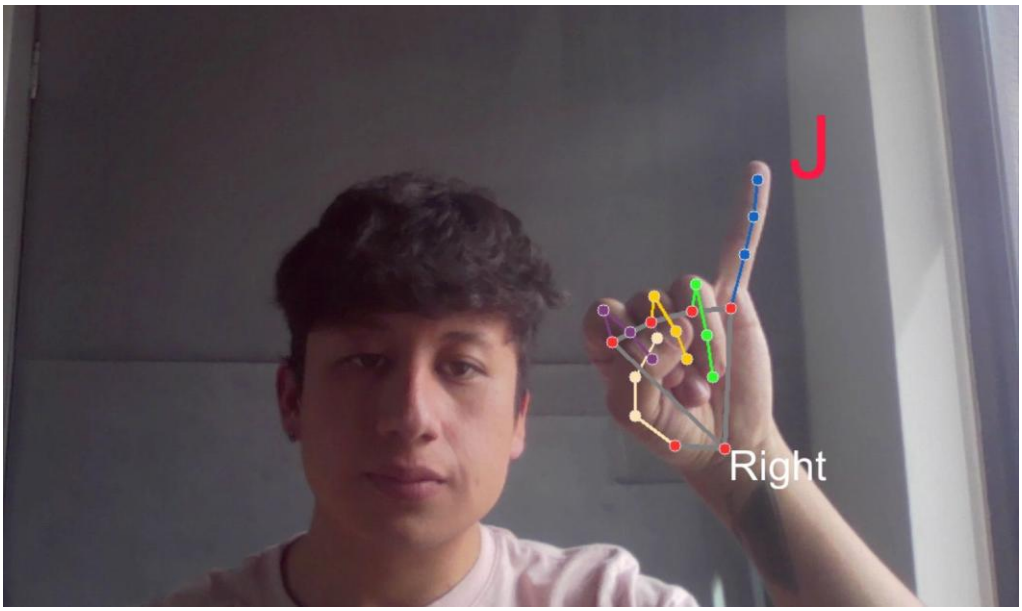
- **Ejemplos de Uso**

Reconocimiento de Letras:

Seleccione el botón Letras.

Realice gestos de letras individuales frente a la cámara.

El sistema mostrará las letras reconocidas en tiempo real.

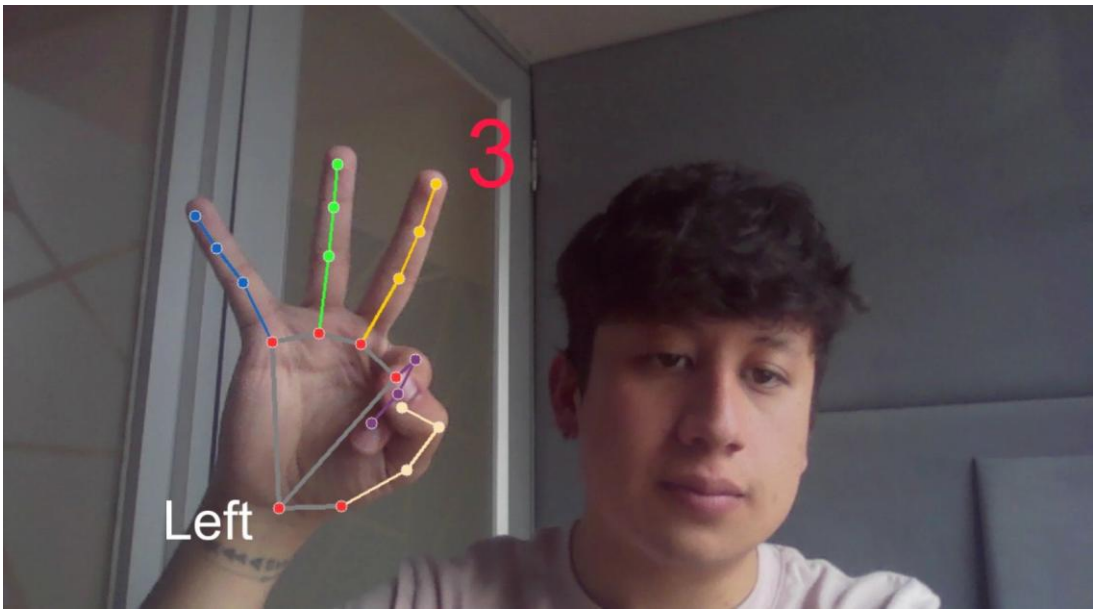


Reconocimiento de Números:

Seleccione el botón Números.

Realice gestos de números (0-10) frente a la cámara.

El sistema mostrará los números reconocidos en tiempo real.

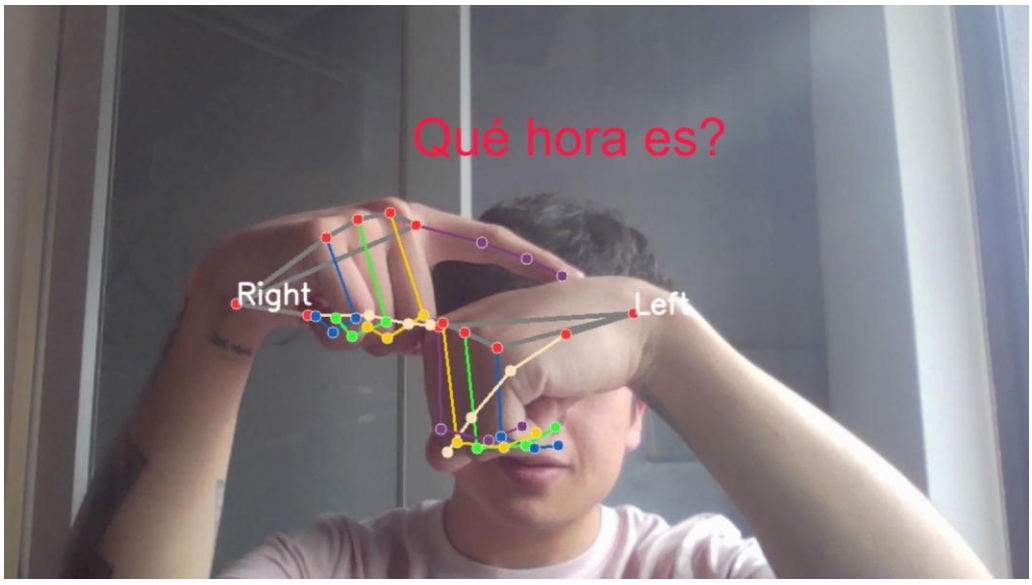


Reconocimiento de Frases:

Seleccione el botón Frases.

Realice gestos de frases comunes frente a la cámara.

El sistema mostrará las frases reconocidas en tiempo real.



- **Resolución de Problemas**

La cámara no se abre:

Asegúrese de que la cámara esté correctamente conectada y no esté siendo utilizada por otra aplicación.

Verifique los permisos de la cámara en la configuración del sistema operativo.

El reconocimiento de gestos no funciona correctamente:

Asegúrese de que la iluminación sea adecuada y que la cámara tenga una vista clara de los gestos.

Verifique que las bibliotecas necesarias estén correctamente instaladas y actualizadas.

La aplicación se cierra inesperadamente:

Revise la terminal o la línea de comandos para mensajes de error.

Asegúrese de estar utilizando la versión correcta de Python y las bibliotecas necesarias.

BIBLIOGRAFÍA

Libros y artículos académicos:

Fernández Escribano, G. (2009). *Extreme Programming Explained*. Addison-Wesley.

García, R. (2020). *Real-time Sign Language Translation: Challenges and Advances*. Journal of Computer Vision, 35(2), 112-125.

Van Rossum, G., & Drake, F. L. (2010). *Python Tutorial*. Addison-Wesley Professional.

DuBois, P. (2008). *MySQL*. Addison-Wesley Professional.

Hills, M., & Klint, P. (2017). *PHP Programming*. Springer.

Stauffer, M. (2019). *Laravel: Up & Running*. O'Reilly Media.

Sitios web y documentación en línea:

TensorFlow. (2024). *TensorFlow Documentation*. Recuperado de https://www.tensorflow.org/

OpenCV. (2024). *OpenCV Documentation*. Recuperado de https://opencv.org/

PyTorch. (2024). *PyTorch Documentation*. Recuperado de https://pytorch.org/

Tkinter. (2024). *Tkinter Documentation*. Recuperado de https://docs.python.org/3/library/tkinter.html

PyQt. (2024). *PyQt Documentation*. Recuperado de https://www.riverbankcomputing.com/software/pyqt/intro

XP (Extreme Programming). (2024). *Extreme Programming Explained*. Recuperado de https://www.extremeprogramming.org/

Python Software Foundation. (2024). *Python Documentation*. Recuperado de https://www.python.org/doc/

Visual Studio Code. (2024). *Visual Studio Code Documentation*. Recuperado de https://code.visualstudio.com/docs

Artículos y blogs relevantes:

Mestre, A. (2015). *The Importance of Front-end Development*. Web Development Journal. Recuperado de https://webdevjournal.com/frontend-importance

Quintana, J. (2023). *Back-end Development: Key Aspects and Best Practices*. Tech Insights Blog. Recuperado de https://techinsightsblog.com/backend-development

Rodríguez, M. (2018). *The Lifecycle of Extreme Programming*. Agile Practices Journal. Recuperado de https://agilepracticesjournal.com/xp-lifecycle

Documentación de herramientas y bibliotecas:

OpenCV Documentation. (2024). OpenCV. Recuperado de https://docs.opencv.org/

TensorFlow Documentation. (2024). TensorFlow. Recuperado de https://www.tensorflow.org/

PyTorch Documentation. (2024). PyTorch. Recuperado de https://pytorch.org/docs/

Tkinter Documentation. (2024). Python Software Foundation. Recuperado de https://docs.python.org/3/library/tkinter.html

PyQt Documentation. (2024). Riverbank Computing. Recuperado de https://www.riverbankcomputing.com/software/pyqt/intro

GLOSARIO

A

Algoritmo de reconocimiento de gestos: Secuencia de pasos computacionales diseñados para interpretar y convertir movimientos de señas en texto o voz.

Aprendizaje Automático (Machine Learning): Campo de la inteligencia artificial que permite a los sistemas aprender y mejorar a partir de datos sin ser explícitamente programados para cada tarea específica.

API (Interfaz de Programación de Aplicaciones): Conjunto de definiciones y protocolos que permite la comunicación entre diferentes sistemas de software.

C

Cámara de alta resolución: Dispositivo utilizado para capturar imágenes con gran detalle, esencial para el reconocimiento preciso de gestos en el sistema de traducción de señas.

CNN (Red Neuronal Convolucional): Tipo de red neuronal utilizada principalmente para el análisis de imágenes, incluyendo el reconocimiento de gestos en lenguaje de señas.

Controlador: Componente en la arquitectura de software que gestiona la lógica del sistema y la interacción con el usuario.

D

Deep Learning (Aprendizaje Profundo): Subcampo del aprendizaje automático que utiliza redes neuronales con muchas capas para modelar patrones complejos en grandes volúmenes de datos.

E

Extreme Programming (XP): Metodología ágil de desarrollo de software que se enfoca en la entrega rápida y continua de software de alta calidad mediante la colaboración constante con el cliente y la mejora continua.

F

Framework: Conjunto de herramientas y bibliotecas que proporciona una estructura estándar para el desarrollo de aplicaciones, facilitando la creación y el mantenimiento del software.

I

Interfaz de Usuario (UI): Parte del sistema con la que interactúan los usuarios, diseñada para ser accesible y fácil de usar.

Integración Continua: Práctica de desarrollo de software que implica la integración frecuente del código en un repositorio compartido para detectar problemas de manera temprana.

J

JSON (JavaScript Object Notation): Formato de intercambio de datos que utiliza texto legible por humanos para representar estructuras de datos, utilizado para almacenar y transmitir datos en el sistema.

L

Lenguaje de Programación Python: Lenguaje de programación de alto nivel conocido por su simplicidad y versatilidad, utilizado en el desarrollo del prototipo del sistema de traducción de lenguaje de señas.

M

Modelo: En la arquitectura de software MVC, el modelo define la lógica de negocio y la estructura de los datos del sistema.

Modelo-Vista-Controlador (MVC): Patrón de diseño que separa la lógica de negocio (modelo), la interfaz de usuario (vista) y el control de la aplicación (controlador) para facilitar la organización y el desarrollo del software.

O

OpenCV (Open Source Computer Vision Library): Biblioteca de visión por computadora de código abierto utilizada para el procesamiento de imágenes y el reconocimiento de gestos en tiempo real.

P

Procesamiento de Imágenes: Conjunto de técnicas utilizadas para mejorar y analizar imágenes digitales, esencial para la captura y interpretación de señas en el sistema.

R

Red Neuronal: Conjunto de algoritmos inspirados en la estructura y el funcionamiento del cerebro humano, utilizados para el reconocimiento de patrones complejos, como los gestos de lenguaje de señas.

Retroalimentación: Proceso de obtener y utilizar la opinión de los usuarios para mejorar continuamente el sistema.

S

Segmentación de Imágenes: Proceso de dividir una imagen digital en múltiples segmentos para simplificar el análisis y reconocimiento de las partes relevantes, como las manos en un gesto.

Sintaxis: Conjunto de reglas que define la estructura de un lenguaje de programación, como Python, utilizado en el desarrollo del sistema de traducción de señas.

T

TensorFlow: Biblioteca de código abierto para el aprendizaje automático, utilizada en el desarrollo de modelos de reconocimiento de gestos.

Tkinter: Biblioteca estándar de Python para la creación de interfaces gráficas de usuario (GUI).

U

Usabilidad: Medida de la facilidad de uso y la eficiencia con la que los usuarios pueden interactuar con el sistema.

V

Visión por Computadora: Campo de la informática que se ocupa de cómo las computadoras pueden ser hechas para obtener una comprensión de alto nivel a partir de imágenes o videos digitales, esencial para el reconocimiento de señas en el sistema.

W

Widget: Componente de una interfaz gráfica de usuario que permite interactuar con el software, como botones o menús desplegables.