Universidad de La Habana
Facultad de Matemática y Computación

# Generación gráfica de señas de la Lengua de Señas Cubana: una primera aproximación

Autor:
**Luis Enrique Dalmau Coopat**

Tutores:

**Dr. Yudivián Almeida Cruz**
**Lic. Leynier Gutiérrez González**

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencia de la Computación

5 de diciembre de 2022

Dedico este trabajo a mis padres, Liliana y Enrique, quienes con su amor y apoyo me han permitido lograr este sueño y a mis abuelos que ya no me acompañan en este plano terrenal , Benita, Luis, Raciel, Tata, Elsa y Kim, quienes estuvieran muy felices de verme cumplir este reto.

Especial dedicación también a toda la comunidad sordo-muda de Cuba por perseverar a pesar de todas las adversidades.

# Agradecimientos

A mi familia, en especial a mis padres; mis abuelos; Vilma, Ada y Enrique; a mis tíos, Ernesto y Belinda y al Moise por siempre apoyarme y guiarme a pesar de lo duro que ha sido el camino.

A mi prometida por acompañarme en estos años tan duros y ser uno de los pilares de mi vida; a sus padres y demás familia política, en especial a mis suegros por acogerme como uno más y correr conmigo a donde fuera siempre que hizo falta.

A mis amigos, Luiso, Miguel, Claudia, Marcos, Laurita, Wata, Yasmín, Jessy, Klever, Penny, Menor, Pollín, Betina, Karl Lewis, Raúl, Buti, Tigre por alumbrarme el camino cuando no era muy claro, además de siempre lograr sacarme una risa y momentos de diversión.

A Idania y a Carmen por siempre ayudarme cuando más lo necesitaba y velar porque cumplamos nuestras metas.

A mis demás compañeros y profesores por todos los conocimientos y experiencias que hacen de mi la persona que soy.

A mis tutores, Yudivián y Leynier por tenerme paciencia y abrirme las puertas a este magnífico tema de investigación.

A todas aquellas personas que de una forma u otra han aportado su granito de arena para que esta investigación sea hoy lo que es.

# Opinión del tutor

Es una responsabilidad de la sociedad fomentar y establecer sistemas que permitan a las personas con discapacidad vivir plenamente. Las personas con limitaciones en el habla y en su capacidad auditiva tienen muchas dificultades en su comunicación con otras personas. En este contexto, las tecnologías de la información y las comunicaciones pueden proporcionar soluciones. Sin embargo, en Cuba no se han desarrollado muchas investigaciones y/o desarrollos en este campo. Por ello, el Grupo de Investigación en Inteligencia Artificial de la Universidad de La Habana comenzó a trabajar en una línea de investigación cuyo objetivo final es crear una plataforma útil para mejorar la comunicación de las personas con discapacidad auditiva y del habla mediante el uso de la Lengua de Señas Cubana.

En este sentido, el trabajo de diploma "Generación gráfica de señas de la Lengua de Señas Cubana: una primera aproximación" del estudiante Luis Enrique Dalmau Coopat es una continuación de la línea de investigación antes mencionada. Como parte de este trabajo se profundiza en el estado del arte de diversos campos relacionados con la escritura de señas, con énfasis en su estructura y composición, el reconocimiento de señas, la tecnología de avatares, la interpolación de movimiento y los modelos generativos donde se observó un aumento en la aceptación de los modelos que utilizan inteligencia artificial y aprendizaje de máquina. Se propone una estructura de métodos y clases para la interpolación y adición adecuadas de una secuencia de señas, logrando así una generación precisa de movimientos suaves y claros del esqueleto del avatar. La propuesta realizada para la graficación de señas de la Lengua de Señas Cubana, que utiliza el corpus generado por investigaciones anteriores, se concreta en un prototipo de clases que utiliza interpolación e integración numérica para resolver los problemas en cuestión. El resultado es un avatar capaz de ejecutar de manera suave y armoniosa los movimientos asociados a cada token de seña, incluso cuando se combinan señas de manera extensa.

Durante la realización del trabajo se enfrentaron disímiles problemáticas cómo la curación y normalización de los datos que evidencian la importancia de continuar trabajando en la recolección de más y mejores datos para futuras investigaciones. Además el estudiante propone ideas y pautas a seguir para desarrollos posteriores que se encarguen de generar avatares realistas utilizando modelos generativos.

El estudiante Luis Enrique Dalmau Coopat trabajó arduamente con una gran determinación, capacidad de trabajo y habilidades, tanto en la gestión como en el desarrollo e investigación. Además, su trabajo tiene un gran valor social por la utilidad de la línea de investigación continuada con este proyecto. Por todo ello, solicitamos que se le otorgue la mejor calificación posible y que, de este modo, obtenga el título de Licenciado en Ciencias de la Computación.

| | |
|---|---|
| Dr. Yudivián Almeida Cruz | Lic. Leynier Gutiérrez González |

# Resumen

Este trabajo sigue un enfoque nunca antes probado en el contexto de las investigaciones de la Lengua de Señas Cubana. Muchos países han avanzado en el aspecto de generación automática de avatares, probándose los transformadores y enfoques no autorregresivos, los cuales han brindado excelentes resultados en los respectivos países de dichas investigaciones pero se hace necesario tener un vasto conjunto de datos de párrafos u oraciones contra la correspondiente lengua de señas. Se presenta una investigación acerca del uso de una propuesta basada en interpolación e integración numérica para la generación continua de avatares para la Lengua de Señas Cubana utilizando el corpus de señas aisladas brindado por el Centro Nacional para el Desarrollo y Superación del Sordo (CENDSOR) como los únicos datos disponibles.

# Abstract

This work follows an approach never tested before in the context of Cuban Sign Language research. Many countries have advanced in the aspect of automatic generation of avatars, testing the transformers and non-autoregressive approaches, which have provided excellent results in the respective countries of those investigations but it is necessary to have big dataset of paragraphs or sentences corresponding to sign language. It is presented an investigation about the use of a proposal based on interpolation and numerical integration for the continuous generation of avatars for the Cuban Sign Language using the corpus of isolated signs provided by the National Center for the Development and Overcoming of the Deaf (CENDSOR, by it's spanish meaning) as the only available data.

# Índice general

# Índice de figuras

# Ejemplos de código

# Introducción

La población sordomuda es un sector de la población no despreciable, en cuanto a desempeño social, a la cual se debe prestar gran atención. Muchas de estas personas sordomudas esconden un verdadero potencial humano el cual no pueden desarrollar en su máxima capacidad debido a los impedimentos sociales en que incurren, a causa del problema comunicativo mayoritariamente. Con este trabajo se quiere dar pasos para disminuir esa brecha comunicativa lo máximo posible para alcanzar un mundo mejor, en el que ser sordomudo no represente una limitación en el proceso de la comunicación social, ni en su desarrollo individual.

Alrededor de 72 millones, de las 8 mil millones de habitantes en este mundo, son personas sordas que utilizan lenguas de señas para comunicarse, según la Federación Mundial de Sordos (WFD, por sus siglas en inglés)[16]. Aproximadamente, más del 80 % de esos 72 millones viven en países en desarrollo, donde las autoridades locales no promueven ni apoyan políticas o acciones para suplir sus necesidades o incluirlos en la sociedad.

En Cuba, la Asociación Nacional de Sordos de Cuba (ANSOC) cuenta con alrededor de 25 mil asociados y aproximadamente 400 intérpretes registrados, de un total de 11 millones aproximadamente de habitantes, según especifica Yoel Moya, Subdirector de Investigación del Centro Nacional de Superación y Desarrollo del Sordo (CENDSOR). Sería asombroso imaginar la cantidad de tiempo, dinero y preparación que se ahorraría el país si se pudieran generar, con el vocabulario existente, avatares para las señas conocidas.

Como es una proporción abrumadora, las cifras del número de personas sordomudas representan menos del 1 % del total de personas, tanto a nivel mundial, como nacional. Por tanto, sería prácticamente imposible para una persona sordomuda llegar a entenderse con cualquier persona en una actividad del día a día.

La solución que se propone es generar lengua de señas en forma de video con un avatar para que, en un futuro, las personas sordomudas que sepan lengua de señas puedan entenderse con la otra parte de la población que no habla dicha lengua. De ser lo suficientemente preciso puede servir, además, a modo de tutorial para aquellas personas que no sepan lengua de señas y puedan aprender mediante la repetición de los gestos del avatar.

En trabajos anteriores se refieren a esta problemática abordando el tema de la comunicación de la siguiente forma:

> "En la actualidad este problema de la comunicación con personas sordomudas es abarcado por disímiles campos de investigación, como la interpretación automática de las lenguas de señas, la generación de avatares señantes, entre otros. Si bien este proceso se puede entender como una traducción interlingual, las características específicas de las lenguas de señas hacen que su traducción e interpretación constituyan un proceso no tan sencillo que involucra componentes visuales. Es por dichos componentes que intervienen además varios campos de investigación como el procesamiento de lenguaje natural y la visión por computadora para asistir al entendimiento y procesamiento correcto de las lenguas de señas."
>
> Gutiérrez−González[11]

# Problemática

En lo que se refiere a la generación de avatares para la lengua de señas existen avances significativos a nivel mundial. La Lengua de Señas Cubana es diferente de otras utilizadas en el mundo y de cualquier idioma hablado conocido. El Español se utiliza en muchos países con algunas modificaciones regionales pero que no impide la capacidad de comunicarnos entre todos. Por el contrario la lengua de señas cubana es única y, por tanto, Cuba es el único país responsable, en realizar estudios, investigaciones y herramientas para acelerar su aceptación.

Aunque en los últimos años en el país se han impulsado las tecnologías de la información y las telecomunicaciones, no se han utilizado suficientemente estos avances en pos de la generación de avatares para lograr una mayor inclusión de la sociedad hipoacústica. Es debido a ello que se hace importante fomentar estudios e implementar plataformas que apoyen e inciten a una mejor inserción en la sociedad de las personas que dependan de la Lengua de Señas Cubana para comunicarse.

# Objetivos del Trabajo

## Objetivo general

- Proponer un modelo que contribuya a la generación automática de avatares gráficos para la traducción automática de la lengua de señas cubana

### Objetivos específicos

- Estudiar el estado del arte en la generación de avatares para las lenguas de señas
- Identificar el conjunto de símbolos y acciones que se conocen por investigaciones previas de la lengua de señas cubana
- Proponer un modelo para las primeras etapas en la generación de avatares gráficos en la lengua de señas
- Realizar un conjunto de experimentos que permitan probar la viabilidad de su propuesta

## Propuesta de solución

Como método de solución se propone un sistema por partes, utilizando, primeramente, un preprocesamiento de los datos existentes obtenidos en investigaciones previas garantizando el correcto etiquetado de las señas. Dicho conjunto de datos es guardado en un corpus de extensión JSON para su posterior uso. Se crea una estructura para guardar la representación de dichas señas (tokens), lo que permite instanciarlos, manejarlos y concatenarlos, garantizando su correcta interpolación y su posterior exportación a vídeo o graficado, utilizando motores gráficos.

## Estructura del trabajo

Este trabajo se divide en 6 partes. El capítulo introductorio plantea la problemática y sus consecuencias en el contexto social correspondiente, describiendo, posteriormente, los objetivos y un breve resumen de la propuesta del trabajo, finalizando con la explicación de su estructura. El capítulo 1 aborda un estudio realizado sobre el estado del arte del campo, mostrando varios conceptos e ideas que se aplican en los capítulos posteriores. El capítulo 2 presenta el diseño de la propuesta basada en distintos tipos de interpolación. En el capítulo 3 se explica la implementación de un prototipo del modelo propuesto, los experimentos para validar la propuesta y los resultados obtenidos. Seguidamente, se exponen las conclusiones del trabajo y un grupo de recomendaciones para investigaciones y desarrollos futuros. Culmina con la lista de las referencias bibliográficas utilizadas para su elaboración.

# Capítulo 1

# Estado del Arte

## 1.1. Lengua de Señas

La lengua de señas es el principal medio de comunicación para los sordos. Es un idioma que utiliza signos o señas en lugar de sonidos para comunicarse. Los signos se producen utilizando las manos, la cara y el cuerpo. Los mismos se usan en todo el mundo y existen más de 300 variantes. Las lenguas de señas son todas diferentes, no son inteligibles entre sí, no son universales y tienen su propia gramática y sintaxis, las cuales difieren de las del lenguaje hablado.

### 1.1.1. Escritura de señas Sutton

Sutton SignWriting, comúnmente conocida como SignWriting, es un sistema de escritura de lenguas de señas desarrollado en 1974 por Valerie Sutton, una bailarina que, dos años antes, había desarrollado la escritura danzaria (DanceWriting). Es muy específica y visualmente icónica, tanto en las formas de los personajes, que son imágenes abstractas de las manos, la cara y el cuerpo, como en su disposición espacial en la página, que no sigue un orden secuencial como las letras que forman las palabras escritas. Algunas formas estandarizadas más nuevas se conocen como el Alfabeto Internacional de Escritura por Señas (ISWA, por sus siglas en inglés).

#### Símbolos

En SignWriting, se utiliza una combinación de símbolos icónicos para formas de manos, orientación, ubicaciones corporales, expresiones faciales, contactos y movimiento[32][7] para representar palabras en lengua de señas.

La cantidad de símbolos es extensa y, a menudo, brinda múltiples formas de escribir un solo signo. Así como tomó muchos siglos para que la ortografía de los diversos

idiomas se estandarizara, la ortografía en SignWriting aún no está estandarizada para ninguna lengua de señas. Sutton originalmente la diseñó para que se escribiera horizontalmente (de izquierda a derecha), como en su idioma nativo, y desde el punto de vista del observador, pero luego lo cambió a vertical (de arriba a abajo) y desde el punto de vista del firmante, para ajustarse a los deseos de los escritores sordos.[31]

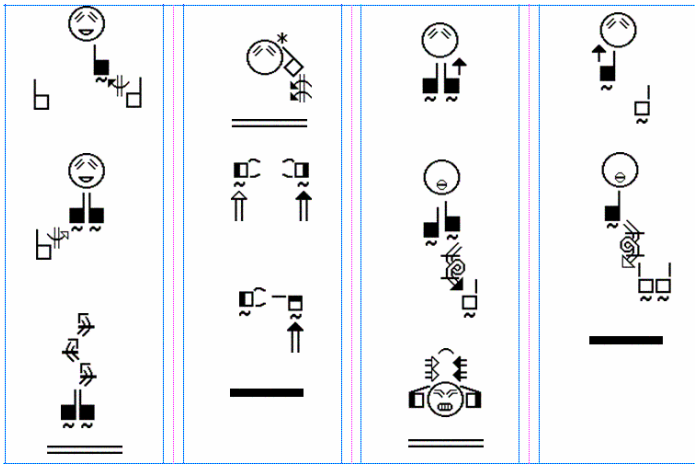


Figura 1.1: Ejemplo de una secuencia de glifos del sistema de escritura de señas Sutton SignWriting. De manera vertical cada una de las cuatro secciones representa una seña

Dado que SignWriting representa la formación física real de los signos en lugar de su significado [Fig 1.1], no se requiere ningún análisis fonético o semántico de un idioma para escribirlo. Una persona que ha aprendido el sistema puede "sentir" un signo desconocido de la misma manera que una persona que habla inglés puede "pronunciar" una palabra desconocida escrita en el alfabeto latino, sin siquiera saber qué significa el signo.[31]

Las palabras pueden estar escritas desde el punto de vista del firmante o del espectador. Sin embargo, casi todas las publicaciones utilizan el punto de vista del firmante y asumen que la mano derecha es dominante. Existen varios símbolos de puntuación que corresponden a comas, puntos, signos de interrogación y exclamación y otros símbolos de puntuación de otras escrituras. Estos se escriben entre signos, y las líneas no se rompen entre un signo y su siguiente símbolo de puntuación.

La escritura de seña facilita enormemente la estandarización de las lenguas de señas, pero, a pesar de ello, no se encuentra popularizada globalmente y no se tiene constancia, al momento del autor realizar este trabajo, de que sea empleada en Cuba, por lo que otras alternativas como el reconocimiento de lenguas de señas pudieran ser de más utilidad.

## 1.2. Reconocimiento de la Lengua de Señas

El reconocimiento de la lengua de señas es ampliamente usado en diversos proyectos actualmente [5]. Se emplea en la identificación de gestos usando conjuntos de datos existentes previamente. Las dos manos, la cabeza, los ojos, los labios y el cuerpo son, en esencia, lo utilizado en las lenguas de señas para realizar infinidad de gestos dedicados a funciones lingüísticas [28]. Al ser los gestos una parte fundamental de este tipo de lenguas, se requiere de su reconocimiento específico. Los métodos de reconocimiento de gestos se pueden dividir en dos grandes grupos, en dependencia si utilizan un enfoque basado en hardware como Kinect u otros tipos de sensores, o un enfoque basado en software [19]. Dentro de este segundo enfoque destaca la detección de gestos por MediaPipe Holistic de Google, logrando muy buenos resultados en dichas tareas [10] [Fig 1.2] .

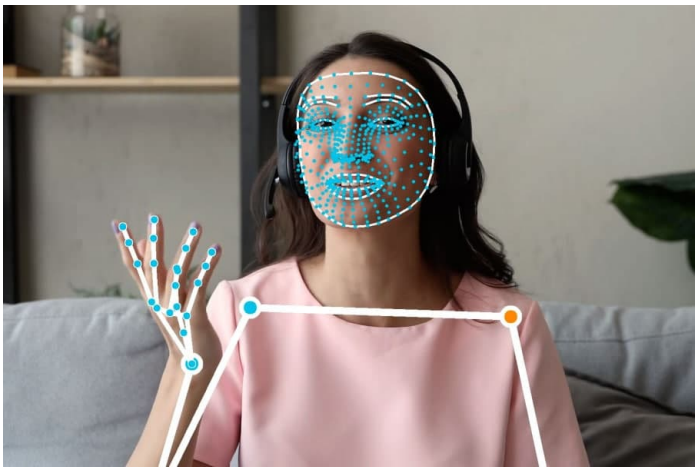


Figura 1.2: Software MediaPipe Holistic de Google, ejecutando un reconocimiento sobre una mujer.

Las diferentes vías de reconocimiento de señas aportan información valiosa para la comprensión kinética, estructural y lingüistica de cada lengua de señas. Esto ayuda en la comprensión en un sentido, del oyente hacia el hipoacúsico, ya que este último es entendido por el oyente pero no en el otro sentido. Es por ello que se hace necesario el uso de formas de reforzar el entendimiento de la persona sorda hacia el oyente.

## 1.3. Tecnología de avatares

La traducción automática de lenguas de señas, aplicando la tecnología de generación de avatares, ha sido empleada en proyectos como ViSiCAST y eSIGN [véase Fig

1.3], apoyados por la Unión Europea [1] [2]. Se utilizan para traducir de las lenguas habladas a las lenguas de señas, lo cual solamente concierne la parte del avatar en sí. Las animaciones para las señas individuales son generadas basándose en las señas no manuales de datos complementarios y en señas individuales. Los procesos para generar los avatares incluyen una serie de pasos bastante complejos.

Figura 1.3: Muestra de los avatares de ViSiCAST, efectuando algunos movimientos

Los aspectos referentes a la velocidad, el ritmo y el tamaño deben ser tomados en consideración utilizando reglas para controlarlos, ser configurados por un humano o como resultado de un modelo de aprendizaje de máquina [20]. Otro de los usos que tienen estos métodos es lograr el anonimato en los videos y, cabe remarcar, que son relativamente nuevos comparándolos con otros métodos de traducción automática de lenguas de señas [15] [29].

Uno de los más grandes desafíos es que los avatares luzcan como humanos. En diversos estudios se comparan los gestos de los avatares con los gestos de los humanos incluyendo algunos trabajos que han obtenido buenos resultados en Reino Unido en cuanto a generar dichos avatares con rasgos humanos utilizando modelos generativos [30]. En Corea, usando el mismo corpus del trabajo anterior, se han logrado avances en un enfoque no auto-regresivo para lograr resultados incluso mejores [13].

Entre las tecnologías más utilizadas para la generación de avatares por animación se encuentran:

- Unity
- Unreal 4
- Blender
- Maya3D
- DirectX

Se han desarrollado lenguajes de marcado para automatizar la generación de avatares de una forma dinámica y flexible [18] [4]. El estado del arte en la generación de avatares no está completamente automatizado, la mayoría de las partes del proceso de generación incluyen intervención humana para lograr buenos resultados. Gran parte de las investigaciones miden la calidad de las animaciones de avatares de forma

perceptual y de estudios comprensivos con participantes sordos-mudos, incluyendo investigaciones metodológicas y recursos compartidos [12] [14].

En cuanto a la generación automática de avatares, han ocurrido recientes avances al poderse utilizar una interfaz de programación de aplicación de uno de los motores gráficos mencionados anteriormente. Se utilizan como medio para enlazar un archivo de captura de movimiento a una armadura predefinida en el motor gráfico, como por ejemplo una jerarquía de visión biótica (BVH, por sus siglas en inglés) a un diseño 3D de una persona [20]. Aún así, estas señas aisladas no son capaces de generar todo el vocabulario posible de la lengua de señas específica, por lo que convendría poseer una manera de crear cualquier secuencia de señas, dado un vocabulario establecido, mediante algún tipo de interpolación entre ellas.

## 1.4. Modelos Generativos

Durante la última década se ha visto un marcado incremento en lo que vienen siendo programas capaces de generar imágenes con un alto nivel de realismo, pero sin copiar nada de ninguna otra parte, siendo "auténticas" de cierta forma. Esto es gracias a los denominados modelos generativos. Una nueva generación de modelos de redes neuronales capaces de aprender a hacer imágenes desde cero.

### 1.4.1. Redes Generativas Adversariales

Las Redes Generativas Adversariales (GANs) han logrado resultados impresionantes en varias tareas de síntesis de imágenes, y se están convirtiendo en un tema popular en la investigación de la visión por computadoras, debido al impresionante rendimiento que han alcanzado en diversas aplicaciones. Este modelo se ha impuesto desde que Goodfellow[9] lo propuso en 2014. Las GANs consisten en un generador G que se utiliza para generar muestras realistas a partir de ruido aleatorio e intenta engañar al discriminador D. El discriminador D se utiliza para identificar si la muestra es real o generada por G. El generador y el discriminador compiten entre sí hasta que el discriminador no puede distinguir entre las imágenes reales y las falsas [Fig 1.4].

### 1.4.2. Redes Generativas Adversariales Condicionales

Las GANs condicionales (CGAN) consisten en un Generador y Discriminador condicionados durante el entrenamiento mediante el uso de alguna información adicional [Fig 1.5]. Esta información auxiliar podría ser, en teoría, cualquier cosa, como una etiqueta de clase, un conjunto de etiquetas o incluso una descripción escrita [17].

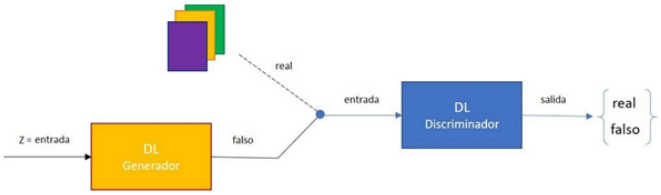



Figura 1.4: Diseño de la arquitectura de redes adversariales generativas. En amarillo con una G se identifica al generador, mientras que el discriminante se identifica con el color azul y una D en el nombre. Arriba se puede apreciar la entrada de datos.

Durante el entrenamiento de CGAN, el Generador aprende a producir ejemplos realistas para cada etiqueta o descripción en el conjunto de datos de entrenamiento, y el Discriminador aprende a distinguir los pares ejemplo-etiqueta falsos de los pares ejemplo-etiqueta reales. [17] Por lo tanto, para engañar al discriminador, no basta con que el generador de CGAN produzca datos de aspecto realista. Los ejemplos que genera también deben coincidir con sus etiquetas. Una vez que el Generador está completamente entrenado, nos permite especificar qué ejemplo queremos que sintetice el CGAN pasándole la etiqueta deseada.[17]

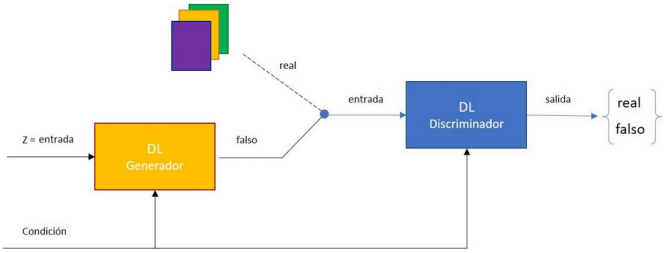



Figura 1.5: Diseño de la arquitectura de redes adversariales generativas condicionales. En amarillo con una G se identifica al generador, mientras que el discriminante se identifica con el color azul y una D en el nombre. Abajo de ellos se puede apreciar la entrada de datos

### 1.4.3. Imágenes a partir de imágenes

La traducción de imagen a imagen mediante redes generativas adversariales ha suscitado una gran atención en la investigación sobre el aprendizaje supervisado y no supervisado. Las GAN de ruido a imagen generan imágenes realistas a partir de muestras aleatorias de ruido, mientras que las GAN de imagen a imagen generan imágenes diversas a partir de imágenes. Se han propuesto muchas variantes de GAN,

que han logrado buenos resultados en este tipo de tareas de traducción. El objetivo de la traducción de imágenes es aprender el mapeo del dominio de la imagen de origen al de la imagen de destino, cambiando el estilo o algunas otras propiedades mientras mantiene el contenido de la imagen[33].

#### Texto a imágenes

En esta formulación, en lugar de utilizar ruido u otra imagen como entrada para el Generador, se utiliza una descripción textual de lo que se desea generar. Dicho texto es transformado primero en una incrustación (embedding) de texto, el cual se concatena con el vector de la imagen a generar y luego se da como entrada al Generador. Esta formulación ayudará al Generador a crear imágenes alineadas con la descripción de entrada en lugar de generar imágenes aleatorias[33].

### 1.4.4. Otras formas de generación de imágenes

Existen otras formas de generación de imágenes como el *image inpainting*, la cual es una técnica de restaurar y reconstruir partes de una imagen basadas en la información de fondo de la misma. La edición de imágenes, la cual manipula principalmente las imágenes a través del color e interacciones geométricas para completar tareas como la deformación y la mezcla de imágenes. Otra forma es la síntesis de imágenes humanas, la cual tiene como objetivo manipular la apariencia visual de las imágenes de los personajes, transfiriendo la pose de un personaje a la pose objetivo, que puede calcularse a partir de otros personajes.[33]

Además, en el último año, se ha visto un fuerte incremento de la adopción masiva de los modelos de difusión para la generación de imágenes, siendo Stable Difussion [26], un modelo de difusión latente, el que encabeza el podio, quedando segundos, Dall-E y Midjourney (otros modelos de difusión) [Fig 1.6]. Este éxito de Stable Diffusion se debe a que, tanto los pesos, como el código, se encuentran disponibles públicamente, siendo además eficiente y menos exigente que sus otros competidores, permitiendo ejecutarlo y obtener buenos resultados en máquinas modestas con al menos 8Gb de VRAM.

A pesar de poder ser capaces de hacer más realistas los vídeos de forma aislada (a través del uso de img2img) sigue siendo necesaria encontrar la forma de concatenar de manera fluida varias señas.

## 1.5. Interpolación

La interpolación es una forma de estimar datos nuevos en función de datos ya conocidos. Una técnica como esta puede ser aplicada para hacer predicciones en el

Figura 1.6: Imagen generada por Stable Diffusion

campo de la economía y otras ramas. La dificultad con aplicarla en ese campo es la volatilidad del mismo que, sumado a los errores de precisión de la técnica, han llevado a los economistas a buscar otros métodos.

### 1.5.1. Interpolación constante por partes

El método de interpolación más simple es ubicar el valor de datos más cercano y asignar el mismo valor. En problemas simples, es poco probable que se use este método, ya que la interpolación lineal (ver a continuación) es más sencilla y devuelve mejores resultados con un menor número de puntos, pero en la interpolación multivariante de mayor dimensión, esta podría ser una opción favorable por su velocidad y simplicidad.

### 1.5.2. Interpolación Lineal

Una de las formas más simples es la interpolación lineal, la cual utiliza dos puntos ya conocidos $(x_a, y_a)$, $(x_b, y_b)$ y se usa la siguiente fórmula para estimar el punto $(x, y)$:

$$y = y_a + (y_b - y_a)\frac{x - x_a}{x_b - x_a} \tag{1.1}$$

Esta ecuación anterior establece que la pendiente de la nueva recta entre $(x_a, y_a)$ y $(x, y)$ es igual a la pendiente de la línea entre $(x_a, y_a)$ y $(x_b, y_b)$.

### 1.5.3. Interpolación Cúbica

Otra técnica de interpolación muy utilizada es la cúbica. Para utilizarla se necesita que la función $f(x)$ sea derivable y conocer los valores cuando $x = 0$ y $x = 1$, se utiliza un polinomio de grado tres y su derivada como se muestra a continuación:

$$f(x) = ax^3 + bx^2 + cx + d \tag{1.2}$$
$$f\prime(x) = 3ax^2 + 2bx + c \tag{1.3}$$

Evaluando en los puntos $x = 0$, $x = 1$ y luego despejando se obtienen los siguientes valores:

$$a = 2f(0) - 2f(1) + f\prime(0) + f\prime(1) \tag{1.4}$$
$$b = -3f(0) + 3f(1) - 2f\prime(0) - f\prime(1) \tag{1.5}$$
$$c = f\prime(0) \tag{1.6}$$
$$d = f(0) \tag{1.7}$$

Con eso se tiene todo lo necesario para la interpolación cúbica.
En caso de que no se conozca la derivada de la función, se puede utilizar la derivada en puntos específicos. Si se tienen los puntos $(x_0, p_0)$, $(x_1, p_1)$,$(x_2, p_2)$,$(x_3, p_3)$:

$$f(0) = p_1 \tag{1.8}$$
$$f(1) = p_2 \tag{1.9}$$
$$f\prime(0) = \frac{p_2 - p_0}{2} \tag{1.10}$$
$$f\prime(1) = \frac{p_3 - p_1}{2} \tag{1.11}$$

Combinando estos valores con la fórmula de la interpolación cúbica se obtiene:

$$a = -\tfrac{1}{2}p_0 + \tfrac{3}{2}p_1 - \tfrac{3}{2}p_2 + \tfrac{1}{2}p_3 \tag{1.12}$$
$$b = p_0 - \tfrac{5}{2}p_1 + 2p_2 - \tfrac{1}{2}p_3 \tag{1.13}$$
$$c = -\tfrac{1}{2}p_0 + \tfrac{1}{2}p_2 \tag{1.14}$$
$$d = p_1 \tag{1.15}$$

Por lo que la fórmula de interpolación queda:

$$f(p_0,p_1,p_2,p_3,x) = (-\tfrac{1}{2}p_0 + \tfrac{3}{2}p_1 - \tfrac{3}{2}p_2 + \tfrac{1}{2}p_3)x^3 + (p_0 - \tfrac{5}{2}p_1 + 2p_2 - \tfrac{1}{2}p_3)x^2 + (-\tfrac{1}{2}p_0 + \tfrac{1}{2}p_2)x + p_1 \tag{1.16}$$

Esta forma de interpolar se conoce como Catmull-Rom spline.

### 1.5.4. Otros métodos de interpolación

Entre otros métodos que quedan sin comentar se encuentran:

- **La interpolación polinomial** la cual no es más que una generalización de la lineal y la cúbica.
- **La interpolación por splines** que utiliza polinomios de bajo grado en cada uno de los intervalos y elige las piezas del polinomio de modo que encajen entre sí sin problemas. La función resultante es llamada spline.
- **Interpolación mimética** la cual satisface las identidades del cálculo vectorial.

### 1.5.5. Interpolación de movimiento

La interpolación de movimiento (interpolación motriz) es una técnica de programación en la animación de personajes basada en datos que crea transiciones entre movimientos de ejemplo y extrapola nuevos movimientos.

Las acciones de ejemplo a menudo se crean a través de fotogramas clave o captura de movimiento. Sin embargo, la creación de fotogramas clave requiere mucha mano de obra y carece de variedades de movimiento, y ambos procesos dan como resultado movimientos que requieren mucho tiempo para modificarse. La interpolación motriz proporciona una alternativa mucho más rápida a la creación de nuevos movimientos a través de los mismos medios.[27]

## 1.6. Investigaciones y proyectos relacionados con la Lengua de Señas Cubana

En el área de las tecnologías de la información y las comunicaciones sobre la Lengua de Señas Cubana se pueden encontrar otros dos proyectos y/o trabajos.

El desarrollo de una aplicación móvil [Fig. 1.7] para dispositivos con sistema operativo Android, la cual esta enfocada en las familias con niños sordo-mudos, para brindarles el vocabulario necesario para interactuar con sus hijos [3] y la tesis de


Figura 1.7: Aplicación Android para la LSC [3]

grado referente al reconocimiento de la lengua de señas cubana usando métodos de traducción basados en inteligencia artificial y aprendizaje de máquina [11].

Se constata la existencia de una única investigación previa relacionada con la interpretación automática de la Lengua de Señas Cubana [11] pero ninguna en cuanto al tema que trata esta investigación.

# Capítulo 2

# Propuesta

En esta sección se abordan los problemas principales presentados y la solución encontrada, para cada uno, en las respectivas secciones y se profundiza sobre nuevos problemas hallados, los cuales fueron resueltos para un mejor efecto final. Por otra parte, se proponen formas de facilitar la utilización e inter-operabilidad de la propuesta presentada.

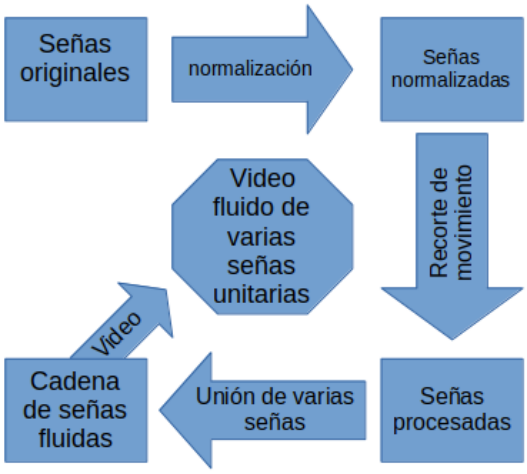



Figura 2.1: Esquema del enfoque a utilizar

Se utilizara un enfoque basado en la interpolación de movimiento para lograr la correcta concatenación y fluidez para la generación gráfica de señas. Dado un conjunto de señas unitarias se deberá efectuar un correcto procesamiento de las mismas y lograr unir varias de ellas en un video de manera fluida, haciendo así posible la generación de señas [véase Fig 2.1].

## 2.1. Definiciones

Se define como un video $V$ a la secuencia de puntos extraídos. $V_i$ constituye el i-ésimo cuadro (*frame*) del video. Un frame $F$ está compuesto por una secuencia de puntos en el espacio, representando las posiciones de las diferentes partes del cuerpo extraídas, $F_i$ representa el i-ésimo punto del frame y $F_{parte}$ representa los puntos asociados a la parte descrita en el frame.

### 2.1.1. Operaciones vectoriales

Las operaciones vectoriales constituye operaciones realizadas sobre los vectores. Estas operaciones pueden representar diferentes acciones a un nivel de abstracción mayor sobre los vectores. Sean $a$ y $b$ vectores o puntos de tres dimensiones entonces:

- $a - b$: Representa el vector de la dirección del punto $b$ hacia $a$.
- $a + b$: Representa la traslación del punto $a$ ($b$) en dirección $b$ ($a$).
- $a \times b$: Representa un vector perpendicular a $a$ y $b$.
- $\cos\theta = \frac{a \cdot b}{||a|| \cdot ||b||}$: Representa el coseno del ángulo $\theta$ entre los vectores $a$ y $b$.
- $\alpha a$: Representa el vector siendo escalado por un factor $\alpha$.

#### Rotación de Rodríguez

La rotación constituye en girar sobre un eje $k$ un vector $v$, $\theta$ grados, donde ambos vectores son unitarios. Este vector rotado puede ser hallado mediante:

$$rotación(v, k, \theta) = v\cos\theta + (k \times v)\sin\theta + k(k \cdot v)(1 - \cos\theta)$$

.

## 2.2. Preprocesamiento de los datos

Al utilizar datos por primera vez siempre es una buena práctica analizarlos y reconocer la estructura que poseen. La gran mayoría de las veces los datos con los que se trabaja vienen originalmente con fallas estructurales o de valores que no pertenecen al dominio que se trabaja. Es por este motivo que se hace fundamental, siempre que se vaya a realizar un trabajo con datos, realizarle su correspondiente análisis estructural y detectar posibles errores que se contengan en los mismos.

### 2.2.1. Obteniendo el conjunto de datos

De anteriores investigaciones contamos con un corpus de señas unitarias las cuales tienen la siguiente estructura :

$$(r, f, p, c)$$

donde :

- $r$ indica la representación a la que se accede de dicha frase nominal en el conjunto de datos.
- $f$ refiere al frame al que se accede de dicha representación
- $p$ representa el punto al que se indexa de dicho frame
- $c$ indica la componente del punto a la que se accede

Una seña unitaria no es más que una secuencia que representa a una frase nominal, dígase como ejemplo la seña que significa "américa del sur".

### 2.2.2. Limitando a una sola seña por frase

Luego de tener el conjunto de frases curadas y limpias, quedaba aún el problema de las palabras repetidas que tienen más de una seña, en dependencia de qué léxico del corpus brindado por el CENDSOR se revise. Para dicho problema se optó por escoger solamente la primera que apareciera, puesto que solo complejizaría más la tarea presentada dado que, para una misma frase en el corpus, podía incurrir en una variación bastante considerable, es decir, la señas para una misma palabra de un intérprete a otro variaban mucho para el caso de la palabra "paz" con 4 tipos de señas distintas (como si de una "falta de ortografía" se tratase).

### 2.2.3. Detección de problemas

A pesar de que todos los videos poseen una estructura similar, manteniéndose un fondo fijo de un color constante, los señantes presentes en los mismos son distintos unos de otros, así como también sus movimientos realizados. Aun refiriéndose a la misma seña, las diferencias no dejaban de aparecer, tanto en estructura corporal, como en velocidad y variaciones de la seña.

En algunos casos, los videos presentaban una secuencia de imágenes sin señante inicialmente, lo que generaba que el modelo dedicado al reconocimiento corporal fallara en dicha tarea, devolviendo un conjunto de ceros, en esos cuadros (frames). Algo similar, ocurría cuando no se detectaban bien las manos del intérprete (ya sea por

salirse de la ventana de enfoque de la cámara o encontrarse de forma perpendicular a la misma), haciendo que estas no aparezcan en algún que otro frame.

Para el primer caso la solución fue, sencillamente, no tomar en cuenta los frames que todos sus puntos fueran nulos; mientras que el segundo caso no era tan trivial puesto que, a pesar de no tener las manos, el cuerpo si se detecta y proporciona información importante del movimiento del señante por lo que se decidió dejar dichos conjuntos para una posterior revisión.

#### Análisis preliminar de los datos

Al analizar los datos se encontraron varios problemas:

- Los puntos asociados a las manos no se encontraban sobre las muñecas.
- Los puntos de las manos no existían en intervalos del video.
- El tamaño del señante varía entre videos.
- El video contiene frames en los cuales no se realiza ningún movimiento.

### 2.2.4. Limpieza de los datos

A todas las palabras se le hace limpieza con expresiones regulares para eliminar la mayoría de imperfecciones posibles en las palabras, el uso de “nn” en lugar de la “ñ ” y la presencia de números puesto que al agrupar las frases de varios léxicos distintos, estos se renombraban. Este procesamiento de las palabras es de utilidad para conformar el dataset lo más preciso posible, evitando la repetición de las frases.

#### Arreglando errores ortográficos

Durante la creación del conjunto de datos se detectaron, además, frases nominales inexistentes y/o con errores ortográficos, corrigiéndose una pequeña cantidad de estas a mano con un diccionario de palabras, dado que la gran mayoría de los errores ortográficos eran tildes ausentadas.

## 2.3. Problema de la visualización de los datos

Los datos ya procesados, del trabajo de Gutiérrez-González [11], son solamente arreglos de dimensión (número de interpretaciones de la seña, (número de frames de la seña, número total de puntos * 3)), lo cual no facilita interpretar o detectar posibles problemas. Para solventarlo se recurrió a graficar los puntos en un espacio tridimensional. Se realizaron las transformaciones pertinentes (rotación por el eje z y usando

un grado de elevación de 90 grados) para que se visualizara de frente al espectador, permitiendo, al tener guardada información de las uniones, la visualización de líneas entre dichos puntos espaciales.

Figura 2.2: Ejemplo parcial de la representación de los datos dentro de uno de los conjuntos de datos. Se puede apreciar la frase nominal “abandonar’ seguida de parte de su representación’

Se utilizan solamente los puntos del cuerpo y de las manos obtenidos a través de la investigación antes mencionada. Se constan de 67 puntos espaciales siendo 25 del cuerpo y la cabeza (obviando el tren inferior de puntos), 21 para la mano derecha e ídem para la mano izquierda [Fig 2.3].

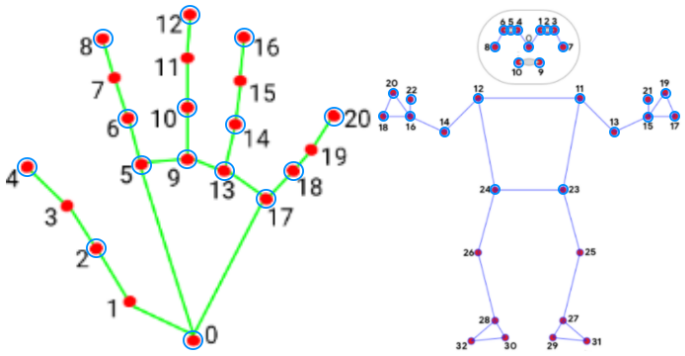
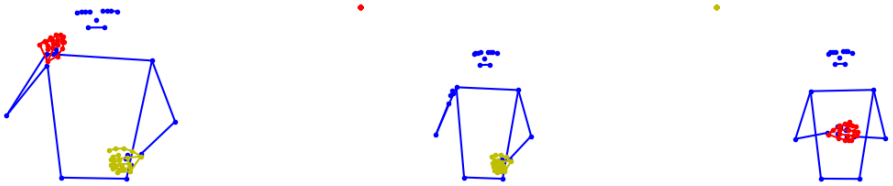


Figura 2.3: Algunos de los puntos que extrae Mediapipe Holistic. De izquierda a derecha apreciamos los puntos de la mano derecha y los puntos del cuerpo.

## 2.4. Problema de manos que no se encontraban sobre las muñecas

Las manos en los datos se encuentran definidas como una secuencia de puntos $F_{mano\_izquierda}$ y $F_{mano\_derecha}$, en ambos casos el primer punto de cada una re-



(a) Mano antes de desaparecer del token "aborto"

(b) Mano derecha desaparecida del mismo token al siguiente frame

(c) Mano izquierda desaparecida del mismo token en otro frame distinto

Figura 2.4: Ejemplo del segundo caso de manos faltantes.

presenta el punto asociado a la muñeca en la mano. A su vez el cuerpo posee otra representación de la muñeca $F_{muñeca\_cuerpo\_izquierda}$ y $F_{muñeca\_cuerpo\_derecha}$. Estos puntos en los datos deberían estar juntos o relativamente cercanos, sin embargo, en los datos se encontraban a una distancia considerable.

Para la solución del problema se calcula el vector dirección entre las representaciones de las muñecas del cuerpo y las muñecas de las manos y se trasladan los puntos de las manos en esta dirección, dejando ambas muñecas en el mismo sitio.

$$d = F_{muñeca\_cuerpo\_izquierda} - F_{muñeca\_mano\_izquierda} \tag{2.1}$$

$$F'_{mano\_izquierda} = F_{mano\_izquierda} + d \tag{2.2}$$

## 2.5. Problema de las manos faltantes

Este problema consiste en que en ciertas secciones del video los puntos asociados a las manos fallan al ser encontrados por los modelos de reconocimiento [véase Fig 2.4]. Este problema se divide en dos casos:

- Faltan en los frames internos de la seña [véase Fig 2.4]
- Faltan las manos en los frames extremos de la seña [véase Fig 2.5]

### 2.5.1. Frames internos

La solución propuesta para el primer caso es interpolar las posiciones intermedias faltantes teniendo como partida los últimos puntos existentes de las manos antes de desaparecer y como llegada los primeros puntos existentes luego de la desaparición. Para esto se define el operador interpolación $I$, el cual toma dos conjuntos de puntos, los puntos de partida y los de llegada, y un número representando la cantidad de elementos a devolver uniformemente escogidos en el intervalo a interpolar. El resultado de aplicar $I(V[i]_{mano\_izquierda}, V[i+k]_{mano\_izquierda}, k-1)$ es el conjunto interpolado de $k-1$ frames entre los frames $i$ e $i+k$. Al aplicar este método en cada intervalo en los cuales no estaba definida las manos se llenan estos huecos.

### 2.5.2. Frames extremos

En el segundo caso no se puede realizar la interpolación, debido a que no existe un conjunto de llegada o de partida. Al probar con distintos conjuntos de señas, resaltó el hecho de que las señas que comenzaban sin manos, durante un largo período de tiempo, causaban errores y fallos en la solución de la sección 2.6. Por lo que la forma de resolver dicho inconveniente devino del uso de una mano falsa, como si de una prótesis se tratara, la cual se compone del mismo número de puntos de una mano real y la misma fue situada gracias a que el cuerpo contiene puntos coincidentes para este mismo propósito.

Para resolver el problema se creó una mano en posición semi-extendida para que actuara de prótesis en los lugares donde ocurre el problema. Esta mano prótesis necesita ser modificada para que se pueda colocar en el lugar correcto en cuerpo. En primer lugar se realiza una rotación de esta de tal forma que quede alineada con el brazo, luego dicha mano es trasladada hacia la muñeca del cuerpo.

Se realizaron transformaciones espaciales, rotaciones y se seleccionaron matrices específicas para operar con los puntos de la mano falsa y así obtener su homologa contraria, además de permitir ajustarla al plano regido por el codo, la muñeca y los tres puntos coincidentes. Todo este conjunto de operaciones, sumado al hecho de utilizar la proporción de cuanto mide el antebrazo respecto a la medida desde la muñeca y la base del dedo medio, permitieron el correcto estiramiento y adaptación de la mano falsa a las distintas estructuras corporales evidenciadas en el conjunto de datos.

$$brazo\_izq = normalizar(F_{muñeca\_cuerpo\_izquierdo} - \dots F_{codo\_izquierdo}) \tag{2.3}$$

$$muñeca\_izq\_dirección = normalizar(F_{muñeca\_mano\_izquierda} - \dots F_{nudillo\_dedo\_medio\_izquierdo}) \tag{2.4}$$

$$eje\_rotación = normalizar(brazo_izq \times muñeca\_izq\_dirección) \tag{2.5}$$

$$ángulo = \arccos brazo\_izq \cdot muñeca\_izq\_dirección \tag{2.6}$$

$$F_{mano\_izquierda} = rotación(F_{mano\_izquierda}, eje\_rotación, ángulo) \tag{2.7}$$

$$d = F_{muñeca\_cuerpo\_izquierda} - F_{muñeca\_mano\_izquierda} \tag{2.8}$$

$$F_{mano\_izquierda} = F_{mano\_izquierda} + d \tag{2.9}$$

## 2.6. Problema de la fluidez en la unión de varias señas

La fluidez en la generación de avatares es de vital importancia, puesto que una de las metas es lograr la mayor semejanza posible al movimiento natural de una persona. Al pensar en concatenar dos señas lo primero que se imagina es un movimiento suave y que se logren identificar ambas secuencias, cosa que no ocurre al unir dos secuencias sin ningún tipo de procesamiento como los videos con que se cuentan. No todos los intérpretes terminaban la seña de la misma manera, ni en la misma posición [véase Fig 2.5], y en algunos casos hasta ocurrían periodos largos de inactividad entre que terminaba la seña y el final del grabación.

La solución encontrada a este problema fue interpolar entre ambas secuencias, obtando por la interpolación cúbica como candidata de mayor suavidad, si se requieren bastantes frames intermedios, y la lineal para casos de pocos frames o poca variación de movimiento. Esta interpolación se realiza escogiendo una cantidad fija de puntos en dependencia del método escogido. Dichos puntos son separados por componentes y se realiza la interpolación de forma individual en función de una lista de valores "t" asociados a cada una. Es decir, una lista t que enumera los valores de x es el argumento de la función a interpolar, dejándose por medio los valores asociados a la cantidad a interpolar, mientras que los valores obtenidos en la lista x serían los valores de la evaluación de dicha función en los valores de t conocidos.

### 2.6.1. Frames variables intermedios

Durante la concatenación también existe un diferencial de movimiento entre el final de una seña y el inicio de otra. El problema aquí es que existen poses que

terminan más distantes de su continuación que otras, haciendo que la transición sea o muy rápida o muy lenta, por esto se decide hacer una transición con frames variables en dependencia de la diferencia. Algunos de los videos terminan reposando ambas manos superpuestas sobre el vientre, mientras que en otros se termina descansando los brazos paralelamente al cuerpo o terminan de cualquier otra forma. Además, la distancia del señante a la cámara es, también, muy variable, por lo que se denota como movimiento dada la diferencia de posición.

### 2.6.2. Unión de señas

La unión de señas constituye la operación de unir dos señas para conformar una sola. Esta operación es realizada mediante el operador de interpolación al unir los videos $V_1$ y $V_2$, una vez realizadas todas las operaciones anteriores, de la siguiente forma:

$$V = I(V_1[n], V_2[0], frames\_de\_intercambio) \tag{2.10}$$

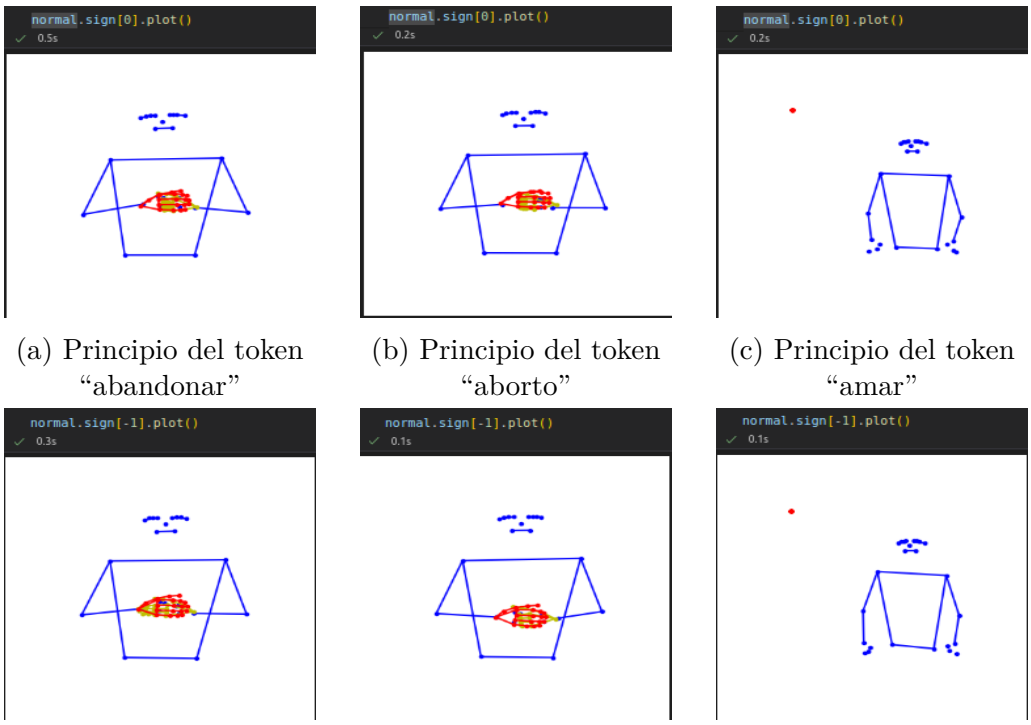
Se observa que hace falta conocer la cantidad de $frames\_de\_intercambio$, este valor debe depender de la diferencia entre $V_1[n]$ y $V_2[0]$ ya que vectores parecidos deben de tener menos frames de transición que vectores más distantes [véase Fig 2.5). Se define diferencia como la norma de la diferencia entre los vectores.

A criterio del autor se seleccionaron unos ejemplos en donde se sabía la variabilidad y la cantidad de frames buenas para el segmento a interpolar. Para obtener este valor se observaron varios valores de diferencia entre vectores y se decidió la cantidad de frames de intercambio para dicha diferencia [véase Tabla 2.1], con varias muestras se creó una función mediante interpolación de estos valores la cual permite conocer el valor de $frames\_de\_intercambio$ dada la diferencia.

Tabla 2.1: Valores a criterio del autor para interpolar según la variación de movimiento entre frames

| **Variación entre frames** | **Cantidad de frames** |
|---|---|
| 0.0 | 1 |
| 0.5 | 7 |
| 4.6 | 14 |

El hecho de usar uno u otro método de interpolación, implica que se seleccionan distintas cantidades de puntos, como se explica en el estado del arte [véase sección 1.5] . Siendo la más dinámica, con una menor cantidad de puntos, la interpolación lineal, pero, mínimo, se cuenta con 24 frames ya que equivalen a 1 segundo de un video de 24 fps(frames por segundo), lo cual es bastante estándar.




(a) Principio del token "abandonar"

(b) Principio del token "aborto"

(c) Principio del token "amar"

(d) Final del token "abandonar"

(e) Final del token "aborto"

(f) Final del token "amar"

Figura 2.5: Ejemplo de la diferencia entre extremos consecutivos de varios tokens, evidenciándose su variabilidad.

### 2.6.3. Problema del tamaño del señante variable durante el video

En los videos analizados se encuentran varios tipos de señante, estos tienen diferentes complexiones y también se encuentran grabados desde diferentes distancias e inclinaciones. Esto trae como consecuencia que los puntos extraídos no tengan la misma configuración. Para esto se normalizan los puntos de cada frame $F$ teniendo como referencia la distancia entre los hombros. Esta distancia en cada frame es mantenido constante a un valor de 0,5. Para lograr esto primero se calcula el factor de escalado del frame para que la distancia entre los hombros sea 0,5 y luego este factor es aplicado a todos los puntos, logrando así una mayor homogeneidad entre los distintos señantes.

$$factor\_de\_normalización = \frac{0{,}5}{||F_{hombro\_derecho} - F_{hombro\_izquierdo}||} \tag{2.11}$$

$$F = factor\_de\_normalización \cdot F \tag{2.12}$$

Con estas operaciones se mantiene una distancia constante de 0,5 en cada frame.

### 2.6.4. Inactividad en la secuencia

Este problema surge debido a la naturaleza de los videos analizados. En estos el señante empieza y termina en una posición de descanso por lo que en esta etapa no ocurre un movimiento relativo a la seña descrita. Esto implica que al realizar la concatenación de videos aparecieran periodos con estas posiciones de descanso, lo cual impedía una concatenación fluida. Esto no conlleva una solución tan trivial como simplemente recortar los frames con una cantidad fija, dado que cada video puede presentar anomalías que lo hagan parecer inactivo o pueden existir videos sin inactividad, con lo cual, al recortar, estaríamos eliminando parcialmente la seña. Para la solución de dicho problema se define una ecuación que mide el movimiento en el video, luego nos quedamos con el intervalo de video que contenga la mayor parte del movimiento total del mismo y que tenga un intervalo lo más pequeño posible.

La ecuación de movimiento se define como:

$$mov(i) = ||V[i+1] - V[i]|| \tag{2.13}$$

Esta ecuación cuantifica el movimiento entre un frame y el próximo y por su definición es no negativa [véase Fig 2.6], por lo tanto hallando su integral en el dominio se puede tener una medida del movimiento en el segmento. Se defina la integral de $mov$ en el intervalo $i$, $j$ como:

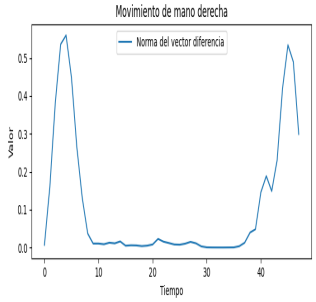
$$integral\_mov(i,j) \tag{2.14}$$

Se desea obtener un intervalo que contenga gran parte del movimiento pero que no sea el video completo, para esto se define el problema de optimización siguiente, donde $n$ es el frame final:

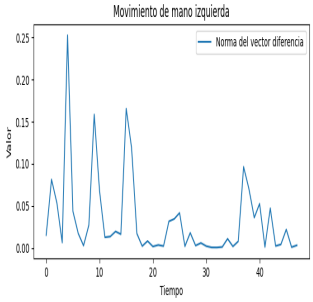
$$\max_{i,j} \frac{integral\_mov(i,j)}{j-i} \tag{2.15}$$

$$s.a: \quad integral\_mov(i,j) > integral\_mov(0,n) \cdot porciento\_de\_movimiento \tag{2.16}$$

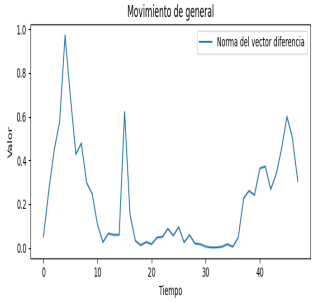
$$j > i \tag{2.17}$$



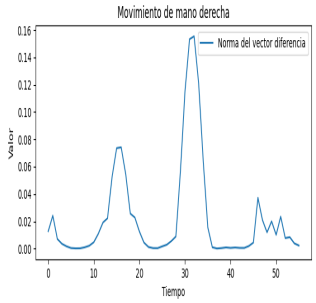

(a) Gráfica de movimiento de la mano derecha del token “amar”



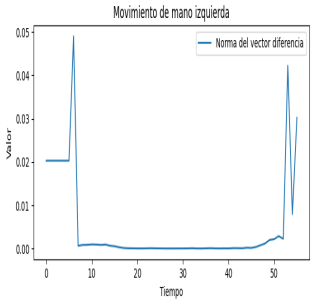

(b) Gráfica de movimiento de la mano izquierda del token “amar”



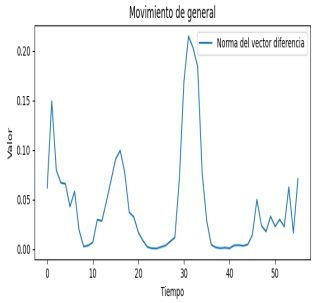

(c) Gráfica de movimiento general del token “amar”




(d) Gráfica de movimiento de la mano derecha del token “aborto”




(e) Gráfica de movimiento de la mano izquierda del token “aborto”




(f) Gráfica de movimiento general del token “aborto”

Figura 2.6: Gráficas de movimiento por manos y general de los tokens “amar” y “aborto”.

Obteniendo el intervalo que solucione el problema anterior se obtiene un nuevo video al quedarse solamente con los frames dentro de este intervalo, que aseguran que el movimiento en este es mayor al *porciento_de_movimiento* seleccionado. Dado que el problema es discreto la integral es calculada por métodos numéricos y el problema de optimización es resuelto probando los valores posibles de los intervalos $i, j$.

Resolver todos los problemas planteados anteriormente son fundamentales para la correcta generación de señas, puesto que, en caso de fallarse en solucionarse alguno de los problemas el resultado pudiera afectar negativamente la fluidez percibida por los usuarios, así como generar distorsiones durante la reproducción.

# Capítulo 3

# Detalles de Implementación y Experimentos

Para poder valorar la viabilidad de la propuesta realizada en este trabajo, es necesaria la implementación de un prototipo del modelo explicado en el capítulo anterior.

## 3.1. Herramientas y tecnologías utilizadas

### 3.1.1. Lenguaje de programación Python

Python es un lenguaje de programación de propósito general y alto nivel desarrollado por Guido van Rossum en 1991. Su filosofía de diseño enfatiza la legibilidad del código con el uso de sangría significativa. Python se tipifica dinámicamente y tiene incorporado un recolector de basura. Admite múltiples paradigmas de programación, incluida la programación estructurada, orientada a objetos y funcional. La más reciente versión liberada al momento de realizarse este trabajo es la 3.11 [25].

Es altamente empleado para ingeniería y análisis de datos, aprendizaje de máquina e inteligencia artificial gracias a sus vasta cantidad de bibliotecas como NumPy, Tensorflow, Keras, Pytorch, SciPy, Pandas y Matplotlib, entre otras creadas tanto por el mismo equipo de trabajo de Python como la propia comunidad. Para desarrollo web cuenta con marcos de trabajo (frameworks) como Django. Es altamente utilizado en la educación al ser de fácil aprendizaje y asimilación logrando así reducir la barrera de entrada al mundo de la programación a todo aquel interesado.

En la actualidad sigue manteniendo el primer puesto de los índices de TIOBE y PYPL ratificando el interés de gran parte de la población y de los empleadores por este lenguaje y las ventajas que ofrece. Grandes organizaciones como Google [23], el CERN [6], la NASA [24], Yahoo [22], Wikipedia, Amazon, Facebook, Instagram

[8], Spotify, entre otros. Para este trabajo se utilizó la versión 3.7 para lograr una retro-compatibilidad alta.

**numpy**

Constituye una biblioteca de Python de código abierto, la cual permite generar, tanto vectores como matrices, de grandes dimensiones y operar de manera cómoda y sencilla con ellos [21]. Consigue esto gracias a que utiliza internamente el lenguaje C para lograr efectuar de forma rápida operaciones muy costosas entre elevadas dimensiones. En las etapas posteriores a la limpiezas de los datos explicada en el capítulo anterior se utiliza Numpy para el trabajo con los datos.

**matplotlib**

Matplotlib es una biblioteca de gráficos creada en 2003 por John D. Hunter para el lenguaje de programación Python y su extensión matemática numérica NumPy. Proporciona una API orientada a objetos para incrustar gráficos en aplicaciones utilizando kits de herramientas GUI de uso general como Tkinter, wxPython, Qt o GTK. Es una biblioteca completa para crear visualizaciones estáticas, animadas e interactivas. Matplotlib hace que las cosas fáciles, pues sigan siendo fáciles y las difíciles sean posibles, como lo es crear gráficos con calidad de una publicación y hacer figuras interactivas que puedan hacer zoom, desplazarse, actualizar.

**scipy**

SciPy es una biblioteca de Python gratuita y de código abierto que se utiliza para la computación científica y la informática técnica. SciPy contiene módulos para optimización, álgebra lineal, integración, interpolación, funciones especiales, FFT (Transformada Rápida de Fourier), procesamiento de señales e imágenes, solucionadores de ODE (Ecuaciones Diferenciales Ordinarias) y otras tareas comunes en ciencia e ingeniería. Esta creada encima de la biblioteca Numpy anteriormente mencionada y fue desarrollada por la compañía Enthought en el año 2001.

**boto3**

Se utiliza el Kit de desarrollador de software (SDK por sus siglas en inglés) de AWS (Servicios Web de Amazon) para Python (Boto3) para crear, configurar y administrar servicios de AWS, como Amazon Elastic Compute Cloud (Amazon EC2) y Amazon Simple Storage Service (Amazon S3). El SDK proporciona una API (Interfaz para Programas de Aplicación) orientada a objetos, así como acceso de bajo nivel a los servicios de AWS.

Este SDK es empleado para acceder a los datos.

#### pathlib

Este es un módulo de Python que ofrece clases que representan rutas de sistemas de archivos con semántica apropiada para diferentes sistemas operativos. Las clases de ruta se dividen entre rutas puras, que proporcionan operaciones puramente computacionales sin Entrada/Salida (I/O por sus siglas en inglés), y rutas concretas, que heredan de rutas puras pero también proporcionan operaciones de I/O.

La clase *Path* de esta biblioteca es ampliamente usada en este trabajo.

#### json

Python tiene el módulo *json* incorporado, que nos permite trabajar con datos JSON (Notación de objetos JavaScript). JSON es un formato de archivo estándar abierto y un formato de intercambio de datos que utiliza texto legible por humanos para almacenar y transmitir objetos de datos que consisten en matrices y pares de atributos y valores. Es un formato de datos común con diversos usos en el intercambio electrónico de datos, incluido el de las aplicaciones web con servidores.

#### mediapipe

Desarrollada por Google [10], Mediapipe es una biblioteca de código abierto la cual utiliza distintos tipos de modelos para solucionar de forma conjunta varios problemas de detección [véase Fig 1.2]. De los modelos presentes en la biblioteca de Mediapipe solo fue utilizado el modelo Holistic, el cual engloba a los modelos para la detección de manos (modelo Hands), detección facial (modelo Face Mesh) y pose corporal (modelo Pose) realizando la sincronización de estos de manera eficiente garantizando un buen resultado final.

Para este trabajo solo fue necesario el uso de los modelos Hands y Pose de Holistic.

### 3.1.2. Google Colaboratory

Google provee un servicio gratuito en el navegador el cual brinda, de manera temporal, recursos computacionales (CPU,TPU,GPU,RAM,HDD) mientras se haga uso activo de los mismos. En dicha plataforma se le permite al usuario escribir y ejecutar código Python en un entorno basado en Jupyter Notebook.Siendo así una herramienta ideal para el aprendizaje de máquinas, el análisis de datos y la educación al posibilitar que personas de pocos recursos puedan acceder de forma fácil a herramientas para el aprendizaje de máquina las cuales muchas de ellas ya vienen incluidas en el entorno o son muy sencillas de instalar en el mismo. Cabe destacar que además del plan gratuito que incluye CPU de última generación, 13 Gb de RAM y 110 Gb de almacenamiento, nos da la posibilidad de utilizar Google Drive para ampliar el

almacenamiento y varios planes de pago para aumentar la capacidad computacional del entorno.

## 3.2. Implementación de un prototipo

Para el desarrollo de los métodos que solucionan las problemáticas resueltas en el capitulo anterior, y concreticen la propuesta ofrecida, se decide utilizar varias clases para un mayor nivel de abstracción y posibilidad de generalización de lo métodos propuestos.

### 3.2.1. Descargar datos y filtrado

Auxiliándose de la biblioteca *boto3* y *json* fue posible recorrer los elementos disponibles en el almacenamiento de Gutiérrez-González[11] para, una vez descargados, filtrar los frames que son enteramente ceros y corregir alguna de las palabras mal escritas que se denotan como inexistentes. Luego se guarda todo el corpus en un archivo *dataset.json* el cual al cargarlo nuevamente es utilizable como diccionario de forma instantánea. Todo este trabajo primario se realiza desde Google Colaboratory para poder conformar mejor el conjunto de datos.

#### Extracción de los Datos

Para la extracción de los datos se utilizó la biblioteca *boto3* con acceso al almacenamiento en la nube donde se encontraban los datos. Para ello se importa el método *client* de *boto3*, el cual es instanciado en la url donde se encuentra el conjunto de datos, además de las claves de acceso, las cuales por privacidad del dueño de las mismas se sustituyen por valores de "X" [véase Código 3.1].

```
from boto3 import client
s3_client = client(
    "s3",
    config=Config(
        signature_version="s3v4",
        retries={"max_attempts": 10},
        s3={"addressing_style": "path"},
    ),
    region_name="ams3",
    endpoint_url="https://ams3.digitaloceanspaces.com",
    aws_access_key_id="XXXXXXXXXXXXXXXX",
    aws_secret_access_key="XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
)
```

Ejemplo de código 3.1: Instanciar cliente s3

Una vez obtenido el cliente configurado se descargan los datos del mismo, conociéndose que se encuentran en el bucket "lsc-corpus", los de las señas aisladas, puesto que son los únicos datos con seguridad de la seña que corresponde a cada frase.

```
from pathlib import Path
root=Path("/content")
bucket="lsc-corpus"
paginator = s3_client.get_paginator('list_objects_v2')
pages = paginator.paginate(Bucket=bucket)
for page in pages:
    for obj in page['Contents']:
        extension = obj['Key'].split('.')[-1]
        name=obj['Key'].split('.')[0]
        if extension  in ['json']:
          drivepath=(root/f"tesis-generacion-lsc/{obj['Key']}").resolve()
          (drivepath/'..').resolve().mkdir(parents=True,exist_ok=True)
          drivepath.touch()
          s3_client.download_file(bucket,obj['Key'],str(drivepath))
```

Ejemplo de código 3.2: Descargar usando el cliente s3

Aquí se realiza un paginado para poder recorrer de manera adecuada los elementos del bucket para luego chequear la extensión de los archivos contenidos en cada página [véase Código 3.2]. Se conocen previamente que los datos anotados de los puntos de interés estaban recogidos en archivos con extensión *json* por cada frase. En la primera parte del código se importa la clase *Path* de *pathlib* para trabajar de manera rápida y efectiva con los directorios que se van creando.

### Limpieza de los datos

Una vez ya se posean los archivos *json* referentes a cada frase del corpus se pueden entonces cargarlos para limpiarlos y organizarlos en un dataset mejor estructurado donde estén todas las frases juntas.

En este código [véase Código 3.3] se utilizan expresiones regulares para la limpieza de las llaves, ya que incurrían en faltas de ortografía, errores tipográficos y de copiado. Al no poseer una forma clara de rectificar las tildes faltantes se decidió dejar así dichas palabras y recomendar el uso de un algoritmo de similitud antes de indexar en el dataset creado. La primera expresión elimina todo lo que esté entre paréntesis en las llaves, puesto que esto no aporta nada a la información semántica de los mismos. Luego se reemplaza todas las “nn” por una “ñ” , se eliminan los dígitos que hayan quedado de manera residual del primer método y todos los guiones entre palabras para que estén de manera individual.

```
import numpy as np
import json
import re
root_tesis = (root/f"tesis-generacion-lsc/").resolve()
cendsor_path = "cendsor-corpus/keypoints/"
wrong_words={ 'barsura':'basura',
              'insipido': 'insípido',
              'investar':'inventar',
              'ostion':'ostion',
              'panciencia':'paciencia',
              'sacapunta':'sacapuntas',
              'zapia':'zapya'}
word_poses_dict={}
for dirpath,dirnames,filenames in os.walk(root_tesis/cendsor_path):
  for filename in filenames:
    key,extension=filename.split('.')

    ############################
    # Limpieza de las llaves
```

```
    ############################

    key=re.sub(r"\s*\(.*\)","",key)
    key= key.replace("nn",'ñ')
    key=re.sub(r"\s?\d","",key)
    key = key.replace("_"," ")
    if key in wrong_words.keys():
      key=wrong_words[key]

    ############################

    with open((root_tesis/cendsor_path/filename).resolve(), 'r', encoding='utf-8') as json_list:
      poseframes=np.array(json.load(json_list))
      json_list.close()

      ##########################################################
      # Limpieza de los frames de los gestos que son 0
      ##########################################################
      filtro = [m!=0.0 for m in  np.mean(poseframes, axis=1)]

      poseframes=poseframes[filtro]

      ##########################################################

      single_gesture = poseframes.tolist()
      try:
        word_poses_dict[key].append(single_gesture)
      except:
        word_poses_dict[key]=[single_gesture]
```

Ejemplo de código 3.3: Cargar los json y limpiar las llaves

En el caso de los gestos, existen en el corpus frames en los cuales los modelos no detectaban nada y por tanto estos devolvían (0,0,0) para todos los puntos de interés de ese frame. Por tanto, se hace uso de la biblioteca *json* para cargar el archivo y de la biblioteca numpy para realizar la limpieza de una manera más cómoda y rápida. Luego de obtener la máscara (filtro) con la información de los frames vacíos, simplemente se indexa en el arreglo numpy guardando su resultado como si fuera ahora el arreglo original.

Como información adicional, se guarda el diccionario tanto en la nube como en el directorio local para su mejor aprovechamiento y eliminarnos pasos extras a la hora de probar el modelo.

### 3.2.2. Creación de las Clases

En esta subsección se trataran de forma más especifica las clases utilizadas en la implementación de la propuesta.

#### Clase TokenLSC

Una seña esta asociada a la clase *TokenLSC*, la cual está conformada por un nombre y una lista de Skeleton, los cuales no son más que una clase para abstraer conceptualmente un frame. La clase Skeleton cuenta con 3 Bodyparts para su inicialización, las cuales deben ser pasadas como argumentos a su constructor. Estas son la conceptualización de los puntos referentes a un tipo de parte del cuerpo, los cuales son representados mediante un enumerador (*enum*) inicialmente. La clase *TokenLSC*

cuenta con una serie de atributos y métodos para solventar de manera correcta los problemas planteados en el capitulo anterior, siendo estos métodos descritos a continuación, así como los correspondientes utilizados de Skeleton y Bodypart para ayudar a su cumplimiento.

La clase *TokenLSC* esta conformada por un nombre (el cual hace referencia al identificador de la seña), una lista de objetos de tipo Skeleton, los cuales a su vez están conformados por 3 partes de cuerpo (son la clase Bodypart creada para este propósito). Además, al construirse la clase *TokenLSC*, también se configuran valores necesarios para el graficado y animación utilizando la biblioteca de Matplotlib.

### Clase Skeleton

La clase Skeleton [véase Código 3.4] es una abstracción para mejorar el manejo de las partes del cuerpo. Corresponden a un frame de la seña en formato de video y contiene propiedades y métodos de utilidad para la solucionar los problemas antes vistos en el pasado capítulo.

```
class Skeleton:
    default_lhand = DefaultLeftHand()
    default_rhand = DefaultRightHand()
    def __init__(self, body: Bodypart, lhand: Bodypart, rhand: Bodypart):
    ...
    @property
    def bodyparts(self):
    ...
    @property
    def all_points(self):
    ...
    def get_position_default_hand(self, is_left: bool, hand_forearm_proportion=1/11.15,
     finger_proportion=1/0.68):
    ...
    def plot(self, figsize=(5.0, 5.0), elev=90, angle=1, dims=None, text=False, joints=True):
    ...
    def _transform_data(self, Data, i=0):
    ...
    def normalize(self):
    ...
```

Ejemplo de código 3.4: Clase de Skeleton

Cada esqueleto posee un método para graficarse en caso de requerirse y además posee los métodos necesarios para ajustar las manos por defecto que poseen en caso de que sus manos originales no existan.

### Clase Bodypart

Los objetos de tipo Bodypart [véase Código 3.5] son una conceptualización de las partes del cuerpo utilizadas en las estructuras internas de las clases Skeleton y *TokenLSC*. Se define un enumerador para clasificar de forma más cómoda y legible los tipos de partes del cuerpo presentadas.

```
from enum import Enum
class TypeOfBodyPart(Enum):
    BODY = 0
    LHAND = 1
    RHAND = 2
```

```
class Bodypart:
    def __init__(self, name: str, type: TypeOfBodyPart, points: ndarray):
    ...
    def normalize(self):
    ...
    def plot(self):
    ...
```

Ejemplo de código 3.5: Clase de Bodypart

```
class DefaultRightHand(Bodypart):
    def __init__(self):
            np_default_hand= np.array([
            [4.32893813e-01,  7.13910162e-01,  1.74915698e-07],
            ...
            [5.24085701e-01,  7.43879616e-01, -1.89451054e-02],])
        super().__init__("Default Right Hand", TypeOfBodyPart.RHAND, )
class DefaultLeftHand(Bodypart):
    def __init__(self):
        rhand = DefaultRightHand().points
        mirror_x_matrix = np.array([
            [-1, 0, 0],
            [0, 1, 0],
            [0, 0, 1],
        ])
        lhand = rhand @ mirror_x_matrix
        super().__init__("Default Left Hand", TypeOfBodyPart.LHAND, lhand)
```

Ejemplo de código 3.6: Clase de las manos por defecto que heredan de la clase Bodypart

Además, se definen las clases para las manos derecha e izquierda por defecto las cuales heredan de la clase Bodypart [véase Código 3.6].

### Clases para BVH

Se utilizaron clases y métodos para intentar la correcta exportación del token a formato BVH [véase Código 3.7] para poder ser utilizado posteriormente en animaciones y renderizados.

```
class BvhNode(object):
    def __init__(
        self, name, offset, rotation_order,
        children=None, parent=None, is_root=False, is_end_site=False
    ):
        if not is_end_site and \
                rotation_order not in ['xyz', 'xzy', 'yxz', 'yzx', 'zxy', 'zyx']:
            raise ValueError(f'Rotation order invalid.')
        self.name = name
        self.offset = offset
        self.rotation_order = rotation_order
        self.children = children
        self.parent = parent
        self.is_root = is_root
        self.is_end_site = is_end_site
class BvhHeader(object):
    def __init__(self, root, nodes):
        self.root = root
        self.nodes = nodes
```

Ejemplo de código 3.7: Clase referente a la jerarquía de BVH

Se investigó la vía descrita por Nguyen y col. [20] sin éxito alguno puesto que no se pudieron instalar las dependencias y bibliotecas necesarias.

### 3.2.3. Métodos desarrollados

#### Métodos para la normalización

Este conjunto de métodos son los necesarios para resolver las problemáticas asociadas con las manos faltantes y desplazadas, además de las diferencias estructurales de los señantes mediante el uso de escalado proporcional dado una medida fija de los hombros. Dichos métodos pueden encontrarse agrupados en el método *normalize*, de la clase implementada [véase Código 3.8].

```
class TokenLSC:
    def __init__(self, name: str, sign: List[Skeleton], with_normalize=True, with_crop=True) ->
    None:
            ...

    def normalize(self, rhand_threshold=2, lhand_threshold=2, previous_cut=2,
     next_cut=2, edge_interpolation_gap=5, hand_forearm_proportion=3/11, finger_proportion=1/0.68)
    :
        if self.isNormalized:
            pass
        self.isNormalize = True
        for i, sk in enumerate(self.sign):
            body, lhand, rhand = sk.bodyparts
            body.normalize()
            lhand.normalize()
            rhand.normalize()
            sk.normalize()
        rhand_mask = [np.linalg.norm(
            x.rhand.points) > rhand_threshold for x in self.sign]
        lhand_mask = [np.linalg.norm(
            x.lhand.points) > lhand_threshold for x in self.sign]
        def get_to_fix_intervals(mask):
            ...
        def fix_interval(intervals: list, body_part, mask: list):
            ...
        rhand_intervals = get_to_fix_intervals(rhand_mask)
        lhand_intervals = get_to_fix_intervals(lhand_mask)
        fix_interval(rhand_intervals, TypeOfBodyPart.RHAND, rhand_mask)
        fix_interval(lhand_intervals, TypeOfBodyPart.LHAND, lhand_mask)
        angle = None
        for sk in self.sign:
            angle = self._scale_normalize(sk, angle=angle)
    ...
```

Ejemplo de código 3.8: Método normalize de la clase TokenLSC

Una vez normalizado un token no se vuelve a normalizar, a menos, que el usuario decida ejecutarlo por su cuenta. Cada vez que un token es creado, es normalizado por defecto, dejando a elección de quien lo use activar o no los argumentos pertinentes durante la construcción.

#### Método para el recorte de movimiento

El método para el recorte de movimiento [véase Código 3.9] por inactividad utiliza la integración numérica para encontrar la cantidad de movimiento para un intervalo dado. Es posible recortar para mantener un determinado porcentaje del movimiento total original de la seña a través del argumento movement_percentage (por defecto es un 80 % del movimiento total el que se mantiene). Además, puede escogerse que a qué tipo de parte del cuerpo se le quiere hacer énfasis, lográndose así un recorte más efectivo a las necesidades del usuario. Los argumentos *difference_mapping* y

*to_maximize* son las funciones *lambda* encargadas de determinar como se cuantifica el movimiento y en base a que criterio se maximiza. Cómo último argumento se deja a elección del usuario escoger entre la integración por el método de Simpson o por el método de Trapezoides.

```
def crop_movement(self,
                  movement_percentage=.8,
                  focus_on: TypeOfBodyPart = None,
                  difference_mapping=lambda x: np.linalg.norm(x) ** 2,
                  to_maximize=lambda value, i, j: value / abs(j-i),
                  integration_method="simpson"):
        """
        integration_method: trapezoid or simpson
        """
        if self.isCropped:
            pass
        self.isCropped = True
        if focus_on is None:
            general = [sk.all_points for sk in self.sign]
            general = [general[i] - general[i+1]
                       for i in range(len(general) - 1)]
            to_focus = general
        elif focus_on == TypeOfBodyPart.BODY:
            rhands = [sk.rhand for sk in self.sign]
            rhands = [rhands[i].points -
                      rhands[i+1].points for i in range(len(rhands) - 1)]
            to_focus = rhands
        elif focus_on == TypeOfBodyPart.LHAND:
            lhands = [sk.lhand for sk in self.sign]
            lhands = [lhands[i].points -
                      lhands[i+1].points for i in range(len(lhands) - 1)]
            to_focus = lhands
        elif focus_on == TypeOfBodyPart.RHAND:
            body = [sk.body for sk in self.sign]
            body = [body[i].points -
                    body[i+1].points for i in range(len(body) - 1)]
            to_focus = body
        else:
            raise ValueError("Invalid focus_on:", focus_on)
        def integrate(samples, a, b):
            ...
        # Integrate the squared norm
        integrate_with = [difference_mapping(x) for x in to_focus]
        integration_values = {}
        maximization_values = {}
        total_movement = 0
        for i in range(len(integrate_with)-1):
            for j in range(i+1, len(integrate_with)):
                value = integrate(integrate_with, i, j)
                if j == i + 1:
                    total_movement += value
                integration_values[i, j] = value
                maximization_values[i, j] = to_maximize(value, i, j)
        i_max = 0
        j_max = 0
        value_max = 0
        # Can add some restrictions if necessary
        for i in range(len(integrate_with)-1):
            for j in range(i+1, len(integrate_with)):
                int_val = integration_values[i, j]
                max_val = maximization_values[i, j]
                current_movement_percentage = int_val / total_movement
                if max_val > value_max and current_movement_percentage >= movement_percentage:
                    value_max = max_val
                    i_max = i
                    j_max = j
        self.sign = self.sign[i_max:j_max+1]
        return i_max, j_max
```

Ejemplo de código 3.9: Método crop_movement de la clase TokenLSC

Una vez escogidos los puntos para enfocar el recorte, se computan los valores del rango de integración al igual que los valores de la función para ser maximizada. Se utiliza el método de fuerza bruta para resolver el problema de optimización presentado, puesto que se cuenta con un dominio finito de frames que son, en lo humanamente

razonable, pocos para utilizar mejoras algorítmicas.

### Métodos para la unión de tokens

Para la concatenación de tokens [véase Código 3.10], se redefine el operador suma de la clase *TokenLSC*, lográndose así una mejor inter-operabilidad entre los tokens de lengua de señas. Una vez 2 tokens son adicionados, el método join_tokens es llamado con los mejores valores por defecto, escogidos de forma perceptual por el autor. El método join_tokens recibe los tokens a ser unidos, así como la cantidad de esqueletos a ser escogidos del primer token y del segundo, con prev_cut y next_cut respectivamente. La cantidad de esqueletos a ser creada también es uno de los parámetros a ser escogidos por el usuario con la variable amount_of_skeletons para garantizar una mayor adecuación a las diversas necesidades futuras que se puedan presentar. Como último argumento se puede escoger el método de interpolación a ser utilizado, por defecto este es lineal dado que es necesario para solventar varias de las problemáticas explicadas en el capítulo anterior.

```
@staticmethod
    def join_tokens(token1, token2, prev_cut=2, next_cut=2, amount_of_skeletons=None, kind='linear
    '):
        sign1, sign2 = token1.sign, token2.sign
        new_name = f"{token1.name}-{token2.name}"
        last_sk = sign1[-1].all_points
        first_sk = sign2[0].all_points
        if amount_of_skeletons is None:
            difference = np.linalg.norm(last_sk - first_sk)
            f = interp1d([0, 0.5, 4.7, 9], [0, 8, 14, 18],
                         assume_sorted=True, kind="linear", fill_value="extrapolate")
            amount_of_skeletons = int(np.ceil(f(difference)))
        elif amount_of_skeletons < 0:
            raise ValueError("amount_of_skeleton is negative")
        list_skeleton = TokenLSC.interpolate(sign1, sign2,
                                             prev_cut=prev_cut,
                                             next_cut=next_cut,
                                             amount_of_skeletons=amount_of_skeletons,
                                             kind=kind)
        new_sign = sign1+list_skeleton+sign2
        return TokenLSC(new_name, new_sign, with_normalize=False, with_crop=False)
    def __add__(self, object):
        '''
        Interpolate here with two LSC tokens
        '''
        if not isinstance(object, TokenLSC):
            raise ValueError(
                f"Invalid type {type(object)} in addition with TokenLSC")
        return self.join_tokens(self, object)
```

Ejemplo de código 3.10: Método utilizados para la unión de tokens TokenLSC

Se utiliza en caso de ser *None* el argumento para la cantidad de esqueletos, este es hallado mediante la interpolación de valores perceptualmente buenos para el autor. Luego se hace un llamado al método *interpolate* (de la clase *TokenLSC* también) el cual describiremos a continuación [véase Código 3.11].

```
@staticmethod
    def interpolate(sign1: List[Skeleton], sign2: List[Skeleton], prev_cut=2, next_cut=2,
    amount_of_skeletons=4, kind='cubic'):
        origin_skeletons = sign1[-prev_cut:]
        end_skeletons = sign2[:next_cut]
        l = prev_cut + amount_of_skeletons+ next_cut
        dims = origin_skeletons[0].all_points.shape
        newbodies = np.zeros((amount_of_skeletons, dims[0], dims[1]))
```

```
8           fullbodies = [sk.all_points for sk in origin_skeletons+end_skeletons]
9           for i, points in enumerate(zip(*fullbodies)):
10              current_kind = kind
11              x = [point[0] for point in points]
12              y = [point[1] for point in points]
13              z = [point[2] for point in points]
14              t = [j for j in range(prev_cut)] +
15                      [prev_cut + amount_of_skeletons + j for j in range(next_cut)]
16              if len(t) < 4:
17                  current_kind = "linear"
18              fx = interp1d(t, x, kind=current_kind)
19              fy = interp1d(t, y, kind=current_kind)
20              fz = interp1d(t, z, kind=current_kind)
21              newx = [fx(p)
22                      for p in range(prev_cut, prev_cut + amount_of_skeletons)]
23              newy = [fy(p)
24                      for p in range(prev_cut, prev_cut + amount_of_skeletons)]
25              newz = [fz(p)
26                      for p in range(prev_cut, prev_cut + amount_of_skeletons)]
27              for j in range(amount_of_skeletons):
28                  X, Y, Z = (newx[j], newy[j], newz[j])
29                  newbodies[j, i, 0] = X
30                  newbodies[j, i, 1] = Y
31                  newbodies[j, i, 2] = Z
32          skeleton_list = []
33          body_dim = origin_skeletons[0].body.shape[0]
34          lhand_dim = origin_skeletons[0].lhand.shape[0]
35          rhand_dim = origin_skeletons[0].rhand.shape[0]
36          for i in range(amount_of_skeletons):
37              body = newbodies[i, :body_dim, :]
38              lhand = newbodies[i, body_dim:body_dim+lhand_dim, :]
39              rhand = newbodies[i, body_dim + lhand_dim:body_dim+lhand_dim+rhand_dim, :]
40              body_part = Bodypart("body", TypeOfBodyPart.BODY, body)
41              left_part = Bodypart("left", TypeOfBodyPart.LHAND, lhand)
42              right_part = Bodypart("right", TypeOfBodyPart.RHAND, rhand)
43              skeleton = Skeleton(body_part, left_part, right_part)
44              skeleton_list.append(skeleton)
45          return skeleton_list
```

Ejemplo de código 3.11: Método estático interpolate de la clase TokenLSC

Este método escoge una serie de esqueletos del final de la primera lista *sign1* y del principio de la segunda lista *sign2* utilizando los argumentos *prev_cut* y *next_cut* respectivamente. Luego se obtienen las dimensiones de los esqueletos para poder crear un arreglo de ceros correspondiente a las dimensiones de la cantidad de esqueletos a ser creados y las dimensiones anteriormente extraídas de los esqueletos. Posteriormente se desenrollan y envuelven todos los puntos de los esqueletos escogidos para interpolar, haciendo que en la variable *points* queden todos los valores referentes al i-ésimo punto de cada esqueleto. La interpolación se realiza por cada componente de cada punto como se explica en el capítulo anterior.

### Método de animación con matplotlib

Poder revisar como se unen o efectúan los distintos tokens, mediante los métodos descritos en subsecciones anteriores, es de vital importancia para determinar el correcto funcionamiento de los mismos. Es por dicha razón que utilizando la biblioteca de matplotlib, se consigue de manera efectiva animar los distintos tokens lo cual mejora la experiencia durante la evaluación durante los experimentos.

El método *animate* [véase Código 3.12] recibe la cantidad de fps (frames por segundo) a utilizar como medida de velocidad de la animación, lo cual es usual en este campo. Las dimensiones de la figura tanto como el ángulo de elevación y de

azimuth se encuentran en los valores óptimos por defecto para la mejor experiencia visual del usuario, aunque este puede utilizar otros valores para dichos argumentos modificando los parámetros *figsize*, *elev*, *angle*. De igual manera, es posible visualizar el índice numérico de cada punto, así como las uniones entre los puntos utilizando los argumentos booleanos *text* y *joints*

```
def animate(self, fps=30, figsize=(7.0, 3.5), elev=90, angle=1, dims=None, text=False, joints=True
    ):
        if not dims:
            dims = self.dims
        skeletons_points = np.array([sk.all_points for sk in self.sign])
        rc('animation', html='jshtml')
        fig_size_x, fig_size_y = figsize
        plt.rcParams["figure.figsize"] = [fig_size_x, fig_size_y]
        plt.rcParams['figure.autolayout'] = True
        fig = plt.figure()
        ax = fig.add_subplot(projection='3d')
        colors = ['b', 'y', 'r', 'k']
        total_frames = skeletons_points.shape[0]
        time = total_frames / fps
        def update(i):
            ax.clear()
            D = self.__transform_data(skeletons_points, i)
            s = 0
            e = 0
            for j, dim in enumerate(dims):
                dim = dim[0]
                e += dim
                x = D[s:e, 0]
                y = D[s:e, 1]
                z = D[s:e, 2]
                ax.plot(D[s:e, 0], D[s:e, 1], D[s:e, 2], f'{colors[j]}.')
                if text:
                    for p in range(s, e):
                        ax.text3D(D[p, 0], D[p, 1], D[p, 2], str(p))
                if joints:
                    for h, k in self.fullbodyjoints[j]:
                        ax.plot([D[s+h, 0], D[s+k, 0]], [D[s+h, 1], D[s+k, 1]],
                                [D[s+h, 2], D[s+k, 2]], f'{colors[j]}')
                s += dim
                ax.set_axis_off()
                if elev and angle:
                    ax.view_init(elev, angle)
            return ax
        return FuncAnimation(fig, update, frames=total_frames, repeat=True, interval=fps)
```

Ejemplo de código 3.12: Método animate de la clase TokenLSC

En este método de animación se seleccionan las dimensiones de las distintas partes así como los puntos por cada frame para poderse utilizar dentro del método update definido de manera interna en el scope de dicho método. Internamente, update realiza transformaciones de rotación a los datos y separa por dimensiones para un mejor acabado gráfico en la proyección final.

### Métodos para importar los tokens

Siempre se puede crear un *TokenLSC* de manera manual, pero se vuelve muy tedioso y se puede incurrir en errores a la hora de manipular los datos. Es por esto que se agrega el método para importar un token desde un directorio que le sea suministrado [véase Código 3.13].

```
    @staticmethod
    def import_from_path(path: str, name: str = '_new-sign_', verbose=False, with_normalize=True,
    with_crop=True) -> 'TokenLSC':
        if '.json' in path:
```

```
4               return TokenLSC.import_token_lsc_from_json(str(path), with_normalize=True, with_crop=
    True)
5           else:
6               return TokenLSC.import_token_lsc_from_video(path, name, with_normalize=True, with_crop
    =True)
7       @staticmethod
8       def import_from_video(path: str, name: str = '_new-sign_', verbose=False, with_normalize=True,
      with_crop=True) -> 'TokenLSC':
9           images = TokenLSC._get_images_from_video(path)
10          with mp.solutions.holistic.Holistic() as holistic:
11              results = []
12              for image in images:
13                  result = holistic.process(image)
14                  results.append(result)
15              sign = TokenLSC._process_results(results)
16              return TokenLSC(name, sign, with_normalize=with_normalize, with_crop=with_crop)
17      @staticmethod
18      def import_from_json(path: str, name: str = '', verbose=False, with_normalize=True, with_crop=
    True) -> 'TokenLSC':
19          json_path = Path(path)
20          json_list = json.loads(json_path.read_text())
21          if len(json_list) > 1:
22              print(
23                  f"More than one representation found at {path}. Selecting first representation.")
24          json_repr = json_list[0]
25          np_repr = np.array(json_repr)
26          if verbose:
27              print("Json shape:", np_repr.shape)
28          np_repr = np.reshape(
29              np_repr, (np_repr.shape[0], np_repr.shape[1] // 3, 3))
30          if verbose:
31              print("Frames shape", np_repr.shape)
32          body = np_repr[:, :25, :]
33          left = np_repr[:, 25:46, :]
34          right = np_repr[:, 46:, :]
35          if verbose:
36              print("Body shape", body.shape)
37              print("Left shape", left.shape)
38              print("Right shape", right.shape)
39          skeletons = []
40          for frame_body, frame_left, frame_right in zip(body, left, right):
41              body_part = Bodypart("body", TypeOfBodyPart.BODY, frame_body)
42              left_part = Bodypart("left", TypeOfBodyPart.LHAND, frame_left)
43              right_part = Bodypart("right", TypeOfBodyPart.RHAND, frame_right)
44              skeleton = Skeleton(body_part, left_part, right_part)
45              skeletons.append(skeleton)
46          if name == '':
47              name = json_path.name.split(".")[0]
48          return TokenLSC(name, skeletons, with_normalize=with_normalize, with_crop=with_crop)
```

Ejemplo de código 3.13: Métodos estático para importar tokens de la clase TokenLSC

El método genérico import_from_path analiza la extensión del directorio que recibe como argumento para decidir cual de los métodos secundarios utilizar para el propósito requerido. El método de import_from_video utiliza los métodos explicados y brindados por Gutiérrez-González [11]. Las ventajas de importar de vídeo es que cualquiera puede hacer uso de la herramienta para, combinándolo con los métodos de exportación, ampliar y extender la base de datos actual de señas cubanas.

### Métodos para exportar los tokens

Como primer método de exportación contamos con export_to_json, el cual recibe como argumento solamente la ruta al archivo donde se desea guardar el token. Otro de los métodos con que se cuenta es el método de exportar a un archivo BVH de captura de movimiento, el cual se logra exportar pero con fallas dado el desconocimiento del autor acerca de los ángulos de rotación requeridos [véase Código 3.14].

```
    def export_to_json(self,path:str):
        sign_array=np.array([sk.all_points for sk in self.sign])
        with open(path, 'w', encoding='utf-8') as json_file:
            json.dump(sign_array,json_file,sort_keys=True, ensure_ascii=False)
            json_file.close()
     def export_to_bvh(self, header=None, output_file=None):
        if not header:
            header = self.get_bvh_header(self.poses3d)
        channels = []
        for frame, pose in enumerate(self.poses3d):
            channels.append(self.pose2euler(pose, header))
        if output_file:
            write_bvh(output_file, header, channels)
        return channels, header
```

Ejemplo de código 3.14: Métodos estático para exportar tokens de la clase TokenLSC

## 3.3. Experimentación

Al desplegar e implementar los métodos y clases, de la sección anterior, se realizaron una serie de experimentos comparativos para evaluar el desempeño de los mismos.

### 3.3.1. Experimento 1

Para el primer experimento se decidió verificar si el umbral escogido para selección de los frames con manos faltantes resultó efectivo, así como evaluar la interpolación realizada en dichos intervalos, comparando los intervalos de los tokens antes y después de normalizarlos.

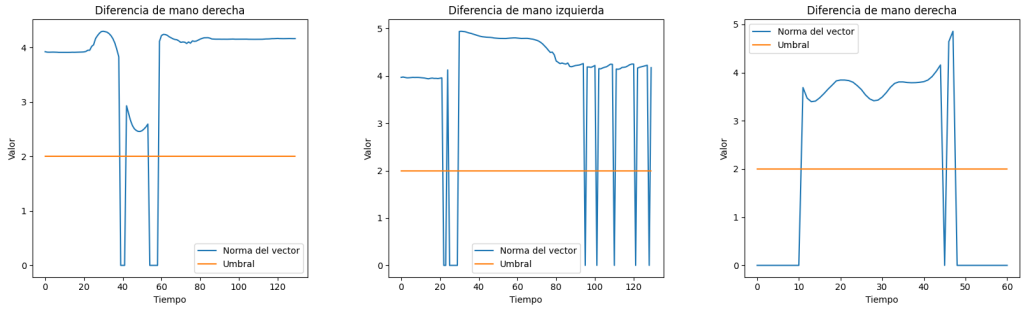
En la Fig 3.1 se puede apreciar la distancia de cada mano con respecto al cero del espacio siendo los picos, los momentos en que desaparecen. Se tomó el valor de distancia 2 con respecto al (0,0,0) como umbral para detectar cuando una mano ha desaparecido para lograr a tiempo interpolarla.

Mientras que en la Fig 3.4 se puede analizar la comparativa con respecto al token sin normalizar.

Este experimento brindó buenos resultados al realizarse con el método de interpolación lineal, mientras que el cúbico, a pesar de aportar mayor suavidad, en intervalos cortos generaba perturbaciones y alargamiento de extremidades.

### 3.3.2. Experimento 2

Para este segundo experimento se evalúa la efectividad del recorte por inactividad en un mismo token mediante la verificación de los extremos del mismo y gráficas que cuantifican el movimiento general y por las manos. En cuanto a la Fig 3.2 se puede verificar como el método de recorte por inactividad está detectando los intervalos de mayor cantidad de movimiento. Siendo estos mayoritariamente los del medio de la grabación, ya que en los extremos es que suele estar en reposo el señante. En la Fig




(a) Umbral en la gráfica de la mano derecha del token "aborto"

(b) Umbral en la gráfica de la mano izquierda del token "aborto"

(c) Umbral en la gráfica de la mano derecha del token "amar"

Figura 3.1: Umbral de detección de las manos faltantes en los tokens "aborto" y "amar" Fig 2.4.

3.3 se podrá apreciar mejor el recorte puesto que se ve el principio y el final actual de cada token de muestra.

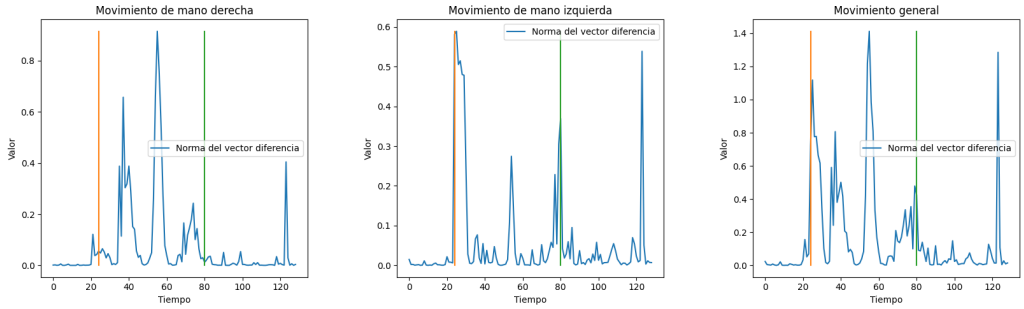
Este experimento demostró excelentes resultados, jugando un papel especial el hecho de escoger de manera efectiva las funciones que cuantifican el movimiento, la función a maximizar y el porcentaje de movimiento total a mantener. Valores distintos a los escogidos por defecto pueden devenir en exceso de recorte o ninguno.

### 3.3.3. Experimento 3

Para este tercer experimento se decide valorar el correcto posicionamiento y ajuste de la mano falsa, así como la interpolación en la suma entre dos tokens. Para ello se eligen al token al correspondiente a la seña de aborto y de la seña amar, puesto que este último carece de manos en ambos extremos en su token original sin normalizar ni realizarle recorte por inactividad.

Se evaluó de positivo el uso de las manos falsas y la interpolación interna para solucionar el problema de las manos faltantes. Como se ve en la Fig 3.4.

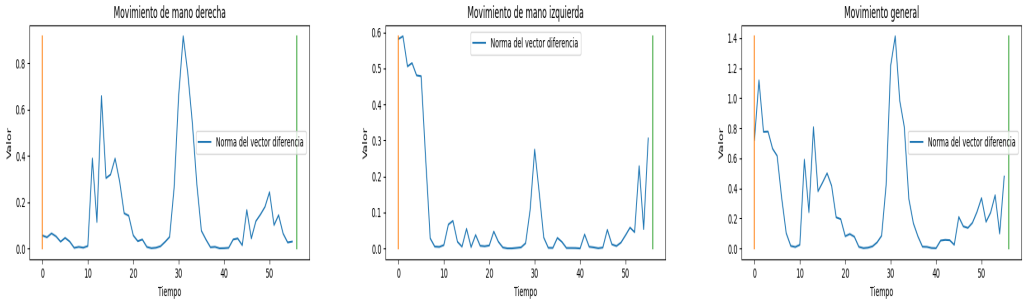
Además de que uno de los mejores resultados obtenidos durante la experimentación fue la secuencia de interpolación entre 2 señas con finales tan marcados. Véase la Fig 3.5 para una mejor apreciación. Este experimento se comportó de manera correcta, a pesar de que la mano no corresponde con la original o presenta una disposición diferente a la que originalmente perteneciera a la imagen. Dado que no existe una referencia en la imagen original se puede evaluar como positivo el desempeño demostrado.




(a) Umbrales para recortar del movimiento de la mano derecha del token

(b) Umbrales para recortar del movimiento de la mano izquierda del token

(c) Umbrales para recortar del movimiento de la mano derecha del token




(d) Recorte de movimiento de la mano derecha del token

(e) Recorte de movimiento de la mano izquierda del token

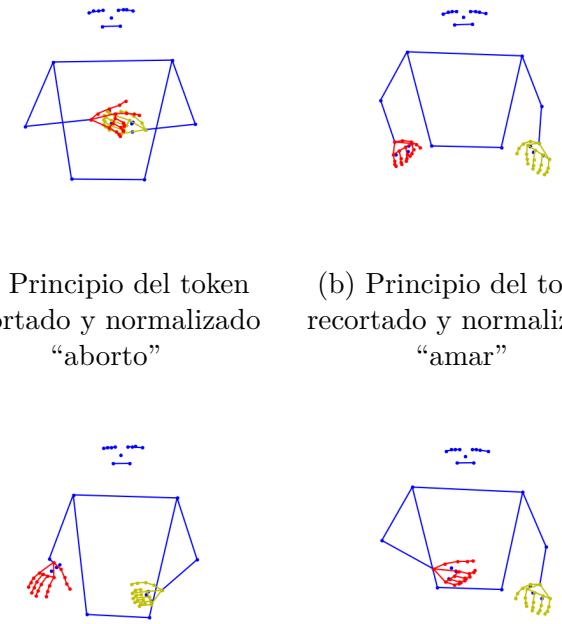
(f) Recorte de movimiento general del token

Figura 3.2: Detección de umbrales en las gráficas de movimiento por manos y general del token “aborto” normalizado. Véase Fig 2.6

### 3.3.4. Experimento 4

Se decidió probar si la solución presentada para exportar a formato BVH era al menos una aproximación inicial. Para ello se evaluó de manera perceptual la secuencia generada por el archivo BVH importándolo a la aplicación del motor gráfico Blender

Los resultados obtenidos con la generación de un archivo BVH no fueron aceptables. Por desconocimiento de algunos de los ángulos de rotación y como definirlos correctamente, solo se pudo armar el esqueleto pero su movilidad deja mucho que desear. Muchas de las uniones no pueden moverse en algunas direcciones a pesar de que hace el intento por replicar la seña de la cual fue hecho [véase Fig 3.6].



(a) Principio del token recortado y normalizado "aborto"

(b) Principio del token recortado y normalizado "amar"

(c) Final del token recortado y normalizado "aborto"

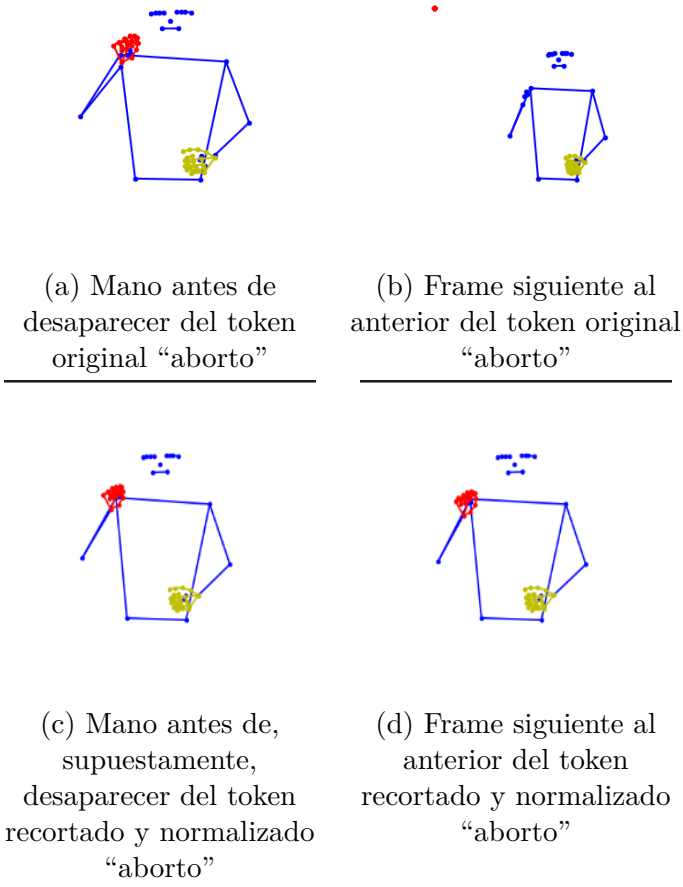
(d) Final del token recortado y normalizado "amar"

Figura 3.3: Resultados obtenidos en cuanto a diferencias mostradas en la Fig 2.5.

## 3.4. Discusión

A partir de los experimentos realizados se pudo concluir que la propuesta brindó buenos resultados fuera del ámbito de la animación. Muchos factores como bibliotecas de python como bpy (Módulo de Blender para Python) que no llegaron a instalarse nunca podrían haber acelerado o incluso hacer muy ligero y transparente el proceso de llevar el *TokenLSC* a BVH e incluso a animación 3D. Pese a los buenos resultados en todos los demás aspectos, aún quedan estructuras y proporciones en las que trabajar, dado que la forma del cuerpo, así como su proporción, dependen mucho de factores como:

- ángulo de inclinación de la grabación
- estructura corporal del señante
- fallos en el modelo de reconocimiento



(a) Mano antes de desaparecer del token original “aborto”

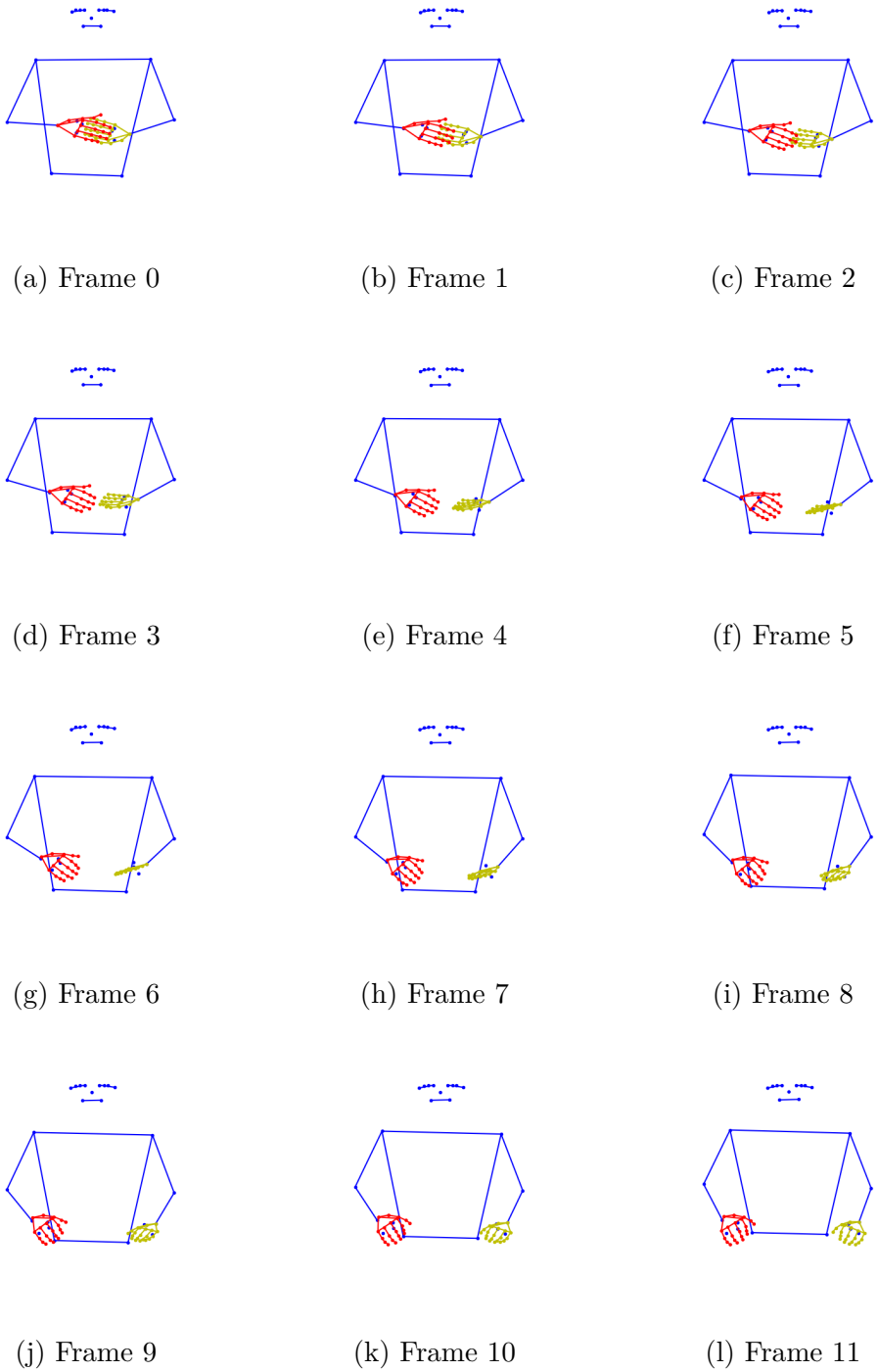
(b) Frame siguiente al anterior del token original “aborto”

(c) Mano antes de, supuestamente, desaparecer del token recortado y normalizado “aborto”

(d) Frame siguiente al anterior del token recortado y normalizado “aborto”

Figura 3.4: Resultados obtenidos en cuanto a diferencias mostradas en la Fig 2.4.

La solución implementada brinda generalización y extensibilidad para su uso en otros fines tanto para generar nuevo contenido audiovisual como para registrar nuevas señas al conjunto de datos.



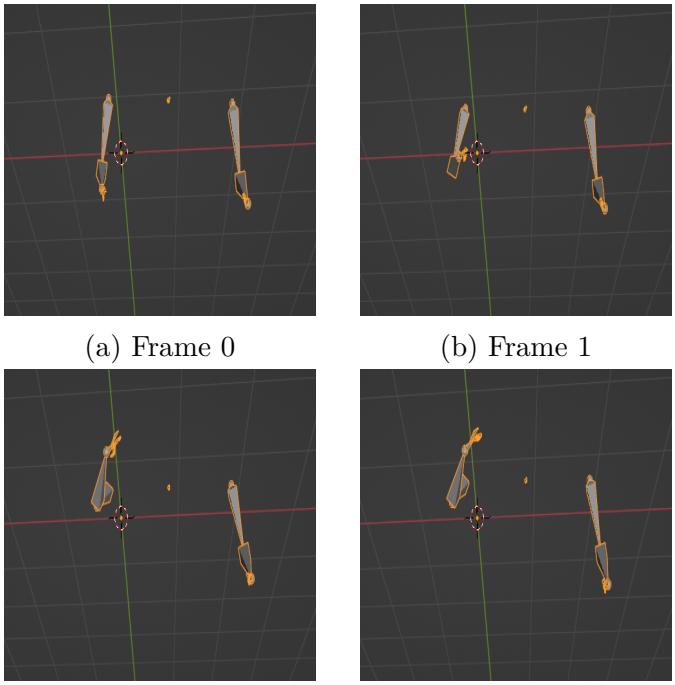
(a) Frame 0 (b) Frame 1 (c) Frame 2

(d) Frame 3 (e) Frame 4 (f) Frame 5

(g) Frame 6 (h) Frame 7 (i) Frame 8

(j) Frame 9 (k) Frame 10 (l) Frame 11

Figura 3.5: Secuencia Interpolada entre el token “aborto” y “amar”. Véase Fig 3.3 para notar los extremos a unirse.



(a) Frame 0 (b) Frame 1

(c) Frame 2 (d) Frame 3

Figura 3.6: Resultado del archivo BVH del token recortado y normalizado “amar”

# Conclusiones

La generación de avatares para la Lengua de Señas Cubana representa el principal objetivo del trabajo. Durante la investigación se generan nuevos métodos, estructuras y corpus que posibilitan una mejor comprensión de esta temática. Se introduce la problemática presente en el contexto social correspondiente y se profundiza en el estado del arte de diversos campos como la escritura de señas, haciendo énfasis en su estructura y composición, reconocimiento de señas, tecnología de avatares, modelos generativos y interpolación de movimiento, detectándose un crecimiento de la aceptación de los modelos que utilizan inteligencia artificial y aprendizaje de máquina. Se analizó el uso de los modelos generativos como una alternativa para la generación automática de avatares hiper-realistas de los cuáles se deja como recomendaciones para futuras investigaciones. Se indagó acerca de los únicos trabajos encontrados que tratan la Lengua de Señas Cubana. Se propuso una estructura de métodos y clases para la correcta interpolación y adición de una secuencia de señas, lográndose así una correcta generación de movimientos suaves y definidos del esqueleto del avatar. Esta propuesta para la generación de avatares para la Lengua de Señas Cubana, utilizando el corpus generado por anteriores investigaciones, fue concretada en un prototipo de clases que utilizan interpolación e integración numérica para resolver los problemas presentados. Se obtuvo un avatar que consigue ejecutar de manera armónica los movimientos asociados a cada token de seña incluyendo cualquier combinación de los mismos sin importar cuán extensa sea.

# Recomendaciones

Con este trabajo se ayuda a una nueva rama de investigación y desarrollo referente a la lengua de señas cubana, ya que forma un pilar esencial en cualquier sistema de traducción de lenguas de señas. Al ser este un trabajo novedoso en el campo de la generación de avatares señantes, se deja como pauta inicial para futuros trabajos y una guía del proceder ante los limitados datos disponibles.

Se sugiere:

- Estudiar la generación de archivos de captura de movimiento como BVH y el cálculo de los vectores de rotación de brazos, manos y dedos.
- Lograr transformar un vídeo de un avatar animado en una persona real haciendo uso de los modelos generativos, en especial el uso de los métodos de imagen a imagen (img2img)
- Desarrollar alternativas con el uso de otros modelos generativos de imágenes para la humanización del avatar presentado.

Se recomienda, a los investigadores de Cuba, lingüistas y especialistas de la lengua de señas y a toda la comunidad:

- Crear conciencia sobre la importancia y necesidad de este tipo de estudios y herramientas para el correcto desarrollo de la comunidad sordo-muda cubana
- Trabajar unánimemente en pos de lograr un corpus correctamente anotado y tanto extenso, como generalizado, para así obtener una herramienta de utilidad para que, en manos de los hipoacúsicos, sea la llave para romper las barreras comunicativas que les impiden desenvolverse y desarrollarse en nuestro país actualmente.

# Bibliografía


[1] 2021. URL: https://www.visicast.cmp.uea.ac.uk/Visicast_index.html (vid. pág. 7).

[2] 2021. URL: https://www.visicast.cmp.uea.ac.uk/eSIGN/index.html (vid. pág. 7).

[3] Yanet Almarales-Wilson. «Aplicación Androide para favorecer la comunicación en Lengua de Señas Cubana». En: *Evento "Pedagogía 2021"* (2021) (vid. págs. 13, 14).

[4] Deepali Aneja, Daniel McDuff y Shital Shah. «A high-fidelity open embodied avatar with lip syncing and expression capabilities». En: *2019 International Conference on Multimodal Interaction*. 2019, págs. 69-73 (vid. pág. 7).

[5] Necati Cihan Camgoz y col. «Sign Language Transformers: Joint End-to-end Sign Language Recognition and Translation». En: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020 (vid. pág. 6).

[6] CERN. 2014. URL: http://cdsweb.cern.ch/journal/CERNBulletin/2006/31/News%20Articles/974627?ln=en (vid. pág. 27).

[7] Michael Everson, Stephen Slevinski y Valerie Sutton. «Proposal for encoding Sutton SignWriting in the UCS». En: abr. de 2013. URL: https://www.unicode.org/L2/L2012/12321-n4342-signwriting.pdf (vid. pág. 4).

[8] Facebook. 2018. URL: https://developers.facebook.com/blog/post/301 (vid. pág. 28).

[9] Ian Goodfellow y col. «Generative adversarial networks». En: *Communications of the ACM* 63.11 (2020), págs. 139-144 (vid. pág. 8).

[10] Google. 2020. URL: https://mediapipe.dev/ (vid. págs. 6, 29).

[11] Leynier Gutiérrez-González. «Plataforma y modelos para la interpretación automática de la Lengua de Señas Cubana». En: (nov. de 2021). URL: https://github.com/leynier/computer-science-bachelors-thesis (vid. págs. 2, 14, 18, 30, 40).

[12] Matt Huenerfauth. «Generating American Sign Language classifier predicates for English-to-ASL machine translation». Tesis doct. Citeseer, 2006 (vid. pág. 8).

[13] Eui Jun Hwang, Jung-Ho Kim y Jong C. Park. «Non-Autoregressive Sign Language Production with Gaussian Space». En: *The 32nd British Machine Vision Conference (BMVC 21)*. British Machine Vision Conference (BMVC). 2021 (vid. pág. 7).

[14] Hernisa Kacorri y col. «Regression analysis of demographic and technology-experience factors influencing acceptance of sign language animation». En: *ACM Transactions on Accessible Computing (TACCESS)* 10.1 (2017), págs. 1-33 (vid. pág. 8).

[15] Sin-Hwa Kang y Jonathan Gratch. «The effect of avatar realism of virtual humans on self-disclosure in anonymous social interactions». En: *CHI'10 Extended Abstracts on Human Factors in Computing Systems*. 2010, págs. 3781-3786 (vid. pág. 7).

[16] Karolina Kozik. Sep. de 2019. URL: `https://www.hrw.org/news/2019/09/23/without-sign-language-deaf-people-are-not-equal#` (vid. pág. 1).

[17] J. Langr y V. Bok. *GANs in Action: Deep learning with Generative Adversarial Networks*. Manning, 2019. ISBN: 9781617295560. URL: `https://books.google.com/books?id=HojvugEACAAJ` (vid. págs. 8, 9).

[18] Marc Erich Latoschik y col. «The effect of avatar realism in immersive social virtual realities». En: *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology*. 2017, págs. 1-10 (vid. pág. 7).

[19] Sushmita Mitra y Tinku Acharya. «Gesture recognition: A survey». En: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37.3 (2007), págs. 311-324 (vid. pág. 6).

[20] Lan Thao Nguyen y col. «Automatic generation of a 3D sign language avatar on AR glasses given 2D videos of human signers». En: *MTSUMMIT*. 2021 (vid. págs. 7, 8, 34).

[21] Numpy. 2012. URL: `https://numpy.org/about/` (vid. pág. 28).

[22] PSF. 2020. URL: `https://wiki.python.org/moin/OrganizationsUsingPython` (vid. pág. 27).

[23] PSF. 2021. URL: `https://www.python.org/about/quotes/` (vid. pág. 27).

[24] PSF. 2021. URL: `https://www.python.org/about/success/usa/` (vid. pág. 27).

[25] PSF. 2022. URL: https://www.python.org/doc/essays/blurb/ (vid. pág. 27).

[26] Robin Rombach y col. *High-Resolution Image Synthesis with Latent Diffusion Models*. 2021. arXiv: 2112.10752 [cs.CV] (vid. pág. 10).

[27] Charles F. Rose, Michael F. Cohen y Bobby Bodenheimer. «Verbs and Adverbs: Multidimensional Motion Interpolation». En: *IEEE Computer Graphics and Applications* 18 (1998), págs. 32-40 (vid. pág. 13).

[28] Wendy Sandler. «Dedicated gestures and the emergence of sign language». En: *Gesture* 12.3 (2012), págs. 265-307 (vid. pág. 6).

[29] Jason M Saragih, Simon Lucey y Jeffrey F Cohn. «Real-time avatar animation from a single image». En: *2011 IEEE International Conference on Automatic Face & Gesture Recognition (FG)*. IEEE. 2011, págs. 117-124 (vid. pág. 7).

[30] Stephanie Stoll y col. «Sign Language Production using Neural Machine Translation and Generative Adversarial Networks». En: *British Machine Vision Conference (BMVC)*. 2018 (vid. pág. 7).

[31] Valerie Sutton. *Lessons in SignWriting*. http://www.signwriting.org, 2002 (vid. pág. 5).

[32] Stuart Thiessen. «A Grammar of SignWriting». M.A thesis. 2011 (vid. pág. 4).

[33] Lei Wang y col. «A state-of-the-art review on image synthesis with generative adversarial networks». En: *IEEE Access* 8 (2020), págs. 63514-63537 (vid. pág. 10).