Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

IE0499 – Proyecto Eléctrico

Desarrollo de un algoritmo de modelaje de plantillas faciales y etiquetado automático de marcadores de un sistema de captura de movimiento

por

Pablo Angulo Carvajal

Ciudad Universitaria Rodrigo Facio

Julio de 2020

Desarrollo de un algoritmo de modelaje de plantillas faciales y etiquetado automático de marcadores de un sistema de captura de movimiento

por

Pablo Angulo Carvajal

B50445

IE0499 – Proyecto Eléctrico

Aprobado por

Dr. rer. nat. Francisco Siles Canales
Profesor guía

M. Sc. Denise Dajles Kellerman
Profesor lector

Ing. Juan José Delgado Quesada
Profesor lector

Julio de 2020

Resumen

Desarrollo de un algoritmo de modelaje de plantillas faciales y etiquetado automático de marcadores de un sistema de captura de movimiento

por

Pablo Angulo Carvajal

Universidad de Costa Rica
Escuela de Ingeniería Eléctrica
Profesor guía: Dr. rer. nat. Francisco Siles Canales
Julio de 2020

La captura de movimiento se ha vuelto una herramienta muy versátil en el mundo moderno, desde su uso tradicional en el mundo de la animación, siguiendo por su utilidad en la investigación las ciencias del movimiento humano, hasta permitir oportunidades nuevas como lo es la telepresencia o actuar como interfaz para controlar una máquina o computadora. Sin embargo, el procesamiento de los datos obtenidos por este método se vuelve engorroso al trabajar la etapa de etiquetación, usualmente a mano, de los marcadores y al lidiar con interferencia y otros fenómenos que ocurren en la captura.

Con el propósito de facilitar el procesamiento de datos de captura de movimiento facial (*facial MoCap*), se propone el desarrollo de un programa que aplique un algoritmo de asociación para la etiquetación automática de marcadores faciales, y permita la formación de nuevas plantillas además del uso de plantillas predeterminadas. Se utiliza el concepto de asociación de grafos bipartitos como base para un algoritmo de etiquetación y se crea una interfaz gráfica para facilitar el uso de este.

Se encuentra que el acercamiento de utilizar asociación de grafos bipartitos para la etiquetación automática de marcadores tiene gran potencial y es eficaz, sin embargo, la implementación presente tiene ciertas limitaciones que podrían solventarse con optimizaciones y/o el uso de grafos multipartitos para tomar en cuenta un marco temporal mayor y lidiar con situaciones que el algoritmo desarrollado no es capaz de enfrentar de manera efectiva.

***Palabras claves**: MoCap, etiquetación, plantillas faciales, modelado facial, grafos bipartitos.*

Acerca de IE0499 – Proyecto Eléctrico

El Proyecto Eléctrico es un curso semestral bajo la modalidad de trabajo individual supervisado, con el propósito de aplicar estrategias de diseño y análisis a un problema de temática abierta de la ingeniería eléctrica. Es un requisito de graduación para el grado de bachiller en Ingeniería Eléctrica de la Universidad de Costa Rica.

Abstract

Desarrollo de un algoritmo de modelaje de plantillas faciales y etiquetado automático de marcadores de un sistema de captura de movimiento

Original in Spanish. Translated as: "Development of an algorithm for facial template modelling and automatic labelling of markers in a motion capture system"

by

Pablo Angulo Carvajal

University of Costa Rica
Department of Electrical Engineering
Tutor: Dr. rer. nat. Francisco Siles Canales
July of 2020

Motion capture has become a very versatile tool in the modern world, from its traditional use in the world of animation, following its usefulness in research dealing with human motion, to allowing new opportunities such as telepresence or acting as an interface to control a machine or computer. However, processing of data obtained by this method becomes cumbersome as one gets to the labelling of the markers, usually done by hand, and dealing with interference and other phenomena that occur in the process of recording.

In order to facilitate the processing of facial motion capture data (*facial MoCap*), this project proposes to develop a program that applies an association algorithm for automatic labelling of facial markers, and allows the formation of new layouts in addition to the use of default templates. The concept of bipartite graph association is used as a basis for an automatic labelling algorithm and a graphical interface is created to facilitate the use of the algorithm.

Thehe approach of using bipartite graph association for automatic labelling of markers has great potential and is proven effective, however, the present implementation has certain limitations that could be solved with optimizations and/or the use of multipartite graphs to take into account a larger time frame and deal with situations that the developed algorithm is not able to deal with succesfully.

***Keywords**: MoCap, labelling, facial layouts, facial modelling, bipartite graphs.*

About IE0499 – Proyecto Eléctrico ("Electrical Project")

The "Electrical Project" is a course of supervised individual work of one semester, with the purpose of applying design and

analysis strategies to a problem in an open topic in electrical engineering. It is a requisite of graduation for the Bachelor of Science in Electrical Engineering, granted by the University of Costa Rica.

Agradecimientos

A mi familia, por su amor y apoyo en todo momento.
A Francisco Siles Canales, por presentarme oportunidades únicas y empujarme a tomarlas.
A Denise Dajles Kellerman y Juan José Delgado Quesada, por toda su guía y ayuda.
Al PRIS-Lab, por abrirme sus puertas y expandir mis horizontes.
A mis amigos, por siempre apoyarme, alegrar mi vida y estar a mi lado.

Índice general

Índice de figuras

Índice de tablas

Nomenclatura

MoCap Captura de Movimiento (*Motion Capture*)

BMI Interfaz Cuerpo-Máquina (*Body-Machine Interface*)

CSV Formato de archivo de valores separados por comas (*Comma Separated Values*)

BVH Formato de archivo para generar modelos tridimensionales de objetos (*BioVison Heirarchical Data*)

PRIS – Lab Laboratorio de Reconocimiento de Patrones y Sistemas Inteligentes, de la Escuela de Ingeniería Eléctrica de la Universidad de Costa Rica (*Pattern Recognition and Intelligent Systems Laboratory*)

Capítulo 1

Introducción

Las máquinas y las computadoras se han vuelto ubicuas en el mundo moderno, de manera que dependemos de ellas para desarrollar nuestro trabajo, nuestras vidas sociales, nuestro aprendizaje y nuestro entretenimiento como seres en sociedad. La forma en la que interactuamos y controlamos con estas máquinas se vuelve necesariamente universal para poder facilitar la operación de estas a distintas personas con capacidades y necesidades variadas, especialmente personas con discapacidades motoras.

Para alcanzar esta universalidad es necesario lograr la redundancia, es decir, tener disponibles varias maneras de realizar una misma tarea, mediante interfaces cuerpo-máquinas distintas a las prototípicas como lo son, por ejemplo, *mouse* y *teclados*. La forma usual de interactuar con estas máquinas y computadoras es mediante interfaces cuerpo-máquina o BMI (Body-Machine interfaces); si bien el *mouse* y el *teclado*, mencionados anteriormente, además de dispositivos como controles analógicos se encuentran en la categoría de BMI, este proyecto hace mención de BMI en referencia a interfaces alternativas a estas interfaces usuales. Las BMI usualmente toman ventaja de la redundancia y la flexibilidad del sistema de control motor del cuerpo; ya que se puede entrenar el cuerpo a re-asociar las distintas formas que se utilizan para controlar la motora para adaptarse a nuevas circunstancias, como lo es la perdida de control de ciertas mociones, extremidades. etc. [4]

El reconocimiento facial y de las expresiones posibles por las caras humanas ha sido un área de estudio por varios años; ya que las expresiones faciales se asocian a las emociones de los humanos y se pueden utilizar desde retroalimentación para el aprendizaje de AI (inteligencia artificial) hasta métodos de control de máquinas. El alcance de métodos basado en modelos e imágenes 2D de caras humanas es limitado ya que usualmente requieren de buena iluminación, y se restringe la posición a únicamente un área pequeña donde opere el sensor de captura de imagen [5]. Un acercamiento mediante modelos 3D como lo es el MoCap facial permite mucho mayor rango de movimiento de un sujeto y puede llegar a capturar mejor algunas de las complejidades de los cambios de expresiones faciales.

Este proyecto propone aportar a la creación de nuevas interfaces cuerpo-máquina que tomen provecho de la identificación de expresiones faciales por marcadores de MoCap mediante la creación de plantillas faciales con marcadores etiquetados de tomas existentes; estas plantillas faciales y la etiquetación automática de los marcadores acelerarían el procesamiento de las tomas faciales para su identificación, lo cual en la actualidad presenta ser un proceso tedioso con la capacidad de ser asignado a una computadora para modelar dichas plantillas faciales. Una interfaz gráfica para designar los parámetros

iniciales para el procesamiento de las capturas y mostrar el resultado permitiría disminuir la barrera de entrada para el uso del software y el algoritmo de creación de plantillas.

1.1. Antecedentes

El rastreo de movimiento y posicionamiento de objetos en un espacio tridimensional ha sido un tema de estudio por mucho tiempo; desde sistemas tan antiguos como lo son los inicios del famoso "radar" [6] hasta el uso de equipo de MoCap como el usado en este proyecto. La tecnología de computación moderna permita utilizar la información de este posicionamiento para modelar cuerpos en un espacio tridimensional virtual, y reconocer poses y movimientos.

En colaboración con el PRIS-Lab, el laboratorio de reconocimiento de patrones y sistemas inteligentes de la Escuela de ingeniería eléctrica de la Universidad de Costa Rica, se realiza el proyecto en cuestión. El PRIS-Lab cuenta con equipo necesario para el rastreo de marcadores de MoCap de OptiTrack, además de un ambiente apto para la investigación y experimentación con este equipo. El sistema de OptiTrack se encuentra en posesión del grupo MOVE del PRIS-Lab, el cual se especializa en el análisis del movimiento humano y el desarrollo de sistemas que toman ventaja del procesamiento de la información de estos movimientos. El grupo MOVE ha trabajado en colaboración con el grupo CORE, de implementación de sistemas inteligentes en la robótica, en proyectos como *Terisa*, el cual consiste en el desarrollo de sistema de para la telepresencia de pacientes con esclerosis lateral amiotrófica, condición que progresivamente deteriora las neuronas y el tejido nervioso responsable por el control voluntario de los músculos del cuerpo. [7]

El PRIS-Lab tiene varios otros proyectos tocando la temática del modelado 3D utilizando MoCap (Normalización de cuerpos tridimensionales) [8] y de telepresencia (Telepresencia robótica utilizando LeapMotion) [9] mostrando como es un ambiente ideal para desarrollar un sistema de modelado facial 3D y sistemas basados en este modelado.

1.2. Alcance

El proyecto consiste en la implementación de un algoritmo apropiado para formar las plantillas faciales a partir de la posición de los marcadores faciales MoCap en las capturas; mediante la identificación de los marcadores por su posición se forma un modelo 3D de la superficie facial al armar uniones entre marcadores y se etiquetan los marcadores para mantener su identificación según la posición en los vértices 3D y acelerar el procesamiento de las capturas en términos de la etiquetación.

Además se desarrolla una interfaz gráfica para facilitar el uso del algoritmo de creación de plantillas el cual permite ingresar el número de marcadores en uso, y la posición de referencia para iniciar la etiqueta de los marcadores. Si se considera relevante se implementaría calibración de marcadores por posiciones o expresiones de referencia, en caso de que esto ayude al algoritmo a realizar su función.

1.3. Justificación

Un algoritmo el cual reduzca el tiempo de etiquetación de marcadores y permita la creación de plantillas faciales es de gran utilidad para proyectos subsecuentes los cuales al momento presente deben etiquetar manualmente los marcadores y no cuentan con el beneficio de formar la plantilla facial simultáneamente en caso de necesitarla para el modelado de la superficie facial.

El proceso de etiquetado manual de marcadores es un proceso tedioso, repetitivo y afectado por el error humano; la automatización de este proceso mediante el uso de algoritmos de identificación por posicionamiento 3D es de gran ayuda para acelerar el procesamiento de las capturas de MoCap y se pretende utilizar para ayudar con la creación de interfaces cuerpo-máquina como se propone en el proyecto de telepresencia *Terisa*, y proyectos como el uso del reconocimiento de expresiones faciales humanas como retroalimentación para máquinas y software o como interfaz alternativa de control.

1.4. Objetivos

1.4.1. Objetivo General

Desarrollar un algoritmo junto a una interfaz básica para la creación de plantillas faciales por capturas de marcadores faciales de MoCap los cuales sean etiquetados automáticamente por el algoritmo en cuestión.

1.4.2. Objetivos Específicos

1. Realizar una investigación bibliográfica acerca de los métodos de tracking de marcadores faciales MoCap, y de algoritmos de clasificación de puntos en un espacio tridimensional.

2. Determinar el software necesario para el procesamiento y identificación de marcadores, e interfaz de uso del modelador de plantillas faciales.

3. Diseñar el algoritmo de identificación, capaz de formar las plantillas faciales etiquetando los marcadores y formando un robusto modelo facial tridimensional.

4. Validar cuantitativamente el algoritmo de creación de plantillas mediante la comparación de los vértices 3D de una captura respecto a un modelo de referencia conocido.

5. Modificar la interfaz para facilitar el uso del modelador de plantillas faciales.

6. Divulgar los resultados mediante un informe técnico, la presentación del informe, un artículo formato IEEE, un vídeo corto y la participación en el PRIS-Seminar 2021.

1.5. Metodología

1. Realizar una investigación bibliográfica en artículos académicos y proyectos en la industria de MoCap de métodos de identificación de marcadores y el rastreo de estos marcadores a lo largo de los movimientos realizados.

2. Hacer un acercamiento al software utilizado para tomar los datos generados por el sistema de rastreo MoCap disponible en el PRIS-Lab.

3. Identificar que software y métodos de programación se utilizan para el rastreo de objetos en ambientes tridimensionales virtuales.

4. Implementar un algoritmo apropiado basado en la investigación bibliográfica previa, mediante el software escogido, que permita identificar varios puntos diferenciados en un ambiente tridimensional virtual.

5. Comparar las plantillas generadas por el algoritmo y la etiquetación de los marcadores con modelos faciales de referencia con posiciones predeterminadas.

6. Instalar del ambiente de desarrollo Tkinter para la creación de la interfaz gráfica como *frontend* de la aplicación.

7. Completar un reporte técnico donde se detalla el desarrollo del proyecto junto con la teoría utilizada como su base.

8. Crear un artículo académico en formato IEEE describiendo el proyecto presente.

9. Realizar un vídeo corto el cual explique el proyecto presente y demuestre su funcionamiento.

Capítulo 2

Marco Teórico

2.1. Captura de Moción (MoCap)

La captura de moción, comúnmente llamado MoCap por su nombre en inglés *Motion Capture*, se refiere al campo de aplicación de tecnología para grabar los datos de la posición y orientación de objetos en el espacio; los objetos de captura pueden ser no humanos o humanos, mas tienden a ser usualmente los últimos. En el MoCap se utilizan distintas tecnologías para grabar las posiciones y movimientos de los cuerpos: [10]

2.1.1. Captura Mecánica

Consiste en el uso de exoesqueletos típicamente metálicos con sensores para detectar la rotación de las articulaciones en el proceso de movimiento. Este tipo de captura no tiene ninguna referencia de la posición absoluta de los cuerpos, sino que se obtiene de los datos de las rotaciones; además no se tiene conciencia de la posición del suelo, requiere de re-calibración frecuente y suele ser incómodo y limitante para los sujetos.

2.1.2. Captura Óptica

Utiliza marcadores reflectivos los cuales son seguidos por varias cámaras, mediante la posición de las cámaras y los marcadores respecto a cada una de ellas se triangula la posición absoluta de cada marcador. Este tipo de captura es sujeta a interferencia de luz y se puede perder el rastreo de algunos marcadores en ciertos movimientos debido a objetos o cuerpos obstaculizar la vista de las cámaras. Este última desventaja se puede solventar estimando la posición de los marcadores perdidos respecto a su posición relativa a marcadores conocidos o a posiciones pasadas antes de perder la visibilidad de estos marcadores perdidos.

2.1.3. Captura Magnética

Mediante la interacción magnética entre bobinas que establecen flujos magnéticos ortogonales y marcadores/recibidores magnéticos puestos sobre el cuerpo en captura. Los objetos no metálicos y no mag-

néticos no interfieren con las capturas, lo cual presenta una ventaja ya que no se da la misma obstaculización que en sistemas ópticos. Por otro lado, los cables, computadoras, y objetos metálicos en el área de captura interfieren, además tienen una velocidad de captura menor a sistemas ópticos.

2.1.4. Captura con Radiofrecuencia

Estos sistemas toman ventaja de la radiofrecuencia y la reflexión de ondas electromagnéticas con cuerpos sólidos para capturar el movimiento, tamaño y distancia de objetos en un área frente a los transceptores de radiofrecuencia; ejemplos de este sistema en uso es el conocido Radar[1] y proyecto Soli de Google. [11] Estos sistemas suelen tener una resolución pobre sobre los objetos que captura, además de ser muy susceptibles a ruido ambiental o interferencia de otros objetos u ondas electromagnéticas presentes.

2.1.5. Captura Infrarroja

Sistemas similares a los ópticos, utilizan reflexión infrarroja para detectar los marcadores reflectivos. Tienen la ventaja de que en condiciones ideales, tienen menor interferencia que sistemas ópticos.

En el PRIS-Lab se encuentra disponible un sistema de la empresa OptiTrack, como lo son 16 cámaras Prime^x 41 de rastreo óptico infrarrojo de marcadores pasivos y activos de alta precisión, cámaras Flex 13 de rastreo óptico de marcadores con precisión sub-milimétrica y el software de interfaz gráfica y procesamiento Motive y ARENA de la misma empresa.

Figura 2.1: Cámaras Flex 13 de OptiTrack [1]

Figura 2.2: Cámara Prime^x 41 de OptiTrack [2]

2.1.6. Motive (*Software*)

Motive es una plataforma de software de la empresa OptiTrack con el propósito de controlar y procesar los datos de sistemas de captura de movimiento. Motive permite la calibración y configuración de los

[1] Radar: *Radio Detection and Ranging*

sistemas de captura además de el post-procesamiento de datos tridimensionales. Para obtener los datos, se utiliza reconstrucción mediante varias imágenes 2D para calcular las coordenadas 3D. Este software permite exportar los datos a varios formatos como CSV, C3D, FBX, BVH y TRC, también permite la transmisión a tiempo real a otros sistemas. [12]

2.1.7. MoCap Facial

El "MoCap Facial" se refiere a la utilización de técnicas y tecnologías de MoCap para obtener datos y modelos de la superficie facial de sujetos humanos. El rastreo y la captura de caras humanas presenta desafíos como el de capturar la complejidad del movimiento de los rostros humanos; además se vuelve necesario tener una mayor cantidad de puntos de rastreo más cercanos entre sí, por lo cual se utilizan marcadores especiales los cuales tienden a ser aproximadamente un tercio del tamaño de los marcadores convencionales de MoCap. Estos marcadores no cuentan con la ventaja de poder utilizar trajes de velcro o adhesivos similares para adherir los marcadores al sujeto, por lo cual vienen con una parte trasera con un adhesivo integrado diseñado para piel humana, el cual limita la cantidad de veces que se pueden utilizar estos marcadores.

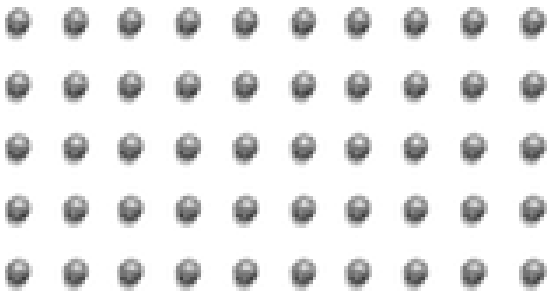

Figura 2.3: Marcadores Faciales de 3mm de OptiTrack [3]

2.2. Etiquetación de Marcadores

Cuando se habla de la etiquetación de marcadores de MoCap se refiere a la asignación de títulos a marcadores específicos con el propósito de poder identificarlos y mantener una continuidad entre los puntos de referencia conforme se desarrollan los movimientos. El etiquetado de los marcadores permite formar el esqueleto de modelo 3D del cuerpo, aunque en veces se usa el esqueleto para etiquetar los marcadores en un principio. [13] Esta etiquetación permite dar identificadores con sentido para interpretación humana e identificación de los marcadores específicos a las computadoras. Previo a la formación de un esqueleto y la etiquetación los datos consisten en solo puntos suspendidos en el espacio sin significancia alguna.

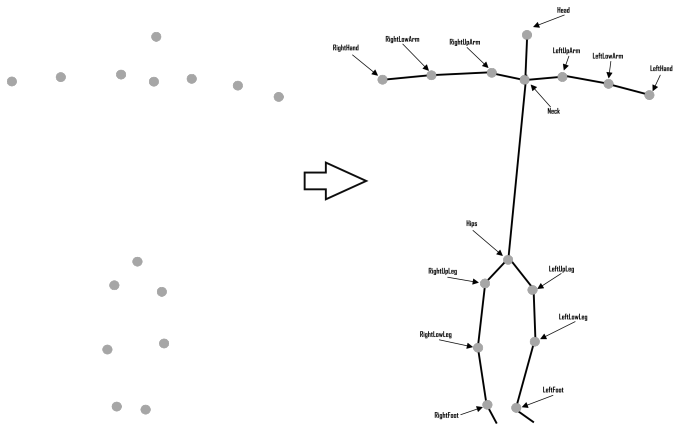

Figura 2.4: Nube de puntos a esqueleto etiquetado [Elaboración Propia]

2.2.1. Oclusión de Marcadores y Marcadores Ausentes

Un problema común que dificulta el lidiar con la etiquetación de marcadores es la obstaculización de los marcadores por objetos en la escena o las mismas partes del cuerpo del sujeto; también se tiene que ciertos movimientos o posiciones imposibilitan un rastreo continuo de algunos marcadores. [10] Estos problemas causan uno mayor el cual es el de marcadores ausentes, ya que no se puede triangular la posición de estos marcadores.
Solucionar la situación de los marcadores ausentes ha sido objeto de estudio por más de 2 décadas, mas una gran parte de los acercamientos utilizan las restricciones de movimiento del cuerpo humano para crear modelos predictivos y estimar la posición de los marcadores que se pierden; tal y como se presenta al utilizar algoritmos constreñidos con un filtro de Kalman en [14] o asignando pesos a predicciones hechas por el algoritmo tomando en cuenta la rigidez de los segmentos entre articulaciones humanas en [15]. Ya que la superficie facial es capaz de movimientos mucho menos rígidos que los de las extremidades humanas, además de la distancia más variable entre marcadores debido a la flexibilidad de la piel humana en la superficie facial, se vuelve necesario tomar distintos acercamientos para solventar el problema de marcadores ausentes o incluso de marcadores mal identificados por su cercanía con otros.

2.3. Grafos Bipartitos

Un grafo bipartito, también llamado bigrafo, es un tipo de modelo de representación por pareja de relaciones entre objetos en el cual los vértices (V) se pueden separar en dos conjuntos independientes (V_1 y V_2) y cada vértice en cada conjunto está conectado a un vértice en el otro conjunto. Si cada cada vértice en V_1 está conectado a cada vértice en V_2 y viceversa, se llama al grafo como grafo bipartito completo. [16]

2.3.1. Algoritmo Húngaro

El algoritmo húngaro, también llamado algoritmo de asociación de Munkres luego de la revisión de James Munkres en 1957, es un algoritmo de optimización combinatoria para resolver el problema de asociación con una complejidad de $O(n^4)$ y es un algoritmo muy popular para resolver la asociación de objetos con la capacidad de representarse en un grafo bipartito completo. [17]

El algoritmo se puede ejemplificar mediante una matriz de costo $n \times n$, donde cada conjunto del grafo bipartito se representa en las filas y columnas.

En este caso se tiene $V_1 = (A_1, B_1, C_1)$ y $V_2 = (A_2, B_2, C_2)$ y una matriz G donde se representa el costo de asociar cada vértice de V_1 con cada vértice de V_2 y viceversa.

	A2	B2	C2
A1	30	25	10
B1	15	10	20
C1	25	20	15

Figura 2.5: Matriz de ejemplo

Paso 1: Se inicia identificando el mínimo valor de cada fila y restándolo de manera que $G'_{i \times j} = G_{i \times j} - min\, G_i$.

30	25	10	→	20	15	0
15	10	20		5	0	10
25	20	15		10	5	0

Figura 2.6: Paso 1: Reducir Filas

Paso 2: Se identifica el mínimo valor de cada columna y se resta de manera que $G''_{i \times j} = G'_{i \times j} - min\, G_j$.

20	15	0	→	15	15	0
5	0	10		0	0	10
10	5	0		5	5	0

Figura 2.7: Paso 2: Reducir Columnas

Paso 3: Se etiquetan las filas y columnas de manera que se cubran todos los ceros utilizando la menor cantidad de etiquetas posibles. Si se tiene n filas y columnas etiquetadas, existe una asociación de costo mínimo; si se tiene menos de n filas y columnas etiquetadas se continúa el algoritmo.

15	15	0
0	0	10
5	5	0

Figura 2.8: Paso 3: Etiquetación y revisión de asociación

Paso 4: De no tener una asociación de costo mínimo luego del paso 3, se identifica el valor mínimo no etiquetado y se resta a todas las filas no etiquetadas y se suma a todas las columnas etiquetadas de manera que no quedan valores negativos.

15	15	0	→	10	10	0
0	0	10		0	0	15*
5	5	0		0	0	0

Figura 2.9: Paso 4: Crear nuevos ceros

Paso 5: Se repite el paso 3, de contar con n filas y columnas etiquetadas se consideran asociaciones de costo mínimo posibles; de no contar con suficientes filas y columnas etiquetadas se repite el paso 4.

10	10	0
0	0	15*
0	0	0

Figura 2.10: Paso 5: Re-etiquetación y revisión de asociaciones

Paso 6: Se escoge una de las asociaciones posibles, el algoritmo asegura que todas las posibles asociaciones tienen un costo total equivalente. El costo total se obtiene sumando los costos de cada asociación individual seleccionada. En este ejemplo se tiene un costo total de 45.

	A2	B2	C2
A1	10	10	0
B1	0	0	15*
C1	0	0	0

Figura 2.11: Paso 6: Selección de asociación

	A2	B2	C2
A1	30	25	10
B1	15	10	20
C1	25	20	15

Figura 2.12: Paso 6: Costo de asociación total

Capítulo 3

Desarrollo

3.1. Estructura del proyecto

El algoritmo se divide en 3 etapas generales para facilitar su desarrollo. Se inicia con una etapa de adquisición de datos mediante el uso de Motive y el sistema de cámaras Flex 13 de OptiTrack con marcadores faciales, donde se exportan los datos en un archivo CSV en formato de Motive. Se hace una lectura y procesamiento de los datos para eliminar información innecesaria para el algoritmo y ordenar los datos para asociarlos a marcadores y series de tiempo. Finalmente se utiliza una interfaz gráfica para asignar las etiquetas, se aplican las transformaciones necesarias a los datos ordenados, se procesan por el algoritmo de asignación y se exportan los resultados a un archivo CSV procesado con los marcadores etiquetados.

3.2. Toma de Datos

Para obtener los datos de captura facial se establece un protocolo a seguir para asegurar la mejor calidad de datos, minimizar ruido, interferencia y permitir que este sea replicable. Para cada toma es necesario anotar la cantidad de marcadores utilizados y el layout de estos sobre el rostro para su uso como argumentos del algoritmo.

3.2.1. Protocolo de toma de datos

1. Preparación del setup experimental

    1. Elección de lugar para filmación: es mejor alejar las cámaras de paredes reflectivas y no tener contraluces; se recomienda sitios con ventanas polarizadas para evitar interferencia de luz externa.
    2. Colocación de trípodes: no abrir muy amplias las bases para evitar impedir el libre paso y subir el tubo para tener oportunidad de colocar varios clamps de soporte de cámaras.
    3. Colocación de clamps.

4. Colocación de cámaras IR[1]: tratar de no hacer el volumen de filmación más grande de lo necesario.
5. Colocación de cámaras RGB[2]: aproximadamente a nivel de los ojos del sujeto en posición estéreo (como si fueran ojos humanos, pero no tan cerca una de la otra).
6. Colocación de luces: si es necesario para RGB sin que genere ruidos para las IR.
7. Colocación de silla: no muy lejos de los trípodes, pero lo suficiente para que la persona sentada con las piernas no mueva los trípodes, pues descalibraría el sistema.

2. Calibración

    1. Calibrar el sistema OptiTrack al menos cada 2 horas, o si hay cambios importantes en la iluminación del espacio experimental, o si hubo movimiento de los trípodes o el volumen de filmación.
    2. Grabar con las cámaras RGB el tablero de calibración para posteriormente extraer información 3D de las cámaras RGB

A partir de este punto, no puede haber movimiento de los trípodes o cámaras, pues descalibran el sistema, ni siquiera tocarlos, por lo que medio milímetro haría las mediciones IR 3D inexactas.

3. Preparación del sujeto de prueba

    1. Sentar a la persona en una silla distinta a la colocada para las filmaciones, pues para la puesta de marcadores es más cómodo tener espacio libre de trabajo, y no cercano a los trípodes.
    2. Explicarle a la persona el procedimiento y duración del experimento, así como tiempos de espera entre cada movimiento solicitado. Indicarle que se puede reiniciar en cualquier momento, que no hay problema si se equivocara (para que esté relajada y así los movimientos sean naturales). Este tipo de información, los posibles riesgos, etc. deben documentarse en un **consentimiento informado** que la persona debe firmar.
    3. Colocar al sujeto la gorra con marcadores y los marcadores faciales según la plantilla de marcadores establecida para la toma.
    4. Revisar la captura en el Motive: el sujeto de prueba se ve centrado y a una distancia adecuada (abarcando el rostro un porcentaje significativo de la imagen) en todas las cámaras.
    5. Revisar que se ve bien en las cámaras RGB: ídem anterior. Colocar el control de las cámaras para grabar en modo movie y desactivar el apagado de la cámara automático, además configurar la resolución, formato de saludos y otros parámetros deseados en los vídeos finales.

4. Preparación del sistema de obtención de datos

[1] IR: espectro infrarrojo de luz
[2] RGB: modelo de colores *Red Green Blue*

1. Iniciar grabación de vídeo RGB en ambas cámaras. Iniciar con el clap de sonido de inicio de grabación para sincronizar vídeos posteriormente.

2. Iniciar grabación IR en el Motive

3.3. Pre-Procesamiento de los datos

Se opta por usar el lenguaje de programación Python 3.8.3 con los paquetes Pandas, NumPy y MatPlotLib para el procesamiento de los datos. Se utilizan estos paquetes debido a la facilidad de trabajar con archivos CSV y manipulación de matrices con varios tipos de variables.
Mediante Pandas se lee el archivo CSV y se copia a una variable tipo DataFrame con el formato mostrado en la siguiente tabla.

Frame	Time	Unlabeled:662	Unlabeled: 662.1	Unlabeled: 662.2	...	Unlabeled:683	Unlabeled: 683.1	Unlabeled: 683.2
0	0.000000	-0.154009	0.327220	0.189062	...	-0.177703	0.225254	0.178559
1	0.011111	-0.153984	0.327221	0.189105	...	-0.177691	0.225269	0.178561
2	0.022222	-0.153974	0.327233	0.189122	...	-0.177683	0.225283	0.178561
...	...	...	...	...	...	...	...	...
3180	35.333333	-0.148974	0.326221	0.181742	...	-0.172728	0.222286	0.173589

Tabla 3.1: Ejemplo de formato de DataFrame utilizado en captura de MoCap facial[Elaboración Propia]

Una vez con el DataFrame se remueve la columna de Frame y Time. Se pasan los datos de este DataFrame a matrices de números NumPy de dimensión *Cantidad de Frames x 3* ya que solo se requieren las coordenadas X,Y y Z de cada trama de captura de las cámaras; estas matrices se asocian cada una a un marcador mediante un diccionario de Python el cual es variable según la cantidad de marcadores contenga el CSV inicial.

Para mostrar estos datos luego de ser procesados se utiliza MatPlotLib para plotear los marcadores en sus posiciones iniciales en las cuales no debería existir ninguna pérdida de marcadores bajo condiciones iniciales ideales según el protocolo.

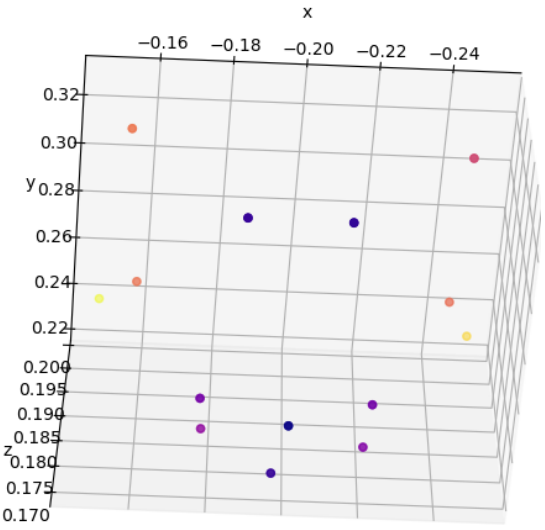

Figura 3.1: Trazo 3D de los 14 marcadores en sus posiciones iniciales en el layout presentado [Elaboriación Propia]

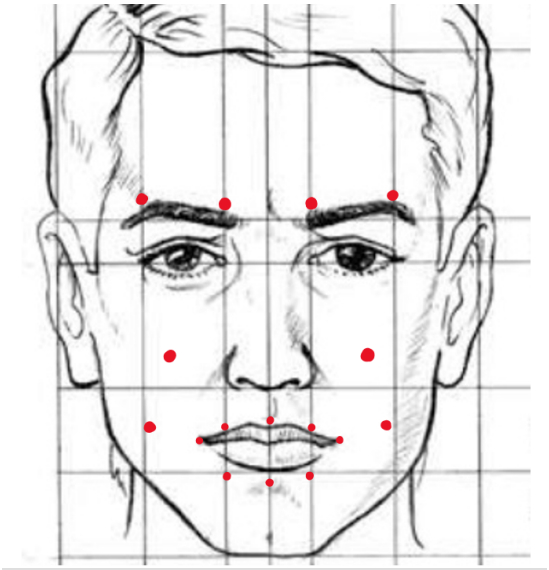

Figura 3.2: Layout de los 14 marcadores utilizados para el trazo 3D mostrado [Elaboración Propia]

A continuación se presenta un trazo 3D de una secuencia que consiste en levantar y bajar las cejas repetidamente, mostrando como se comportan y mueven los puntos a lo largo de una captura con un movimiento conocido.

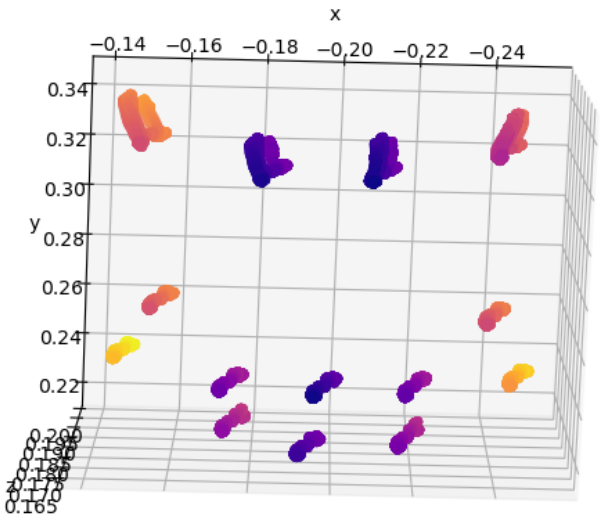

Figura 3.3: Trazo 3D de secuencia de levantar cejas [Elaboración Propia]

3.4. Interfaz Gráfica

Se utilizan las herramientas de Tkinter en Python, ya que esta herramienta ya se encuentra integrada en las instalaciones estándar de Python por lo cual no requiere ninguna instalación extra y corre en las plataformas de Windows, Linux y Mac OS.

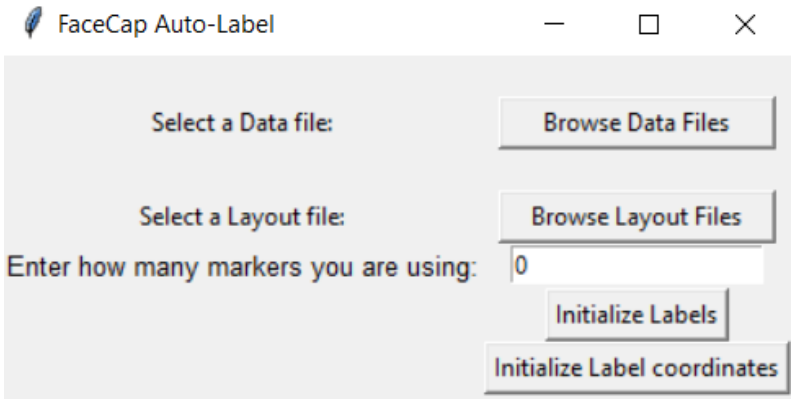

Figura 3.4: Interfaz Gráfica: Menú Principal para inicializar plantillas, etiquetas y coordenadas [Elaboración Propia]

Se inicia seleccionando el archivo CSV con los datos de las coordenadas para rastrear. Se utiliza un formato de las etiquetas para iniciar la captura con forma de *marker#* donde # se sustituye por el número de marcador que se está rastreando; en el caso de Motive este software utiliza este formato por defecto. Luego se selecciona una de las imágenes de plantillas para utilizar, para agregar nuevas plantillas solo es necesario añadir la imagen de la plantilla a la carpeta *"Layouts"* en la carpeta principal del programa.

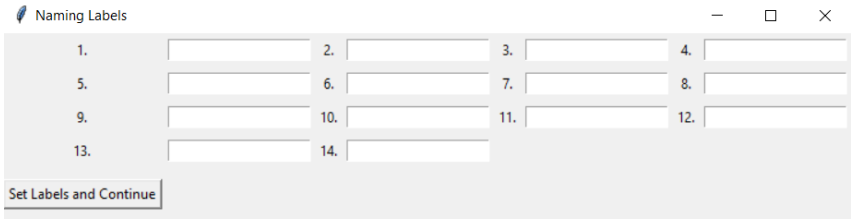

Figura 3.5: Interfaz Gráfica: Inicialización de etiquetas [Elaboración Propia]

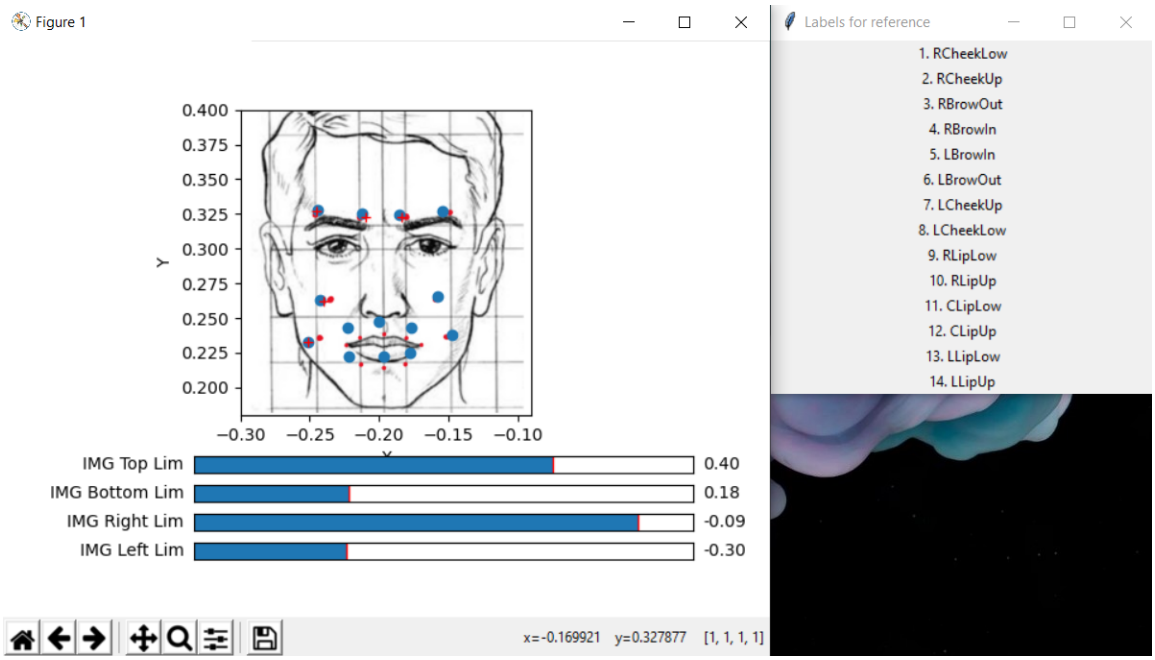

Figura 3.6: Interfaz Gráfica: Inicialización de coordenadas con plantilla [Elaboración Propia]

Luego de seleccionar los archivos base, se indica la cantidad de etiquetas a utilizar. Con la cantidad de etiquetas se inicializa los nombres de las etiquetas en una ventana que aparece al presionar *"Initialize Labels"*, el orden es importante ya que al asociar estas etiquetas con las coordenadas indicadas es necesario seguir el mismo orden.
Con las etiquetas agregadas al programa, se procede a inicializar las coordenadas con sus etiquetas apropiadas. Para seleccionar los puntos se utiliza la función GInput de MatPlotLib; esta función permite seleccionar una dada cantidad de puntos en una gráfica al hacer click izquierdo lo cual crea una cruz marcando el punto seleccionado, se puede de-seleccionar el último punto con click derecho y terminar la selección de manera temprana con click medio, una vez seleccionados todos los puntos las cruces desaparecen y se puede cerrar la ventana.

Al cerrar la ventana principal de la interfaz gráfica, se procesan los datos y se guardan los datos en un archivo CSV.

3.5. Procesamiento

El procesamiento de los datos consiste en la preparación del formato de estos para ser recibidos por algoritmo de asociación, la asociación trama por trama y la unificación de las etiquetas con las coordenadas y los resultados de asociación en una estructura de DataFrame del paquete Pandas de Python para luego exportarse a un archivo CSV.

Para iniciar se toman los valores iniciales de las etiquetas dadas por el usuario en la interfaz gráfica, luego con el resto de tramas se separan los datos trama por trama, luego de lo cual se separan los marcadores individualmente en cada trama. En orden cronológico se asocia cada trama con la anterior tomando como referencia para la primera asociación las coordenadas iniciales asociadas a las etiquetas del usuario.

```
# Function to run the matching algorithm to the marker dict frame by frame
def MarkerDictHungMatch(dataFilePath: str  = None, labelList: list = None, usrCoords: list = None):#, dictToPass: dict = None):

    dictMarkersFull = PrepareData(Path(dataFilePath))

    firstCoords = DictToInitPosList(dictOfMarkers = dictMarkersFull)
    initialCoords = From2DTo3D(xyPoints= usrCoords, xyzPoints= firstCoords)
    # print('InitialCoordCount: ', len(initialCoords), '\n')

    frameCount = len(dictMarkersFull['marker1'])
    labelCount = len(labelList)
    # print('Labelcount: ', labelCount, '\n')
    markerCount = len(dictMarkersFull.keys())

    # initialize the result dict with the initial coords with the labels as keys
    resultMatchedDict = {}
    for k in range(labelCount):
        resultMatchedDict[str(labelList[k]) + 'X'] = list()
        resultMatchedDict[str(labelList[k]) + 'Y'] = list()
        resultMatchedDict[str(labelList[k]) + 'Z'] = list()

        resultMatchedDict[str(labelList[k]) + 'X'].append(initialCoords[k][0])
        resultMatchedDict[str(labelList[k]) + 'Y'].append(initialCoords[k][1])
        resultMatchedDict[str(labelList[k]) + 'Z'].append(initialCoords[k][2])

    lastFrameData = initialCoords
    for i in range(1,frameCount):
        # making list with current frame coordinates for each marker
        myFrameList = list()
        for key in dictMarkersFull:
            myFrameList.append(list(dictMarkersFull[key][i,:]))

        # frame matching function
```

```
34          # get result from matching the last frame with the unlabelled
                frame
35          matchedFrame = FrameHungarianMatching(lastFrameList=
                lastFrameData, activeFrameList= myFrameList, labelVertex=
                labelList)
36
37          # adding the matched values to corresponding label in result
                dictionary
38          for k in range(labelCount):
39              resultMatchedDict[str(labelList[k]) + 'X'].append(
                    matchedFrame[labelList[k]][0])
40              resultMatchedDict[str(labelList[k]) + 'Y'].append(
                    matchedFrame[labelList[k]][1])
41              resultMatchedDict[str(labelList[k]) + 'Z'].append(
                    matchedFrame[labelList[k]][2])
42
43          # updating last frame data
44          tempList = list()
45          for labelKey in labelList:
46              tempList.append(matchedFrame[labelKey])
47          lastFrameData = tempList
48
49      dictToPass = resultMatchedDict
50      return resultMatchedDict
```

Luego se crea la matriz de costo para el algoritmo con una estructura de diccionario de diccionarios, de manera que las llaves del primer nivel indican la etiqueta , equivalentes a las columnas, y las llaves del segundo nivel constan de números de índice para representar cada marcador aún no asociado; las llaves de segundo nivel contienen cada una como costo la distancia euclidiana de la coordenada de la etiqueta de referencia al marcador en cuestión. Para la asociación se utiliza la implementación de Ben Chaplin en el paquete “hungarian-algorithm” disponible por el administrador de paquetes Pip. [18] Luego de esto la función del algoritmo devuelve una lista de tuplas con cada etiqueta asociada con el apropiado índice de marcador además de el costo de la asociación, se utiliza el este índice para añadir las coordenadas a la etiqueta apropiada.

```
1  # Function that takes two lists of coordinates each representing a
       frame, and matches them with labels using Hungarian Algorithm
2  # (WARNING: this func adapts the coordinates' format and matches it to
       the last frame,
3  # last frame list MUST be in the same order as the label list/vertexs)
4  def FrameHungarianMatching(lastFrameList: list = None, activeFrameList
       : list = None, labelVertex: list = None):
5      markerCount = len(activeFrameList)
6      labelCount = len(labelVertex)
7
8      markerVertex = activeFrameList
9      # dict to store the final matched coords
```

```
10      matchedCoords = {}
11      # cleaned marker list without NaNs
12      markerListClean = [list(s) for s in markerVertex if np.isnan(np.
            sum(s)) == False]
13      cleanMarkerCount = len(markerListClean)
14      # if there are less markers than labels, copy last frame coords
15      if cleanMarkerCount < labelCount:
16          for k in range(labelCount):
17                  matchedCoords[labelVertex[k]] = lastFrameList[k]
18
19      else:
20          hungEntryDict = {}
21          # Making the new dict to be used with the hungarian algorithm
                matching func
22          # the Last Frame Coordinate list HAS to be matched and ordered
                already
23          for i in range(markerCount):
24              # Create dict key for label vertex
25              if i < labelCount:
26                  hungEntryDict[str(labelVertex[i])] = {}
27              # Create dummies for extra marker spaces
28              elif labelCount <= i <= markerCount:
29                  hungEntryDict['dummy{}'.format(i)] = {}
30              # Loop to fill with a dict of the marker vertex and the
                    weight function of
31              for j in range(markerCount):
32                  # Dummy labels are non important
33                  if i >= labelCount:
34                      hungEntryDict['dummy{}'.format(i)]['{}'.format(j)]
                            = int(100*np.sum(max(lastFrameList)))
35                  # NaN values mean these coordinates aren't for this
                        frame
36                  elif np.isnan(np.sum(activeFrameList[j])):
37                      hungEntryDict[str(labelVertex[i])]['{}'.format(j)]
                            = int(100*np.sum(max(lastFrameList)))
38                  # The weigth of the match is the eucledian distance
                        between the last frame's point and the new frame's
                        one
39                  else:
40                      hungEntryDict[str(labelVertex[i])]['{}'.format(j)]
                            = EucDist(
41                          point1= lastFrameList[i],
42                          point2= [markerVertex[j][0], markerVertex[j
                                ][1], markerVertex[j][2]]
43                      )
44
45          hungMatchedList = algorithm.find_matching(hungEntryDict,
```

```
            matching_type='min', return_type='list')

        # If matching fails then assign with costly direct eucledian distance comparisson
        if type(hungMatchedList) == type(bool()):
            # If there are equal number of markers and labels use min Euc Dist comparisson
            if cleanMarkerCount == labelCount:
                for k in range(labelCount):
                    minDist = int(sys.maxsize)
                    resInd = int()
                    for l in range(labelCount):
                        currentDist = EucDist(
                            point1= lastFrameList[k],
                            point2= [markerVertex[l][0], markerVertex[l][1], markerVertex[l][2]]
                        )
                        if(minDist > currentDist):
                            minDist = currentDist
                            resInd = l
                    matchedCoords[labelVertex[k]] = list(markerVertex[resInd])

            # if there are more or less markers than labels repeat last frame
            else:
                for k in labelCount:
                    matchedCoords[labelVertex[k]] = lastFrameList[k]

        # Matching succesful then do this
        else:
            for item in hungMatchedList:
                matchedCoords[item[0][0]] = list(markerVertex[ int(item[0][1]) ])

    # print('Matched Coords: \n', matchedCoords)
    return matchedCoords
```

RCheekLowX	RCheekLowY	RCheekLowZ	RCheekUpX	RCheekUpY	...	LLipUpX	LLipUpY	LLipUpZ
-0.250965	0.23298	0.196249	-0.242958	0.262678	...	-0.176847	0.243076	0.176141
-0.250958	0.232998	0.196248	-0.242933	0.26271	...	-0.176834	0.243094	0.176147
-0.250955	0.233015	0.196237	-0.242936	0.262742	...	-0.176824	0.243117	0.176165
...	...	...	...	...	...	...	...	...
-0.245993	0.230349	0.189965	-0.238316	0.259832	...	-0.171921	0.239737	0.170635

Tabla 3.2: Ejemplo de DataFrame resultante al completar el procesamiento [Elaboración Propia]

Capítulo 4

Resultados

4.1. Comprobación

Se utiliza los datos del archivo “alzar cejas1.csv”, el cual corresponde a una captura en ambiente ideal del movimiento de alzar las cejas, tomada con la plantilla de 14 marcadores predeterminada del programa. Esta toma no cuenta con ninguna pérdida de marcadores, interferencia o cambios de identidad entre marcadores rastreados por lo cual es una buena referencia para comprobar la funcionalidad del algoritmo para rastrear marcadores. Ya que en el proceso de preparar los datos para asociación se separan las coordenadas de sus etiquetas, coordenadas consecuentes o pasadas, y que el algoritmo en sí trabaja trama por trama comparando todos los marcadores con solo sus posiciones relativas entre sí; si el algoritmo logra reconstruir el DataFrame original se puede comprobar su fidelidad.

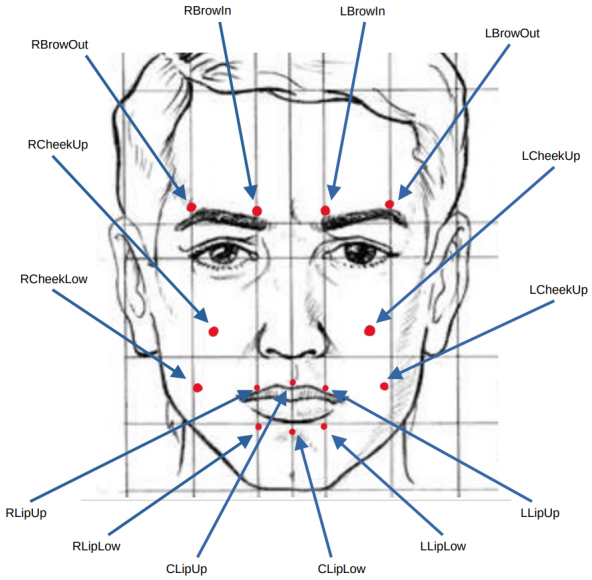

Figura 4.1: Plantilla de 14 marcadores y etiquetas utilizadas [Elaboración Propia]

Tomando en cuenta que se utiliza la primer trama para inicializar el algoritmo se puede utilizar esta primera trama para asociar los marcadores etiquetados en el producto final con los marcadores sin etiqueta del archivo “alzar cejas1.csv” original.

Etiqueta	Diferencia (m)	Etiqueta	Diferencia (m)	TOTAL (m)
RCheekLow	$2{,}78x10^{-17}$	LCheekLow	$1{,}93x10^{-17}$	
RCheekUp	$2{,}50x10^{-17}$	RLipLow	$2{,}33x10^{-17}$	
RBrowOut	$2{,}45x10^{-17}$	RLipUp	$2{,}36x10^{-17}$	
RBrowIn	$2{,}39x10^{-17}$	CLipLow	$1{,}96x10^{-17}$	
LBrowIn	$5{,}55x10^{-17}$	CLipUp	$1{,}96x10^{-17}$	
LBrowOut	$1{,}95x10^{-17}$	LLipLow	$1{,}91x10^{-17}$	
LCheekUp	$1{,}98x10^{-17}$	LLipUp	$2{,}36x10^{-17}$	
				$2{,}15x10^{-17}$

Tabla 4.1: Diferencia entre marcadores en “alzar cejas1.csv” y el resultado del algoritmo de asociación

Para obtener la diferencia en metros se promedia la diferencia trama a trama entre los marcadores de “alzar cejas1.csv” y los marcadores etiquetados por el algoritmo; es de notar que debido a transformaciones de Python y del programa para analizar estos resultados, en cada trama, se dan pequeñas desigualdades de coordenadas que deberían ser idénticas por redondeos. Estas desigualdades tienen un magnitud tan pequeña que se pueden omitir.

Se realiza una segunda comparación con otro archivo sin marcadores perdidos, el archivo se llama “fruncir ceno1.csv” y utiliza la misma plantilla y etiquetas.

Etiqueta	Diferencia (m)	Etiqueta	Diferencia (m)	TOTAL (m)
RCheekLow	$2{,}43x10^{-17}$	LCheekLow	$1{,}99x10^{-17}$	
RCheekUp	$2{,}55x10^{-17}$	RLipLow	$2{,}34x10^{-17}$	
RBrowOut	$2{,}34x10^{-17}$	RLipUp	$2{,}33x10^{-17}$	
RBrowIn	$2{,}49x10^{-17}$	CLipLow	$1{,}98x10^{-17}$	
LBrowIn	$1{,}99x10^{-17}$	CLipUp	$1{,}93x10^{-17}$	
LBrowOut	$2{,}14x10^{-17}$	LLipLow	$2{,}00x10^{-17}$	
LCheekUp	$1{,}92x10^{-17}$	LLipUp	$1{,}94x10^{-17}$	
				$2{,}17x10^{-17}$

Tabla 4.2: Diferencia entre marcadores en “fruncir ceno1.csv” y el resultado del algoritmo de asociación

En ambos casos, las desigualdades se pueden omitir por lo cual la diferencia entre marcadores continuamente rastreados por el equipo, y las asociaciones del algoritmo son pequeñas por no decir nulas. De esta manera se comprueba que el algoritmo logra hacer asociaciones eficazmente de trama a trama cuando se cuenta con suficientes marcadores en cada instancia que se aplica.

4.2. Comparación

Se compara el desempeño del algoritmo con un set de datos deliberadamente alterado, cubriendo y revelando el marcador *LCheekLow* repetidas veces en la captura. El archivo original reconoce 22 marcadores, pero la plantilla utilizada es de 19 marcadores, esta plantilla es una de las que tiene el programa por defecto. Para mostrar una alternativa, se tiene un set con una asignación manual de las etiquetas, en las secciones donde no se reconocen suficientes marcadores, menos a 19, se procede a copiar el último valor del marcador perdido hasta que nuevas coordenadas puedan tomar su lugar.

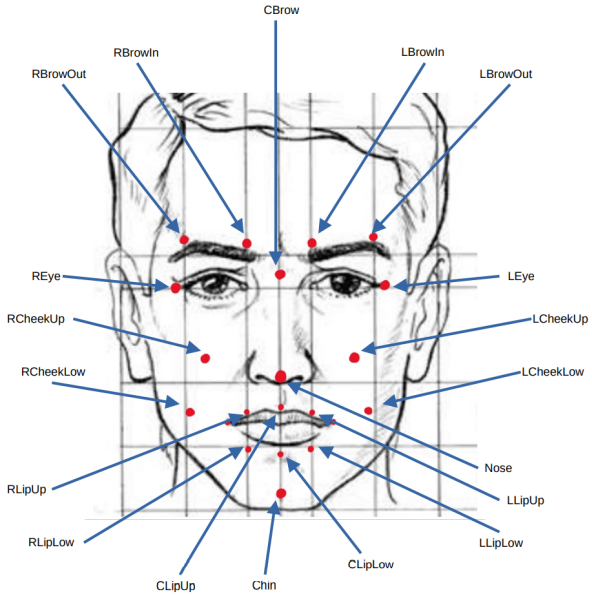

Figura 4.2: Plantilla de 19 marcadores y etiquetas utilizadas [Elaboración Propia]

Debido a que las capturas se realizaron en un ambiente ideal con poca interferencia, el laboratorio del PRIS-Lab, el único marcador perdido fue el cubierto deliberadamente, y los marcadores extra corresponden a momentos en los cuales se vuelve a revelar el marcador *LCheekLow* pero el equipo de captura no logra reconocerlo con la etiqueta original.

Etiqueta	Diferencia Manual (m)	Diferencia Algoritmo (m)	Dif. Manual vs Algoritmo (m)
RCheekLow	0	0.001357	0.001357
RCheekUp	0	0.001625	0.001625
REye	0	0.001653	0.001653
RBrowOut	0	0.002127	0.002127
RBrowIn	0	0.002304	0.002304
CBrow	0	0.002313	0.002313
LBrowIn	0	0.002388	0.002388
LBrowOut	0	0.002425	0.002425
LEye	0	0.002118	0.002118
LCheekUp	0	0.002186	0.002186
LCheekLow	0.274559	0.274559	$1{,}7904x10^{-17}$
RLipLow	0	0.001643	0.001643
RLipUp	0	0.001710	0.001710
CLipLow	0	0.001757	0.001757
CLipUp	0	0.001911	0.001911
LLipLow	0	0.001977	0.001977
LLipUp	0	0.001999	0.001999
Chin	0	0.001725	0.001725
Nose	0	0.002124	0.002124

Tabla 4.3: Diferencia entre marcadores en "tapar cachete derecho abajo2.csv" y el resultado del algoritmo de asociación

Al comparar la diferencia en posición de cada marcador trama a trama, se obtiene la diferencia promedio de toda la captura entre las posiciones del set original, el set con asociación manual y el set asociado por el algoritmo.

	Promedio (m)
Manual vs Algoritmo en LCheekLow	1.7904x-17
Captura total Manual	0.016311
Captura total Manual sin LCheekLow	0.000000
Captura total Algoritmo	0.014450
Captura total Algoritmo sin LCheekLow	0.001964
Captura total Manual vs Algoritmo	0.001860

Tabla 4.4: Diferencia promedio entre asignación del algoritmo y asignación manual

Ya que en el set original las coordenadas de *LCheekLow* se vuelven nulas tras perderse este la primera vez, se compara también el set con asociación manual al set asociado por el algoritmo; además se considera la diferencia promedio en las posiciones de los marcadores sin tomar en cuenta *LCheekLow* para ambos casos. Debido a transformaciones de Python y del programa utilizado para analizar los datos, se presentan pequeñas desigualdades por redondeos pequeños en las coordenadas que deberían ser idénticas. La magnitud de estas desigualdades es tan pequeña que se pueden omitir.

Capítulo 5

Conclusiones y recomendaciones

5.1. Conclusiones

1. Se reconocen varios acercamientos que se han presentado en estudios pasados para resolver el problema de rastreo y etiquetación de marcadores de MoCap, desde utilizar restricciones del movimiento del cuerpo humano, hasta la aplicación de redes neuronales y machine learning para lidiar con la flexibilidad del MoCap facial.

2. Se utiliza Python por su robustez como lenguaje de programación, la rapidez con la que se puede iterar distintas versiones del código y su gran documentación en linea, además de contar con paquetes como Tkiner, Pandas, MatPlotLib, NumPy y hungarian-algorithm.

3. Se identifica el algoritmo de asociación húngaro y la representación de datos como grafos bipartitos como una alternativa con mucho potencial para resolver el problema de asignación de etiquetas.

4. Se prueba el funcionamiento del algoritmo al comparar sus resultados con capturas ideales donde no se pierde ningún marcador, y capturas deliberadamente modificadas, de manera de que el marcador perdido se conoce y se puede realizar una asignación manual de etiquetas. Además se reconocen las limitaciones del la implementación realizada en este proyecto.

5. Se crea una interfaz gráfica para preparar los datos, asignar una plantilla y etiquetas a una posición inicial de referencia para el algoritmo.

6. Se preparó un informe técnico y una presentación para la promoción del proyecto, adicionalmente, se tiene planeado un artículo en formato IEEE y la participación en el PRIS-Seminar 2021.

5.2. Recomendaciones

- El algoritmo está limitado a la comparación entre la trama a asociar y una trama de referencia, por esta razón instancias donde se pierde uno o más marcadores por varias tramas consecutivas pueden sufrir en la disminución de la eficacia del algoritmo; un acercamiento predictivo o la

utilización de grafos multipartitos tomando en cuentas múltiples tramas pasadas pueden aliviar este efecto.

- El algoritmo utilizado tiene ciertas limitaciones intencionales para mejorar su velocidad de desempeño, como lo es la replicación de la última trama completa al reconocer menos marcadores que el mínimo para aplicar el algoritmo; se pueden realizar procesos alternativos en estos casos para limitar la cantidad de marcadores copiados de trama a trama.
- Si bien Tkinter tiene ventajas al ser liviano y estar integrado en las instalaciones de Python en varios sistemas operativos, otras plataformas de desarrollo de interfaces gráficas pueden permitir la creación de una interfaz más moderna y completa.

Apéndice A

Código

A.1. Archivo principal

Código del archivo principal del programa en Python, la versión más actualizada se puede encontrar en GitHub a la dirección (https://github.com/Pab-Ang/Pab_FaceCap)

```
from numpy.lib.npyio import save
import myFuncs as Fn
from pathlib import Path
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from tkinter import *
from tkinter import filedialog

userWindow = Tk()
userWindow.title('FaceCap Auto-Label')
# Get the Data (CSV) file location from user
dataFileName = StringVar()
Label(userWindow, text="Select a Data file:").grid(column = 0, row =
    0, pady=20)
dataFileBtn = Button(userWindow, text='Browse Data Files', width=18)
dataFileBtn.config(command=lambda:
    Fn.File_selection(dataFileName,"Select A Data File", [("CSV File",
        "*.csv")] )
    )
dataFileBtn.grid(column = 1, row = 0, pady=20)

# Get the Layout file location from the user
layoutFilePath = StringVar()
Label(userWindow, text="Select a Layout file:").grid(column = 0, row =
     2)
layoutFileBtn = Button(userWindow, text='Browse Layout Files', width
    =18)
layoutFileBtn.config(command=lambda:
```

```
26     Fn.File_selection(layoutFilePath, "Select A Layout File", [("PNG
           File", "*.png"),("JPEG File", "*.jpeg"),("JPG File", "*.jpg")] )
27     )
28 layoutFileBtn.grid(column = 1, row = 2)
29
30 usrMarkerCount = IntVar()
31 Label(userWindow, text='Enter how many labels you are using:', font=('
       bold', 10)).grid(column= 0, row= 3)
32 Entry(userWindow, textvariable=usrMarkerCount).grid(column= 1, row=3)
33
34 labelStringVar = StringVar()
35 initialLabelsBtn = Button(userWindow, text='Initialize Labels')
36 initialLabelsBtn.config(command =lambda:
37     Fn.InputLabelsWindow(usrLabelCount= int(usrMarkerCount.get()),
           StringVarToPass= labelStringVar)
38     )
39 initialLabelsBtn.grid(column = 1, row = 4)
40
41 initialUsrCoords = []
42 initialPlotBtn = Button(userWindow, text='Initialize Label coordinates
       ')
43 initialPlotBtn.config(command =lambda:
44     Fn.PlotInitialLayout(
45         dataFilePath= dataFileName.get(),
46         layoutPath= layoutFilePath.get(),
47         usrLabelCount = usrMarkerCount.get(),
48         listOfLabels = Fn.CsStringToStringList(labelStringVar.get()),
49         listForCoords = initialUsrCoords
50         )
51     )
52 initialPlotBtn.grid(column = 1, row = 5)
53
54 userWindow.mainloop()
55 #
       ----------------------------------------------------------------------------------------------------------------------------------
56
57 initialUsrCoordCount = len(initialUsrCoords)
58 print("USER INPUT COORDINATES", "| Count:", initialUsrCoordCount, '\n
       ')
59 for coord in initialUsrCoords: print(coord)
60
61 # print( '\n', "Initial Coordinates in order: \n")
62 # orderedList = Fn.From2DTo3D(xyPoints= initialUsrCoords, xyzPoints=
       firstCoords)
63 # for elem in orderedList: print(elem)
64
```

```
# ordered data
usrLabelList = Fn.CsStringToStringList(labelStringVar.get())

resultFinal = Fn.MarkerDictHungMatch(dataFilePath =dataFileName.get(),
    labelList=usrLabelList, usrCoords= initialUsrCoords)
resultFinalDF = pd.DataFrame(resultFinal)
print('Resultado Final: \n', resultFinalDF)

# saveWin = Tk()
# saveWin.title('Save As')
export_file_path = filedialog.asksaveasfilename(title= 'Save As',
   initialdir= Path.cwd(), defaultextension='.csv', filetype=[("CSV
   File", "*.csv")])
resultFinalDF.to_csv(export_file_path, index = False, header=True)
```

A.2. Archivo de funciones

Archivo de funciones de Python utilizadas en el programa principal

```
from pathlib import Path
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.widgets import Slider
from tkinter import *
from tkinter import filedialog
from hungarian_algorithm import algorithm

#FUNCTIONS AND CLASSES

# Recieves CSV file path for Mocap Data and returns a Dict of the
   markers as keys and position matrixes as values (FramesxXYZ)
def PrepareData(sessionFilePath):
    # Use pandas to read CSV and save to DataFrame
    # Drop unnecesary rows like Format, HexDecLabel and PositionLabel
       rows
    path_toData = Path(sessionFilePath)
    markersDF = pd.read_csv(path_toData.resolve(), skiprows
       =[0,1,2,4,5,6])
    # print(markersDF)

    # Rename Frame and Time header labels as they are read as Unnamed
    markersDF.rename({"Unnamed: 0":"Frame"}, axis="columns", inplace=
       True)
    markersDF.rename({"Unnamed: 1":"Time"}, axis="columns", inplace=
       True)
```

```
23      # Drop Time column | Unnecesary.
24      markersDF.drop(["Time"], axis=1, inplace=True)
25
26      # Count number of Rows
27      #FullRowCount = len(markersDF.index)
28
29      # Count quantity of markers
30      FullColumnCount = len(markersDF.columns)
31      # Each marker has x(t), y(t) and z(t) vectors
32      FullMarkerCount = (FullColumnCount-1)/3
33      FullMarkerCount = int(FullMarkerCount)
34      print('\nInitial Marker Count:', FullMarkerCount, '\n')
35
36      # Create Frame numpy Array
37      #FrameArray = markersDF.iloc[:, 0].to_numpy(copy=True)
38
39      # Create a Dict to store marker arrays
40      markerDict ={}
41      for i in range (1,FullMarkerCount+1):
42          markerDict['marker{}'.format(i)] = np.empty
43
44
45      # Copying each markers XYZ to a different NumPy matrix
46      for i in range(1, FullColumnCount, 3):
47      # Select only the columns of interest for the marker and copy them
            to activeMarker
48          activeMarker = markersDF.iloc[:, i:i+3].to_numpy(copy=True)
49
50          # The number of marker active in each loop
51          currentMarkerIndex = (i-1)/3 +1
52          currentMarkerIndex = int(currentMarkerIndex)
53
54          # add the active marker matrix to the corresponding marker
                dictionary key
55          markerDict['marker{}'.format(currentMarkerIndex)] =
                activeMarker
56      return markerDict
57
58  # Function to plot in 2D the first frame of FaceCap Data with a
        reference layout image
59  # markerDict in format ['markerN':((X1,Y1,Z1),(X2,Y2,Z2),...), '
        markerN+1':((X1,Y1,Z1),(X2,Y2,Z2),...)]
60  def PlotInitialLayout(dataFilePath: str, layoutPath: str,
        usrLabelCount: int, listOfLabels: list = None, listForCoords: list
        =None):
61
```

```
62      DisplayLabelsWindow(labelsList= listOfLabels, labelCount=
            usrLabelCount)
63
64      preMarkerDict = PrepareData(Path(dataFilePath))
65      # print('\n Marker Arrays')
66      # for key in preMarkerDict:
67      #     print(key)
68      #     print(preMarkerDict[key])
69
70      preMarkerCount = len(preMarkerDict.keys())
71
72      if(preMarkerCount < usrLabelCount):
73          NotEnoughMarkerData()
74          return None
75
76      # First frame Data in X Dim
77      xfdata = np.array((preMarkerDict['marker1'][0,0], preMarkerDict['
            marker2'][0,0]))
78      for i in range(3, usrLabelCount+1):
79          xfdata = np.concatenate(( xfdata,[preMarkerDict['marker{}'.
                format(i)][0,0]]),axis=0)
80      #First frame Data in Y Dim
81      yfdata = np.array((preMarkerDict['marker1'][0,1], preMarkerDict['
            marker2'][0,1]))
82      for i in range(3, usrLabelCount+1):
83          yfdata = np.concatenate(( yfdata,[preMarkerDict['marker{}'.
                format(i)][0,1]]),axis=0)
84      #First Frame Data in Z Dim
85      zfdata = np.array((preMarkerDict['marker1'][0,2], preMarkerDict['
            marker2'][0,2]))
86      for i in range(3, usrLabelCount+1):
87          zfdata = np.concatenate(( zfdata,[preMarkerDict['marker{}'.
                format(i)][0,2]]),axis=0)
88
89      # print("First X Data: \n", xfdata, "\n \n", "First Y Data: \n",
            yfdata)
90      # 2D plot of the first frame data
91      minX = np.amin(xfdata)
92      minY = np.amin(yfdata)
93      maxX = np.amax(xfdata)
94      maxY = np.amax(yfdata)
95      print("Min X | Max X | Min Y | Max Y \n" , minX, maxX, minY, maxY)
96      # Layout image for reference
97      faceimg = plt.imread(Path(layoutPath))
98
99      #creating subplots to use slider widgets on window
100     fig, ax = plt.subplots()
```

```
101     plt.subplots_adjust(left=0.1, bottom=0.35)
102     plotFigure = plt.scatter(xfdata, yfdata, cmap = 'plasma', picker =
            usrLabelCount)
103     # extent in data units in order imshow(img, zorder=0, extent=[left
            , right, bottom, top])
104     # Default testing values [-0.30, -0.09, 0.18, 0.40]
105     layoutFig = plt.imshow(faceimg, zorder=0, extent=[-0.30, -0.09,
            0.18, 0.40])
106     plt.xlabel('X')
107     plt.ylabel('Y')
108
109     minX_Slider = plt.axes([0.25, 0.1, 0.65, 0.03])
110     sl_minX = Slider(minX_Slider, 'IMG Left Lim', valmin=1.5*minX,
            valmax=0.5*minX, valinit=-0.30)
111
112     maxX_Slider = plt.axes([0.25, 0.15, 0.65, 0.03])
113     sl_maxX = Slider(maxX_Slider, 'IMG Right Lim', valmin=1.5*maxX,
            valmax=0.5*maxX, valinit=-0.09)
114
115     minY_Slider = plt.axes([0.25, 0.2, 0.65, 0.03])
116     sl_minY = Slider(minY_Slider, 'IMG Bottom Lim', valmin=0.5*minY,
            valmax=1.5*minY, valinit=0.18)
117
118     maxY_Slider = plt.axes([0.25, 0.25, 0.65, 0.03])
119     sl_maxY = Slider(maxY_Slider, 'IMG Top Lim', valmin=0.5*maxY,
            valmax=1.5*maxY, valinit=0.40)
120
121     def updateLims(val):
122         leftLim = sl_minX.val
123         rightLim = sl_maxX.val
124         bottomLim = sl_minY.val
125         topLim = sl_maxY.val
126
127         layoutFig.set_extent([leftLim, rightLim, bottomLim, topLim])
128
129     sl_minX.on_changed(updateLims)
130     sl_maxX.on_changed(updateLims)
131     sl_minY.on_changed(updateLims)
132     sl_maxY.on_changed(updateLims)
133
134     # Picker for picking initial points
135     coordList = plt.ginput(n=usrLabelCount, show_clicks =True)
136     # fig.canvas.mpl_connect('pick_event', lambda event:
137     #  onpick(event, xArray= xfdata, yArray= yfdata, zArray= zfdata)
138     #  )
139
140     for elem in coordList:
```

```
141         listForCoords.append(list(elem))
142     plt.show()
143     return listForCoords
144 
145 # Function to plot Dict data in 3D
146 def PlotDict3D(markerDict: dict = None):
147     markerCount = len(markerDict.keys())
148 
149     # Data in X Dim
150     xdata = np.array(markerDict['marker1'][:,0])
151     for i in range(2, markerCount+1):
152         xdata = np.concatenate((xdata,markerDict['marker{}'.format(i)
            ][:,0]),axis=0)
153     # Data in Y Dim
154     ydata = np.array(markerDict['marker1'][:,1])
155     for i in range(2, markerCount+1):
156         ydata = np.concatenate((ydata,markerDict['marker{}'.format(i)
            ][:,1]),axis=0)
157     # Data in Z Dim
158     zdata = np.array(markerDict['marker1'][:,2])
159     for i in range(2, markerCount+1):
160         zdata = np.concatenate((zdata,markerDict['marker{}'.format(i)
            ][:,2]),axis=0)
161 
162     ax = plt.axes(projection='3d')
163     ax.scatter3D(xdata, ydata, zdata, c=zdata, cmap='plasma')
164     ax.set_xlabel('x')
165     ax.set_ylabel('y')
166     ax.set_zlabel('z')
167     # Z+ pointing into screen | X- pointing to right screen border
168     plt.show()
169 
170 # def onpick(event, xArray, yArray, zArray, label: string, storageDict
    : dict):
171 #         ind = event.ind
172 #         print('Picked point at coordinates:', ind, xArray[ind],
    yArray[ind], zArray[ind])
173 
174 #Label input pop-up window
175 def InputLabelsWindow(usrLabelCount: int, StringVarToPass: StringVar):
176     entryList = []
177     popWin = Toplevel()
178     popWin.title("Naming Labels")
179 
180 
181     if(usrLabelCount <4):
```

```
182         notEnoughLabels = Label(popWin, text= "More than 4 labels are
                needed\n Close Window and resume")
183         notEnoughLabels.pack(pady=15,padx=10)
184     #dimensions for the window change if even or odd number of labels
185     elif (usrLabelCount % 2 == 0):
186         entryCounter=0
187         winColumnCount = int(4)
188         winRowCount = int(np.ceil(usrLabelCount/winColumnCount))
189         #Row Entry creation loop
190         for y in range(winRowCount):
191             #Column Entry creation loop
192             for x in range(1,2*winColumnCount,2):
193                 if(entryCounter >= usrLabelCount):
194                     break
195                 Label(popWin, text="{}.".format(entryCounter+1)).grid(row=
                        y, column=x-1, pady=0, padx=0)
196                 myEntry = Entry(popWin)
197                 myEntry.grid(row=y, column=x, pady=5, padx=5)
198                 entryList.append(myEntry)
199 
200                 entryCounter+=1
201 
202         passListBtn = Button(popWin, text='Set Labels and Continue',
                command=lambda:
203         ListToStringVar(listOfEntries= entryList, passStringVar=
                StringVarToPass)
204         )
205         passListBtn.grid(row= winRowCount +1, column= 0, pady=10)
206 
207 
208     else:
209         entryCounter=0
210         winColumnCount = int(3)
211         winRowCount = int(np.ceil(usrLabelCount/winColumnCount))
212         #Row Entry creation loop
213         for y in range(winRowCount):
214             #Column Entry creation loop
215             for x in range(1,2*winColumnCount,2):
216                 if(entryCounter >= usrLabelCount):
217                     break
218                 Label(popWin, text="{}.".format(entryCounter+1)).grid(row=
                        y, column=x-1, pady=0, padx=0)
219                 myEntry = Entry(popWin)
220                 myEntry.grid(row=y, column=x, pady=5, padx=5)
221                 entryList.append(myEntry)
222 
223                 entryCounter+=1
```

```
224
225         passListBtn = Button(popWin, text='Set Labels and Continue',
                command=lambda:
226             ListToStringVar(listOfEntries= entryList, passStringVar=
                    StringVarToPass)
227             )
228         passListBtn.grid(row= winRowCount +1, column= 0, pady=10)
229
230 # Function to display a pop-up window with labels ordered for
        reference
231 def DisplayLabelsWindow(labelsList: list = None, labelCount: int =
        None):
232     if labelsList is None:
233         Exception("No label list to display")
234     else:
235         lblWin = Toplevel()
236         lblWin.title("Labels for reference")
237
238         for i in range(labelCount):
239             textToDisplay = str("{}".format(i+1) + ". " + labelsList[i
                    ])
240             lblLabel = Label(lblWin, text=textToDisplay)
241             lblLabel.pack(padx=120)
242
243 # Function to select file on button press, wintitle is a Str and
        winfiletype is a tuple on format ("Title", "*.extension"),("Title2
        ", "*.extension2")
244 def File_selection(filenameVar: StringVar, wintitle: str, winfiletype)
        :
245     filename =  filedialog.askopenfilename(initialdir=Path(), title=
            wintitle, filetypes=winfiletype)
246     try:
247         print("Selected:", filename)
248         filenameVar.set(filename)
249     except:
250         print("No file selected")
251
252 # Function to transform a list of TKinter entries to a StringVar of
        comma separated strings
253 def ListToStringVar(listOfEntries: list, passStringVar: StringVar):
254     stringList=[]
255     entryCounter = 1
256     print("List of Labels")
257
258     for entry in listOfEntries:
259         stringList.append(str(entry.get()))
260
```

```
261         print(str(entryCounter) + ". " + str(entry.get()) )
262         entryCounter+=1
263 
264     stringComma = ','.join(stringList)
265     # to modify external variable in TKinter
266     passStringVar.set(stringComma)
267 
268 # Function to pass comma separated string to a list of strings
269 def CsStringToStringList(commaSepString: str):
270     return commaSepString.split(",")
271 
272 # Function to transform a list of strings to a single string
    separating each original object with a newline
273 def StringListToNewLineString(stringList: list):
274     nlString = ''
275 
276     for x in stringList:
277         nlString = nlString + str(x) + '\n'
278 
279     return nlString
280 
281 # Function to display warning message | Warning there are less markers
    in the data than there are labels
282 def NotEnoughMarkerData():
283     warningWin = Toplevel()
284     warningWin.title("WARNING!")
285 
286     myLabel = Label(warningWin, text="MARKER COUNT INSUFFICENT \n
        CHECK DATA FILE AND/OR LABEL COUNT")
287     myLabel.pack()
288     return None
289 
290 
291 # Function to compare XY points to XYZ points proyected to XY in
    distance:
292 def From2DTo3D(xyPoints: list, xyzPoints: list):
293     # 2d Array needs to have just as many elements as the 3D Array
294     labelCount = len(xyPoints)
295     markerCount = len(xyzPoints)
296     if (labelCount > markerCount):
297         print("More Labels than markers in data")
298         return None
299 
300     resultList = []
301 
302     for i in range(labelCount):
303         minDist = int(sys.maxsize)
```

```
304         resInd = int()
305         for j in range(markerCount):
306             activeDist = EucDist(point1= [ xyPoints[i][0], xyPoints[i
                ][1] ] ,point2= [ xyzPoints[j][0], xyzPoints[j][1] ])
307             if(minDist > activeDist):
308                 minDist = activeDist
309                 resInd = j
310 
311         resultList.append(xyzPoints[resInd])
312 
313     # print(resultList)
314     return resultList
315 
316 # Function to calculate distance in 2D points
317 def EucDist(point1=None, point2=None):
318     point1 = np.array(point1)
319     point2 = np.array(point2)
320 
321     dist = np.linalg.norm(point1 - point2)
322 
323     return dist
324 
325 # Function to transform the Dictionary way of storing initial marker
    position into a 2 dimensional np.array (matrix)
326 # with row= marker and column= x/y/z
327 def DictToInitPosList(dictOfMarkers: dict = None):
328 
329     markerCount = len(dictOfMarkers.keys())
330     keylist = list(dictOfMarkers.keys())
331 
332     # Pass the marker lists into a list of their initial positions
333     # as of Python 3.7 and newer Dicts are order-perserving
334     completeList = []
335     for key in dictOfMarkers:
336         activeMarkerInitList = dictOfMarkers[key][0,:].tolist()
337         completeList.append(activeMarkerInitList)
338 
339     # remove zero-ed and nan elements
340     initialPosList = [s for s in completeList if np.isnan(np.sum(s))
        == False]
341 
342     return initialPosList
343 
344 # Function that takes two lists of coordinates each representing a
    frame, and matches them with labels using Hungarian Algorithm
345 # (WARNING: this func adapts the coordinates' format and matches it to
    the last frame,
```

```
346 # last frame list MUST be in the same order as the label list/vertexs)
347 def FrameHungarianMatching(lastFrameList: list = None, activeFrameList
        : list = None, labelVertex: list = None):
348     markerCount = len(activeFrameList)
349     labelCount = len(labelVertex)
350 
351     markerVertex = activeFrameList
352     # dict to store the final matched coords
353     matchedCoords = {}
354     # cleaned marker list without NaNs
355     markerListClean = [list(s) for s in markerVertex if np.isnan(np.
            sum(s)) == False]
356     cleanMarkerCount = len(markerListClean)
357     # if there are less markers than labels, copy last frame coords
358     if cleanMarkerCount < labelCount:
359         for k in range(labelCount):
360                 matchedCoords[labelVertex[k]] = lastFrameList[k]
361 
362     else:
363         hungEntryDict = {}
364         # Making the new dict to be used with the hungarian algorithm
                matching func
365         # the Last Frame Coordinate list HAS to be matched and ordered
                already
366         for i in range(markerCount):
367             # Create dict key for label vertex
368             if i < labelCount:
369                 hungEntryDict[str(labelVertex[i])] = {}
370             # Create dummies for extra marker spaces
371             elif labelCount <= i <= markerCount:
372                 hungEntryDict['dummy{}'.format(i)] = {}
373             # Loop to fill with a dict of the marker vertex and the
                    weight function of
374             for j in range(markerCount):
375                 # Dummy labels are non important
376                 if i >= labelCount:
377                     hungEntryDict['dummy{}'.format(i)]['{}'.format(j)]
                            = int(100*np.sum(max(lastFrameList)))
378                 # NaN values mean these coordinates aren't for this
                        frame
379                 elif np.isnan(np.sum(activeFrameList[j])):
380                     hungEntryDict[str(labelVertex[i])]['{}'.format(j)]
                            = int(100*np.sum(max(lastFrameList)))
381                 # The weigth of the match is the eucledian distance
                        between the last frame's point and the new frame's
                        one
382                 else:
```

```
383                     hungEntryDict[str(labelVertex[i])]['{}'.format(j)]
                            = EucDist(
384                         point1= lastFrameList[i],
385                         point2= [markerVertex[j][0], markerVertex[j
                                ][1], markerVertex[j][2]]
386                     )
387
388         # print('Algorithm Entry \n',hungEntryDict)
389         hungMatchedList = algorithm.find_matching(hungEntryDict,
                matching_type='min', return_type='list')
390         # print('Matched List: \n', hungMatchedList)
391
392         # If matching fails then assign with costly direct eucledian
                distance comparisson
393         if type(hungMatchedList) == type(bool()):
394             # If there are equal number of markers and labels use min
                    Euc Dist comparisson
395             if cleanMarkerCount == labelCount:
396                 for k in range(labelCount):
397                     minDist = int(sys.maxsize)
398                     resInd = int()
399                     for l in range(labelCount):
400                         currentDist = EucDist(
401                             point1= lastFrameList[k],
402                             point2= [markerVertex[l][0], markerVertex[
                                    l][1], markerVertex[l][2]]
403                         )
404                         if(minDist > currentDist):
405                             minDist = currentDist
406                             resInd = l
407                     matchedCoords[labelVertex[k]] = list(markerVertex[
                            resInd])
408
409             # if there are more or less markers than labels repeat
                    last frame
410             else:
411                 for k in labelCount:
412                     matchedCoords[labelVertex[k]] = lastFrameList[k]
413
414         # Matching succesful then do this
415         else:
416             for item in hungMatchedList:
417                 matchedCoords[item[0][0]] = list(markerVertex[ int(
                        item[0][1]) ])
418
419     # print('Matched Coords: \n', matchedCoords)
420     return matchedCoords
```

```
421
422 # Function to run the matching algorithm to the marker dict frame by
        frame
423 def MarkerDictHungMatch(dataFilePath: str  = None, labelList: list =
        None, usrCoords: list = None):#, dictToPass: dict = None):
424
425     dictMarkersFull = PrepareData(Path(dataFilePath))
426
427     firstCoords = DictToInitPosList(dictOfMarkers = dictMarkersFull)
428     initialCoords = From2DTo3D(xyPoints= usrCoords, xyzPoints=
            firstCoords)
429     # print('InitialCoordCount: ', len(initialCoords), '\n')
430
431     frameCount = len(dictMarkersFull['marker1'])
432     labelCount = len(labelList)
433     # print('Labelcount: ', labelCount, '\n')
434     markerCount = len(dictMarkersFull.keys())
435
436     # initialize the result dict with the initial coords with the
            labels as keys
437     resultMatchedDict = {}
438     for k in range(labelCount):
439         resultMatchedDict[str(labelList[k]) + 'X'] = list()
440         resultMatchedDict[str(labelList[k]) + 'Y'] = list()
441         resultMatchedDict[str(labelList[k]) + 'Z'] = list()
442
443         resultMatchedDict[str(labelList[k]) + 'X'].append(
                initialCoords[k][0])
444         resultMatchedDict[str(labelList[k]) + 'Y'].append(
                initialCoords[k][1])
445         resultMatchedDict[str(labelList[k]) + 'Z'].append(
                initialCoords[k][2])
446
447     lastFrameData = initialCoords
448     for i in range(1,frameCount):
449         # making list with current frame coordinates for each marker
450         myFrameList = list()
451         for key in dictMarkersFull:
452             myFrameList.append(list(dictMarkersFull[key][i,:]))
453
454         # frame matching function
455         # print("Frame number: ", i, '\n')
456         # get result from matching the last frame with the unlabelled
                frame
457         matchedFrame = FrameHungarianMatching(lastFrameList=
                lastFrameData, activeFrameList= myFrameList, labelVertex=
                labelList)
```

```
458
459         # adding the matched values to corresponding label in result
                dictionary
460         for k in range(labelCount):
461             resultMatchedDict[str(labelList[k]) + 'X'].append(
                    matchedFrame[labelList[k]][0])
462             resultMatchedDict[str(labelList[k]) + 'Y'].append(
                    matchedFrame[labelList[k]][1])
463             resultMatchedDict[str(labelList[k]) + 'Z'].append(
                    matchedFrame[labelList[k]][2])
464
465         # updating last frame data
466         tempList = list()
467         for labelKey in labelList:
468             tempList.append(matchedFrame[labelKey])
469         lastFrameData = tempList
470
471     dictToPass = resultMatchedDict
472     return resultMatchedDict
```

Bibliografía

[1] OptiTrack. Flex 13. [Online]. Available: https://optitrack.com/products/flex-13/

[2] ——. Prime^x 41. [Online]. Available: https://optitrack.com/products/primex-41/

[3] ——. Facial marker 3mm. [Online]. Available: https://optitrack.com/products/motion-capture-markers/#mcp1125

[4] Maura Casadio, Rajiv Ranganathan, and Ferdinando A. Mussa-Ivaldi, "The body-machine interface: A new perspective on an old theme," *Journal of motor behavior*, vol. 44, 2012.

[5] Jun Wang, Lijun Yin, Xiaozhou Wei, and Yi Sun, "3D facial expression recognition based on primitive surface feature distribution," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, 2006, pp. 1399–1406.

[6] P. Sim. (2014) Making waves: Robert watson-watt, the pioneer of radar. [Online]. Available: https://www.bbc.com/news/uk-scotland-tayside-central-27393558

[7] Melissa Fallas Sanabria, "Esclerosis lateral amiotrófica," *Revista Médica de Costa Rica y Centroamérica*, vol. 67(591), pp. 89–92, 2010.

[8] N. A. Jiménez, "Desarrollo de un algoritmo para la normalización deun cuerpo tridimensional utilizando archivos bvh," *Technical report*, 2019.

[9] J. P. Ávila López, "Desarrollo de un sistema de telepresencia robóticaintegrando osvr, leap motion y nao para personas conmotora reducida," *Technical report*, 2017.

[10] M. Furniss. Motion capture. MIT Communications Forum. [Online]. Available: http://web.mit.edu/comm-forum/legacy/papers/furniss.html#5

[11] J. Lien, N. Gillian, M. Karagozler, P. Amihood, C. Schwesig, E. Olson, H. Raja, and I. Poupyrev, "Soli: Ubiquitous gesture sensing with millimeter wave radar," *ACM Transactions on Graphics*, vol. 35, pp. 1–19, 07 2016.

[12] OptiTrack. Motive documentation. [Online]. Available: https://v20.wiki.optitrack.com/index.php?title=Motive_Documentation

[13] J. Meyer, M. Kuderer, J. Müller, and W. Burgard, "Online marker labeling for fully automatic skeleton tracking in optical motion capture," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 5652–5657.

[14] V. Joukov, J. F. S. Lin, K. Westermann, and D. Kulić, "Real-time unlabeled marker pose estimation via constrained extended kalman filter," in *Proceedings of the 2018 International Symposium on Experimental Robotics*, J. Xiao, T. Kröger, and O. Khatib, Eds. Cham: Springer International Publishing, 2020, pp. 762–771.

[15] X. Deng, S. Xia, W. Wang, Z. Wang, L. Chang, and H. Wang, "Automatic gait motion capture with missing-marker fillings," in *2014 22nd International Conference on Pattern Recognition*, 2014, pp. 2507–2512.

[16] M. L. Zepeda-Mendoza and O. Resendis-Antonio, *Bipartite Graph*. New York, NY: Springer New York, 2013, pp. 147–148. [Online]. Available: https://doi.org/10.1007/978-1-4419-9863-7_1370

[17] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the Society for Industrial and Applied Mathematics*, vol. 5, no. 1, pp. 32–38, 1957. [Online]. Available: http://www.jstor.org/stable/2098689

[18] Ben Chaplin. hungarian-algorithm. [Online]. Available: https://github.com/benchaplin/hungarian-algorithm