INSTITUTO TECNOLÓGICO DE BUENOS AIRES – ITBA
ESCUELA DE INGENIERÍA Y GESTIÓN

Markerless Tracking

AUTORES: Lóránt Mikolás (Leg. Nº 57347)
Agustín Calatayud (Leg. Nº 57325)

DOCENTES TITULAR O TUTOR: Marcela Alejandra Guerrero

TRABAJO FINAL PRESENTADO PARA LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN INFORMÁTICA

Lugar: Ciudad Autónoma de Buenos Aires
Fecha: 22/07/20

Índice

1. Introducción **2**

2. Consideraciones y contexto **3**

3. Implementación **4**

3.1. Calibración de cámaras 4

3.2. Video mp4 a partir de set de imágenes .png 7

3.3. DeepLabCut 7

3.3.1. Redes Convolucionales 7

3.3.2. Configuración de DeepLabCut 8

3.3.3. Selección de puntos a trackear y etiquetar 9

3.4. Entrenamiento de la red neuronal 10

3.5. Aplicación de filtros en 2D 13

3.6. Análisis de .csv obtenidos 16

3.7. Triangulación de puntos en 3D 17

3.7.1. Triangulación Stereo con OpenCV 18

3.7.2. Triangulación Multi Camera PyMVG 20

3.7.3. Triangulación SFM con OpenCV 21

3.7.4. Comparación de los Resultados de las triangulaciones 21

Análisis cualitativo 21

Análisis cuantitativo 22

3.8. Aplicación de filtros temporales en 3D 24

4. Mejoras futuras **25**

4.1. Selección de red 25

4.2. Selección de markers 26

4.3. Construcción de archivo .FBX 26

4.4. Triangulación con más de 8 cámaras 26

4.5. Ejecución en tiempo real 27

4.6. Actualizaciones de DeepLabCut 27

4.7. Optimización de la elección de cámaras 27

5. Conclusión **28**

6. Anexo **30**

6.1. Video mp4 a partir de set de imágenes .png 30

6.2. Instalación del ambiente de Anaconda y ejecución de DeepLabCut 30

6.3. Descarga de los pesos de la red pre-entrenada MPII 32

6.4. Configuración de bodyparts del archivo yaml de DeepLabCut 33

7. Bibliografía y referencias **36**

1. Introducción

El objetivo de este proyecto es desarrollar un sistema de captura de movimiento de humanos con la ayuda de algoritmos de aprendizaje automático y triangulación, que sustituya el software licenciado actualmente utilizado y que evite el uso de un traje con marcas (*markers)*. Este proceso es conocido como *markerless motion capture*.

Las capturas generadas por este tipo de sistemas tiene un gran número de aplicaciones: estudio del movimiento de pacientes que padecen de alguna enfermedad o discapacidad, animaciones para el armado de películas y videojuegos, procesos de optimización logística, entre otros.

El desarrollo de una solución simple y eficiente de tipo *markerless* permitiría alcanzar cuatro metas fundamentales: la independencia del uso de trajes con markers infrarrojos; permitir la utilización de cualquier cámara comercial para la captura; poner al alcance de múltiples estudios y universidades un programa *open source* colaborativo; y disminuir el costo de entrada asociado al alto precio de los sistemas propietarios de software, cámaras especializadas y trajes.

Con respecto a nuestro proyecto, en primer lugar, se utilizará el *framework* de *deep learning* para estimación de poses *DeepLabCut* [1] a fin de entrenar un modelo que permita *trackear* puntos de interés en humanos sin el uso de *markers*. A su vez, se espera que el modelo entrenado permita resolver oclusiones parciales.

En segundo lugar, usando los puntos 2D obtenidos para cada cámara, se proyectan los puntos de interés en 3D usando la información de calibración de las mismas proveída en [2]. Este proceso se conoce como triangulación y para llevarla a cabo se implementaron tres metodologías diferentes de las cuáles se eligió aquella que produzca los mejores resultados.

Por último, los datos triangulados son procesados por filtros para ser suavizados y corregir irregularidades. El resultado puede ser utilizado como base para cualquiera de las aplicaciones mencionadas anteriormente.

2. Consideraciones y contexto

El presente trabajo fue realizado a lo largo de un año. Inició en Agosto del 2019 y finalizó en Junio del 2020. La investigación comenzó en la sala de captura de movimiento de la sede del Distrito Tecnológico (Parque Patricios) del ITBA. Ésta se encuentra a cargo de Marcela Guerrero, tutora del presente trabajo. Lóránt Mikolás había tenido la oportunidad de participar en algunos proyectos de la sala como becario y Agustín Calatayud tuvo la oportunidad de conocerla en las primeras reuniones del trabajo.

En el transcurso del año, se aplicaron múltiples herramientas, lenguajes y conceptos aprendidos en la universidad para alcanzar el objetivo propuesto. En particular, aquellos aprendidos en materias como "Métodos Numéricos Avanzados", "Física II", "Sistemas de Inteligencia Artificial" y "Aprendizaje Automático". También fue necesario dedicar gran parte de nuestro tiempo al aprendizaje de nuevos conceptos como la triangulación o el uso de redes neuronales convolucionales.

Resulta importante destacar que la pandemia del COVID-19 que inició a principios del año y la posterior cuarentena, impusieron una limitación al equipo de trabajo a la hora de volver a la sala de la universidad para llevar a cabo nuevas pruebas y capturas. Por lo tanto, la información recabada se limita a las 6 escenas capturadas en la sala durante el año 2019. Algunas de las pruebas que se podrían haber incluido son: la grabación de nuevos individuos y escenas, el aumento del número de cámaras utilizadas para las capturas, el análisis comparativo contra el sistema utilizado actualmente, entre otras.

Para finalizar, se menciona que el trabajo fue realizado en su mayor parte de manera remota en los hogares particulares de cada miembro del equipo, utilizando las computadoras personales para el análisis y procesamiento. La comunicación fue realizada con la ayuda de *Google Meet* y correo electrónico. Para el intercambio de archivos, se utilizaron "Unidades de Equipo" de *Google Drive*. Además, estuvo siempre a disposición del equipo de trabajo por parte de la tutora la posibilidad de consultar dudas e inquietudes a docentes de la universidad, alumnos y exalumnos.

3. Implementación

En esta sección se explicarán las diferentes partes de la implementación realizada así como las decisiones tomadas a lo largo del desarrollo.

Se divide el desarrollo en dos etapas principales: la obtención de la posición de los puntos de interés a partir del análisis de la red neuronal y la triangulación de los puntos en el espacio. Antes de desarrollar estas dos etapas, vamos a explicar el primer paso realizado para poder llevar a cabo la captura de las imágenes: la calibración.

3.1. Calibración de cámaras

Para empezar, se deben calibrar las cámaras para realizar la proyección y las capturas. Para ello se utilizó el programa proveído por el ex alumno del ITBA Braulio Sespede [2]. En total, cada escena es capturada por 8 cámaras. Entonces se cuenta con diferentes puntos de vista desde los cuales puede ser visto el sujeto a capturar. La calibración de las cámaras permite obtener las intrínsecas y extrínsecas de cada una de ellas.

Por un lado, las extrínsecas indican la rotación y traslación de las cámaras. Se debe tener en cuenta que la rotación viene dada por la fórmula de rotación de Rodrigues, que permite con una simple terna definir la orientación de las cámaras. La rotación expresa cómo rotar el sistema de coordenadas del mundo (‘world coordinates`) al sistema de coordenadas de la cámara (‘camera coordinates’). Mientras que la traslación entre los sistemas de coordenadas viene dada en el sistema de coordenadas de la cámara. Es decir, expresa donde está el punto (0,0,0) de las coordenadas del mundo pero en el sistema de coordenadas de la cámara. Ambos sistemas se pueden ver representados en la siguiente figura:

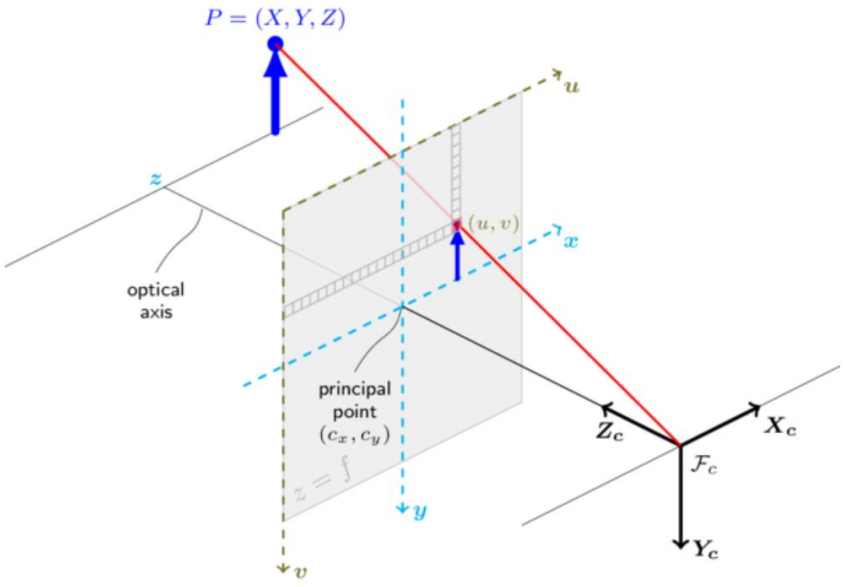

Figura 1: representación del sistema de la cámara (imagen extraída de [12])

Por otro lado, las intrínsecas indican la información propia de la cámara. Estos parámetros son: la distancia focal de las cámaras, formato del sensor y el ”punto principal”. Dado que no existen cámaras ideales es de suma importancia corregir los fotogramas obtenidos usando la información sobre la distorsión del lente.

Todos estos parámetros pueden ser usados en su conjunto para armar lo que se conoce como matriz de proyección. Dicha matriz al ser multiplicada con cualquier punto en el espacio da como resultado las coordenadas en píxeles del plano de la imagen.

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} \mathbf{X} \\ \mathbf{Y} \\ \mathbf{Z} \\ 1 \end{bmatrix} \tag{1}$$

Donde u y v como se observa en la ecuación 1 representan los píxeles correspondientes; fx y fy la distancia focal; cx y cy el punto principal; r_{ij} la matriz de rotación; t_i la traslación y X, Y, Z las coordenadas del punto en el mundo.

Es importante resaltar que, para realizar las operaciones matriciales que permite llevar un punto de las coordenadas del mundo a su posición en la cámara se deben utilizar

coordenadas homogéneas. Su objetivo principal es facilitar la operatoria agregando una cuarta coordenada al sistema euclídeo de tres (x, y ,z)[1].

A su vez se presenta a continuación el error de reproyección de cada cámara, obtenido a partir de la calibración de las mismas, en la siguiente tabla:

Número de Cámara	Error de Reproyección
0	0.1151
1	0.1380
2	0.1808
3	0.1313
4	0.1701
5	0.1656
6	0.1572
7	0.1809

Tabla 1: errores de reproyección de las 8 cámaras:

En las siguientes secciones se explicará la importancia de la elección de cámaras para la triangulación tomando en consideración variables de las mismas como las presentadas en la Tabla 1.

Para finalizar, se menciona que este programa tiene como *output* también los fotogramas (*frames*) de las diferentes cámaras para una captura de vídeo (los fotogramas por segundo obtenidos pueden ser ajustados).

[1] Para más información sobre ellas consultar [10]

3.2. Video mp4 a partir de set de imágenes .png

Se generaron videos a partir de los fotogramas de las capturas usando *ffmpeg* de *linux* [8]. Los comandos utilizados y la resolución de las imágenes pueden ser consultados en la sección 6.1. del Anexo.

La generación de videos fue necesaria debido a un error encontrado en el código de *DeepLabCut* que imposibilitaba el análisis directo de imágenes. Para solventar el problema se encontró que el análisis de videos por el programa, aún siendo en blanco y negro, no generaba problemas.

Tal como se mencionó la operación se puede realizar en un sistema operativo Unix o se puede instalar el *Windows Subsystem* para *Linux* en *Windows 10* [13].

3.3. DeepLabCut

DeepLabCut es un framework programado en *Python* que permite estimar poses de animales mediante el uso de redes neuronales convolucionales de tipo residual. En esta sección, se explicará porqué fue elegido y los pasos a seguir para desarrollar la solución propuesta.

3.3.1. Redes Convolucionales

Se conoce a una red neuronal convolucional (CNN) como un algoritmo de aprendizaje automático que recibe como entrada una imagen y que a través de operaciones de convolución permite extraer diferentes características de interés de la misma. Es capaz de capturar dependencias temporales y espaciales en una imagen mediante el uso de filtros o *Kernels*. Los *Kernel* aplicados permiten extraer las características de la imagen, tanto de alto como de bajo nivel (bordes, esquinas, etc.). Además la arquitectura permite reducir el costo computacional del análisis de las imágenes.

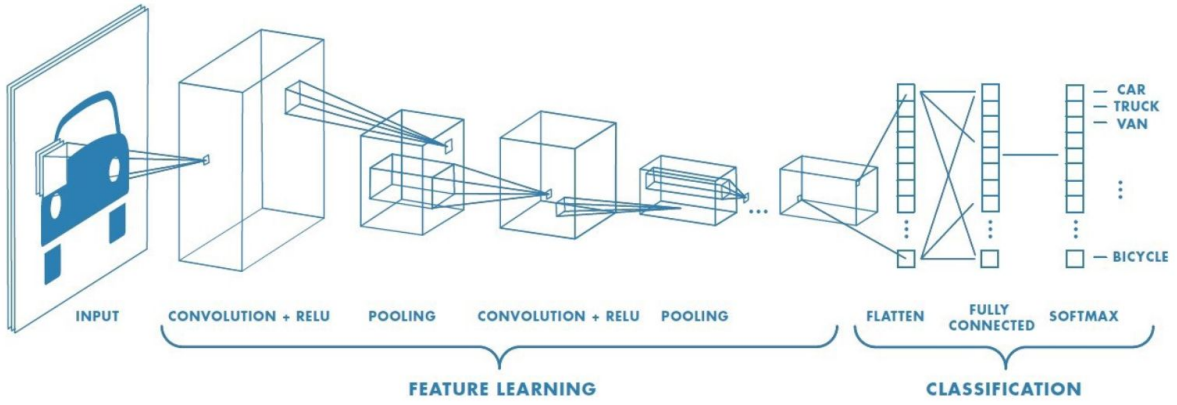

Figura 2: arquitectura de red convolucional (extraído de [15]).

El resultado de insertar un vector con los bytes de la imagen a un perceptrón multinivel no sería suficiente para capturar las características de interés con precisión.

Sobre el resultado de cada convolución se aplica una función *RELU* que permite reemplazar los valores negativos por 0.

Luego, se realiza la operación de *Pooling* que extrae los features dominantes que son independientes de la posición y rotación. El resultado es una matriz con menos datos ya que cada elemento extraído consiste en un promedio o el máximo de un parche.

Este proceso se repite múltiples veces dependiendo de la arquitectura. Cada capa consiste en el resultado de una convolución y una operación de *pooling*.

Por último, se convierte el resultado en un vector columna que es ingresado en un Perceptrón multicapa que realiza *backpropagation* y que gracias a la técnica de clasificación *Softmax* puede brindar el output final.[2]

3.3.2. Configuración de DeepLabCut

Para la configuración de *DeepLabCut* se siguió la guía que se encuentra en [1]. Se optó por la instalación del ambiente de *conda* para el uso de los módulos de *Python*. El ambiente permite la ejecución de la red neuronal en la placa de vídeo *Nvidia* para una notable mejora en *performance.* Entrenar la red utilizando el procesamiento de un CPU puede tardar

[2] Para una explicación más detallada del proceso consultar la referencia [15].

múltiples días (dependiendo de su capacidad de procesamiento) mientras que en la placa de video se pueden obtener buenos resultados en menos de 24 horas.

Para más información sobre cómo se instaló y ejecutó *DeepLabCut*, así como, sus comando generales consultar la sección 6.2. del Anexo.

3.3.3. Selección de puntos a trackear y etiquetar

La selección de puntos es de suma importancia ya que determinará el número de casos de uso en los cuales será utilizable la información exportada por el trabajo. Es decir, considerando que el output generado podría ser utilizado en animaciones, resulta fundamental seleccionar puntos que permitan obtener datos que puedan ser convertidos a formatos estándar de programas de animación. Es por esto que optamos por emular la modalidad *baseline* del traje de *Optitrack* [5]. El *baseline* determina 37 puntos (o etiquetas) dispuestos de manera que se pueda determinar la orientación de las diferentes partes del cuerpo a través de ellos.

Los markers se indican en el archivo *config.yaml* de *DeepLabCut* (ver *Anexo*). Se eligieron 11 capturas para etiquetar de las cuales se tomaron 20 fotogramas. Por lo tanto, se etiquetaron 220 imágenes en total. Las mismas se corresponden a tres cámaras o dos cámaras de una misma escena (formando diferentes ángulos con los puntos de interés).

Por una cuestión de velocidad y para mejorar la captura se utilizó el método K-medias para elegir las imágenes. Dicho método de agrupamiento nos permite elegir los fotogramas más significativos y diferenciados para que la red neuronal pueda generalizar mejor.

El algoritmo de *K-Medias* busca analizar y entender las relaciones existentes entre las variables observadas. Construye una partición del conjunto de imágenes en *K* conjuntos diferentes en donde se busca minimizar la función de distancia de los elementos dentro de cada grupo o *cluster*. Una vez que genera los *K* agrupamientos, se elige una imagen de cada uno de ellos para agregar al conjunto de imágenes a etiquetar.

Vale la pena mencionar que en algunos casos puede ser recomendable elegir qué imágenes etiquetar de manera manual sea por la naturaleza del inicio random de *K-medias* o por su costo computacional en un conjunto grande de imágenes. Entonces, *DeepLabCut* ofrece la posibilidad de realizar la selección de las imágenes manualmente (sección 6.2.).

Una vez que se eligieron los *frames* es posible comenzar con el proceso de etiquetado de los *markers*.

Con la interfaz gráfica de de *DeepLabCut* es sencillo etiquetar *frame* por *frame* los puntos indicados en el archivo de configuración. Con el objetivo de generar un buen etiquetado, tuvimos en cuenta que se debe mantener un criterio estricto y etiquetar en forma consistente todas las imágenes. Por ejemplo, si se piensa al pecho como un conjunto de píxeles en la imagen se debe intentar etiquetar consistentemente el mismo punto en todas ellas. Adicionalmente, consideramos importante evitar etiquetar puntos que no se encuentran a la vista u oclusionados por otras partes del cuerpo.

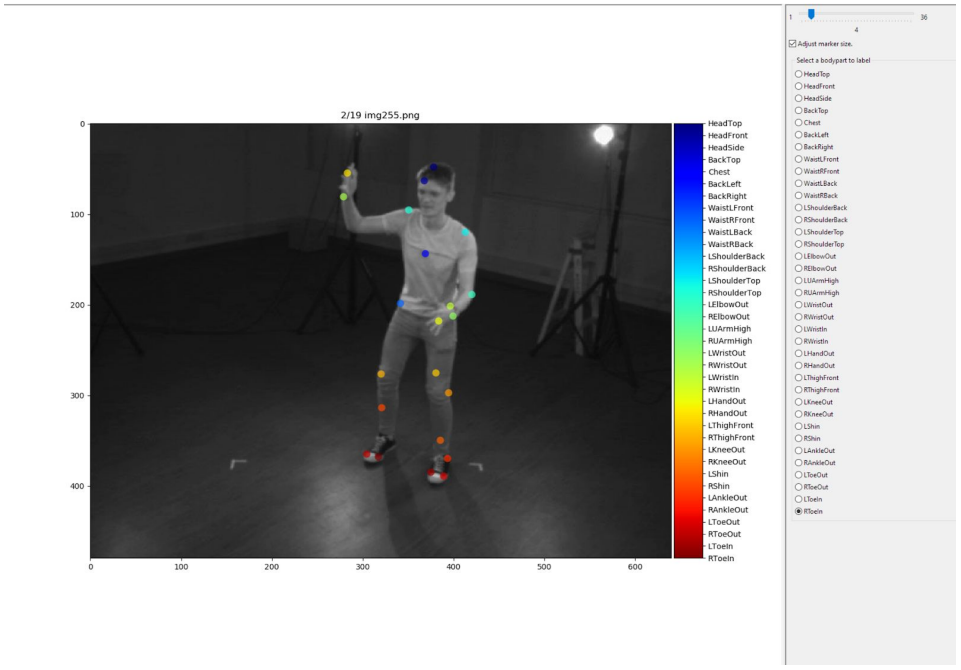

Figura 3: captura de pantalla del etiquetado de los puntos del Baseline de Optitrack.

3.4. Entrenamiento de la red neuronal

Como se mencionó, *DeepLabCut* utiliza redes convolucionales. En particular, se hace uso de *ResNets* (redes neuronales residuales). Estas redes neuronales residuales se basan en el algoritmo desarrollado por Insafutdinov et al. llamado *DeeperCut*. Las *ResNets* permiten resolver el problema de desvanecimiento de gradientes al utilizar *skip connections* o atajos que saltean algunas capas.

Para empezar, se realizó un primer entrenamiento con 100 imágenes con un resultado subóptimo. Los puntos de interés se desplazaban rígidamente y a veces se marcaban en lugares erróneos. Es por esto que se optó por duplicar la base de datos para el entrenamiento a 200 imágenes. Posteriormente, se agregaron 20 imágenes más obteniendo así un total de 220 imágenes etiquetadas. Consideramos que es posible refinar aún más este proceso, con el etiquetado de *frames* más representativos de las capturas y ampliando aún más la base de entrenamiento con más imágenes.

Consideramos de suma importancia resaltar que la base de información para el etiquetado se tomó de 5 escenas grabadas en las que participaron 4 personas diferentes (para tener una mayor variación en los datos de entrada) y de las cuáles se extrajeron 2 o 3 capturas de cámaras distintas. Obteniendo así el total de 220 imágenes etiquetadas previamente mencionado.

Se probaron 3 entrenamiento diferentes con resultados diversos. Los mismos se detallan a continuación.

En primer lugar, se probó entrenar la *Resnet 101* utilizando como base los pesos de la red neuronal entrenada para detectar humanos encontrada en [7], llamada MPII. Para su aplicación, se tuvieron que descargar los datos e indicar en el archivo de configuración que se utilicen los pesos correspondientes.

El entrenamiento se realizó en una *GeForce GTX 1070 (8 GB VRAM)* y tuvo una duración de aproximadamente 24 horas, obteniendo por lo menos 500.000 iteraciones para garantizar buenos resultados. Vale la pena mencionar que *DeepLabCut,* en su configuración *default*, guarda *checkpoints* cada 50.000 iteraciones. Por lo tanto, se podría retomar el entrenamiento desde cualquiera de estos *checkpoints* guardados.

En segundo lugar, se entrenó utilizando una *Resnet 50*, en vez *101*, y como base la *default* (`resnet_v1_50.ckpt`) por un total de 550.000 iteraciones.

Por último. se entrenó con una *Resnet 101*, pero con un conjunto de *labeling* más estricto. Es decir, se tomaron las imágenes que habían utilizado para el primer entrenamiento pero se sacaron puntos en donde era posible argumentar subjetivamente que el punto no era visto por esa cámara. Esto resultó en menos información de *input* para la red.

Los resultados de los errores de entrenamiento y testeo obtenidos para cada red se pueden ver desplegados en la siguiente tabla:

	Entrenamiento 1	Entrenamiento 2	Entrenamiento 3
Porcentaje de Entrenamiento del dataset	95	95	95
Error de entrenamiento (px)	0.85	0.9	1.18
Error de testeo (px)	8.32	6.17	11.54
P-cutoff usado	0.1	0.1	0.1
Error de entrenamiento con p-cutoff	0.85	0.9	1.18
Error de testeo con p-cutoff	7.78	4.64	5.5

Tabla 2: errores de entrenamiento y testeo para los diferentes entrenamientos realizados.

Como se puede observar en la Tabla 2 los mejores resultados son conseguidos con el segundo entrenamiento por lo que será a partir de aquel que se continuará con el resto de la investigación realizada.

Posteriormente, analizamos el resto de los vídeos para determinar la calidad del entrenamiento cualitativamente. En ellos se pudo apreciar que los mejores resultados se obtuvieron para el entrenamiento 2.

Finalmente, los resultados de la pérdida promedio por iteración para el entrenamiento seleccionado se pueden ver en el siguiente gráfico:

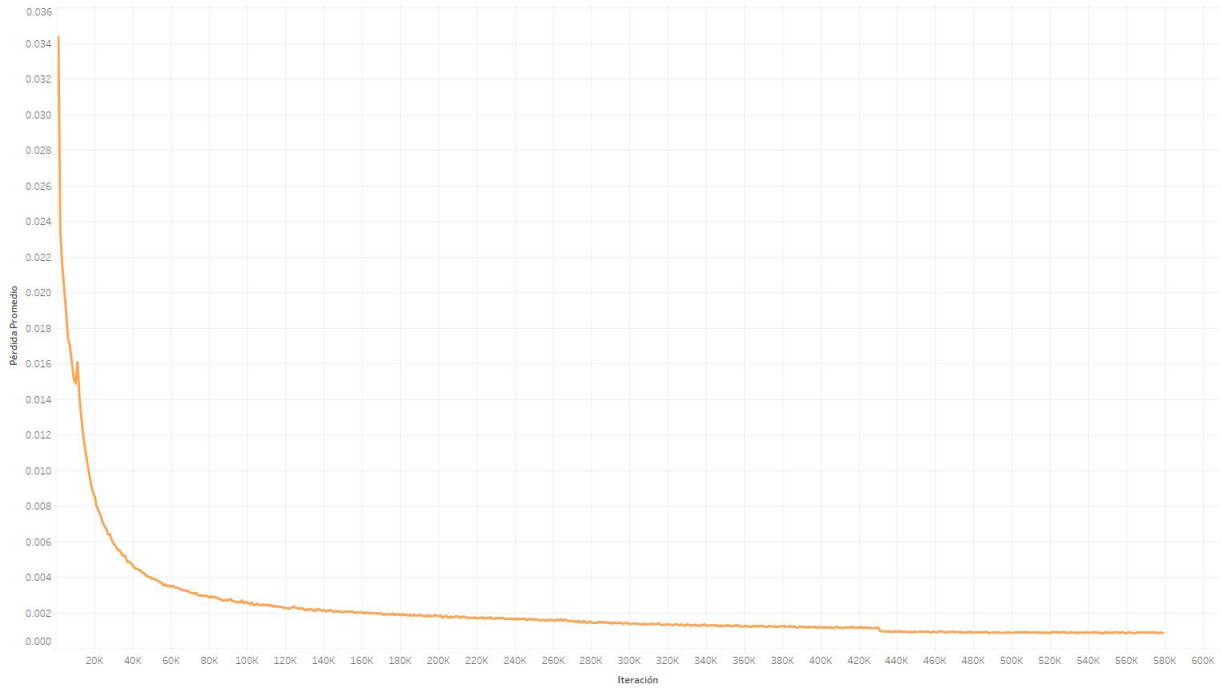

Figura 4: pérdida promedio (*loss*) en función del número de iteraciones del entrenamiento 2.

Se puede observar en la Figura 4 que la cantidad de iteraciones en la que se detuvo el entrenamiento hace que la pérdida promedio sea pequeña y se encuentre en un estadio asintótico.

3.5. Aplicación de filtros en 2D

Como se mencionó anteriormente, utilizando la red entrenada, se pueden realizar predicciones con videos nuevos entregados a la misma. Una de las representaciones obtenidas consiste en cómo se mueven a lo largo de la imagen los puntos de interés. Es decir, fotograma a fotograma en qué píxel se encuentra cada punto de interés.

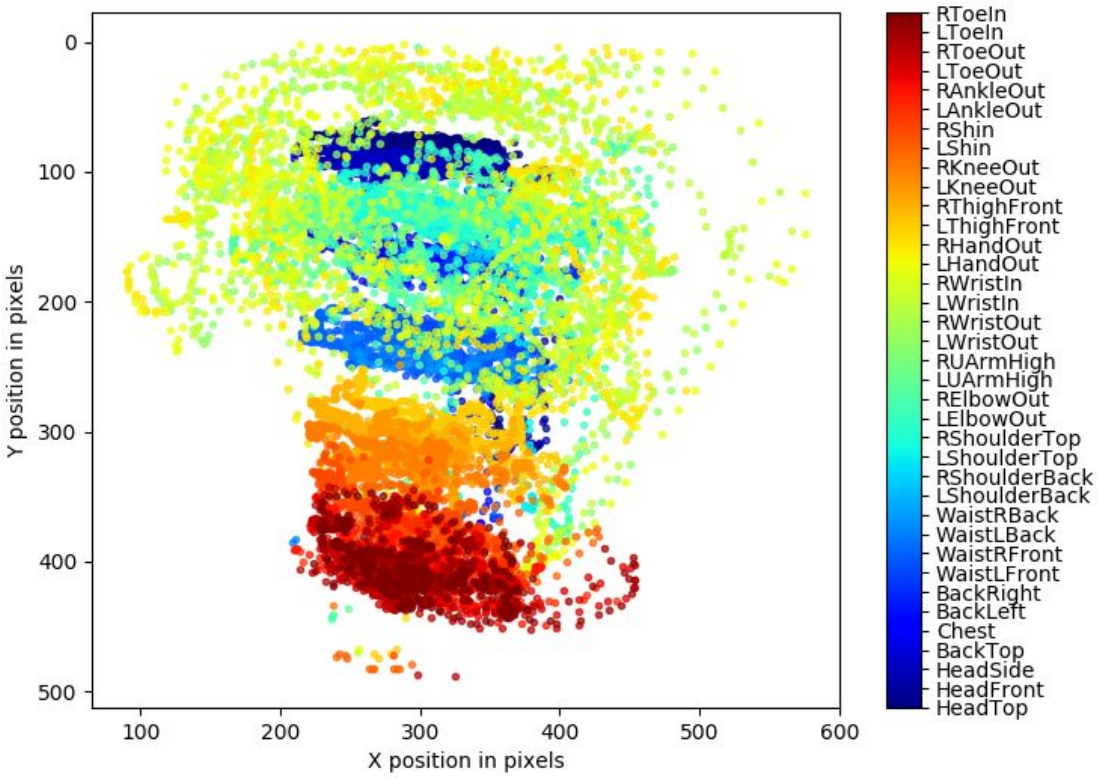

Figura 5: posición x e y en píxeles de los puntos interés.

Como se puede observar en la Figura 5, los *markers* con mayor variabilidad en cuanto a su posición se encuentran en las manos y los pies del sujeto.

En una primera instancia se utilizó la predicción directa de la red para obtener la posición de los puntos de interés. Esto probó ser poco preciso para algunos puntos, que dada su posición y la visión de la cámara en cuestión, eran ubicados por la red en lugares alejados de su verdadera posición en algunos fotogramas. En otras palabras, en algunos fotogramas se incurría un error grande en la predicción de la posición del punto de interés 2D.

Por esta razón se decidió aplicar un filtro a las predicciones. Para hacer esto se optó por un *median filter*. Se debe considerar que las posiciones de los puntos están asociados al movimiento del cuerpo del sujeto analizado y que no se deberían encontrar saltos significativos de las posiciones entre un *frame* y el siguiente. Por lo tanto, dado un tamaño de ventana apropiada para el filtro, se puede conseguir que puntos predichos en posiciones incompatibles sean ajustados con la mediana (el comando utilizado se encuentra en el Anexo 6.2.).

En cuanto a la diferencia del resultado se observó lo siguiente:

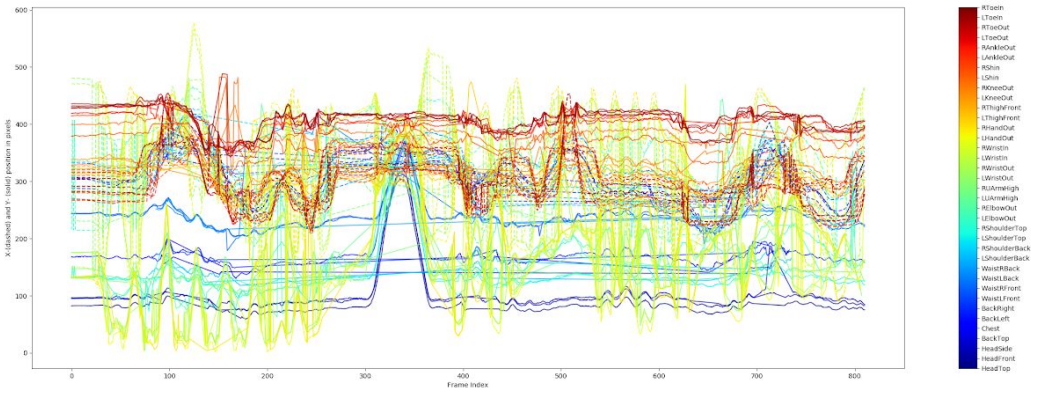

Figura 6: trayectoria de los markers en píxeles sin filtrar por la media para la escena 5 y la cámara 0 (para ver en mayor tamaño consultar [16]).

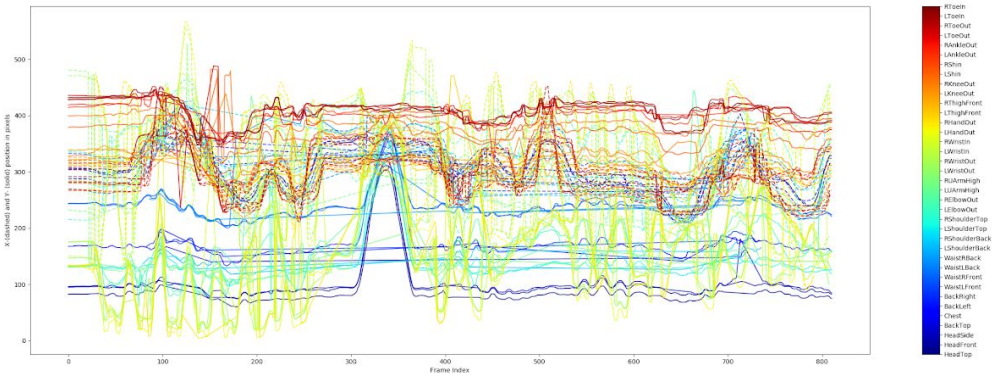

Figura 7: trayectoria de los markers en píxeles filtrados por la media para la escena 5 y cámara 0 (para ver en mayor tamaño consultar [16]).

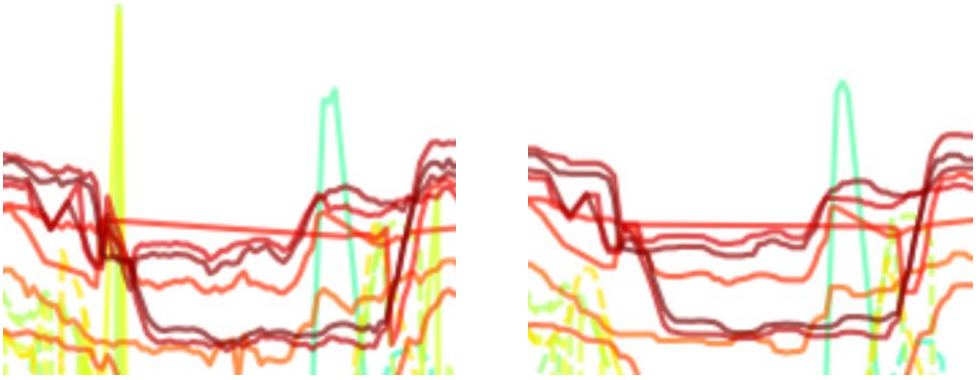

Figura 8: comparación en aumento de una región de la Figura 6 y 7 (a la izquierda el gráfico original y a la derecha el filtrado).

Si se realiza un aumento en las diferentes secciones del gráfico (Figura 8), es posible apreciar que irregularidades como el pico verde a la izquierda son corregidos por el filtro lo cual resulta en un análisis de puntos más regular. Además, cualitativamente observamos que con el filtro aplicado se consiguieron mejores resultados que sin él.

3.6. Análisis de .csv obtenidos

A partir del análisis de los videos se obtiene un archivo .csv con la posición *x* e *y* de los puntos de interés en pixels en cada *frame* (o fotograma). Además cada punto tiene un *likelihood* correspondiente. En otras palabras, la probabilidad de que el punto se encuentre donde lo determina *DeepLabCut*.

Si se grafican los resultados de la predicción en cada frame se obtienen resultados como en la siguiente figura:

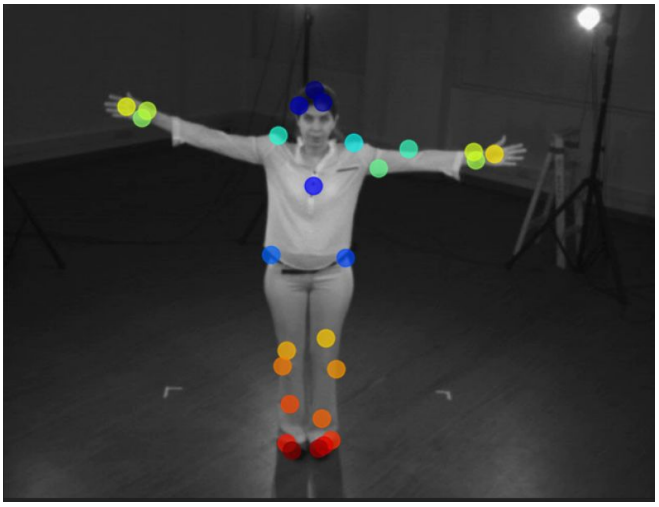

Figura 9: puntos de interés para un frame predichos por la red.

Se realizó un script en *Python 3* que carga en estructuras la información de cada punto de interés. Algunas cámaras, verán un punto de interés en un determinado *frame* mientras que otras no. Por lo tanto, para tener una triangulación adecuada, *frame* por *frame* se debe elegir

la información de las cámaras que hayan detectado el punto con el mayor *likelihood* además de las combinaciones que den un error de reproyección menor.

3.7. Triangulación de puntos en 3D

A partir de los archivos *.csv* (que tienen la posición x e y en pixels de los puntos de interés) y la información de calibración se debe realizar la proyección en el espacio (*x, y, z*) del punto. Para estudiar los conceptos matemáticos detrás de este proceso se utilizaron las clases encontradas en [10].

Antes que nada, se debe remover la distorsión provocada por la cámara para obtener la posición real de los pixels *x* e *y*.

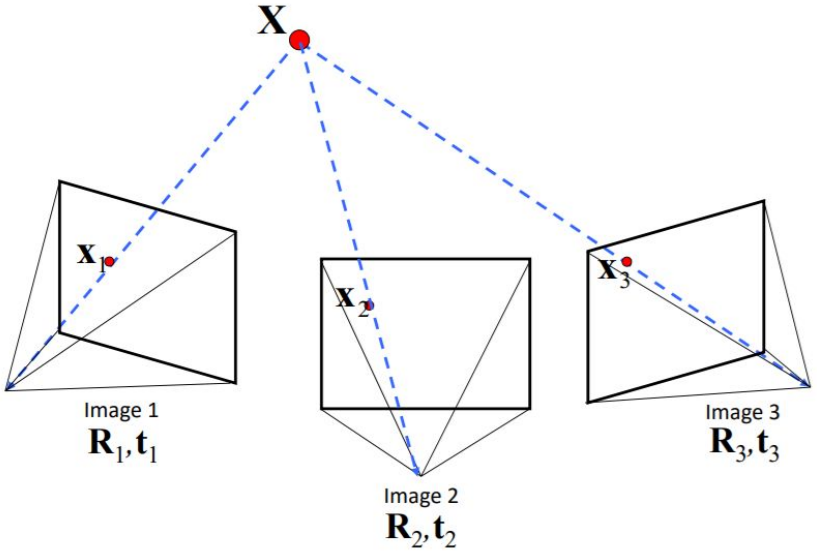

Figura 10: representación de la proyección (imagen extraída de [9]).

Existen múltiples métodos para triangular el punto en el espacio. En nuestra investigación probamos tres variantes que serán explicadas en las siguientes subsecciones.

Una vez implementada la proyección, se recomienda utilizar algún filtro para suavizar el movimiento de los puntos.

Para la verificación rápida de una triangulación correcta, se propuso el uso del software de visualización *Ovito* [11]. El mismo permite generar videos haciendo el rendering de múltiples puntos en sus coordenadas (*x,y,z*) para todos los *frames.*

3.7.1. Triangulación Stereo con OpenCV

Para un primer acercamiento a la triangulación se utilizó la librería *cv2* de *Python* que ofrece múltiples funciones para asistir la triangulación.

En primer lugar, para cada etiqueta de cada frame se eligieron un par de cámaras de manera de armar un conjunto estéreo que lo pueda triangular. Para mejores resultados se optó por tener cámaras que se encuentren al lado. Es decir, se eligieron pares de cámaras consecutivas que no tengan ninguna otra cámara entre sí. Como se ve en la siguiente Figura 11 el par (3,4) o (7,0):

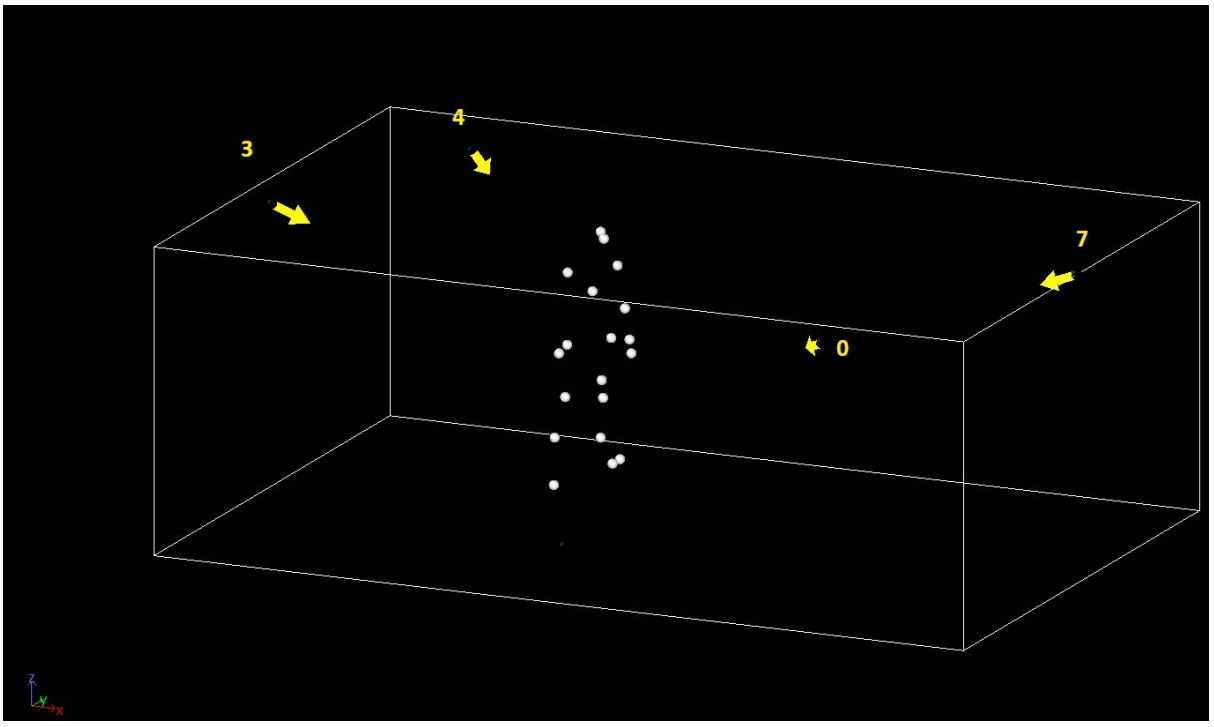

Figura 11: pares de cámaras visualizados en *Ovito*.

En la Figura 11, las flechas representan la orientación de las cámaras y los puntos las etiquetas trianguladas.

En segundo lugar, se debe calcular la traslación y rotación relativa para rectificar las imágenes. Para calcular la traslación relativa se hace la siguiente operación:

$$relativeT = R_o(-R_1^t T_1 - (-R_0^t T_0)) \qquad (2)$$

Donde en la ecuación (2) $relativeT$ representa la traslación relativa, R_i la rotación de la cámara i, T_i la traslación de la cámara *i* (*i* es 0 para la cámara izquierda y 1 para la cámara derecha). Se debe tener en cuenta que para las matrices rotación:

$$R_i^t = R_i^{-1} \qquad (3)$$

Con lo cual siempre que se multiplica por la transpuesta se indica que se realiza la transformación inversa. Entonces analizando (2) se puede ver que se está llevando T_0 y T_1 a coordenadas del mundo y luego se las resta para calcular su traslación relativa. Por último, se las multiplica por R_o , es decir, que se lleva la traslación relativa al sistema de la cámara izquierda ya que por convención se toma la izquierda como sistema de referencia.

Para calcular la rotación relativa simplemente se realiza:

$$relativeR = R_1 R_o^t \qquad (4)$$

Donde en la ecuación (4) $relativeR$ es la rotación relativa, R_1 y R_0 las rotaciones de las cámaras tal como se indicó previamente. Esta transformación resultante simplemente señala que se rota hacia el sistema de referencias del mundo aplicando la inversa y luego se rota nuevamente para llevar al sistema de referencias de la cámara 1.

Con esta información, la matriz de la cámara, los coeficientes de distorsión y el tamaño de la imagen se puede hacer el llamado a la función de Rectificación Stereo de *CV2*. Esto permite obtener las matrices de rotación $Rrec_1$, $Rrec_2$, y las matrices proyección P_1 y P_2 .

Luego, se llama a la función para remover la distorsión a los puntos con las matrices $Rrec_i$, P_i , K_i y los coeficientes de distorsión para la cámara *i*.

Al obtener los puntos sin distorsiones se puede entonces invocar la función de triangulación con las matrices P_1 , P_2 y los puntos correspondientes para cada cámara.

La función triangulatePoints de *OpenCV* utiliza DLT (*Direct Linear Transformation*) y el método propuesto en [17].

Por último, como el resultado de esta triangulación se obtiene en el sistema de coordenadas de la cámara 0 se debe aplicar una transformación para obtener el resultado final en el sistema de coordenadas del mundo. Para ello se realiza primero un pasaje a coordenadas homogéneas a euclídeas y se aplica la siguiente operación sobre el resultado:

$$Punto\ Triangulado = -R_0^t\,(Rrec_1^t\,X + T_0) \qquad (5)$$

Vale la pena mencionar que los mejores resultados cualitativos para esta configuración fueron obtenidos al usar un par frontal constante para triangular la mayor cantidad de frames. Es decir, no variar el par frontal elegido entre un *frame* y otro. El par frontal que tenga el mejor likelihood promedio para todos los markers de todos los frames fue seleccionado:

$$\text{máx}\Big(\sum_{c\,\varepsilon\,cámaras\ contiguas}\ \sum_{f\,\varepsilon\,frames}\ \sum_{m\,\varepsilon\,markers} likelihood_{fm}\Big) \qquad (6)$$

Cuando las dos cámaras del par frontal no alcanzan el umbral de likelihood de 0,5 se toman las cámaras opuestas ("par trasero"). Esto generalmente ocurría cuando se tenían que triangular markers en la espalda del individuo.

3.7.2. Triangulación Multi Camera PyMVG

Para realizar un triangulación con un mayor número de cámaras (no restringido a la triangulación stereo) se probó como primera alternativa la solución ofrecida por la librería PyMVG de Python (ver [18]).

Para elegir qué cámaras usar para triangular se determina un umbral base de *likelihood*. Si existen al menos dos cámaras que superen ese umbral se las elige. En caso de que no se supere el umbral, este se decrementa en un delta de likelihood hasta que existan al menos dos cámaras que lo superen para ese *marker* específico.

El algoritmo logra tomar cuatro cámaras en promedio para la triangulación de los diferentes markers con lo cual se cuenta con más información que en el caso *stereo* para ubicar el punto de interés en el mundo.

3.7.3. Triangulación SFM con OpenCV

En simultáneo, se puso a prueba la solución de triangulación que ofrece la librería *Structure From Motion* de *OpenCV Contrib* (repositorio en [19]). La misma permite realizar la triangulación con *n* cámaras. Dado que la función de triangulación se encontraba únicamente en la versión de *C++* de *OpenCV Contrib* se reescribieron los fragmentos de código que guardan la información de las cámaras y los *frames* en *C++* (así como el código de la impresión de los resultados).

Internamente la función también hace uso de DLT y la descomposición en valores singulares en donde se resuelve la siguiente ecuación (1).

Para la resolución del sistema de ecuaciones lineales, utiliza SVD que resulta en las coordenadas homogéneas buscadas. Luego, esas coordenadas se transforman en coordenadas cartesianas y son retornadas por el algoritmo.

La elección de cámaras se realiza con el mismo criterio que en la subsección 3.7.2.

3.7.4. Comparación de los Resultados de las triangulaciones

En esta sección nos propusimos comparar los diferentes métodos de triangulación y determinar cuál de ellos genera los mejores resultados.

Análisis cualitativo

En primer lugar, se menciona que los resultados obtenidos para la triangulación *stereo* con *OpenCV* no son precisos. Los *markers* triangulados saltan erráticamente entre de una posición a otra entre *frame* y *frame*. Si bien se puede apreciar el movimiento del individuo analizado, los resultados están lejos de ser usables para cualquier objetivo final. Además existen puntos oclusionados que no pueden ser vistos por ningún par contiguo de cámaras.

En segundo lugar, la triangulación *PyMVG* mejoró notablemente los resultados por sobre la metodología *stereo* al usar frecuentemente más de dos cámaras. El movimiento de los puntos es similar al realizado por el individuo grabado. Existen algunos, en las manos y los pies particularmente, que mantienen un movimiento errático muchas veces saltando entre

posiciones entre *frame* y *frame*. Aunque el resultado no se considera lo suficientemente preciso, es remarcable que se puede discernir el movimiento original en las posiciones de los *markers*.

En tercer lugar, la solución *SFM* ofreció resultados peores que la triangulación *PyMVG*. Los *markers* presentan una variación muy elevada de posición en coordenadas del mundo. Esto se le atribuye a la implementación específica del método de triangulación DLT.

Por último, se cree que la imprecisión que persiste para los métodos multicámara se debe, en gran parte, a la imprecisión que tiene la red neuronal para determinar la posición de los *markers* en las diferentes imágenes en 2D. Un error menor que diez píxeles a la hora determinar la posición (x,y) del *marker* puede provocar errores grandes en la triangulación. Además, cabe destacar que seleccionando un conjunto de cámaras más óptimo para cada *marker* los resultados podrían haber mejorado aún más.

Análisis cuantitativo

Para realizar una comparación cuantitativa se decidió usar la información de una captura de un *ChArUco board.* El board es una excelente forma de evaluar los diferentes métodos usados para triangular ya que se conocen sus dimensiones exactas y gracias a los patrones que presenta es fácil de *trackear* para las cámaras.

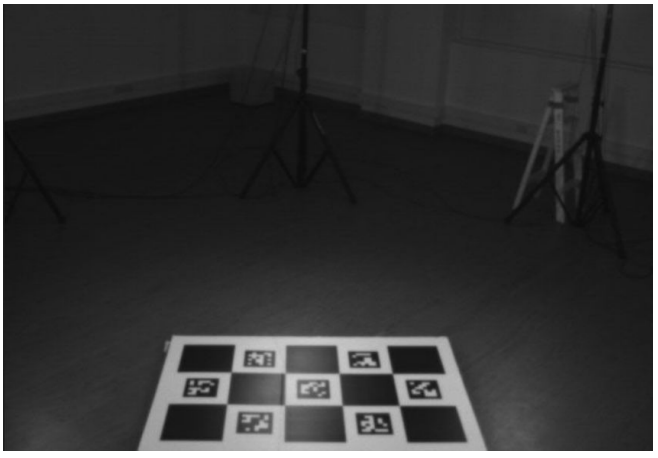

Figura 12: *ChArUco board* capturado.

El equipo de la sala de captura imprimió el patrón y colocó soportes de madera para otorgarle firmeza al *board.* También se le agregaron manijas para poder ser alzado con facilidad.

En primer lugar, se tuvo que obtener las coordenadas (x,y) de los *corners* del *board* para las diferentes cámaras para todos los frames. Para ello se utilizaron las funciones de *detectMarkers* y *refineDetectedMarkers* de *OpenCV*.

En segundo lugar, se triangularon los puntos obtenidos con los tres métodos de triangulación obteniendo así la posición de los *corners* del *board* triangulados para todos los *frames*. Con las coordenadas (x,y,z) de los *corners* es ahora posible comparar los métodos.

En tercer lugar, se tomó el promedio de los puntos (x,y,z) obtenidos para las diferentes cámaras por el software de calibración (ver [2]) como el *ground truth* de la posición de los *corners*. Esta triangulación se hizo individualmente por cada cámara ya que se cuenta con información precisa de las medidas del *board* con lo cual no hace falta usar dos o más cámaras para triangular. En otras palabras, se conoce la escala. Entonces, de esta forma tenemos una medida precisa contra la cual comparar los valores obtenidos para las diferentes triangulaciones.

Por último, se computó el error cuadrático medio con respecto al *ground truth* para los tres métodos de triangulación:

$$\frac{1}{F\,C} \sum_{f\,\varepsilon\,frame} \sum_{c\,\varepsilon\,corners} (norm(X'_{cf}) - norm(X_{cf}))^2 \qquad (7)$$

Donde F es el número de *frames,* C es el número de corners, X' la triangulación obtenida y X es el *ground truth.*

A continuación se presenta una tabla con los resultados:

	PyMVG	Stereo	SFM
Error	511.14	73788.12	11534.93

Tabla 3: error cuadrático para cada los tres tipos de triangulación probados.

También se computó el porcentaje de *frames* para los cuales se pudo observar corners para cada método:

	PYMVG	Stereo	SFM
Ratio de visibilidad	0.75	0.41	0.75

Tabla 4: ratio de visibilidad para los diferentes métodos de triangulación.

En conclusión, se puede ver que los resultados más precisos se obtuvieron para el método de *PyMVG*; luego el *SFM* y por último el *Stereo*. Esto se corresponde con los resultados cualitativos observados. También se puede apreciar que al usar cámaras contiguas en el método *stereo* (con el criterio frontal y trasero explicado anteriormente) se suelen encontrar corners con menos frecuencia que en los otros métodos. Esto pone en evidencia la ventaja de hacer uso de un mayor número de cámaras en los métodos de *PyMVG* y *SFM*. Por último, se resalta que el mayor error del método *SFM* con respecto a *PyMVG* se atribuye a su implementación.

3.8. Aplicación de filtros temporales en 3D

Una vez realizada la triangulación de los puntos se decidió suavizar el movimiento de los mismos en el tiempo. Se optó por añadir este procesamiento debido a que algunos de los puntos triangulados, ya sea por el error de reproyección o por errores en la posición predicha por la red neuronal, realizaban saltos incompatibles con el movimiento humano de un *frame* a otro.

Teniendo este problema en cuenta, se decidió aplicar un filtro que pueda resolver estas irregularidades a fin de suavizar la transición entre estados de forma coherente con el movimiento. El filtro seleccionado se denomina *Kalman filter*, el cual es ampliamente utilizado en este tipo de aplicaciones.

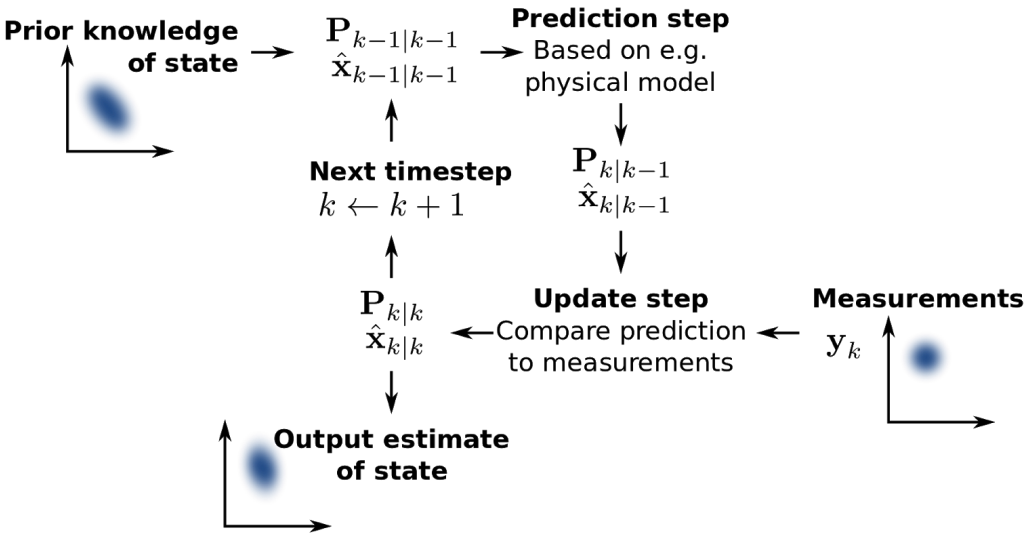

Figura 13: Proceso de filtrado del *Kalman Filter* (imagen extraída de [20]).

El filtro utiliza un modelo físico, en este caso un *MRUV*, para predecir el próximo paso y actualizar el punto obtenido por la triangulación con la predicción realizada de forma tal que la predicción tenga una ponderación mayor que lo triangulado. De esta forma se puede corregir de manera parcial las triangulaciones incoherentes con el movimiento realmente realizado.

Considerando que los saltos abruptos serían anatómicamente imposibles, el filtro es una forma de disminuir el efecto de estos errores haciendo que su salto sea menos abrupto y que se produzca un movimiento más suave.

4. Mejoras futuras

En esta sección se presentan las mejoras que se puede realizar sobre la solución realizada. Dichas mejoras abarcan diferentes componentes del trabajo y podrían complementar ciertos aspectos como la eficiencia y la precisión.

4.1. Selección de red

La selección de la red es un aspecto de suma importancia ya que la generalización de la misma constituye la base de los datos utilizados para la triangulación. Se podrían probar

diferentes tipos de redes residuales con más o menos capas y analizar sus resultados. Además, *DeepLabCut* en cada versión nueva lanzada agrega redes pre entrenadas con el fin de facilitar el tracking de diferentes especies. Por ejemplo, recientemente agregaron un modelo orientado a el *tracking* de humanos. Estos modelos se pueden usar como base para el análisis de los puntos de interés que el usuario desee.

4.2. Selección de markers

Para el entrenamiento de la red en este trabajo se optó por utilizar el *Baseline* de *Optitrack*. Como se explicó, el mismo cuenta con 37 puntos que conforman las diferentes extremidades de un ser humano. El resultado de la solución podría variar con la selección de otros puntos de interés. Se podría tomar un menor número de puntos que sean más fácil de generalizar para la red o simplemente tomar puntos que puedan ser generalizados con más precisión por la red.

4.3. Construcción de archivo .FBX

El resultado de la triangulación (las coordenadas x,y,z) podría ser usado para la construcción de un archivo *.fbx*. Este formato propietario de *Autodesk* sirve para la construcción de animaciones en múltiples herramientas (como por ejemplo los motores gráficos *Unreal Engine* y *Unity*).

Para ello es necesario construir un esqueleto teniendo en cuenta la posición y orientación de cada hueso. Esto podría ser implementado con la ayuda de la documentación del *SDK* de *.fbx* [3].

4.4. Triangulación con más de 8 cámaras

Para la triangulación con múltiples cámaras se usaron 8 en total en simultáneo pero agregar más cámaras de captura permitiría detectar puntos que no pueden ser captados por las cámaras actuales al salir del *frame*. Además, puntos que son oclusionados por alguna extremidad podrían ser visualizados por otra cámara. El aumento del número de cámaras (por ejemplo, 16 en vez 8) aumentaría la precisión de la triangulación.

4.5. Ejecución en tiempo real

En la solución actual la ejecución primero se obtienen los puntos en 2D a partir de la red para todos los frames y luego se realiza la triangulación de todos los frames. Podría realizarse una implementación en donde para cada frame consecutivo se obtengan los puntos 2D correspondientes y luego su triangulación para presentar resultados en tiempo real. Se podrían analizar redes computacionalmente más eficientes o elegir un método de triangulación en base a la velocidad de ejecución.

4.6. Actualizaciones de DeepLabCut

DeepLabCut es un framework que se encuentra actualmente en desarrollo y recibe actualizaciones de manera continua. Por ejemplo, como parte de la actualización 2.2 *beta* se agregó la opción de estimar poses y hacer *tracking* de múltiples animales en simultáneo. Constantemente se cargan nuevas funcionalidades que podrían mejorar los resultados obtenidos o ser relevantes para la solución.

4.7. Optimización de la elección de cámaras

En cuanto a la elección de cámaras al momento de triangular, se tomó como referencia las cámaras que mejor veían al punto. Es decir, las que mejor *likelihood* otorgaban sus puntos en 2D según la red. Esto, aunque muy fiable en 2D, no necesariamente provee una proyección en 3D con el mínimo error. Una posible mejora es proponer algoritmos de optimización para encontrar la mejor combinación de cámaras para la triangulación. Este conjunto debería ser el que menor error de reproyección consiga y sería conveniente que no se realice un escaneo de las opciones mediante fuerza bruta.

5. Conclusión

En conclusión, fue posible construir un primer acercamiento a la captura del movimiento de humanos en 3D mediante el uso de redes neuronales convolucionales y algoritmos de triangulación. Además, se usó un conjunto de algoritmos para mejorar la calidad de los resultados y para facilitar las operaciones. Se investigó una amplia gama de frameworks y configuraciones posibles para alcanzar el objetivo buscado.

Cabe resaltar que los resultados no alcanzaron un nivel de calidad comparable a los obtenidos con métodos que hacen uso de trajes con *markers* infrarrojos para facilitar la detección de los movimientos. Esto puede ser atribuido a múltiples factores.

Para empezar, la posición 2D en píxeles predicha por la red no es exacta. Errores de unos pocos píxeles en las cámaras utilizadas para triangular pueden inducir errores aún más grandes en la triangulación.

También se debe tener en cuenta que el entrenamiento utilizado para la predicción de las posiciones puede ser mejorado de diferentes maneras. Por ejemplo, puede ser ampliado el *dataset* usado para el entrenamiento de la red o cambiarla por otra red pre-entrenada que genere mejores resultados. A su vez, se destaca la posibilidad de usar una selección de puntos de interés o *markers* diferente que sea más sencillo de "aprender" para la red. Tampoco debe ser descartada la posibilidad de cambiar el *framework* utilizado por otro método de seguimiento de puntos de interés que sea más efectivo o por una nueva versión del mismo.

El laboratorio de captura de movimiento de la sede Distrito Tecnológico del ITBA cuenta con 16 cámaras y únicamente 8 fueron utilizadas en el proyecto. El uso de un mayor número de cámaras podría evitar predicciones erróneas sobre *markers* oclusionados y aumentar la precisión de la triangulación realizada. Incluso la resolución de las cámaras utilizadas es un factor a tener en cuenta que puede impactar en el resultado final del proyecto. Otro factor que podría influir de manera positiva en la triangulación es optimizar la elección de las cámaras.

Se cree que a partir de este primer acercamiento sería posible construir una implementación en el futuro con resultados similares a los que se obtienen con los sistemas propietarios que se venden hoy en día. Sobre la estructura actual se podría buscar obtener un sistema que realice el proceso en tiempo real o que ofrezca como formato de *output* un

archivo estándar *.fbx* para ser utilizado en motores gráficos como *Unity* o *Unreal Engine*. Este material escrito y generado podrá ser utilizado por varios alumnos que les interese expandir sobre este proyecto en cualquiera de las direcciones mencionadas.

Para finalizar, cabe resaltar la satisfacción del equipo con el trabajo realizado a pesar de las limitaciones impuestas por el contexto de pandemia. Se destaca el gran número de conocimientos adquiridos en múltiples áreas a lo largo del año: el uso de redes convolucionales; el funcionamiento físico de las cámaras; los algoritmos de triangulación; los algoritmos de *clusterización* como K medias; y el uso de filtros de procesado de datos como lo es el filtro de la mediana o el filtro de Kalman. También resultó indispensable la aplicación de conceptos aprendidos en varias materias a lo largo de la carrera para llevar a cabo un proyecto de investigación anual integrador que puso en perspectiva todo lo aprendido en los últimos cinco años por parte del equipo, dándole así un cierre al ciclo universitario.

6. Anexo

En este anexo se explicarán ciertos comandos utilizados, así como los archivos de configuración para ser copiados.

6.1. Video mp4 a partir de set de imágenes .png

Para la transformación de los fotogramas en vídeos utilizar el comando *ffpmeg*. Usar 15 como valor para el parámetro "-crf" (determina la calidad) y elegir una resolución de 640x480 píxeles para el video. Ejecutar el comando dentro del directorio en el cual se encuentran los frames de la captura que se intentan transformar en un vídeo.

```
> cd <path-to frames>
> ffmpeg -r 24 -f image2 -s 640x480 -i %d.png -vcodec libx264 -crf 15
-pix_fmt yuv420p <path-to-file>.mp4
```

6.2. Instalación del ambiente de *Anaconda* y ejecución de DeepLabCut

Para empezar descargar *Anaconda* desde [14] (64-Bit Graphical Installer). También se deben instalar los drivers de la placa de vídeo necesarios.

Luego en la terminal de comandos de *Windows* ejecutar:

```
> git clone https://github.com/AlexEMG/DeepLabCut.git
> cd <path-al-repositorio>\DeepLabCut\conda-environments
> conda env create -f DLC-CPU.yaml
> active DLC-GPU
```

Con esto habremos creado el ambiente de conda con todos los módulos y librerías necesarias para la ejecución.

Por último para ejecutar las funciones de DeepLabCut se puede abrir un interprete *ipython*:

```
> ipython
```

Luego, se siguieron paso por paso los comandos indicados en la guía de *DeepLabCut* (resulta importante leer la documentación oficial y el resto de este informe previo al comienzo de un entrenamiento o de un nuevo proyecto):

1. Dentro de ipython importar el módulo de deeplabcut:

```
> import deeplabcut
```

2. Crear un nuevo proyecto de deeplabcut:

```
> deeplabcut.create_new_project('project_name','experimenter',['path of video 1','path of video2',..])
```

3. Determinar el path al archivo de configuración de la red:

```
> config_path = 'yourdirectory/project_name/config.yaml'
```

4. Completar los parámetros deseados en el archivo de config.yaml y extraer los frames utilizados para el aprendizaje de la red:

```
> deeplabcut.extract_frames(config_path)
```

O para una selección manual:

```
> deeplabcut.extract_frames(config_path, 'manual')
```

5. Label frames/etiquetar los frames:

```
> deeplabcut.label_frames(config_path)
```

6. Check labels [OPTIONAL]:

```
> deeplabcut.check_labels(config_path)
```

7. Separar el conjunto en datos usados para entrenamiento y para testeo:

```
> deeplabcut.create_training_dataset(config_path)
```

8. Entrenar la red:

```
> deeplabcut.train_network(config_path)
```

9. Evaluar la red entrenada:

```
> deeplabcut.evaluate_network(config_path)
```

10. Análisis de videos:

```
> deeplabcut.analyze_videos(config_path, ['path of video 1','path of video2', ...])
```

11. Filtro de mediana sobre los puntos 2D:

```
> deeplabcut.filterpredictions(config_path,[`path_al_video'], shuffle=1)
```

12. Graficar los resultados (trajectories):

```
> deeplabcut.plot_trajectories(config_path, ['path of video 1','path of video2',..])
```

13. Crear videos etiquetados:

```
> deeplabcut.create_labeled_video(config_path, ['path of video 1','path of video2',..])
```

6.3. Descarga de los pesos de la red pre-entrenada MPII

Se descargaron los pesos con los siguientes comandos de Unix:

```
> curl -L -O
https://datasets.d2.mpi-inf.mpg.de/deepercut-models-tensorflow/mpii-single-resnet-101.data-00000-of-00001
> curl -L -O
https://datasets.d2.mpi-inf.mpg.de/deepercut-models-tensorflow/mpii-single-resnet-101.meta
> curl -L -O
https://datasets.d2.mpi-inf.mpg.de/deepercut-models-tensorflow/mpii-single-resnet-101.index
```

Luego, se movieron los pesos descargados a la carpeta correspondiente: "C:\Users\...\...\envs\ecog\Lib\sitepackages\deeplabcut\pose_estimation_tensorflow\models\pretrained" y se modificó el *config.yaml* como se muestra en [1]. Es decir, se configuró para que el framework tome los pesos indicados como pesos iniciales.

6.4. Configuración de *bodyparts* del archivo *yaml* de DeepLabCut

Configuración del archivo *yaml* de *DeepLabCut. S*e debe tener en cuenta que los comentarios marcados con "#" deben ser removidos previo a la ejecución para el correcto funcionamiento del *framework.*

```
# Head markers
- HeadTop      # Place the marker on the head tip.
- HeadFront    #      Place the marker at the center of the forehead.
- HeadSide     #      Place the marker on the either side of the head, slightly above the ear. When capturing multiple actors with similar proportions, place the marker on opposite side for clearer distinction.
# Torso Markers
- BackTop      #      Place the marker on the spine right below the neck. For best results, LUArmHigh-LShoulderBack-BackTop-RshoulderBack-RUArmHigh markers should align roughly in a straight line when the actor is in T-pose. Also, note that this marker must be placed at an elevation higher than the chest marker for the torso segment to be tracked properly.
- Chest #      Place the marker at the center of the sternum, about 5 cm below where the neck starts.
- BackLeft #   These two markers are located symmetrically on each side of the back, slightly below the lowest end of the shoulder blade (scapular bone).
- BackRight
# Waist Markers
# Note that the waist markers are the key markers in modeling the pelvis bone, which is the major segment governing the other subsequent skeleton segments. For best results, avoid placing the waist markers in a rectangle shape. When tracking multiple actors with similar proportions, introduce an offset to one of the WaistBack marker, or the WaistCMarker (included only with the 13 additional markerset), to create an asymmetrical arrangements.
- WaistLFront #      Placed the marker on the protruding bone located on the front of the pelvis (left/right anterior iliac spine bone). The prominence can be palpated from left and right side of the waist.
- WaistRFront
- WaistLBack #      Place the WaistLBack and WaistRBack markers above left/right hip; about 10 cm above the hip joint.
- WaistRBack
# Shoulder Markers
# When placing the shoulder markers, ask the actor to stand in the T-pose in order to place the markers on accurate locations. It could also be helpful to ask the actor to do few rounds of arm abduction and adduction to palpate for the shoulder axis. These markers determine the width of the shoulder and respective relationship with the upper arm segment.
- LShoulderBack      #      Place the marker on the shoulder joint (glenohumeral joint) on the back.
- RShoulderBack
```

- LShoulderTop # Place the marker at the top of each shoulder where you can palpate the protruding bone, which is usually located just before where the upper arm start when in T-pose. More specifically, the prominence is on the distal end of the clavicle bone (acrominoclavicular joint).
- RShoulderTop
Arm Markers
- LElbowOut # Upper Arm Place the marker on the lateral side of the elbow joint, slightly towards the upper arm. More specifically, on the lateral epicondyle of the distal end of the humerous. Ask the actor to flex and extend the arm few times to confirm that markers are placed along the elbow axis and maintain stationary throughout the range of motion cycles.
- RElbowOut # Upper Arm
- LUArmHigh # Upper Arm Place the marker on the back of the upper arm near the groove between the triceps. For best results, LUArmHigh, LShoulderBack, BackTop, RshoulderNack, and RUArmHigh should align roughly in a straight line when in T-pose.
- RUArmHigh # Upper Arm
Hand Markers
#For best results, place the hand markers so that the shape of the marker arrangement is asymmetrical itself and also unique to the shape on the other hand. Since the wrist marker placement is fixed along the wrist axis, offset the HandOut or the HandIn markers to create the unique arrangements. For more robust and simple tracking, hand rigid-bodies can be attached on the hand to replace the markers and guarantee the asymmetrical and non-deforming marker placement.
- LWristOut # Fore Arm / Hand Place the marker on the outside (lateral side) of the wrist axis. (distal end of the radius bone). Note that the wrist axis is not always located on the protruding bone but 1-2 cm more towards the hand.
- RWristOut # Fore Arm / Hand
- LWristIn # Fore Arm / Hand Place the marker on the inner prominence (medial side) of the wrist axis (distal end of the ulnar bone).
- RWristIn # Fore Arm / Hand
- LHandOut #Hand Place the marker slightly below the pinky knuckle. More specifically, between distal end of the fifth and fourth mertacarpal bones.
- RHandOut # Hand
Leg Markers
- LThighFront # Upper Leg Place the marker on the front side of the thigh, along the midline. For best results, slightly offset the height of the left and right markers to introduce the asymmetry.
- RThighFront # Upper Leg
- LKneeOut # Upper Leg Place the marker on the outter prominence of the knee joint; on the lateral epicondyle of the femur. Palpate the knee for the protruding bone from the upper leg, and place it right on the axis. Ask the actor to flex and extend the knee few times to confirm that the markers are placed along the axis and maintain stationary throughout the range of motion cycles.
- RKneeOut # Upper Leg
- LShin # Lower Leg Place the marker on the shin bone; on the midline of the lower leg. For best results, slightly offset the height of the left and right markers to introduce the asymmetry.
- RShin # Lower Leg
Foot Markers

- LAnkleOut # Lower Leg/Foot: Place the marker on the protruding bone outside of the ankle joint; on the lateral end of the malleolus bone.
- RAnkleOut # Lower Leg/Foot:
- LToeOut # Foot Place the marker just before the little toe joint; on the distal end of the fifth metatarsal bone.
- RToeOut # Foot
- LToeIn # Foot Place the marker just before the big toe joint; on the distal end of the first metatarsal bone.

7. Bibliografía y referencias

[1] Repositorio de *DeepLabCut*: https://github.com/DeepLabCut/DeepLabCut/.

[2] Repositorio del software de captura y calibración por Braulio Sespesde; https://github.com/bsespede/3D-pose

[3] Autodesk .FBX SDK: http://download.autodesk.com/us/fbx/FBX_SDK_Help/files/fbxsdkref/class_k_fbx_skeleton.html

[4] https://kapeli.com/cheat_sheets/Conda.docset/Contents/Resources/Documents/index

[5] *Baseline* de *Optitrack*: https://v21.wiki.optitrack.com/index.php?title=Baseline_(37)

[6] https://v21.wiki.optitrack.com/index.php?title=Properties:_Skeleton

[7] Human pose database: http://human-pose.mpi-inf.mpg.de/

[8] https://hamelot.io/visualization/using-ffmpeg-to-convert-a-set-of-images-into-a-video/

[9] Diapositivas de la proyección: https://www.csie.ntu.edu.tw/~cyy/courses/vfx/16spring/lectures/handouts/SFMedu_brief.pdf

[10] *Playlist* de fotogrametría: www.youtube.com/playlist?list=PLgnQpQtFTOGRsi5vzy9PiQpNWHjq-bKN1

[11] Software de visualización de puntos 3D: https://www.ovito.org/

[12] *Open CV* triangulation: https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

[13] WSL Windows 10: https://docs.microsoft.com/en-us/windows/wsl/install-win10

[14] Anaconda: https://www.anaconda.com/products/individual

[15] Redes Convolucionales: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

[16] Gráficos en tamaño original: https://drive.google.com/drive/folders/1bX_HlJDN0msG4yDjzA3R3AdubMfh9FKU?usp=sharing

[17] Richard Hartley and Andrew Zisserman. Multiple view geometry in computer vision. Cambridge university press, 2003.

[18] Documentación de la librería PyMVG: https://pymvg.readthedocs.io/en/latest/pymvg_file_format.html

[19] Repositorio de Structure From Motion para triangulación: https://github.com/opencv/opencv_contrib/tree/master/modules/sfm

[20] Kalman Filter: https://en.wikipedia.org/wiki/Kalman_filter

[21] Repositorio del proyecto: https://bitbucket.org/itba/pf-markless3d/src/master/