ngeniería Informática Escuela Politécnica Superior de Jaén

UNIVERSIDAD DE JAÉN

Escuela Politécnica Superior de Jaén

Trabajo Fin de Grado

ANÁLISIS BIOMECÁNICO DE SALTOS DE LONGITUD A PARTIR DE SECUENCIAS DE VIDEO

Alumno: Daniel Beltrán Cárdenas

Tutor: Prof. D. Manuel José Lucena López
Prof. D. José Manuel Fuertes García

Dpto: Informática

Septiembre, 2020

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don Manuel José Lucena López y Don José Manuel Fuertes García , tutor del Proyecto Fin de Carrera titulado: Análisis biomecánico de saltos de longitud a partir de secuencias de video, que presenta Daniel Beltrán Cárdenas, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2020

El alumno:

Daniel Beltrán Cárdenas

Los tutores:

Firmado digitalmente por LUCENA LOPEZ, MANUEL JOSE.
Fecha: 09.09.2020 - 20:58:38.

Firmado digitalmente por FUERTES GARCIA JOSE MANUEL - 26478036E
Fecha: 2020.09.09 20:23:25 +02'00'

Manuel josé Lucena López y Jose Manuel Fuertes García

Índice

1. INTRODUCCIÓN, MOTIVACIÓN Y OBJETIVOS DEL PROYECTO....6

1.1 Introducción....6

1.2 Motivación....6

1.3 Objetivos del proyecto....7

1.4 Metodología desarrollada....8

1.5 Estructura de la memoria....10

2. VISIÓN POR COMPUTADOR....12

3. DETECCIÓN DE LA POSE HUMANA EN IMÁGENES Y SECUENCIAS DE VÍDEO....16

3.1 Introducción....16

3.2 Movimiento del atleta....17

3.3 Elección de una red neuronal válida....17

3.4 Cómo funciona OpenPose....19

3.5 OpenPose PythonAPI....26

3.6 Instalación de OpenPose....27

3.6.1 Prerrequisitos....27

3.6.2 Instalación....28

4. PROPUESTA Y DESARROLLO DE LA SOLUCIÓN ADOPTADA....31

4.1 Planificación....32

4.2 Detección del atleta....32

4.2.1 Análisis....32

4.2.2 Restricciones materiales....33

4.2.3 Restricciones de la cámara....33

4.2.4 Restricciones de posición....34
4.2.5 Diseño....35
4.3 Batida del atleta y su detección....36
4.3.1 Análisis....36
4.3.2 Diseño....37
4.4 Calibración....38
4.4.1 Análisis....38
4.4.2 Diseño....40
4.5 Cálculo de la altura del salto....42
4.5.1 Análisis....42
4.5.2 Diseño....42
4.6 Cálculo del ángulo de batida....43
4.6.1 Análisis....43
4.6.2 Diseño....44
4. 7 Cálculo de la velocidades....45
4.7.1 Análisis....45
4.7.2 Diseño....45
4. 8 Optimización del análisis....46
4.8.1 Análisis....46
4.8.2 Diseño....46
4.8.3 Resultado de la optimización....47
4. 11 Resultados....48
5. DISEÑO E IMPLEMENTACIÓN....49
5.1 Diseño....49
5.1.1 *Sketch.*....50

5.1.2 *WireFrame*.....50
5.1.3 *Mockup*.....51
5.1.4 Interfaz.....51
5.1.5 Diagrama de clases.....54
5.2 Implementación.....55
5.2.1 Clase Interfaz.....55
5.2.2 Clase controlador.....57
5.3 Consideraciones futuras.....58
6. CONCLUSIONES.....59
7. MANUAL DE USUARIO.....60
7.1 Ejecutar aplicación y cargar un vídeo.....60
7.2 Identificar la distancia de referencia que se utilizará y calibración de vídeo.....62
7.3 Ejecución del análisis.....64
7.4 Guardar datos del análisis.....65
Bibliografía.....66

1. INTRODUCCIÓN, MOTIVACIÓN Y OBJETIVOS DEL PROYECTO.

1.1 Introducción.

Este primer capítulo, se centrará en introducir los aspectos más básicos del proyecto, será justificada la elección del mismo al igual que la necesidad de una aplicación de este tipo. Para finalizar, se mencionarán las metas que se han perseguido tanto a nivel personal como profesional.

1.2 Motivación.

Hoy en día, cada vez es más frecuente el uso de tecnologías en el deporte, ya sea para la mejora de la técnica de los atletas, ayudas para la valoración física, estudios biomecánicos, etc. En este caso en concreto, el salto del longitud, al tratarse de un estudio biomecánico, existen centros especializados para esta función. Pero, aunque son muy útiles, pocos atletas son los que tienen acceso a esta tecnología.

Es necesario explicar en que consiste un estudio biomecánico para poder justificar la elección de este proyecto.

Un estudio biomecánico es la grabación desde diferentes perspectivas del atleta mientras realiza determinado ejercicio, en este caso, el salto. Tras estudiar todos los vídeos, se obtendrán datos como una correcta o incorrecta posición del tronco, la amplitud de zancada, la altura del salto… Que ayudarán al atleta a mejorar el rendimiento y hacer más eficiente y eficaz el esfuerzo.

Dicho anteriormente, muchos atletas no tienen acceso a este recurso y que, actualmente existen tecnologías que nos permiten procesar e interpretar imágenes con gran

precisión, se puede considerar un proyecto novedoso e interesante de cara a una mejora del rendimiento deportivo y del futuro laboral.

1.3 Objetivos del proyecto.

El objetivo principal del proyecto se basa en la realización de una aplicación capaz de, teniendo como entrada un vídeo de un salto de longitud, devolver, como resultados, datos que sean útiles para el atleta y su entrenador.

Dado que, el objetivo del proyecto es el procesamiento de imágenes y la detección de un atleta realizando un salto de longitud, se necesitarán tecnologías. Una de ellas será el empleo de una red neuronal capaz de detectar el movimiento de las mismas (*traking*). Para las funciones más complejas, relacionadas con el tratamiento de imágenes, se ha utilizado una librería de visión por computador conocida como *OpenCV.* Y, como este proyecto deberá tener su propia interfaz, se usará la librería *Tkinter* de *Python*.

Los objetivos a realizar en este proyecto son los siguientes:

- Uso de herramientas de procesamiento de imágenes y de redes neuronales.
- Estudio de la forma en la que se deben grabar los vídeos que procesará la aplicación, así como sus restricciones y/o recomendaciones.
- Detección del atleta que realiza el salto y posición de sus articulaciones y extremidades en el vídeo.
- Cálculo de datos trigonométricos, distancias y la obtención de ángulos mediante el procesado de las imágenes.
- Cálculo de datos biomecánicos, como pueden ser velocidades o variaciones de movimiento de las extremidades.

A nivel personal, se ha perseguido adquirir conocimientos de programación sobre los tecnologías mencionadas anteriormente. Durante la realización del proyecto, se han obtenido conocimientos de diversa índole referentes a la programación de las tecnologías. Abarcando,

no solo los necesarios para el proyecto objeto de estudio, si no también todos aquellos para la utilización de las técnicas desarrolladas, todo ello fruto del esfuerzo personal y partiendo desde cero en algunos aspectos como la utilización de una red neuronal.

También se ha buscado satisfacer una curiosidad personal acerca del mundo de la *visión por computador o visión artificial:* se ha partido de con un conocimiento previo gracias a algunas asignaturas impartidas en el transcurso del Grado y mediante el estudio de librerías, como *OpenCV*, habiendo conseguido adquirir una visión general de este campo.

1.4 Metodología desarrollada.

Siguiendo los patrones y las tendencias de diseño más modernas hasta la fecha, el prototipo de aplicación que se ha realizado, es un modelo de desarrollo incremental.

Desde el comienzo se estimaron las principales funciones en la aplicación:

- Capacidad de detectar la pose humana mediante la ayuda de una red neuronal.
- Capacidad de sacar datos relevantes para el atleta tras el análisis.
- Posesión un método de calibración para la cámara utilizada.

Estas funcionalidades se dividieron en bloques de trabajo. Los bloques se han ido implementado de forma independiente para finalmente integrarlos y formar la aplicación. En la figura 1, se puede observar un esquema de la metodología utilizada, donde cada incremento quedará reflejado en la aplicación como una funcionalidad a la que podemos acceder desde el menú principal.

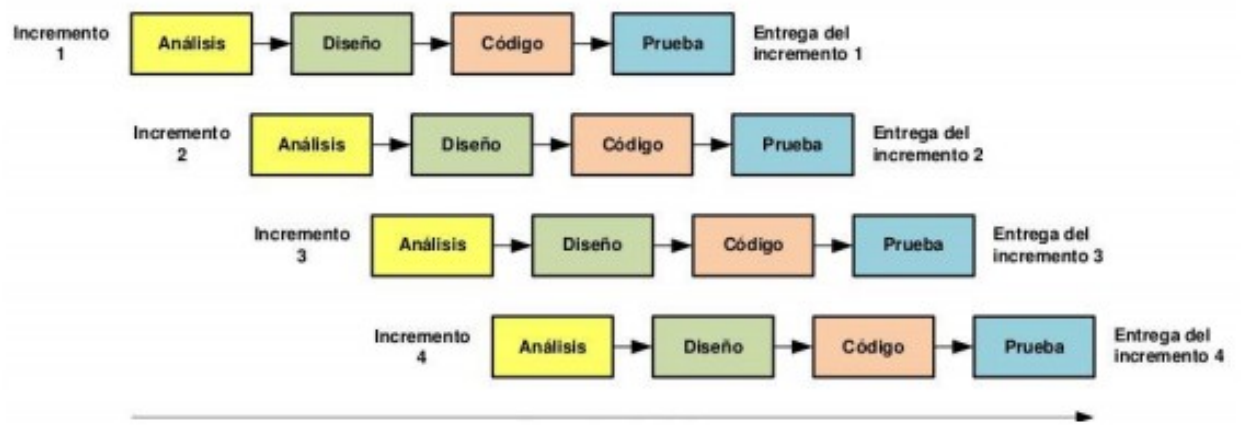

Figura 1: Fases del modelo incremental

Dicho esto, los problemas de partida que se encuentran son:

1. Encontrar un método de grabación que se usará a la hora de realizar un vídeo de un saltador.
2. Hallar una red neuronal funcional que se adapte al problema que se plantea.
3. La posterior instalación y configuración de las herramientas y bibliotecas a utilizar.
4. Implementación de las funciones que anteriormente mencionadas y desarrollo de las mismas en incrementos.

Como se puede comprobar, en las tres primeras fases no se hace ningún desarrollo de la aplicación, ya que, inicialmente se debe realizar un trabajo de investigación y revisión de documentación técnica actualizada sobre los aspectos relacionados con el problema, como son, la realización de un salto de longitud, los elementos que se dispone para la grabación del citado salto, las redes neuronales que pueden ser útiles para este problema y la configuración e instalación de las mismas.

Una vez realizados los trabajos anteriormente detallados, se comenzará el desarrollo desde el primer incremento hasta el último, debiendo cubrir todas y cada una de las

funcionalidades mencionadas anteriormente. En la figura 2 se muestra un *diagrama de Gantt* con las tareas seguidas.

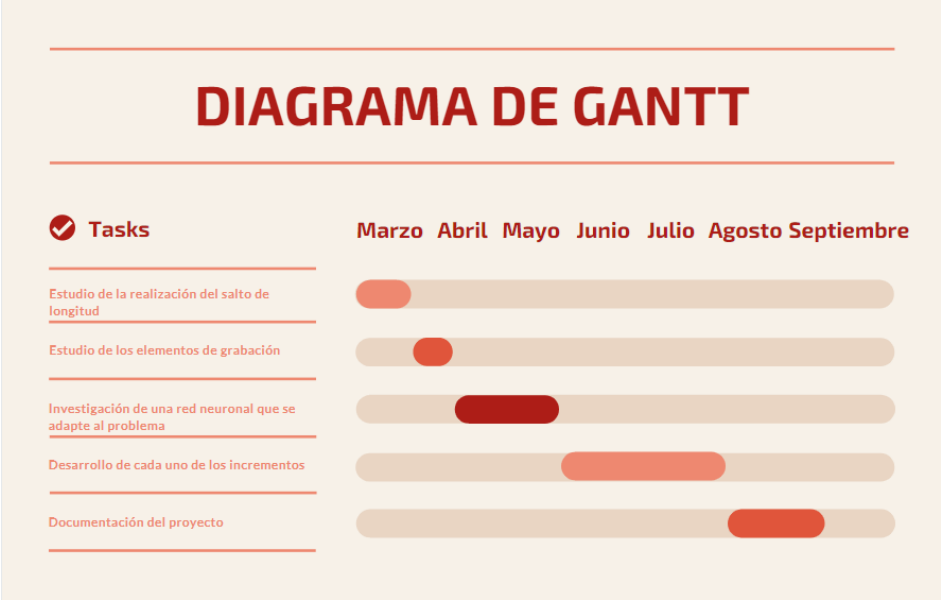

Figura 2: Diagrama de Gantt con las fases del proyecto

1.5 Estructura de la memoria.

Dada la distribución en capítulos de la documentación utilizada, se expondrá una breve introducción de cada uno de ellos.

- En el primer capítulo, se dará una visión general del proyecto, introducción, motivación del mismo y planificación.
- En el segundo capítulo, se presentará la visión por computador y una breve introducción a la estimación de pose.
- En el tercer capítulo, se describirá la red neuronal empleada, sus funciones y la elección de esta sobre otras.

- En el cuarto capítulo, se explicarán las decisiones tomadas para la solución adoptada, así como las restricciones y uso de la cámara a la hora de grabar los vídeos para su posterior uso en la aplicación.
- En el quinto capítulo, se explicarán las decisiones de diseño e implementación seleccionadas.
- En el sexto capítulo, se hablará sobre las conclusiones objetivas y personales.
- En el séptimo capítulo, se expondrá el manual de usuario.

El final del documento incluye un anexo con la bibliografía utilizada.

2. VISIÓN POR COMPUTADOR.

La visión artificial, también conocida como visión por computador, es una disciplina científica que incluye métodos para adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información numérica o simbólica, para que puedan ser tratados por un ordenador.

Tal y como los humanos usan sus ojos y cerebros para comprender el mundo que les rodea, la visión artificial trata de producir el mismo efecto para que los ordenadores puedan percibir y comprender una imagen o secuencia de imágenes y actuar según convenga en una determinada situación.

La visión por computador es una tarea de proceso de información con bien definidas etapas de entrada y salida. La entrada consiste en una matriz de valores, que representan las proyecciones de una escena tridimensional registrada por una cámara o dispositivo similar de toma de imagen. Varias matrices de entrada proporcionan información en varias bandas espectrales (color) o desde múltiples puntos de vista. La salida deseada es una descripción concisa de la escena representada en la imagen, la naturaleza exacta de la cual depende según los objetivos y expectativas del observador. Generalmente implica una descripción de los objetos y sus interrelaciones, pero también puede incluir información tal como las estructuras tridimensionales de superficies, sus características físicas (forma, textura, color, materia y la ubicación de sombreados y fuentes de luz). [7]

Entre los ejemplos de aplicaciones de la visión por computador figuran los sistemas para:

- Inspección automática: en aplicaciones de fabricación…
- Asistencia a los seres humanos en tareas de identificación: un sistema de identificación de especies...

- Detección de eventos: para la vigilancia visual o el conteo de personas...
- Modelado de objetos o entornos: análisis de imágenes médicas o modelado topográfico...
- Navegación: mediante un vehículo autónomo o un robot móvil...
- Organizar la información: para indexar bases de datos de imágenes o secuencias de vídeo...

De todas sus múltiples aplicaciones, la que se utilizará para el desarrollo del proyecto será la detección de las posiciones de las articulaciones (pose).

La estimación de las poses humanas se refiere al proceso de detectar poses en una imagen, esto también se conoce como la localización de las articulaciones humanas. Esencialmente implica la predicción de las posiciones de las articulaciones de una persona en una imagen o vídeo. También, es importante señalar, que la estimación de poses tiene varias subtareas, como son: la estimación de una pose en concreto, la estimación de poses en una imagen con muchas personas, la estimación de poses en lugares concurridos y/o la estimación de poses en vídeos.

También, hay una distinción clave que debe hacerse entre la estimación de la postura 2D y 3D. La estimación de la pose 2D, simplemente estima la ubicación de los puntos clave en el espacio 2D en relación con una imagen o un cuadro de vídeo. El modelo estima una coordenada X e Y para cada punto clave. La estimación de la pose 3D funciona para transformar un objeto de una imagen 2D en un objeto 3D añadiendo una coordenada de profundidad, Z. A continuación, el modelo de con un enfoque 2D, se muestra en la figura 3.

Figura 3: Modelo con un enfoque 2D

La estimación de la pose difiere de otras tareas comunes de visión por computador en algunos aspectos importantes. La detección de objetos consiste en un cuadro delimitador que abarca el objeto, a diferencia de la estimación de la posición, que también predice la ubicación precisa de los puntos clave asociados al objeto.

Este problema resulta difícil, ya que la detección de una pose humana depende de muchas cuestiones como pueden ser articulaciones grandes, articulaciones pequeñas y apenas visibles, oclusiones, ropa y cambios de iluminación; a diferencia de la detección de objetos en los que no influyen tantas variables.

Actualmente, esta rama de la visión por computador está en constante crecimiento, y cada vez son más los desarrolladores que se atreven a crear sus propios modelos o mejorar los ya existentes.

A lo largo de los últimos años, han ido desarrollándose técnicas para la detección de la pose humana, distinguiendo entre detección de pose humana en 2D o 3D. A continuación, se citan algunas de ellas [8]:

- Estimación de pose 3D:
    - Mehta, D., Sotnychenko, O., Mueller, F., Xu, W., Sridhar, S., Pons-Moll, G., Theobalt, C. (3DV 2018). Single-Shot Multi-Person 3D Pose Estimation From Monocular RGB [9]. Saarland Informatics Campus (Stanford University). Proponen

un método para la pose 3D que permite la inferencia de poses de cuerpo entero bajo fuertes oclusiones parciales.

- Pavllo, D., Feichtenhofer, C., Grangier, D., & Auli, M (ArXiv 2018). 3D human pose estimation in video with temporal convolutions and semi-supervised training [10]. Facebook AI research y Google Brain. Demuestran que las poses 3D pueden ser estimadas a partir de de puntos claves 2D con redes con un entrenamiento semi-supervisado.
- Varol, G., Ceylan, D., Russell, B., Yang, J., Yumer, E., Laptev, I., & Schmid, C. (ECCV 2018). BodyNet: Volumetric Inference of 3D Human Body Shapes [11]. Adobe Research (USA) y Inrie (Francia.) Proponen una red capaz de predecir el volumen del cuerpo junto con la pose 3D.

- Estimación de pose 2D:
  - Ke Sun, Bin Xiao, Dong Liu, Jingdong Wang (CVPR 2019). Deep High-Resolution Representation Learning for Human Pose Estimation [12]. University of Electronic Science and Technology of China. Están interesados en la detección de pose con un enfoque de alta resolución, para aumentar la precisión resultante.
  - Osokin, D. (ArXiv 2018). Real-time 2D Multi-Person Pose Estimation on CPU: Lightweight OpenPose [13]. Intel. Este trabajo adaptan la arquitectura OpenPose para para usarla en dispositivos y captar en tiempo real la pose humana
  - Bin, Xiao, Haiping Wu, Yichen Wei (ECCV 2018). Simple Baselines for Human Pose Estimation and Tracking [14]. University of Electronic Science and Technology of China. Este trabajo comprueba como de buenos son métodos más simples para la detección de la pose, sin usar redes con altos niveles de profundidad.

3. DETECCIÓN DE LA POSE HUMANA EN IMÁGENES Y SECUENCIAS DE VÍDEO.

3.1 Introducción.

Para el problema que se plantea, analizar un atleta que realiza un salto de longitud, primeramente se debe entender como es la realización de este salto. La definición según *wikipedia sobre el salto de longitud* [15] dice lo siguiente: "*El salto de longitud o salto largo es una prueba actual del atletismo que consiste en recorrer la máxima distancia posible en el plano horizontal a partir de un salto tras una carrera*"; El salto de longitud consta de 4 fases [2], las cuales podemos observar en la figura 4:

1. Fase de carrera: El atleta empieza a acelerar hasta encontrar su velocidad máxima aproximándose al punto de batida.
2. Fase de impulso o batida: En este momento el atleta realiza el salto vertical intentando tener la menor pérdida de velocidad posible.
3. Fase de vuelo: Donde el atleta se mantiene en el aire durante el salto.
4. Fase de caída: Tras el vuelo, el atleta toca el área de caída con los pies e intenta que su cuerpo pase por delante de la marca que dejan sus pies.

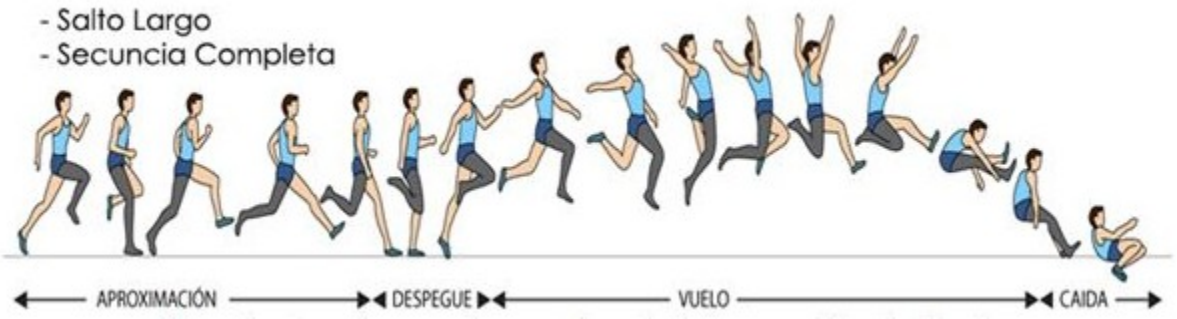

Figura 4: Fases del salto de longitud

Por tanto, el salto que se realiza es un movimiento rectilíneo uniformemente acelerado y luego se transforma en un lanzamiento parabólico, es decir, el atleta se mueve en una trayectoria recta.

3.2 Movimiento del atleta

Como se ha expuesto anteriormente, el atleta se mueve en una trayectoria recta. Dado un plano en el eje X e Y y otro con los ejes X, Y y Z, las trayectorias serían como en las siguientes figuras 5 y 6:

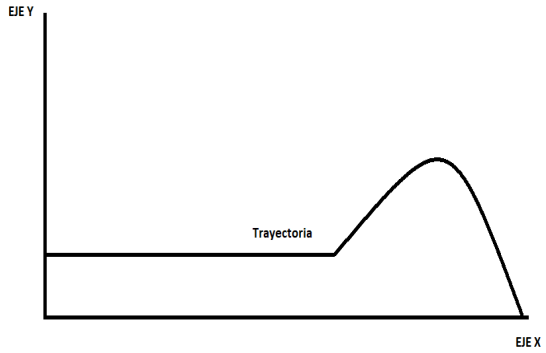

Figura 2: Vista en dos dimensiones (X e Y) de la trayectoria del atleta

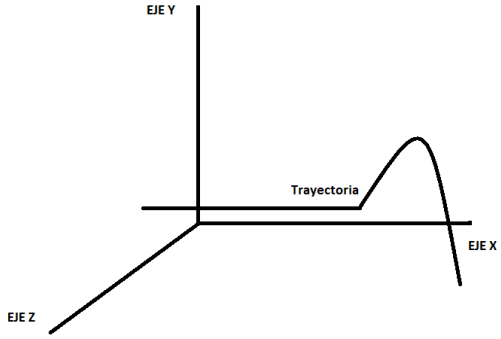

Figura 6: Vista en 3 dimensiones (X , Y y Z) de la trayectoria del atleta

Se puede observar que el eje Z, o en este caso la profundidad, no varía desde la perspectiva del observador estando este paralelo al punto de batida del salto, por tanto, el atleta realmente se está moviendo solo en 2 dimensiones (2D).

3.3 Elección de una red neuronal válida.

Una vez explicado como funciona el salto de longitud, su movimiento dentro del plano y que enfoque puede ser válido a la hora de seleccionar un red, se procederá a la elección de esta.

Para el problema, se necesitan las coordenadas de determinadas articulaciones (cadera, rodilla y pie) para realizar ciertos cálculos biomecánicos, por tanto, una red basada en el enfoque 2D es suficiente, ya que no se requiere de un modelo 3D.

Como se expuso anteriormente, hay una gran cantidad de redes basadas en este enfoque (2D), por lo que, la manera que utilizaremos para discernir entre ellas será:

- El lenguaje de programación de nuestro proyecto, *Python*.
- No tener que instalar paquetes de librerías adicionales más allá de la red neuronal.
- Que el enfoque de la red sea en 2D.

Teniendo en cuenta estas restricciones, la librería que nos ofrece *OpenPose* [3] cumple con las características del proyecto; esta librería, aunque tiene una breve instalación que se verá más adelante, tiene una cantidad de funciones bastante amplia, desde la detección de múltiples personas en tiempo real, hasta la detección de poses en una sola imagen. Aunque, lo más interesante de esta librería son 3 características en concreto:

- La API de *Python*, la cual es sencilla ya que tan solo será necesario mandarle a la red la imagen a procesar, y esta devolverá los datos que indiquemos.
- Su modelo *BODY* 25 [5], el cual consiste en detectar la pose humana en 25 puntos, incluyendo la cadera y todas las partes del pie (inicio del pie y fin del pie) a diferencia de otros modelos, necesarios a la hora de detectar una pose en concreto, este modelo se podrá ver mas adelante en la figura 17.
- Su salida tras procesar una imagen, la cual devuelve de forma sencilla la posición de cada una de las articulaciones que detecta.

Además, los autores [4] de este proyecto continúan actualizando constantemente esta librería, por lo que se siguen implementando mejoras en cada una de sus funcionalidades. Esto puede ser muy útil aunque será necesario ir comprobando sus actualizaciones para futuros incrementos de nuestro prototipo.

3.4 Cómo funciona OpenPose.

Veamos su estructura, la siguiente figura 7:

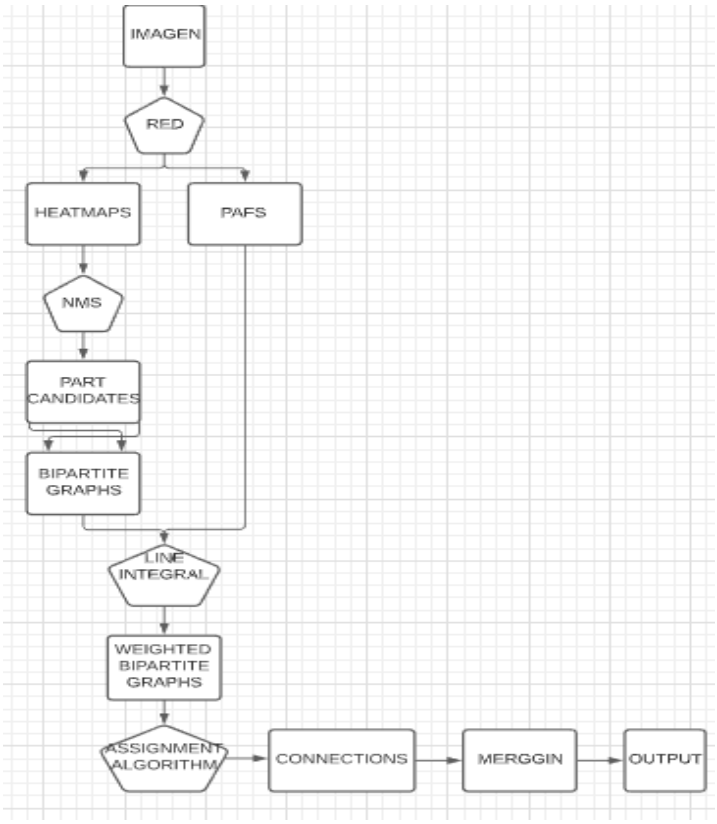

Figura 7: Diagrama de flujo de OpenPose

OpenPose recibe como entrada una imagen RGB. Como la aplicación deberá recibir como entrada un video, se deberá diseccionar el vídeo en *frames* y tratarlos individualmente.

Como vemos en el gráfico, la red funciona de la siguiente manera, teniendo como entrada un imagen RGB, la red pasará a la citada imagen por 2 filtros: *heatmaps* y PAFS.

Los ***heatmaps* o mapas de calor** son matrices que almacenan la confianza que la red tiene en que una parte del modelo (esqueleto) realmente esté ahí; hay una matriz por cada parte del esqueleto del modelo, en este caso el BODY 25, el cual consta de 25 partes más 1 matriz extra para el fondo.

Por tanto, teniendo una imagen de un saltador en la fase de vuelo, figura 8.

Figura 8: Atleta en la fase de vuelo

A continuación observamos la matriz encargada de almacenar la confianza del muslo de la pierna izquierda. La matriz que se generaría para el ejemplo sería de este modo, figura 9.

Figura 9: Recreació de un heatmap

Los píxeles que contengan parte del muslo del atleta o se aproximen a él tendrán mayor valor que los que estén en los alrededores o se alejen del mismo.

Los **PAFS (*part affinity fields*)** son matrices que se encargan de dar información sobre la orientación y posición de las partes, estas vienen en parejas; para cada parte existe un PAF en la dirección “X” y un PAF en la dirección “Y”. Hay 50 PAF asociados a cada una de las parejas. Como se puede ver en la figura 10, tenemos 2 PAFS diferentes y cada uno muestra valores opuestos.

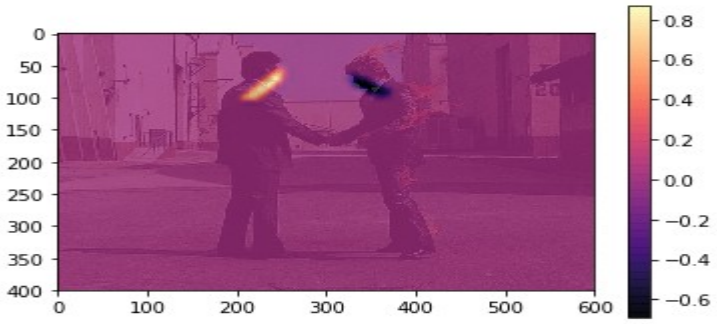

Figura 10: Se puede ver como las muestra de PAF es diferente para los sentidos opuestos

El siguiente paso, es aplicar un **algoritmo de *Non-Max suppression* (NMS)** ya que, aunque los heatmaps aportan información de confianza, es necesario transformar esa confianza en certeza.

Se debe de extraer la ubicación de las partes de un heatmap, o dicho de otro modo, extraer los máximos locales de este.

Se aplicará un **algoritmo de supresión no máxima (NMS)** para obtener esos picos, en la figura 11 se observa procedimiento a seguir.

1. Comienza en el primer píxel del *heatmap*.
2. Rodea el píxel con una ventana de lado 5 píxeles y encuentra el valor máximo en esa área.
3. Sustituye el valor del píxel central por ese máximo.
4. Desliza la ventana un píxel y repite estos pasos después de que se haya cubierto todo el *heatmap*.
5. Compara el resultado con el *heatmap* original. Los píxeles que permanecen con el mismo valor son los picos que buscamos. Suprime los otros píxeles poniéndolos con un valor de 0.

Algorithm 1 Non-Max Suppression

```
1:  procedure NMS(B,c)
2:      B_nms ← ∅
3:      for b_i ∈ B do
4:          discard ← False
5:          for b_j ∈ B do
6:              if same(b_i, b_j) > λ_nms then
7:                  if score(c, b_j) > score(c, b_i) then
8:                      discard ← True
9:          if not discard then
10:             B_nms ← B_nms ∪ b_i
11:     return B_nms
```

Figura 11: Algoritmo de supresión no máxima

Una vez aplicado el algoritmo y encontrado los candidatos a ser una parte del cuerpo, es necesario conectarlos formando pares de partes. Aquí entra la **teoría de grafos**. Para una imagen dada, hemos encontrado un conjunto de candidatos a articulación de cuello y un conjunto de candidatos a articulación de hombro derecho. Para cada cuello hay una posible asociación (candidato a conexión) con cada uno de los hombros derecho. Por lo que, se obtiene un **grafo bipartito completo** (figura 12), donde los vértices son los candidatos a partes y las aristas las candidatas a ser una conexión entre estas.

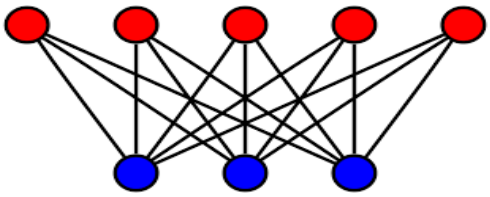

Figura 12: Grafo bipartito completo

Encontrar la mejor coincidencia entre los vértices de un grafo bipartito es un problema conocido en la teoría de grafos como el problema de la asignación [16], para resolverlo, cada arista deberá tener un peso.

Aquí es donde los **PAF** entran en el flujo del proceso. Se procede a calcular la línea integral a lo largo del segmento que conecta cada par de candidatos a la parte, sobre los **PAF** correspondientes (X e Y) para ese par. Una línea integral mide el efecto de un campo dado (en este caso, los **PAFs**) a lo largo de una curva dada (en este caso, las posibles conexiones entre las partes candidatas), es decir, lo que estamos haciendo realmente es añadirle pesos, como en la figura 13, a las conexiones entre dos partes para resolver el **problema de la asignación**.

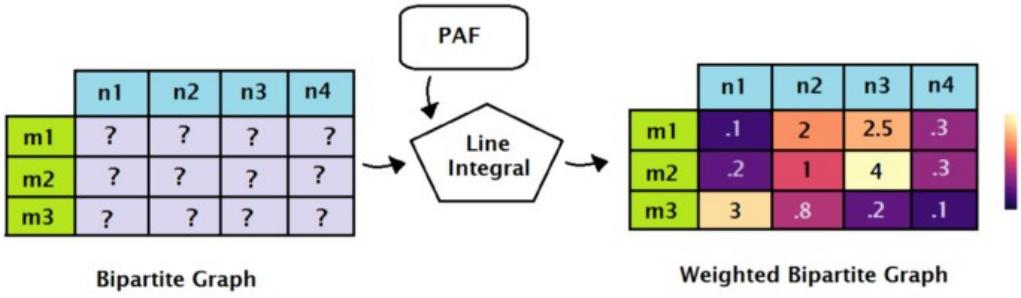

Figura 13: Gracias a los PAFS podemos añadir pesos a las conexiones

El **grafo bipartito** ponderado, muestra todas las posibles conexiones entre candidatos de dos partes teniendo una puntuación para cada conexión La misión ahora, es encontrar las conexiones que maximicen la puntuación total, resolviendo el problema de la siguiente manera.

1. Ordenar cada posible conexión por su puntuación.
2. La conexión con la puntuación más alta es, de hecho, una conexión final.
3. Pasar a la siguiente conexión posible. Si ninguna parte de esta conexión ha sido asignada a una conexión final antes, esta es una conexión final.
4. Repetir el paso 3 hasta que se terminen las conexiones.

Igualmente, en la figura 14 se puede apreciar de forma visual.

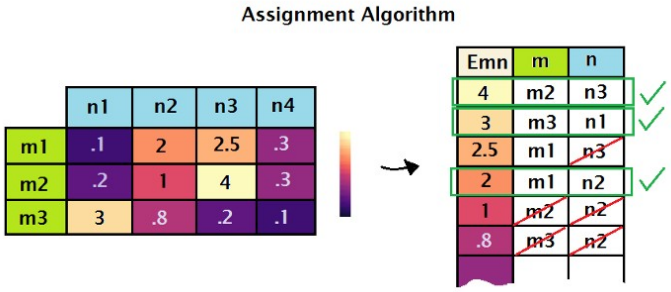

Figura 14: Algoritmo de asignación

El paso final, es transformar estas conexiones detectadas en los esqueletos finales. Se comenzará con una suposición ingenua: al principio, cada conexión pertenece a un humano diferente; De esta manera, existe el mismo número de humanos que conexiones que se han detectado.

Como podemos ver en la figura 15, se deja que los humanos sean una colección de conjuntos {H1, H2 …., Hk}. Cada uno de estos conjuntos contiene, al principio, dos partes. Una parte será descrita como una tupla de un índice, una coordenada en dirección "X" y una coordenada en dirección "Y".

$$\text{Humans} = \{H_1, H_2, \ldots, H_k\}$$
`where`
$$k := \text{number of final connections}$$
$$H_i = \{(m_{idx}, m_x, m_y), (n_{idx}, n_x, n_y)\}$$

Figura 15: Conjunto de humanos inicial

Aquí viene la fusión: si los humanos H1 y H2 comparten un índice de parte con las mismas coordenadas (están compartiendo la misma parte) entonces H1 y H2 son, por lo tanto, los mismos humanos. Así que ambos conjuntos se fusionan en H1 eliminando el H2. Un algoritmo sencillo como se puede apreciar en la figura 16.

```
if   H1 ∩ H2 ≠ ∅
then
  H1 = H1 ∪ H2
  delete(H2)
```

Figura 16: Algoritmo de fusión

Se continuará haciendo esto por cada pareja de humanos hasta que ninguna de ellas comparta una parte.

Finalmente, nuestra **salida** es una colección de conjuntos humanos, donde cada humano es un conjunto de partes, donde cada parte contiene su índice, sus coordenadas relativas y su puntuación.

La forma de la **salida** para la detección de un humano la siguiente tabla 1:

Tabla 1: Salida que devuleve OpenPose

Eje x	Eje y	Score
120	204	0.52
121	211	0.96
114	211	0.88
108	223	0.88
106	236	0.72

Donde los ejes X e Y son las coordenadas del pixel y el *Score* es la confianza de la red de que esa articulación se encuentra donde devuelve que está.

Los puntos del modelo BODY 25 (figura 17) son los siguiente:

0. Nariz.
1. Cuello.
2. Hombro derecho.
3. Codo derecho.

4. Muñeca derechas.
5. Hombro izquierdo.
6. Codo izquierdo.
7. Muñeca izquierda.
8. Cadera central.
9. Cadera derecha.
10. Rodilla derecha.
11. Tobillo derecho.
12. Cadera izquierda.
13. Rodilla izquierda.
14. Tobillo izquierda.
15. Ojo derecho.
16. Ojo izquierdo.
17. Oreja derechas.
18. Oreja izquierda.
19. Pulgar del pie izquierdo.
20. Meñique del pie izquierdo.
21. Tacón izquierdo.
22. Pulgar del pie derecho.
23. Meñique del pie derecho.
24. Tacón derecho.

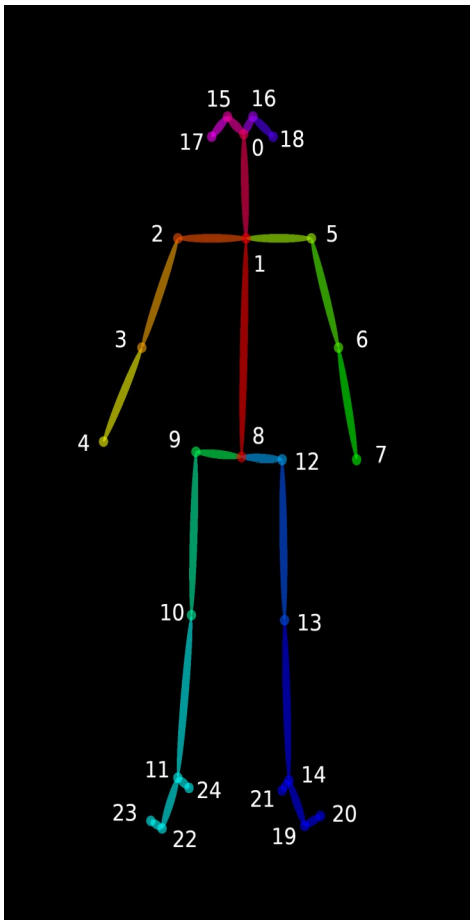

Figura 17: Modelo de BODY 25

3.5 OpenPose PythonAPI.

Respecto al uso de esta librería en *OpenPose* tan solo deberemos usar la clase *pyopenpose*, una vez hecho esto, debemos crear un *datum,*

```
datum = op.Datum()
```
→ (hemos importado *pyopenpose* como op)

leer una imagen con `imageToProcess = cv2.imread(imagePath)` *OpenCV*,

mandar la imagen a la red `datum.cvInputData = imageToProcess`

y reemplazar los valores de *datum* (si los tuviera) `opWrapper.emplaceAndPop([datum])` .

Finalmente, para acceder a la matriz con los valores de las coordenadas de las articulaciones del modelo tan solo debemos acceder a `datum.poseKeypoints`.

Explicado de una forma más simple, tan solo se debe enviarle la imagen a procesar a la red y esta nos devolverá la matriz de los valores de las coordenadas con las articulaciones del modelo.

3.6 Instalación de OpenPose.

Aunque solo necesitamos la librería de *OpenPose* para su uso, si que es necesario una breve y sencilla instalación de la red en la computadora donde se vaya a ejecutar. Para ello, uno de los prerrequisitos principales es: que el ordenador tenga un mínimo de 8GB RAM libre si usamos el modo de usar solo la CPU o nuestro PC tenga una grafica dedicada Nvidia o AMD.

Para el proyecto, se ha usado la primera opción.

Para la instalación, aunque el proceso es casi el mismo, hay pequeñas diferencias en el uso del modo de, solo CPU o, modo de usar una gráfica, por lo que se irán comentando las diferencias. Además, para este proyecto, *OpenPose* se ha instalado en Windows 10 aunque existe una versión para Ubuntu/Linux.

3.6.1 Prerrequisitos

- Es necesario tener instalado en el ordenador la herramienta Cmake GUI [17] la cual es posible descargar en su última versión desde su sitio web.

- Igualmente se debe tener visual *studio comunity* [18] o *enterprise* [19] instalado con todas las banderas relacionadas con C++ habilitadas al seleccionar los componentes a instalar.
- Requisitos previos de la versión de la GPU de Nvidia:
  - Instale CUDA 8.0/10.0 después de que se instale Visual Studio 2015/2017 para asegurarse de que la instalación de CUDA generará todos los archivos necesarios para VS. Si CUDA ya estaba instalado, vuelva a instalarlo.
- Requisitos previos de la versión de la GPU de AMD:
  - Descargar los controladores oficiales de AMD para Windows desde AMD – Windows.
- El paquete *libviennacl* viene empaquetado dentro de *OpenPose* para Windows (es decir, no se requiere ninguna otra acción).
- Además, es necesario comprobar que la gráfica este testeada y verificada para su uso en *OpenPose* en las cuestiones de su repositorio GIT.

3.6.2 Instalación

La instrucciones de instalación de *OpenPose* vienen en su propio repositorio en el apartado de instalación [6] por lo que, es recomendable comprobarlo ya que, pueden haber realizado actualizaciones que, como dijimos antes, esta librería sigue en constante actualización.

Ahora se muestra como es el proceso de instalación para Windows 10.

1. Ejecutar *CMake* GUI y seleccionar el directorio *OpenPose* como directorio de origen del proyecto, y un subdirectorio inexistente o vacío (por ejemplo, *build*) donde se generarán los archivos *Makefile* (Ubuntu) o la solución *Visual Studio* (Windows), como en la figura 18. Si el *build* no existe, le preguntará si lo crea. Pulse *Sí*.

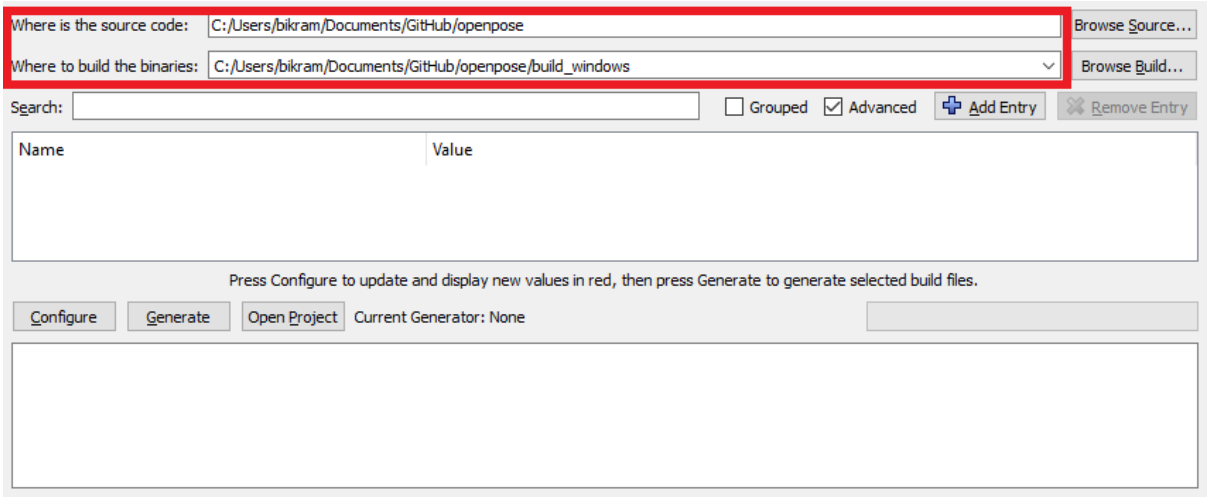

Figura 18: Colocar a OpenPose como directorio origen y colocar la build

2. Presiona el botón Configurar, configura en tu versión de 64 bits de *Visual Studio* (Windows), y presiona *Finalizar*. Nota para los usuarios de Windows: *CMake*-GUI ha cambiado su diseño después de la versión 14. Para versiones anteriores a la 14, normalmente se selecciona *Visual Studio XX 20XX Win64* como el generador (X depende de su versión VS), mientras que, el conjunto de herramientas opcionales a utilizar debe estar vacío. Sin embargo, las nuevas versiones de *Cmake,* requieren que seleccione sólo la versión VS como el generador, por ejemplo, *Visual Studio 15 2017*, y luego debe elegir manualmente x64 para la plataforma Opcional para el generador. Ver las siguientes figuras 19 y 20.

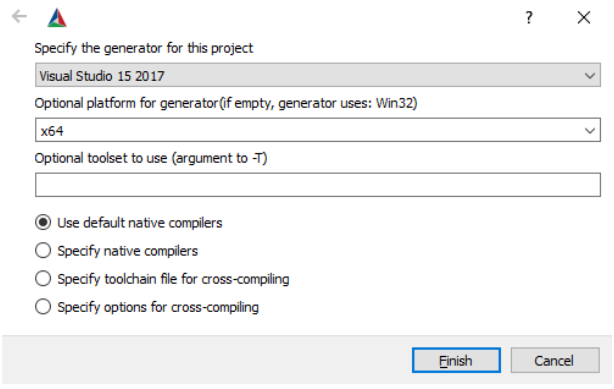

Figura 19: Versiones igual o superior a a 14

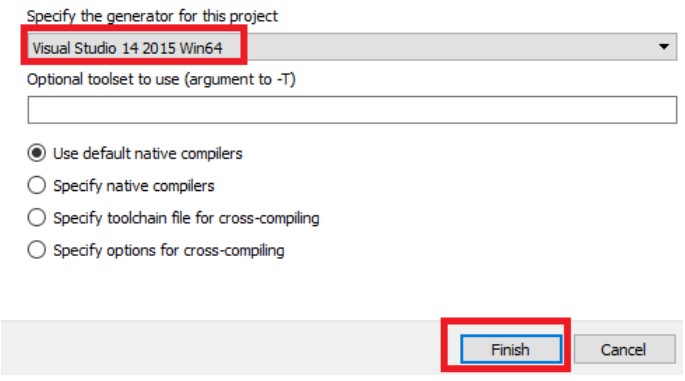

Figura 20: Versiones inferiores a las 14

3. Si este paso tiene éxito, el texto *Configuración hecha* aparecerá en el cuadro inferior de la última línea. De lo contrario, aparecerá un texto rojo en el mismo cuadro inferior que en la figura 21.

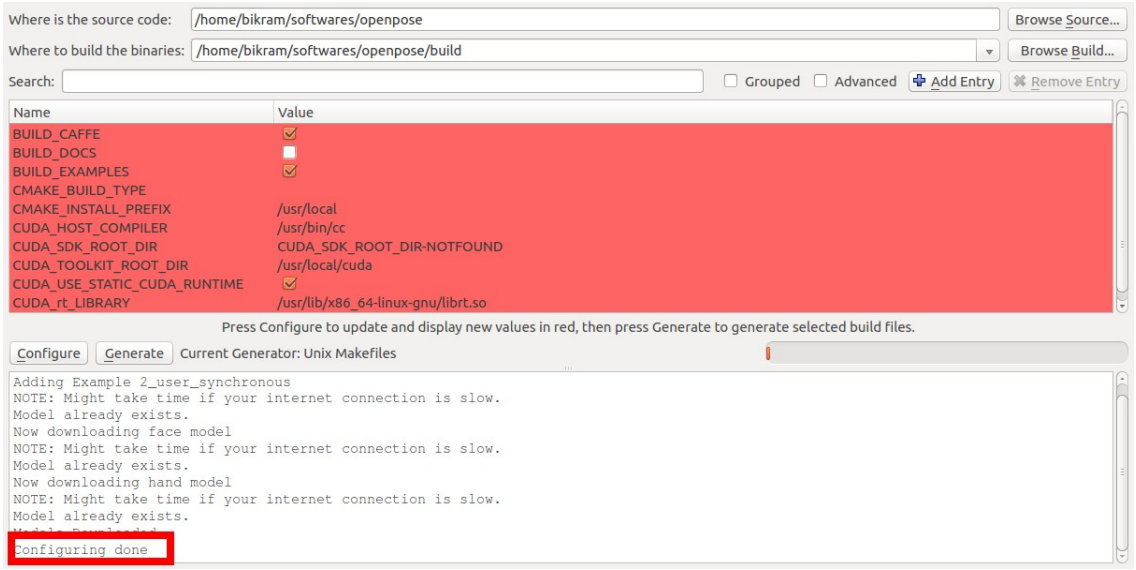

Figura 21: Aviso con la configuración hecha

4. Para activar *pythonAPI* y el modo CPU deberemos buscar en la imagen superior que la cajita de *BUILD_PYTHON* está seleccionada y en *GPU_MODE* pondremos en la selección de modo *CPU_ONLY* como en las siguientes imagenes. En caso de querer usar gráfica NVIDIA O AMD tan solo cambiar en *GPU_MODE* de *CPU_ONLY* a *CUDA* (graficas NVIDIA) o *OPENCL* (gráficas AMD). Tal como se muestra en la figura 22.

Figura 22: Comprobar que estas opciones están habilitadas

5. Presiona el botón *Generar* y procede a la construcción de *OpenPose*. Ahora puede cerrar el *Cmake*. Encuadrados en la figura 23.

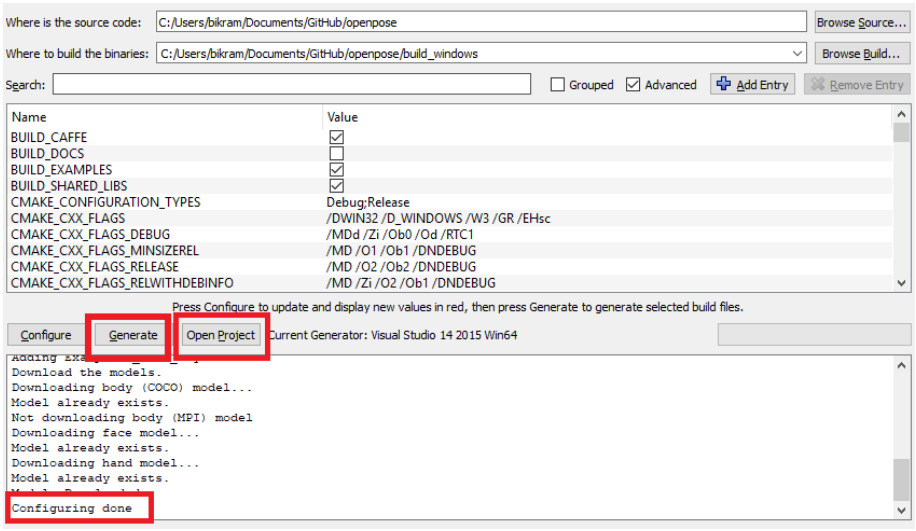

Figura 23: Tras haber realizado los pasos anteriores, pulsaremos en generar y cerraremos Cmake GUI

6. Una vez hecho esto, se abrirá la solución de *Visual Studio* (Windows), llamada *build/OpenPose.sln.* Luego, establezca la configuración de *Debug* a *Release* y presione el icono del triángulo verde (alternativamente presione F5).

4. PROPUESTA Y DESARROLLO DE LA SOLUCIÓN ADOPTADA.

Puesto que la metodología de trabajo es un modelo iterativo basado en incrementos, se expondrá las fases de análisis y diseño de cada uno de los incrementos, antes se mostrará una planificación seguida para el desarrollo de los mismo.

4.1 Planificación.

Cada incremento será un subapartadado en este capítulo, realización de los incrementos fueron los siguientes:

1. Utilización de la *PythonAPI* de *OpenPose* y detección del atleta.
2. Detección de la pose de batida del atleta.
3. Calibrado de la cámara.
4. Cálculo de la altura del salto.
5. Cálculo de ángulo de batida.
6. Cálculo de la velocidad antes de la batida.
7. Optimización del análisis.

4.2 Detección del atleta

4.2.1 Análisis

El primer incremento será el funcionamiento de OpenPose y la detección del atleta en la imágenes. Dado que, ya se ha visto cual es el movimiento de un atleta en el salto de longitud y la red más apropiada para este problema, solo queda estudiar la realización de la grabación de los vídeos.

Como se ha explicado antes, el atleta mantiene su trayectoria durante todo el salto. Puesto que, lo más interesane a estudiar son los apoyos finales y el salto, se usará una posición fija de la cámara en paralelo al punto de batida del salto, así, el movimiento que realiza el atleta desde la perspectiva de la cámara es en 2D [1]. Del mismo modo, si la cámara está fija en vez de acompañar al atleta durante todo el salto con un giro, se evita tener que ir calculando todos lo planos por los que va pasando el atleta desde la perspectiva de la cámara. La imágenes que capta la misma y el movimiento del atleta, son el mismo

plano, tan solo se varía la profundidad del mismo, tal como se explica en la siguiente figura 24.

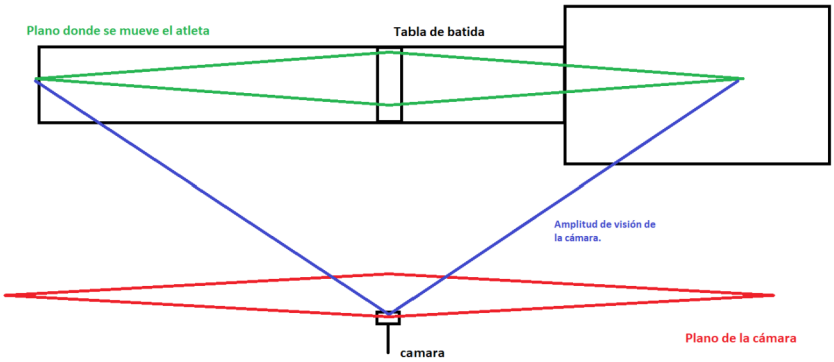

Figura 24: Ambos planos son los mismo, lo único que cambia es la profundidad entre estos

Como se puede observar, el plano de la cámara y del atleta es el mismo, lo único que cambia en este es el eje Z (profundidad), pero el eje X e Y se mantienen iguales.

4.2.2 Restricciones materiales.

El material sería una cámara con un trípode, de forma que esté situada de forma fija apuntando el punto de batida, evitando así que haya movimientos molestos debido a vibraciones en la imagen.

Además, para calcular distancias se debe usar una distancia de referencia dentro de la imagen, de lo cual se hablará más tarde pero, será necesario un objeto de referencia visible y fácil de detectar dentro de la imagen, ya sea varillas o soporte metálicos.

4.2.3 Restricciones de la cámara.

La calidad de los vídeos es importante debido a que, una imagen de baja resolución podría provocar que la red no detecte al atleta, por tanto, para evitar que la red pueda confundirse a la hora de buscar partes del cuerpo, deberá grabarse con una calidad igual o

superior a 854x480 pixeles, puesto que una calidad de imagen peor que esta, no devuelve buenos resultados.

4.2.4 Restricciones de posición.

Dado que, lo más interesante a estudiar son los apoyos finales y el salto, ya que es donde el atleta tiene que evitar a toda costa disminuir su velocidad de despegue y conseguir un buen ángulo de batida para poder llegar lo más lejos posible, la cámara se colocará en paralelo al punto de batida para tener el momento del salto del atleta en el centro de la imagen. Dado que, las cámaras actuales producen una pequeña distorsión cerca de los bordes de la imagen, si la batida del atleta se produce en el centro de la imagen (figura 25), minimizaremos esta pequeña distorsión.

Figura 25: Atleta realizando la batida en el centro de la imagen

También es importante considerar la distancia que habrá de la cámara al punto de batida, esta deberá ser entre [7-10m]. Colocar la cámara demasiado cerca no muestra la carrera de aproximación del atleta y no se vería la caída del salto y colocarla demasiado

lejos provoca que partes del cuerpo (pie, antebrazos, etc.) sean difíciles de detectar para la red.

Por último, es importante fijar la altura a la que será colocado el trípode y la cámara. Esta altura oscilará entre 1,30m para que se encuentre al nivel visual del ojo humano. Por otra parte, esto evitará que el atleta se vea cerca del borde inferior y de esta manera, tratar de evitar la distorsión.

4.2.5 Diseño

Cuando el usuario haya hecho las grabaciones, analizará los vídeos. El usuario deberá cargar en la aplicación un vídeo, este se dividirá en frames para poder procesarlos uno a uno. Luego deberá iniciar el análisis y dará comienzo la comunicación del controlador con la *Python API,* para cada frame del vídeo, se le pedirá con la *python API* a la red, que procese el frame, mientras siga habiendo frames que analizar, se seguirán haciendo peticiones. Como se puede ver en la figura 26.

```
8   Para cada Frame del vídeo
9
10      Mandar frame mediante API (opwrapper)
11      Acceder a los resultados
12
13      Si atleta detectado
14          continuar con el procesamiento
```

Figura 26: Se comprobará si el atleta es detectado para continuar con el procesamiento del frame

4.3 Batida del atleta y su detección.

4.3.1 Análisis

Dado los puntos de cuello, cadera y las piernas del atleta, se calcularan vectores para comprobar si el atleta está realizando una batida de un salto de longitud, en caso positivo se avisará a la aplicación. Para entender la pose veamos los siguientes ejemplos de la figura 27:

Figura 27: Pose de batida de 2 atleta de alto nivel

La pierna con la que bate el atleta está totalmente en extensión junto con el pie impulsando el suelo, la pierna contraria esta realizando un contra movimiento lanzando la rodilla de forma vertical. Esta pose se puede hacer mejor o peor, dependiendo del nivel del atleta, por lo que la comprobación de los parámetros de la pose deberá estar entre unos márgenes, pudiendo así detectarla aunque el atleta no tenga un alto nivel como ocurre ahora en la siguiente figura 28.

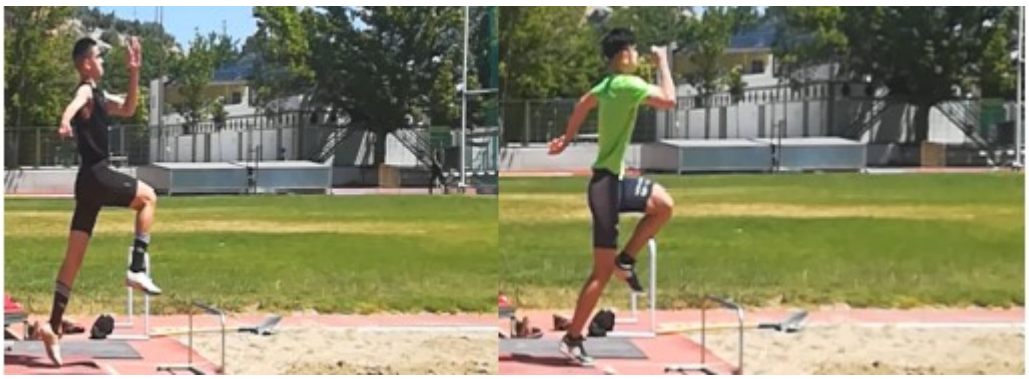

Figura 28: Batida de dos atletas con gestos no tan pulidos

4.3.2 Diseño

Por tanto, la detección de la pose se hará de manera que:

- Compruebe que la pierna de impulse este en extensión o parcialmente en extensión.
- El pie de impulso este parcialmente perpendicular a la misma pierna.
- La pierna en contra movimiento este parcialmente perpendicular al tronco del atleta
- La parte inferior de la pierna en contra movimiento este parcialmente perpendicular a la superior

Como podemos ver en el diagrama de flujo de la figura 29.

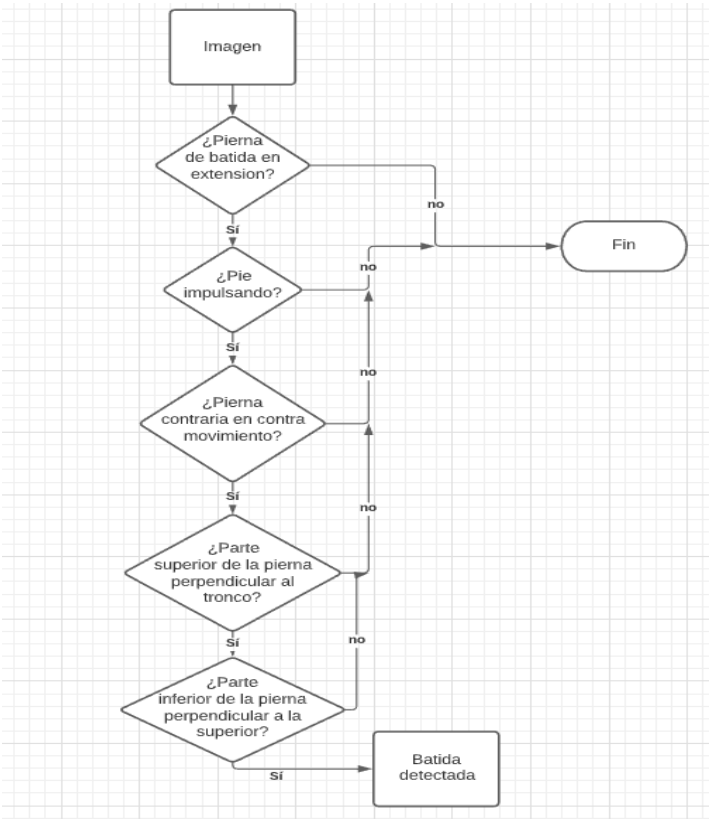

Figura 29: Diagrama de flujo para la detección de un atleta en pose de batida

4.4 Calibración.

4.4.1 Análisis

Como se ha mencionado anteriormente, las cámaras actuales producen una ligera distorsión en los bordes de las imágenes que capturan, mediante el posicionamiento de la cámara, hemos intentado minimizar que la distorsiones se produzcan, pero esto no evita que se sigan produciendo a medida que el atleta se acerca a estos. Por tanto, se ha implementado un método de calibración para las imágenes.

Este método supone que cuanto más lejos se está del centro de la imagen, las distorsiones son mayores, aumentando de forma lineal.

Para saber cuanta distorsión tiene una cámara se necesitará una imagen, esta imagen deberá mostrar un objeto fácil de identificar (conos, varillas, etc.) los cuáles se colocarán de forma horizontal a una distancia de un metro entre ellos, ocupando así todo el ancho de la imagen como en la figura 30.

Figure 30: Conos distanciados a 1m

Ahora bien, es necesario 4 coordenadas , 2 para medir la distancia entre 2 conos (figura 31) lo más próximo del centro de la imagen, cada coordenada coincidirá con el eje central del cono, y otros 2 para medir la distancias entre 2 conos lo mas próximo al borde de la imagen, (figura 31).

Como ahora es conocida la distancia en píxeles de 1m en el centro de la imagen y la distancia en píxeles de 1m en el borde la imagen, se puede calcular la diferencia que hay entre estas. Con esta diferencia se puede calcular la progresión lineal que tiene la distorsión en la imagen. Esta progresión viene dada dividiendo el tamaño horizontal de la imagen entre la diferencia en píxeles. Ahora tenemos cada cuanto se produce una distorsión de un píxel.

Una vez se obtiene la progresión lineal de la distorsión, para calcular la distorsión de un píxel, tan solo habrá que calcular la diferencia en píxeles que hay desde la coordenada del mismo hasta el centro en el eje X, esta diferenciase dividirá entre la progresión que se ha calculado anteriormente, y devolverá el número de píxeles que se ha distorsionado la coordenada.

Figure 31: Cogeremos la distancia entre los conos en el recuadro

4.4.2 Diseño

Para realizar estas operaciones de calibración se seguirá el siguiente diagrama de flujo (figura 32).

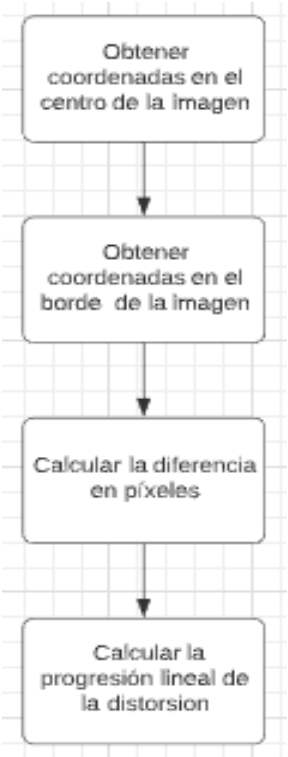

Figure 32: diagrama de flujo seguido

Una vez se obtenga la progresión lineal, para calcular la coordenada rectificada se seguirá los siguientes pasos en la figura 33, siendo el valor mitad, el centro de la imagen en el eje horizontal.

```
diferencia = abs|xl - mitad|
distorsion = diferencia / progresion

Si xl > mitad
    coord_rec = xl - distorsion
Si no
    coord_rec = xl + distorsion
```

Figure 33: Algoritmo para el cálculo de la corrección del píxel

4.5 Cálculo de la altura del salto.

4.5.1 Análisis

Para poder medir distancias dentro de la imagen, será necesario una distancia de referencia en la misma, por ejemplo, unas varillas en el centro de la imagen a una distancia de 1m; al tener esta referencia es posible saber cualquier distancia entre dos puntos dentro de la imagen.

Si se dispone de x1, y1 del primer objeto de referencia y el x2, y2 del segundo objeto de referencia cualquier distancia entre x3, y3 y x4, y4, figura 34 podrá ser calculada mediante la siguiente fórmula, figura 35.

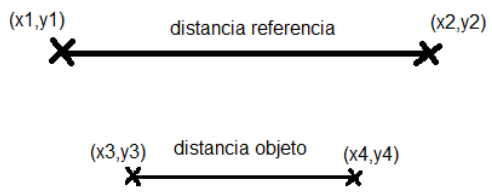

Figure 34: Distancia de referencia y la distancia objeto a medir

$$referencia_{pixel} = \sqrt[2]{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

$$objeto_{pixel} = \sqrt[2]{(x_3 - x_4)^2 + (y_3 - y_4)^2}$$

$$objeto_{uds} = \frac{objeto_{pixel} * medidaReal}{referencia_{pixel}}$$

Figura 35: x1, x2, y1, y2 hacen referencia a las coordenadas de los extremos de la referencia y x3, x4, y3, y4 a las cotas de la distancia objetivo

Ahora bien, además se deberá aplicar la corrección simplemente sustituyendo las X_I por las X_I rectificadas.

4.5.2 Diseño

Para calcular la altura del salto, solo será necesario el punto de altura máxima del salto y medir esta distancia desde la cadera del saltador (eje de gravedad) hasta el suelo. Para hacer esto, basta con que se guarde la mayor altura registrada de la cadera del atleta

durante el salto y posteriormente calcular esa distancia mediante las fórmulas expuestas en la figura 35. En la figura 36 se puede observar los pasos seguidos para el cálculo de esta.

```
8   Para cada Frame procesado
9
10      Si altura de la cadera > registrada
11          Guardar altura
12
13  Medir altura
```

Figure 36: Guardaremos la mayor altura registrada

4.6 Cálculo del ángulo de batida.

4.6.1 Análisis

Normalmente, el ángulo de batida depende mucho del estilo de salto del atleta, a parte de datos morfológicos de este, los saltadores en los que predomina la fuerza, emplean ángulos mayores, siendo más apreciables los cambios de ritmo y de amplitud de los últimos pasos. Los más veloces, emplean ángulos y trayectorias de vuelo más tensas. Las variaciones en los dos últimos pasos son menos significativos. Por tanto, es un dato que depende de la interpretación pero que es relevante; para calcular esto habrá que tener en cuenta dos *frames*:

- El punto de la cadera en la pose de batida del atleta.
- La altura máxima registrada de la cadera del atleta.

Ambos se pueden observar en la figura 37, por tanto, el cálculo del ángulo del salto es simplemente una función trigonométrica.

$$\tan\alpha = a/c$$

$$\alpha = \tan^{-1}(a/c)\ donde\ \alpha\ es\ el\ ángulo\ de\ batida$$

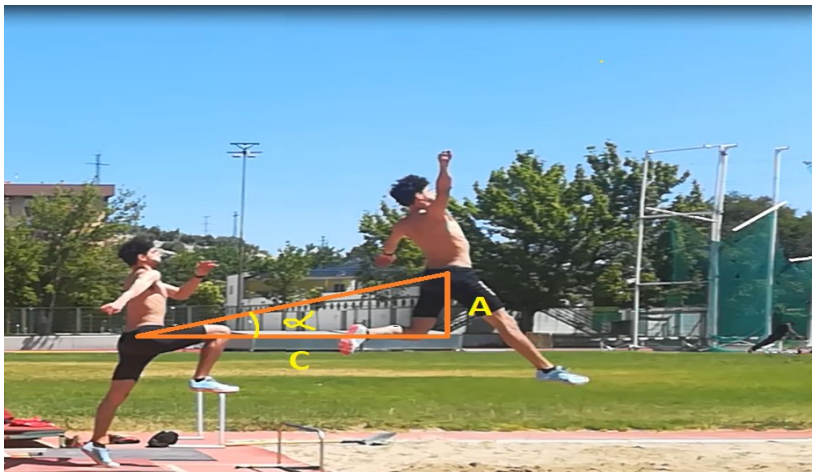

Figura 37: El cálculo del ángulo se vuelve un problema trigonométrico

4.6.2 Diseño

Para realizar el cálculo del ángulo es necesario 3 puntos:

- Punto de la cadera en la altura máxima (P_1), el cual se obtiene en el incremento anterior.
- Punto de la cadera en la batida (P_2), obtenido de incrementos anteriores también.
- Punto (P_3) que forma un triángulo rectángulo con P_1 y P_2.

En la figura 38 tenemos la disposición de los puntos.

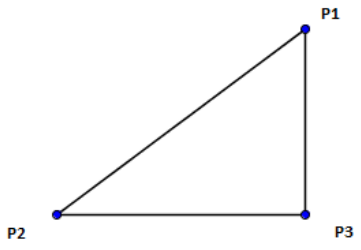

Figure 38: Puntos necesarios para calcular el ángulo

Por tanto deberemos según la figura 39, calcular estos puntos, medir la distancias entre estos y calcular el valor del ángulo mediante la fórmula visto en la fase de análisis.

```
1  Obtener P1 y P2
2  Calcular P3 a partir de P1 y P2
3  calcular angulo mediante la tangente
```

Figure 39: Pasos a seguir para el cálculo de el ángulo

4. 7 Cálculo de la velocidades.

4.7.1 Análisis

Para un atleta, es interesante saber la velocidad con la que llega al punto de batida, por lo que, también se ha añadido a la hora del análisis.

Sabemos que $velocidad = distancia / tiempo$. Se cogerán las coordenadas de la cadera del *frame* anterior a la detección de la pose, y las coordenadas de la cadera en la detección de la misma, y mediremos la distancia que hay entre estos dos puntos.

También se conocen los fotogramas por segundo del vídeo ya que nos los dará *OpenCV*, el tiempo será la inversa de los fotogramas. Por tanto, como ya sabemos la distancia y el tiempo, tan solo tendremos que sustituir los valores en la fórmula de la velocidad.

4.7.2 Diseño

Si se observa el algoritmo de la figura 40, los pasos a seguir para calcular la velocidad es sencillo.

```
1  P1 -> Coordenadas de la cadera en la batida
2  P2 -> Coordenadas de la cadera del frame anterior
3  Medir distancia d entre P1 y P2
4  Tiempo es la inversa de los fps
5  velocidad = d/Tiempo
```

Figure 40: Cálculo de la velocidad

4. 8 Optimización del análisis.

4.8.1 Análisis

A la hora de grabar los vídeos, hay un breve lapso de tiempo de unos 3 o 4 segundos desde que se le da a iniciar el vídeo, hasta que el atleta comienza a saltar, esto hace que un vídeo en el que solamente hay 50 *frames,* donde el atleta sale en este, hay X *frames* donde no ocurre nada, por tanto, se ha añadido dos formas de optimizar este problema.

La primera, es dejar al usuario que pueda elegir el *frame* desde donde empieza a ejecutar el análisis, esto puede ser un poco molesto para el usuario ya que él mismo tiene que elegir donde empezar el análisis.

La segunda segunda opción, no interfiere el usuario y se hace de forma automática. Se deberá comprobar si el atleta está en la imagen, si este no se encuentra, deberemos saltar los *frames* hasta encontrarlo, una vez lo encontremos,

4.8.2 Diseño

Cuando se ejecute el análisis, si no se detecta al atleta en el *frame*, se saltará 10 *frames*, este proceso lo se repetirá mientras el atleta no sea detectado. Cuando el atleta sea detectado, pero aún está lejos de la batida, lo se irán saltando solamente 3 *frames* Cuando el atleta este cerca del punto de batida, el programa empezará a avanzar *frame* a *frame*. Para comprender mejor como funciona esta solución veamos un diagrama, figura 41.

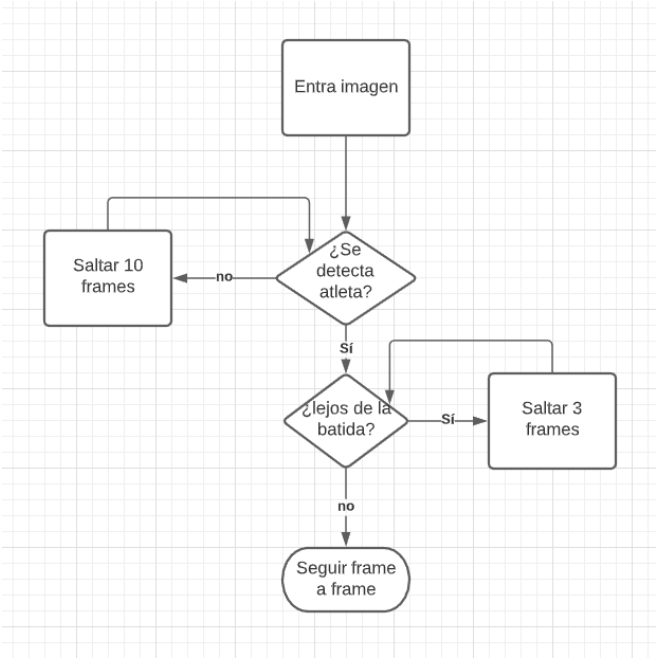

Figure 41: Diagrama de flujo para optimizar el análisis

4.8.3 Resultado de la optimización

Para cada *frame*, la red tarda en analizarlo entre 3,4; 3, 8 segundos dependiendo del tamaño de la imagen y si hay una persona en ella o no. Normalmente, los vídeos suelen ser de unos 250 *frames* dependiendo de la duración del vídeo, esto hace que en el peor de los casos, un vídeo pueda llegar a durar $250 \times 3{,}8 = 950\, segundos$ que hacen un total de 15,7 minutos. Sin embargo, si se aplica esta optimización, hace que se desechen una gran cantidad de *frames* sin información, para este caso 200 *frames*, por tanto, $250 - 200 = 50\, frames$ y $50\, x\, 3{,}8 = 190\, segundos$ que hacen un total de 3,1minutos. Por lo que, esta pequeña optimización da muy buenos resultados bajando el tiempo de espera de más de 10 minutos. Ahora se expondrá un gráfico comparativo (figura 42) donde para un mismo vídeo con X *frames*, la diferencia en minutos de duración que hay cuando se aplica la optimización a cuando no.

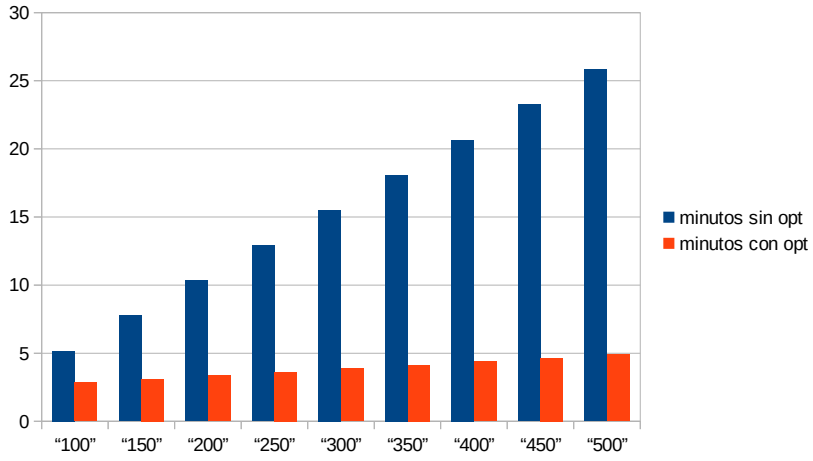

Figure 42: Gráfica comparativa entre los tiempos al aplicar la optimización a cuando no

Como se puede observar, al subir la cantidad de *frames*, la duración del análisis es mayor si la red analiza todos y cada uno de los *frames*, pero cuando se aplica la optimización esto no ocurre, ya que esta irá saltando *frames* de 10 en 10 hasta que detecte al atleta.

4. 11 Resultados.

Los resultados se devolverán en formato CSV, donde estarán la velocidad del atleta en m/s, la altura del salto y el ángulo de este. También, se añadirá el vídeo procesado con una estela de la cadera del atleta, así se podrá ver de forma visual como de brusca es la variación de la altura de la cadera del atleta, ya que a este le interesa que sea la menor posible. La vista de estos resultados se pueden apreciar en la tabla 2 y la figura 43.

Tabla 2: Salida de la aplicación

Altura_cm	Ángulo_batida	velocidad_batida_m/s
171	12	7.53

Figura 43: Video de salida

5. DISEÑO E IMPLEMENTACIÓN.

En esta sección se presentan las fases de diseño e implementación que se ha llevado a cabo.

5.1 Diseño.

Para el diseño de la aplicación, el usuario podrá acceder a cualquier parte de la misma desde el *menú bar*, este tiene el siguiente flujo, figura 44.

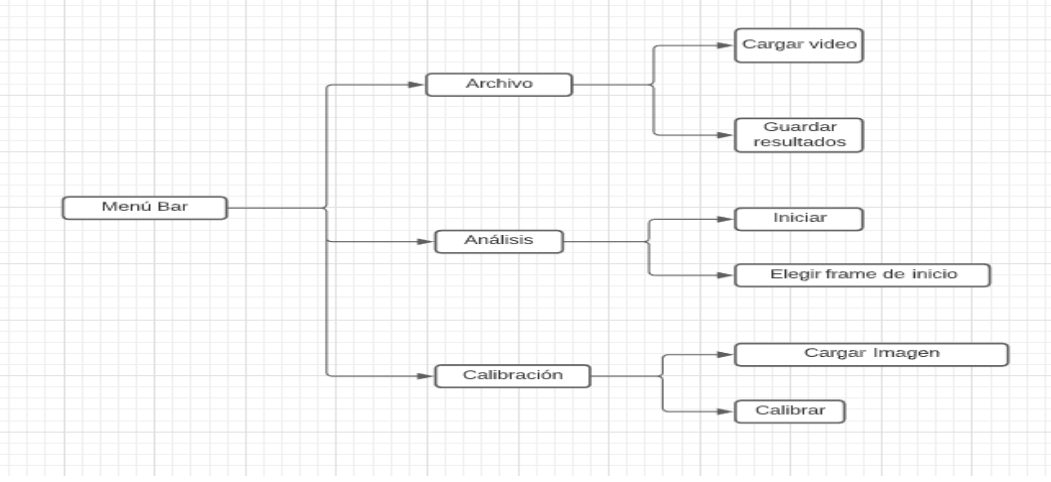

Figura 44: Diagrama del menú

5.1.1 *Sketch.*

En primer lugar, se realizó un *skecth* con las primeras ideas generales de la interfaz (figura 45) buscando donde irían colocado los botones, donde se mostraría la imágenes, como se elegiría los *frames* o vídeos, etc.

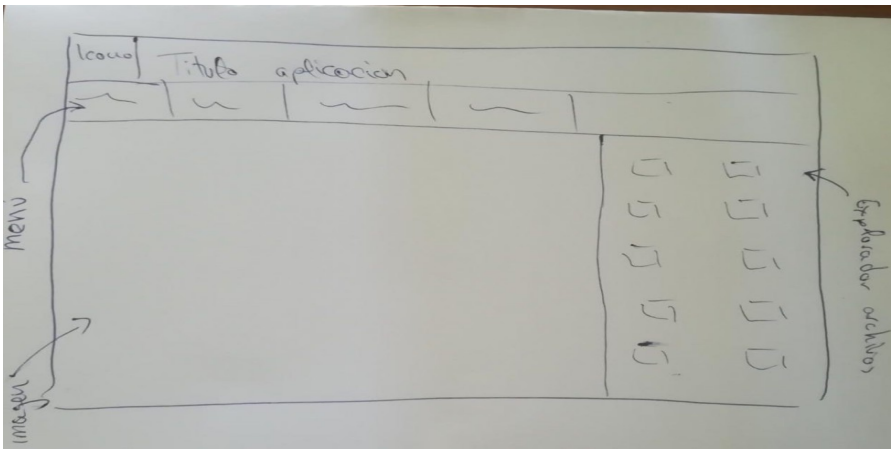

Figura 45: Sketch inicial

5.1.2 *WireFrame*.

Posteriormente, se realizó un *WireFrame* con algunas modificaciones (figura 46) y ver como quedarían los espacios en estos.

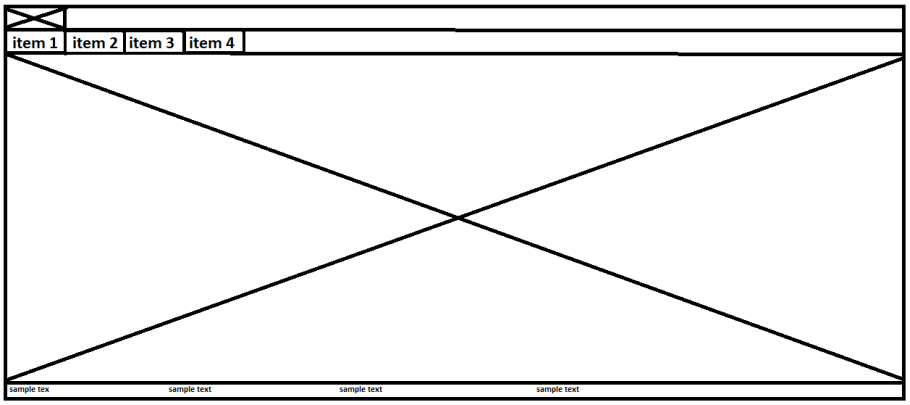

Figura 46: WireFrame para comprobar marcos y tamaños

5.1.3 *Mockup*.

Por último, se realizó un *mockup (figura 47)* con el modelo final que tendrá la interfaz.

Figura 47: Mockup resultante tras varios ajustes

5.1.4 Interfaz.

Se ha optado por una interfaz sencilla e intuitiva de forma que se pueda acceder a cualquier función de la aplicación directamente desde el menú, se puede apreciar en la figura 48 y 49.

Figura 48: Estado de la interfaz al abrir la aplicación

Figura 49: Estado de la interfaz tras carga un vídeo

Como se puede apreciar, la interfaz está separada por 3 secciones:

- Archivo: en esta sección se tendrá acceso a la carga de datos, en este caso un vídeo y el guardado de resultados, figura 50.

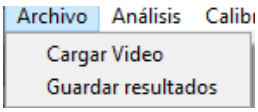

Figura 50: Submenú de archivo

- Análisis: esta sección esta dedicada a la ejecución del análisis del vídeo y elección del *frame* de inicio del mismo, figura 51.

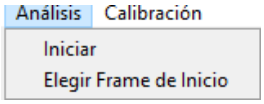

Figura 51: Submenú de análisis

- Calibración: esta sección da acceso a el método de calibración de la cámara. Por un lado, en primer lugar, tendremos que cargar la imagen con la que queremos calibrar la cámara y posteriormente hacer clic en calibrar, figura 52.

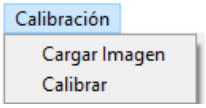

Figura 52: Submenú de calibración

Por otro lado, se dispone un gran hueco en blanco que se usará para mostrar la imagen a calibrar o el *frame* donde el usuario deberá indicar cual es la distancia de

referencia. Además, habrá un *status bar* en la zona inferior de la interfaz para que el usuario pueda ver en zona de la imagen está clicando, podemos observar en la zona inferior de la figura 53.

Figura 53: Interfaz con el status bar actaulizado

Por otro lado, la aplicación también ofrece mensajes emergentes para cada vez que se termine de realizar una acción, así, el usuario siempre sabrá en todo momento que la orden que ha dado se ha ejecutado con éxito. Ej: figura 54.

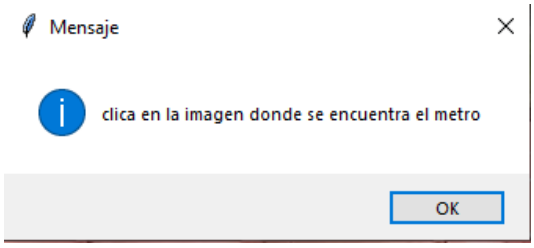

Figura 54: Avisos emergentes

5.1.5 Diagrama de clases.

A continuación se muestra en la figura 55 el diagrama UML utilizado

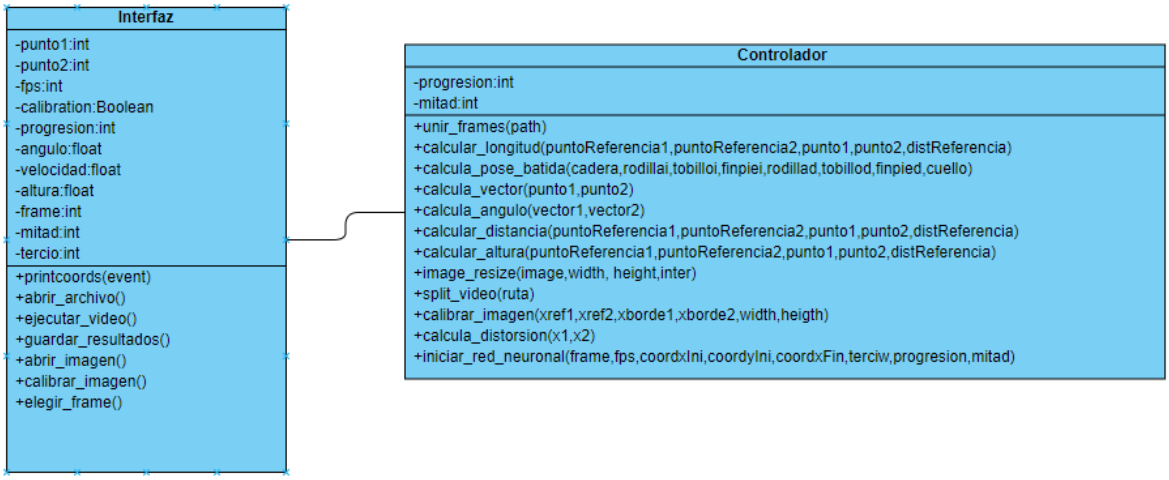

Figura 55: Diagrama UML

5.2 Implementación.

Como se puede observar el diagrama de clases, solo tenemos 2 clases implementadas:

5.2.1 Clase Interfaz.

Se encarga de la comunicación con el usuario, toma de datos del usuario como pueden ser coordenadas en la imagen así como la carga de un vídeo y el guardado del análisis final.

Para esta clase tendremos los atributos:

- Punto1: marca el primer punto de referencia de la distancia.
- Punto2: marca el segundo punto de referencia de la distancia.
- fps: Fotogramas por segundo del vídeo a analizar.
- Calibration: Para saber si la calibración se hará manual o se usará la que viene por defecto.

- Progresión: es la progresión en píxeles con la que se va distorsionando la imagen conforme se acerca al borde.
- Ángulo: ángulo del salto, resultado del análisis final.
- Velocidad: velocidad del atleta antes del saltar, resultado del análisis final.
- Altura: altura del atleta durante el salto, resultado del análisis final.
- *Frame*: *frame* en el que empezará el análisis.
- Mitad: valor al dividir el valor de la imagen en horizontal por la mitad, para poder acceder a la mitad de la imagen más rápido.
- Tercio: valor al dividir la imagen en 3 secciones, para acceder a estos valores más rápido.

Ahora se hablará de las funciones de la misma clase:

- Printcoords(event): Esta función coge el evento de clicar dentro del *canvas* donde se encontrará la imagen del atleta, así podrá indicar mediante clics donde se encuentra la distancia de referencia, tras esto, actualizará el *status bar* en la zona inferior.
- abrir_archivo(): Esta función abrirá el explorador de archivos del sistema para que el usuario pueda elegir el vídeo que desea analizar, los *frames* del vídeo se dividirán y serán guardados en una carpeta lista para su análisis. Para finalizar, coge el primer *frame* del vídeo y lo muestra por pantalla para que el usuario indique la distancia de referencia.
- *ejecutar_video():* Inicia el análisis del vídeo mandando la ruta de la carpeta donde se encuentran los *frames* al controlador.
- *guardar_resultados():* Guarda los resultados del análisis en un csv, junto con el vídeo analizado, donde indique el usuario mediante el explorador de archivos del sistema.

- *abrir_imagen():* abre el explorador de archivos para que el usuario elija la imagen que desea calibrar, tras esto, la imagen se mostrará en el canvas y el usuario podrá clicar en este.
- *calibrar_imagen():* Tras el usuario haber indicado distancias de referencia para la calibración, se calcula la distorsión en los bordes.
- *elegir_frame():* Se abrirá el explorador de archivos del sistema en la carpeta donde se encuentran los *frames* del vídeo, el usuario deberá elegir uno de estos *frames*.

5.2.2 Clase controlador.

Clase que se encargará de la comunicación con *OPENPOSE API* y la realización del análisis del vídeo, tras haber realizado el análisis le devolverá a la clase interfaz una terna con los valores del análisis a la *clase Interfaz.*

Para esta clase se encuentran los atributos:

- Progresión: la progresión de la distorsión de los píxeles conforme se acerca a los bordes
- Mitad: valor al dividir el valor de la imagen en horizontal por la mitad, para poder acceder a la mitad de la imagen más rápido.

Ahora hablaremos de las funciones de la misma clase:

- *Unir_frames(): E*sta función se encarga de unir los *frames* del vídeo analizado para convertirlo en un vídeo archivo .mp4.
- *Calcular_longitud():* Dado 2 puntos en la imagen y la distancia de referencia, calcula mediante las fórmulas mostradas anteriormente, la distancia en el origen de coordenadas X, sin tener en cuenta la coordenada Y.

- *Calcular_pose_batida():* Dado los puntos de cuello, cadera y piernas del atleta se calcularan vectores para comprobar si el atleta está realizando una batida de un salto de longitud, en caso positivo devuelve *True*.
- *Calcula_vector():*Dado dos puntos, calcula el vector que forman entre ellos dos.
- *Calcula_angulo():* Calcula el ángulo que hay entre 2 vectores.
- *Calcula_distancia():* Dado 2 puntos y la distancia de referencia, calcula la distancia entre estos dos.
- *Calcular_altura():* Dado 2 puntos y la distancia de referencia, calcula la distancia entre estos dos teniendo en cuenta solamente el eje Y.
- *Image_resize():* Redimensionar la imagen para que no sea mayor a 1200p.
- *Split_video():* Esta función divide el vídeo que se le pasa mediante su ruta en *frames*, y los guarda en una carpeta llamada *data,* para su posterior uso.
- *Calibrar_imagen():* dado 4 puntos en el eje x,2 para la distancia en el centro de la imagen, 2 para la distancia en el borde de la imagen y el tamaño de la imagen calcula cuanto va distorsionándose la imagen conforme avanza al borde.
- *Iniciar_red_neuronal:* Se ira mandando a la *OPENPOSE API* cada *frame* para que lo vaya analizando, posteriormente se comprobará si el atleta está en la imagen, se calculará poses si es necesario y se calcularan otros datos como altura, ángulo, velocidad del atleta.

5.3 Consideraciones futuras.

Ya que esta aplicación no deja de ser un prototipo, aquí se consideran mejoras futuras para esta, ya que, debido a restricciones de tiempo, no se han podido implementar. Algunas de estas son:

- Indicar de forma visual en el canvas las pulsaciones del usuario, ya sea con una cruz, un círculo...
- Que el usuario a la hora de elegir el *frame*, acceda directamente al vídeo y pueda verlo *frame* a *frame*, decidiendo así desde cual empezar.
- Los métodos de calibrar y iniciar análisis no sean visibles hasta que el usuario no haya realizado las acciones necesarias.
- Conforme inicie la aplicación, avise al usuario de que la cámara no está calibrada en caso de que sea la primera vez que entre, tras calibrarla guardar su configuración.
- Que el usuario pueda ver una barra de progreso conforme se va realizando el análisis.

6. CONCLUSIONES.

En este trabajo, se ha llevado a cabo un estudio de la visión computador centrado en la detección de la pose humana. Dado la gran información que he obtenido sobre este tema, he descubierto la dificultad que tiene una máquina para poder detectar objetos o humanos, lo cual es una tarea trivial para las personas y, a parte, lo complicado que es hacer que una máquina sea capaz de detectar personas u objetos.

Al principio, mientras indagaba y leía documentos, al no entender el funcionamiento de este tipo redes, era bastante complicado encontrar una que realmente funcionará para el problema planteado. Poco a poco, fuí comprendiendo su funcionamiento y encontrando nuevas librerías, cada una con sus peculiaridades, hasta finalmente dar con *OpenPose.*

Pienso que es una librería bien cuidada y bien documentada, por lo que, fue bastante fácil el utilizarla. Por otro lado, al ser 2D se adaptaba muy bien al problema. Actualmente, es una rama que me ha llamado mucho la atención y en la que me gustaría seguir adentrándome, ya que desconocía mucho sobre este campo.

Respecto a la solución desarrollada, pienso que es una aplicación bastante sencilla, se pueden implementar muchas mejoras y añadir nuevas funciones pero, obviamente, por

restricciones de tiempo era imposible. Lo he encontrado como un reto con el cual no me había cruzado hasta ahora. El entender el problema, investigar sobre varias soluciones y comprobar cual es la más efectiva y eficiente, es algo tedioso pero reconfortante una vez se llega a la solución.

Por último, como atleta y en este caso, especializado en el salto de longitud, esta es una herramienta que ofrece datos que, de otro modo, se necesitarían herramientas muy costosas, lo cual la mayoría de atletas no son capaces de permitirse. Por tanto, encuentro la aplicación muy útil y estoy seguro que seguiré trabajando en ella.

7. MANUAL DE USUARIO.

En esta sección vamos a explicar como utilizar aplicación.

7.1 Ejecutar aplicación y cargar un vídeo.

En primer lugar, para ejecutar la aplicación basta con hacer doble clic en el archivo ejecutable *Interfaz.pyw* y nos saldrá la siguiente interfaz de la figura 56.

Figure 56: Interfaz al inicio de la aplicaión

A continuación, para cargar un vídeo se debe ir al menú superior (figura 57) y hacer clic en “Archivo” y posteriormente en “Cargar vídeo”, aparecerá el explorador de archivos del sistema y se podrá elegir nuestro vídeo a analizar.

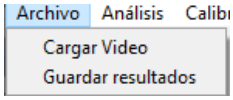

Figura 57:

Submenú

Tras unos breves segundos, la aplicación nos avisará que se ha cargado el vídeo y mostrará un *frame* de este en la interfaz de la siguiente manera Ej: figura 58.

Figura 58: Estado de la interfaz tras cargar un vídeo

7.2 Identificar la distancia de referencia que se utilizará y calibración de vídeo.

En primer lugar, se debe asegurar que la distancia de referencia que se utilizará está a 1m. Cuando se haya cargado un vídeo y tengamos ya un *frame* de este en la interfaz, se debe clicar en la imagen indicando donde empieza la distancia de referencia que usaremos y posteriormente donde acaba. Conforme se vayan haciendo clics, se irá actualizando el *status bar* de la zona inferior de la interfaz con las coordenadas del clic. En caso de equivocación, simplemente deberemos volver a clicar en la imagen y se seguirá actualizando el *status bar* como en la figura 59 y 60.

Figura 59: Status bar

Figura 60: Status bar

Cuando se configure la distancia deseada, se podrá calibrar la cámara clicando en el menú superior el apartado de “Calibracion” y posetiormente en “Cargar imagen”, como se puede ver en la figura 61.

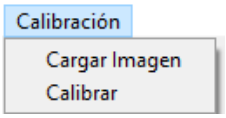

Figura 61: Submenú

Conforme se clique en “cargar imagen” se volverá abrir el explorador del sistema y se seleccionará la imagen que se vaya a utilizar para calibrar la cámara tal tras esto, se nos abrirá la imagen como el ejemplo de la figura 62.

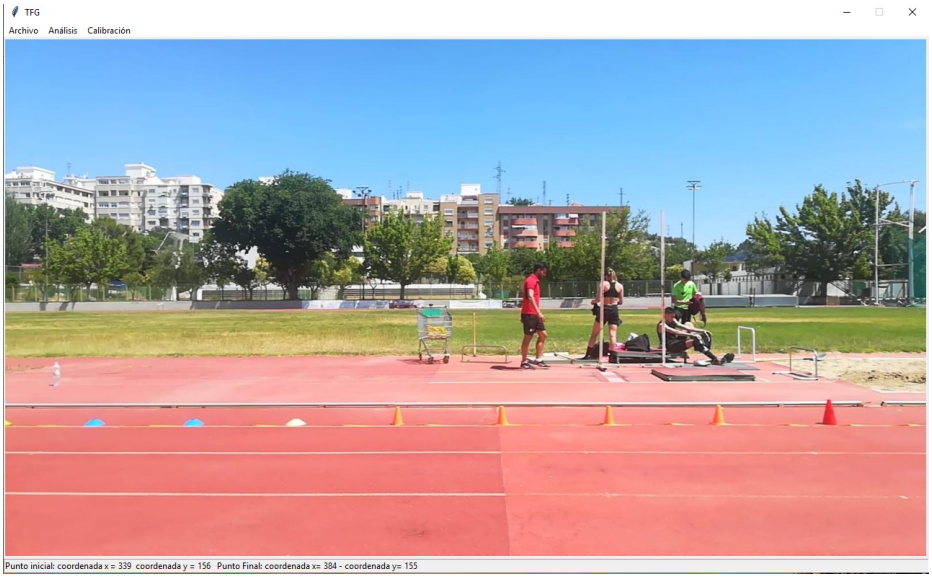

Figura 62: Interfaz tras añadir una imagen para el calibrado

Ahora bien, para calibrar la cámara es necesario hacer 4 clics, 2 para la distancia que hay entre los 2 indicadores que usaremos (conos, varillas, etc) en el centro de la imagen y otros 2 para la distancia entre los indicadores en el borde, de la misma forma que antes, si se comete algún error, no pasa nada, ya que se puede seguir clicando y el *status bar* seguirá actualizándose como en la figura 63. Es importante asegurarse de que la imagen tiene la misma calidad que el vídeo que va a ejecutar.

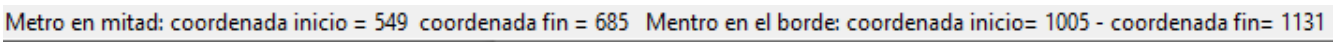

Figura 63: Status bar

Una vez quede como se desea, solo será necesario que hacer clic en el menú "Calibracion" y luego en "calibrar", tras unos segundos la aplicación avisará cuando haya finalizado. Ej: figura 64.

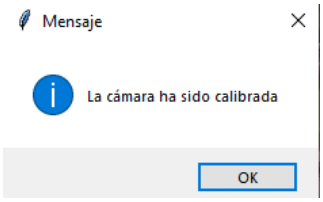

Figura 64: Mensaje emergente

De la misma forma, la aplicación lleva un calibrado por defecto en caso de no poder o no querer usarse este.

7.3 Ejecución del análisis.

Cuando se tengan todos los pasos anteriores realizados, es la hora de ejecutar el análisis, tan solo es necesario ir al menú y pulsar en "análisis" y en "Iniciar" o bien, si se desea que el análisis empiece desde un cierto *frame,* se debe pulsar primero en "elegir *frame* de inicio" y luego en "iniciar" (figura 65). En caso de elegir un *frame* de inicio, se debe

comprobar de que el *frame* que elige sea antes de que el atleta realice la batida, si no, funcionará ya que el análisis de un salto empieza en este en la batida.

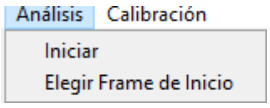

Figura 65: Submenú

7.4 Guardar datos del análisis.

Tras la finalización del análisis la aplicación avisará como en la figura 66.

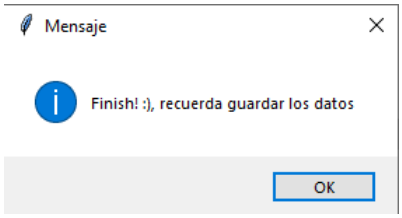

Figura 66: Mensaje emergente

Se debe guardar los datos haciendo clic en “Archivo” y luego en “guardar resultados” como se muestra en el submenú de la figura 67.

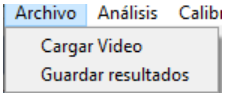

Figura 67: Submenú

Se abrirá de nuevo el explorador de archivos de sistemas y se elegirá donde guardar los datos, estos vendrán en formato “csv”. De la misma forma, el vídeo con el post-

procesamiento también será guardado en el mismo directorio donde se haya guardado el csv.

Bibliografía

[1] - Análisis cinemático de la batida en saltadores

recuperado de: https://www.efdeportes.com/efd181/analisis-cinematico-de-saltadores-de-longitud.htm

[2] Resumen del salto de longitud

recuperado de: https://www.uaeh.edu.mx/scige/boletin/prepa4/n3/m2.html

[3] OpenPose. Gines Hidalgo, Zhe Cao, Tomas Simon, Shih-En Wei, Hanbyul Joo, y Yaser Sheikh.

recuperado de: https://github.com/CMU-Perceptual-Computing-Lab/openpose

[4] OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields: Gines Hidalgo, Zhe Cao, Tomas Simon, Shih-En Wei, Hanbyul Joo, y Yaser Sheikh.

Recuperado de: https://arxiv.org/pdf/1812.08008.pdf

[5] Keypoints OpenPose

Recuperado de: https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/output.md

[6] Instalación OpenPose

Recuperado de: https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/prerequisites.md

[7] H.G. Barrow and J.M tenenbaum. Computer vision systems

Recuperado de: http://web.mit.edu/cocosci/Papers/Barrow-Tenenbaum78.pdf

[8] Awesome Human Pose Estimation

Recaperado de: https://github.com/cbsudux/awesome-human-pose-estimation

[9] Mehta, D., Sotnychenko, O., Mueller, F., Xu, W., Sridhar, S., Pons-Moll, G., Theobalt, C. (3DV 2018). Single-Shot Multi-Person 3D Pose Estimation From Monocular RGB.

Recuperado de: https://arxiv.org/pdf/1712.03453.pdf

[10] Pavllo, D., Feichtenhofer, C., Grangier, D., & Auli, M (ArXiv 2018). 3D human pose estimation in video with temporal convolutions and semi-supervised training

Recuperado de: https://arxiv.org/pdf/1811.11742.pdf

[11] Varol, G., Ceylan, D., Russell, B., Yang, J., Yumer, E., Laptev, I., & Schmid, C. (ECCV 2018). BodyNet: Volumetric Inference of 3D Human Body Shapes.

Recuperado de: https://arxiv.org/pdf/1804.04875v3.pdf

[12] Ke Sun, Bin Xiao, Dong Liu, Jingdong Wang (CVPR 2019). Deep High-Resolution Representation Learning for Human Pose Estimation

Recuperado de: https://arxiv.org/pdf/1902.09212.pdf

[13]Osokin, D. (ArXiv 2018). Real-time 2D Multi-Person Pose Estimation on CPU: Lightweight OpenPose.

Recuperado de: https://arxiv.org/pdf/1811.12004.pdf

[14] Bin, Xiao, Haiping Wu, Yichen Wei (ECCV 2018). Simple Baselines for Human Pose Estimation and Tracking.

*Recuperado de:*https://openaccess.thecvf.com/content_ECCV_2018/papers/Bin_Xiao_Simple_Baselines_for_ECCV_2018_paper.pdf

[15] Definición de wikipedia sobre salto longitud

Recuperado de: https://es.wikipedia.org/wiki/Salto_de_longitud

[16] Problema de la asignación

Recuperado de: *https://es.wikipedia.org/wiki/Problema_de_la_asignaci%C3%B3n*

[17] Cmake GUI

Recuperado de: *https://cmake.org/*

[18] Visual Studio Community

Recuperado: https://visualstudio.microsoft.com/es/vs/community/

[19] Visual Studio Enterprise

Recuperado de: *https://visualstudio.microsoft.com/es/vs/enterprise/*