UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN MECATRÓNICA

TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

"SISTEMA DE ANÁLISIS DE PERFORMANCE DE UN VELOCISTA DE ESCALADA DEPORTIVA MEDIANTE VISIÓN ARTIFICIAL"

Trabajo de titulación previo a la obtención del título de Ingeniero en Mecatrónica

LÍNEA DE INVESTIGACIÓN: Desarrollo de prototipos biomecatrónicos

AUTOR:

Edisson Andres Vargas Tupiza

DIRECTOR:

MSc. Cosme Damián Mejía Echeverría.

Ibarra-Ecuador 2026

UNIVERSIDAD TÉCNICA DEL NORTE
DIRECCIÓN DE BIBLIOTECA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO	
APELLIDOS Y NOMBRES:	Vargas Tupiza Edisson Andres

DATOS DE LA OBRA	
TÍTULO:	"Sistema de análisis de performance de un velocista de escalada deportiva mediante visión artificial"
AUTOR:	Vargas Tupiza Edisson Andres
FECHA:	23/07/2026
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** □ **POSGRADO**
TÍTULO POR EL QUE OPTA:	Ingeniero en Mecatrónica
DIRECTOR /ASESOR:	MSc. Cosme Damián Mejía Echeverría PhD. Carlos Xavier Rosero Chandi

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 23 días del mes de julio de 2026

EL AUTOR:

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
EDISSON ANDRES VARGAS TUPIZA

Firma:……
Nombre: Vargas Tupiza Edisson Andres

Universidad Técnica Del Norte

Facultad de Ingeniería en Ciencias Aplicadas

Certificación del director del trabajo de grado

En mi calidad de director del trabajo de grado “Sistema de análisis de performance de un velocista de escalada deportiva mediante visión artificial”, presentado por el egresado Edisson Andres Vargas Tupiza, que opta por el título de Ingeniero en Mecatrónica, certifico que el mencionado proyecto fue realizado bajo mi dirección.

Ibarra, 23 de julio de 2026

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
COSME DAMIAN MEJIA ECHEVERRIA

MSc. Cosme Damián Mejía Echeverría.
Director de Tesis

Universidad Técnica Del Norte

Facultad de Ingeniería en Ciencias Aplicadas

APROBACIÓN DEL COMITÉ CALIFICADOR

El Tribunal Examinador del trabajo de titulación, “Sistema de análisis de performance de un velocista de escalada deportiva mediante visión artificial”, Elaborado por EDISSON ANDRES VARGAS TUPIZA, previo a la obtención del título de INGENIERO EN MECATRÓNICA, aprueba el presente informe de investigación en nombre de la Universidad Técnica del Norte:

Ibarra, 23 de julio de 2026

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
COSME DAMIAN MEJIA ECHEVERRIA

MSc. Cosme Damián Mejía Echeverría
Director de Tesis

Validar únicamente en FirmaEC.
Firmado electrónicamente por:
CARLOS XAVIER ROSERO CHANDI

PhD. Carlos Xavier Rosero Chandi
Asesor de Tesis

Dedicatorias

A mis padres, Enrique y Lidia, mi principal fuente de motivación y constancia; gracias por enseñarme que con sacrificio y disciplina se pueden lograr grandes objetivos.

A mis hermanos, por brindarme apoyo en todas las veces que lo necesité y demostrarme que el verdadero significado de la familia, siempre estará presente entre nosotros.

A mis sobrinos, por estar siempre presentes y contagiarme de su alegría, en momentos difíciles, su presencia bastaba para confortarme y seguir afrontando obstáculos y continuar con mis objetivos.

A mis amigos, a quienes los considero como mi propia familia, demostrando que las verdaderas amistades, durarán eternamente.

Agradecimiento

Agradezco a toda mi familia, padres, hermanos y sobrinos; ellos fueron el motor de todo mi esfuerzo, ellos me apoyaron en todo el transcurso que duró el logro de este objetivo.

A mi tutor, MSc. Cosme Mejía, le agradezco inmensamente por haberme guiado en toda esta experiencia de la elaboración de este proyecto; le agradezco por haberme comprendido y apoyado en circunstancias difíciles que se presentaron.

A todos los compañeros y amigos que hice en todo el tiempo de estadía en la universidad. me demostraron que cualquier persona puede lograr grandes objetivos, pero si haces amigos que tienen ese mismo objetivo, se lo logrará más rápido.

A todos los docentes, que compartieron su conocimiento y experiencia, haciendo que este proceso sea una experiencia enriquecedora y memorable.

A todas esas grandiosas personas, deportistas y entrenadores de la escuela de escalada de Imbabura, que me brindaron de su colaboración sin cuestionamiento alguno, fueron una parte muy importante para la creación y validación de este proyecto.

TABLA DE CONTENIDOS

Identificación de la obra..... I

Constancias..... I

Certificación del director del trabajo de grado.....II

Aprobación del comité calificador..... III

Dedicatorias IV

Agradecimiento V

Índice de figuras..... XI

Índice de tablas XIII

Resumen XIV

Abstract XV

Introducción 1

1.1 Planteamiento del problema 1

1.2 Objetivos.....2

1.2.1 Objetivo General 2

1.2.2 Objetivos Específicos 2

1.3 Alcance 2

1.4 Justificación 2

Marco referencial 4

2.1 Antecedentes 4

2.2 Bases Teóricas 7

2.2.1 Escalada Deportiva 7

2.2.2 Tipos de Escalada Deportiva 7

2.2.3 Escalada deportiva de molalidad velocidad. 8

2.2.4 Estudio de casos: Hombres y Mujeres en esta disciplina. 9

2.2.5 Biomecánica. 12

2.2.6 Tecnologías de inteligencia artificial. 19

2.2.7 Visión Artificial: fundamentos y aplicaciones. 20

Marco metodológico 24

3.1 Modelo de la Investigación 24

3.2 Diseño de la Investigación 24

3.2.1 Fase 1: Conceptualización de aspectos a considerar en el análisis del rendimiento del deportista 24

3.2.2 Fase 2: Diseño del sistema con visión artificial. 25

3.2.3 Fase 3: Implementación del prototipo del sistema. 26

3.2.4 Fase 4: Validación del sistema en circunstancias reales. 26

Resultados y análisis 27

4.1 Recolección y análisis de datos preliminares 27

4.1.1 Resultados y estudio de la encuesta 28

4.1.2 Obtención de material audiovisual 29

4.1.3 Parámetros que se considerarán en el análisis del rendimiento 30

4.1.4 Parámetros que se requieren conocer como resultado del análisis 31

4.2 Especificaciones del sistema a diseñar 32

4.2.1 Requerimientos funcionales 32

4.2.2 Restricciones operativas 32

4.3 Propuestas de solución 33

4.3.1 Diagrama de caja negra 33

4.3.2 Descomposición funcional 34

4.3.3 Matriz morfológica 36

4.3.4 Propuesta de solución Nro. 1 39

4.3.5 Propuesta de solución Nro. 2 40

4.3.6 Selección de la mejor propuesta 41

4.3.7 Ponderación final 45

4.3.8 Descripción general de la solución 45

4.4 Selección del protocolo o arquitectura 46

4.4.1 Flujo de procesamiento de datos del sistema 47

4.5 Desarrollo y diseño del sistema 47

4.5.1 Interfaz 48

4.5.2 Implementación de la detección de pose y seguimiento del deportista 50

4.5.3 Detección de presas 51

4.5.4 Obtención de métricas y resultados 52

4.6 Análisis de datos 54

4.6.1 Comparación de métricas de rendimiento 54

4.6.2 Interpretación de resultados del análisis 55

4.7 Pruebas de funcionamiento y validación del sistema 57

4.7.1 Pruebas de funcionamiento con entornos controlados 58

4.7.2 Validación del sistema en circunstancias reales 61

Conclusiones y recomendaciones 63

5.1 Conclusiones 63

5.2 Recomendaciones 64

Bibliografía.....65

Anexos ……...……...……..67

ÍNDICE DE FIGURAS

Fig 2.1 Escalada deportiva: Un escalador realizando su participación en competencias [13]. 7

Fig 2.2 Récords mundiales de la escalada deportiva de velocidad: (a) La escaladora Aleksandra Miroslaw y (b) el escalador Samuel Watson obteniendo el récord mundial [13]. 11

Fig 2.3 Récords olímpicos de la escalada deportiva de velocidad: (a) La escaladora Aleksandra Miroslaw y (b) el escalador Bassa Mawem obteniendo el récord olímpico [13]. 12

Fig 2.4 Planos y ejes de referencia y términos de orientación asociados: 1. Plano medial; 2. Plano sagital; 3. Plano transversal; 4. Ventral; 5. Plano frontal; 6. Medial; 7. Lateral; 8. Craneal; 9. Caudal; 10. Dorsal; según [15]. 14

Fig 2.5 Movimiento de rotación de un objeto (no deformable). M1: posición1; M2: Posición2; $\boldsymbol{\theta 1}$ y $\boldsymbol{\theta 2}$: angulaciones del desplazamiento; $\boldsymbol{\Delta\theta}$: diferencia de angulación; $\boldsymbol{t1}$ y $\boldsymbol{t2}$: tiempos de desplazamiento; $\boldsymbol{O}$: origen del eje de rotación; según [15]. 16

Fig 2.6 En un desplazamiento lineal el punto $\boldsymbol{O1}$ se desplaza en un eje horizontal, mas no en un eje vertical; según [15]..... 17

Fig 2.7 La inteligencia artificial abarca un gran número de disciplinas tecnológicas, en las cuales se van mejorando con el pasar del tiempo [16]. 20

Fig 2.8 Etapas del procesamiento de imágenes para un sistema de visión artificial. 21

Fig 4.1 Aspectos que se utilizan para medir el rendimiento de un deportista de escalada de velocidad: (a) resultados según el rol del encuestado y (b) resultados según la experiencia del encuestado. 29

Fig 4.2 Herramientas tecnológicas y métodos utilizados por deportistas y entrenadores para el

análisis de rendimiento..... 30

Fig 4.3 Diagrama de caja negra para el sistema a realizar. 33

Fig 4.4 Descomposición funcional del diagrama de caja negra basado en el principio de procesamiento de imágenes y los resultados requeridos. 34

Fig 4.5 Estructura en clases y funciones para la creación de la interfaz. 46

Fig 4.6 Diagrama de flujo de procesamiento de datos del sistema. 47

Fig 4.7 Ventana principal de la interfaz de usuario realizada. 48

Fig 4.8 Ventana asignada a la visualización de la obtención de métricas. 49

Fig 4.9 Ventana que muestra el análisis general y la obtención de resultados..... 49

Fig 4.10 Detección de pose y seguimiento del escalador. 51

Fig 4.11 Pruebas de predicciones del modelo creado para detección de presas. 52

Fig 4.12 Visualización de las métricas obtenidas. 53

Fig 4.13 Graficación de la trayectoria del escalador. 53

Fig 4.14 Métricas obtenidas que se visualizan en la interfaz 54

Fig 4.15 Diseño final de la ventana que muestra los resultados del análisis al usuario. 57

Fig 4.16 Realización de pruebas de funcionamiento utilizando material audiovisual obtenido por el tesista……59

Fig 4.17 Casos donde el sistema diseñado falla por el material audiovisual que no cumple con el formato establecido: (a) En video muestra presas de la otra ruta y (b) el video no muestra la pared completa y posee iluminación demasiado alta. 60

Fig 4.18 Generación de error de análisis por diferente tamaño de video obtenido por el usuario. . 61

Fig 4.19 Realización de pruebas para validación del sistema. 62

ÍNDICE DE TABLAS

Tabla 2.1 TOP 5 DE ESCALADORAS DE VELOCIDAD 10

Tabla 2.2 TOP 5 DE ESCALADORES DE VELOCIDAD.....10

Tabla 4.1 Descripción de los parámetros que se necesitarán previamente.....31

Tabla 4.2 Resultados del análisis del sistema que se diseñará 31

Tabla 4.3 Descripción de las funciones parciales y enlistado de sus componentes. 35

Tabla 4.4 Matriz morfológica para propuestas de soluciones. 37

Tabla 4.5 Subfunciones del sistema que se diseñará en la propuesta Nro. 1 39

Tabla 4.6 Subfunciones del sistema que se diseñará en la propuesta Nro. 2 40

Tabla 4.7 Criterios de diseño que se utilizarán en la evaluación.....41

Tabla 4.8 Ponderación de criterios 42

Tabla 4.9 Valoración resultante de la ponderación de criterios 42

Tabla 4.10 Ponderación de las propuestas para el criterio C3.....43

Tabla 4.11 Ponderación de las propuestas para el criterio C1.....43

Tabla 4.12 Ponderación de las propuestas para el criterio C2.....43

Tabla 4.13 Ponderación de las propuestas para el criterio C8.....43

Tabla 4.14 Ponderación de las propuestas para el criterio C7.....44

Tabla 4.15 Ponderación de las propuestas para el criterio C6.....44

Tabla 4.16 Ponderación de las propuestas para el criterio C4.....44

Tabla 4.17 Ponderación de las propuestas para el criterio C5.....44

Tabla 4.18 Ponderación final.....45

Tabla 4.19 Puntos de referencia utilizados en el sistema 50

Tabla 4.20 Registro de datos 55

RESUMEN

Aplicar la tecnología que se ha estado desarrollando a lo largo de los años hacia el deporte, ha traído excelentes resultados a los deportistas, incrementando la probabilidad de mejorar su rendimiento, y con eso conseguir importantes logros. El deporte de escalada de velocidad es practicado por un gran número de personas dentro del país, especialmente en la ciudad de Ibarra, donde se forman excelentes deportistas que han conseguido importantes logros nacionales e internacionales. A pesar de que los deportistas consiguen grandes resultados, uno de los factores que ha limitado en conseguir su máximo potencial es el de no poseer un sistema que sea capaz de analizar el performance de un escalador de manera cuantitativa. El objetivo de este proyecto de titulación es diseñar un prototipo de un sistema que analice el performance de un velocista de escalada deportiva utilizando tecnologías de visión artificial, que brinde de resultados cuantitativos a deportistas y entrenadores, y que con ellos sean capaces de identificar aspectos a favor y en contra en la práctica de su disciplina.

Como resultado se obtiene una aplicación de escritorio la cual compara a un escalador de referencia con el escalador que se requiere analizar, y muestra al usuario gráfica y numéricamente aspectos de tiempo, velocidad, trayectoria y cinemática del cuerpo humano; finalmente, las pruebas de funcionamiento de este sistema concluyen que estos resultados contribuyen al análisis del performance del velocista de escalada deportiva.

Palabras clave: Sistema, Performance, Escalada de velocidad, visión artificial.

ABSTRACT

Applying technology that has been developed over the years to sports has brought excellent results for athletes, increasing the probability of improving their performance and thereby achieving significant accomplishments. Speed climbing is practiced by a large number of people within the country, especially in the city of Ibarra, where excellent athletes are trained who have achieved significant national and international accomplishments. Although athletes achieve great results, one of the factors that has limited them from reaching their full potential is the lack of a system capable of quantitatively analyzing a climber's performance. The objective of this degree project is to design a prototype system that analyzes the performance of a speed sport climbing sprinter using artificial vision technologies, providing quantitative results to athletes and coaches, enabling them to identify strengths and weaknesses in the practice of their discipline.

The result is a desktop application that compares a reference climber with the climber to be analyzed, and shows the user graphical and numerical aspects of time, speed, trajectory, and kinematics of the human body. Finally, the performance tests of this system conclude that these results contribute to the analysis of the performance of the speed sport climbing sprinter.

Keywords: System, Performance, Speed climbing, artificial vision.

Capítulo I

Introducción

1.1 Planteamiento del problema

La escalada deportiva es una actividad de alto rendimiento que radica en realizar ascensos sobre paredes utilizando la fuerza física. Una de las modalidades de este deporte es la de velocidad que consiste en escalar una pared estandarizada internacionalmente para presionar un pulsador, en el menor tiempo posible, después de un pitido de arranque [1].

Durante el entrenamiento de esta modalidad, se debe analizar la evolución de deportista mediante el estudio de una serie de aspectos como: su movimiento biomecánico, el tiempo de escalada por etapas y el recorrido en la ruta. Esto tiene como propósito evaluar el rendimiento del deportista, de manera que pueda obtener mejores resultados en las competencias futuras.

Actualmente, estos aspectos se monitorean de forma manual ya que no existen dispositivos tecnológicos especializados en este ámbito deportivo y, en términos de investigación, se ha avanzado muy poco. Esa situación genera complicaciones al momento de realizar el estudio de los aspectos antes mencionados, y como consecuencia de ello, no es posible saber con exactitud el avance del rendimiento del deportista.

Se ha comprobado que, mediante el uso de nuevas tecnologías aplicadas al deporte, el rendimiento de los deportistas aumenta, logrando obtener un buen desempeño en competencias. Sin embargo, estas tecnologías aún no alcanzan a cubrir a todos los deportes con sus diversos métodos de entrenamiento.

En este trabajo propuesto por el Ing. Cosme Mejía, desde la línea de investigación de desarrollo de prototipos biomecatrónicos, se busca utilizar conceptos y técnicas de la carrera de ingeniería Mecatrónica para proponer una alternativa de solución que intente resolver el problema.

1.2 Objetivos

1.2.1 Objetivo General

- Desarrollar un sistema para el análisis de performance de un velocista de escalada deportiva mediante la utilización de visión artificial.

1.2.2 Objetivos Específicos

- Conceptualizar los aspectos que se consideran en la evaluación del rendimiento del velocista de escalada deportiva en la práctica de la actividad.
- Diseñar un sistema con visión artificial para la obtención de datos de la escalada del deportista.
- Implementar un prototipo funcional del sistema.
- Validar el sistema bajo circunstancias reales.

1.3 Alcance

El presente proyecto comprende una etapa de investigación que permita conceptualizar los aspectos más importantes a tomar en cuenta en el entrenamiento y movimiento sobre la pared de un velocista de escalada deportiva. A partir de esta información se procederá a diseñar el sistema con visión artificial para obtener una serie de datos que se utilizarán para estudiar el performance de un velocista de escalada deportiva. Posteriormente, se construirá un prototipo funcional que permita demostrar el funcionamiento en circunstancias reales.

1.4 Justificación

En el ámbito social, al aplicar los conceptos, métodos y técnicas de la ingeniería en el deporte, se generaría un mejor desempeño en las actividades deportivas que practican los miembros de la comunidad. En la escalada deportiva de velocidad podría incentivar a las personas que han estado interesadas en este tipo de deportes, a seguir una carrera deportiva, brindándoles la oportunidad de convertirse en deportistas élite; y a la vez, incrementar la competencia y el desarrollo del deporte.

En el ámbito deportivo, este sistema optimizaría el nivel en esta disciplina, debido a que facilitará el estudio hacia el deportista para incrementar su rendimiento, y posteriormente, encontrar las mejores condiciones para que éste compita.

En el ámbito investigativo y de avances tecnológicos, este proyecto abriría campo a nuevas investigaciones relacionadas con la escalada deportiva, las que pueden abarcar temas de seguridad y/o integridad y salud física.

En el ámbito económico, se pretende desarrollar un sistema que sea accesible ya que, por lo general, los recursos económicos destinados al ámbito deportivo son muy limitados.

Capítulo II

Marco referencial

2.1 Antecedentes

La aplicación de la tecnología hacia el deporte ha hecho que cada vez se implementen métodos que se centran en el estudio del performance de un deportista. Dentro de la escalada deportiva de velocidad se ha desarrollado algunas investigaciones para el mejoramiento de esta disciplina. A continuación, se describen una serie de antecedentes que aportaran a la realización de la investigación planteada.

En los juegos olímpicos de la Juventud realizados en 2018, se investigó sobre el rendimiento de los deportistas en el transcurso del evento de escalada de velocidad, con el objetivo de analizar la cinemática de la trayectoria de sus pelvis, que es el área donde se encuentra su centro de masa corporal. Esto se realizó con 42 deportistas (21 hombre y 21 mujeres) de distintos países y edades comprendidas entre 14 y 18 años; se utilizó una cámara fijada en una plataforma de 3 ejes colocada de manera frontal a la pared y controlada mediante cables; dando como resultado diferencias en el rendimiento por la capacidad de generar menos desplazamientos laterales para aumentar fluidez, mantener niveles de velocidad altos en todas las fases de la escalada y evitar la disminución de la velocidad en cada movimiento del cuerpo del deportista [1].

Se presenta un estudio acerca del análisis del movimiento y su visualización tridimensional de un deportista al momento de realizar la escalada de velocidad. Esto se implementó mediante la medición geométrica en tercera dimensión (3D) que requería la integración de dos o más señales bidimensionales (2D) lo que genera la obtención de

resultados en tercera dimensión (3D); esto se realizó mediante el uso de dos drones que realizaban el seguimiento del atleta en el transcurso del ascenso. Esto generó como resultado la obtención de un perfil de velocidad 3D a partir de una marca en el arnés del atleta, indicando fases críticas donde es necesario estudiarlas [2].

Un trabajo juntó los avances de visión por ordenador y el aprendizaje profundo para la investigación en el análisis deportivo basado en video y esqueleto centrado en la escalada deportiva de velocidad. Esto se realizó mediante un conjunto de datos de 362 actuaciones de escalada de velocidad de 55 atletas que posteriormente se las transformó a secuencias de esqueletos 2D; lo que demuestra la elevada capacidad de describir y utilizar los datos de esqueletos 2D para analizar a todos los atletas según su estilo de escalada [3].

De una forma general, se ha propuesto nuevos sistemas de medición de movimientos y fuerza en 3D para la escalada deportiva, esto se implementa mediante la medición y el análisis de los movimientos geométricos y dinámicos de los escaladores; generando como resultado la verificación de que las habilidades de escalada de un deportista se incrementan mediante el uso de un sistema que analice su rendimiento [4].

También se puede verificar que el uso de visión artificial en el deporte beneficia al rendimiento del deportista, ya que se llevó a cabo un estudio para aplicar esta tecnología en el análisis y la evaluación de las posturas y movimientos de los deportistas en el ejercicio de sentadillas. Esto se llevó a cabo mediante una técnica de aprendizaje automático supervisado que consiste en recopilar un conjunto de datos de posturas correctas o incorrectas del ejercicio de sentadilla, los cuales se comparan con un video de la realización de este ejercicio por parte del deportista, y se determina si este realiza correctamente el ejercicio o no; y se ofrece recomendaciones sobre como corregir la técnica en caso de que el deportista realice incorrectamente el ejercicio. Se concluye que la visión artificial mejora el rendimiento deportivo, debido a que, proporcionar información visual a los deportistas sobre su ejecución de entrenamiento genera un entrenamiento más efectivo y a su vez, motiva al deportista [5].

Por otra parte, se ha evidenciado la necesidad que tiene este deporte en aplicarse las nuevas tecnologías que cada vez se van creando y/o mejorando, ya que se realizó una investigación con el fin de analizar posturas y movimientos que genera la práctica de escalada deportiva mediante la utilización de un Microsoft Kinect para obtener animaciones realistas de escalada 3D. En esta investigación se llevó a cabo mediante la división de un ciclo de escalada en cuatro fases y gracias a un escalador profesional que participó, se analizó al escalador en cada una de estas fases obteniendo buenos

movimientos para las animaciones de movimiento y posturas, lo cual servirá de una manera pedagógica a escaladores principiantes [6].

Presentemente, se analiza el movimiento de un deportista de escalada deportiva de velocidad mediante un video y se toma una conclusión dependiendo de la experiencia de los entrenadores, por lo que se propuso un método novedoso que permite analizar secuencias de movimiento y técnicas mediante la combinación de secuencias de video pregrabadas con un algoritmo de detección de características y puntos clave del cuerpo humano y condiciones de contorno normalizadas. Esto permite considerablemente el análisis del entrenamiento y competición de un deportista de escalada deportiva de velocidad, ayudando a la par a atletas y entrenadores [7].

Un artículo presentó una plataforma portátil para detección y un marco de análisis para evaluar el rendimiento de manera general de la escalada de un deportista durante el entrenamiento utilizando un único sensor 3d en miniatura basado en un acelerómetro que se lleva en la oreja, lo que hace que las medidas detectadas como fluidez de movimiento, fuerza y resistencia puedan cuantificar el nivel de desempeño general del deportista. Se demuestra que el sensor utilizado es fácil de utilizar y que las señales obtenidas se correlacionan de una buena manera con las técnicas del deportista, y con ello se puede evaluar la progresión del rendimiento con respecto a su entrenamiento [8].

Se creó un sistema que permite detectar los picos de fuerza en cada una de las presas de escalada, tanto en los pies como en las manos, y con ellos se registra la posición y los movimientos del cuerpo de la persona que práctica el deporte. Esto de llevó a cabo mediante la equipación de cada presa con un sensor de fuerza de tres ejes además de la combinación con un sistema de captura de movimiento tridimensional basado en una cámara de profundidad para detectar la fuerza de los dedos, la temperatura, entre otros.; con los datos obtenidos se estudiaron los valores atípicos o que tenían mayor desviación ya que estos valores podrían representar una mala sujeción en el sensor o la presencia se signos de fatiga y se buscó resolver con el fin de mejorar la técnica, el rendimiento y la prevención de lesiones del deportista de escalada [9].

Por último, se realizó un estudio estadístico de una serie de datos generados con la aplicación de algoritmos de procesamiento de imágenes y redes neuronales a un conjunto de videos de competiciones y seguido de ello se realizó un análisis de velocidad y trayectoria del centro de gravedad en secciones particulares de la ruta. Esto se llevó a cabo mediante la combinación de los parámetros calculados y las variables aleatorias para la implementación de un modelo lineal, este modelo permite predecir el tiempo final de diferentes atletas y determinar su correlación con las variables mencionadas. Se

concluye que, conociendo ciertos parámetros como la trayectoria y la velocidad en las secciones de la ruta, mediante el análisis del modelo lineal se obtiene una correlación del tiempo final con la técnica de salida ejecutada [10].

2.2 Bases Teóricas

2.2.1 Escalada Deportiva

La escalada o escalada deportiva se define como una sucesión de movimientos corporales acíclicos, con la finalidad de desplazar el centro de gravedad en una dirección de avance, lo cual puede ser de ascenso, descenso o travesía; y también, busca mantener el equilibrio tanto en posición estática como dinámica [11] , como se muestra en la figura 2.1.



Fig 2.1 Escalada deportiva: Un escalador realizando su participación en competencias [13].

2.2.2 Tipos de Escalada Deportiva

Dentro del deporte de la escalada deportiva, hay una gran cantidad de modalidades, pero dentro de las competiciones de los juegos olímpicos hay tres tipos de competiciones en las que se compiten [12]:

2.2.2.1 Escalada de velocidad.

Los deportistas que se enfrentan escalan una pared de 15 metros para ver quien llega más rápido. Esta pared está estandarizada y tiene una inclinación de 5 grados; además, los deportistas están sujetados con un sistema de cuerdas desde la parte superior para evitar accidentes.

2.2.2.2 Escalada de dificultad.

En este tipo de escalada, los deportistas tienen un reto de intentar llegar a la cima de una pared en seis minutos. Aunque la pared de esta modalidad puede ser de la misma altura que de la modalidad de velocidad, su diferencia radica en que la ascensión de esta pared es más compleja, debido a que el rocódromo y los agarres son diferentes para cada competición, con agarres difíciles de sujetar y una pared con huecos y salientes pronunciadas, además de una inclinación de hasta sesenta grados.

2.2.2.3 Escalada Búlder.

Esta modalidad se practica solo dentro de una pared de 4.5 metros de altura, los deportistas no llevan cuerda de seguridad y deben escalar esta pared varias veces. La pared más corta está equipada con cuatro o cinco rutas fijas, las cuales también se las llama “Problemas”. El deportista debe escalar varias rutas en un tiempo determinado de normalmente cinco o seis minutos, pudiendo tomar descanso entre “Problemas”.

2.2.3 Escalada deportiva de molalidad velocidad.

El deporte consiste en que, dos escaladores, asegurados por la parde de arriba de una pared de 15 metros de altura, escalan rutas paralelas estandarizas; y con ello, el escalador más rápido será el ganador cuando este apriete un interruptor que los escaladores deben golpear al llegar a la cima para detener el cronómetro que mide hasta las centésimas de segundo [13].

2.2.3.1 Lineamientos de paredes homologadas.

Dentro de la escalada deportiva de modalidad de velocidad se tiene una serie de especificaciones que deben cumplir las paredes donde se van a desarrollar dicho deporte, ya sea de entrenamiento y/o obligatoriamente en competencias validadas por la IFSC. Las especificaciones de la pared son las siguientes [13]:

- Altura:
    - 15,00 m
    - Los rocódromos comienzan a 20 cm del suelo y deben continuar al menos 50 cm por encima del último panel.

- Ancho:
    - Dos rocódromos de 3,00 m, uno al lado del otro.
    - Un espacio de 1,00 m a cada lado del rocódromo, libre de cualquier construcción.
    - Los rocódromos pueden estar adyacentes o separados, pero la separación no debe superar los 50 cm.
- Punto de protección superior individual (ITRS).
    - Los puntos de protección superior individual se instalan en una estructura metálica entre 500mm y 1000mm verticalmente y horizontalmente hacia afuera desde el borde superior del panel.
    - Este ITRS deberá estar situado en el eje central de la ruta con una tolerancia de +/- 50mm.
- Generalidades
    - El rocódromo debe constar por una serie de paneles de 1500 mm de alto y 1500 mm de ancho.
    - Las pardes de escalada deben sobresalir regularmente 5° con una tolerancia media de -0,5°+1,5°.
    - El botón de finalización debe estar situado a 14.800 mm del suelo con una tolerancia de +/-20mm.
    - El color del rocódromo deberá ser de un gris claro.
    - El rocódromo deberá estar libre de cualquier tipo de logos.

2.2.4 **Estudio** de **casos: Hombres y Mujeres en esta disciplina.**

2.2.4.1 Top 5 de escaladores de velocidad.

Actualmente, según [13], en esta disciplina las clasificaciones mundiales de la escalada deportiva modalidad de velocidad tanto para hombres como para mujeres se está clasificándose como se muestra en la **Tabla 2.1** y **Tabla 2.2**:

Tabla 2.1 **TOP 5 DE ESCALADORAS DE VELOCIDAD**

N°	Nombre	País	Rango	Evento	Ciudad	Fecha
1	Aleksandra Kalucka	Polonia	1	Olympic Qualifier Series Budapest 2024	Budapest	23-06-2024
2	Natalia Kalucka	Polonia	9	Olympic Qualifier Series Budapest 2024	Budapest	23-06-2024
3	Emma Hunt	USA	1	IFSC World Cup Salt Lake City 2024	Salt Lake City	06-05-2024
4	Rajiah Sallsabillah	Indonesia	3	Olympic Qualifier Series Budapest 2024	Budapest	23-06-2024
5	Aleksandra Miroslaw	Polonia	1	IFSC World Cup Wujiang 2024	Wujiang	14-04-2024

Nota: Top 5 de las personas de género femenino para la escalada de velocidad [13].

Tabla 2.2 **TOP 5 DE ESCALADORES DE VELOCIDAD**

N°	Nombre	País	Rango	Evento	Ciudad	Fecha
1	Peng Wu	China	1	Olympic Qualifier Series Budapest 2024	Budapest	23-06-2024
2	Matteo Zurloni	Italia	4	IFSC World Cup Salt Lake City 2024	Salt Lake City	06-05-2024
3	Samuel Matson	USA	1	IFSC World Cup Salt Lake City 2024	Salt Lake City	06-05-2024
4	Rahmad Adi Mulyono	Indonesia	23	IFSC World Cup Wujiang 2024	Wujiang	14-04-2024
5	Rishat Khaibullin	Kazajistán	6	Olympic Qualifier Series Budapest 2024	Budapest	23-06-2024

Nota: Top 5 de las personas de género masculino para la escalada de velocidad [13].

2.2.4.2 Récords Mundiales.

Para la disciplina de escalada de velocidad de mujeres se tiene que, el 15 de septiembre del 2023, en las IFSC EUROPEAN QUILIFIER ROME 2023, Aleksandra Miroslaw de nacionalidad polaca, logró el récord mundial con el tiempo de escalada de 6.24 s. En las competiciones de esta disciplina para hombres el récord mundial lo tiene Samuel Watson de nacionalidad estadounidense, ya que en las IFSC World Cup Wujiang 2024 logró un tiempo de escalada de 4.79s [13]; los cuales se muestra en la figura 2.2.

(a) (b)



Fig 2.2 Récords mundiales de la escalada deportiva de velocidad: (a) La escaladora Aleksandra Miroslaw y (b) el escalador Samuel Watson obteniendo el récord mundial [13].

2.2.4.3 Récords Olímpicos.

En las competiciones olímpicas de escalada deportiva de modalidad de velocidad se tiene que, dentro de las competiciones femeniles, Aleksandra Miroslaw de nacionalidad polaca posee el récord olímpico de 6.84 s logrado el 6 de agosto de 2020 en los Olympic Games Tokyo 2020. Mientras que, Bassa Mawem de nacionalidad francesa logró también en los Olympic Games Tokyo 2020 el récord olímpico de escalada de 5.45s en el día 3 de agosto de 2020 [13]; en la figura 2.3 se puede visualizar a estos competidores.

(a) (b)



Fig 2.3 Récords olímpicos de la escalada deportiva de velocidad: (a) La escaladora Aleksandra Miroslaw y (b) el escalador Bassa Mawem obteniendo el récord olímpico [13].

2.2.5 Biomecánica.

La biomecánica es considerada como una disciplina que estudia y elabora análisis físicos de sistemas biológicos, con ello, los movimientos del cuerpo humano. Cuando esta disciplina se aplica al deporte, su objetivo es la caracterización y la mejora de las técnicas del movimiento a partir de conocimientos científicos y también, con el progreso de las técnicas de medición, almacenamiento y procesamiento de datos; con esto, actualmente esta disciplina tiene una importancia relevante ya que realiza innumerables aportaciones al deporte, como: el mejoramiento de la técnica y el rendimiento, evitar lesiones, entre otros [14].

2.2.5.1 Sistemas de referencia del cuerpo humano.

Según [15], los sistemas de referencia permiten situar los segmentos corporales unos respecto a otros y el cuerpo en su entorno para facilitar la descripción y el estudio del movimiento. Por eso, para el estudio de la biomecánica, se aplican dos métodos de referenciación para la anatomía del cuerpo humano:

- Sistema de referencia terrestre (modelo geocentrado):

  Cualquier cuerpo u objeto en la tierra está sometido a la gravedad, por ende, su magnitud y dirección son constantes lo que hace que se puedan obtener cierta cantidad de magnitudes físicas para un análisis; aun así, este sistema de referencia es ineficaz para describir los movimientos del cuerpo.

- Sistema de referencia a partir del cuerpo (modelo egocentrado):

Un método mucho más práctico es delimitar a nuestro propio cuerpo como un sistema de referencia, para orientarnos y desplazarnos en el espacio. Este sistema de referencia nos permite situarnos en relación con los demás o con los objetos o viceversa, situar a los demás o a los objetos en relación con la nuestra. La orientación a partir del cuerpo nos pone en relación con el entorno; sin embargo, se necesita una posición de referencia inicial a partir de la cual se organiza todo el sistema de referencia.

2.2.5.2 Anatomía funcional.

Para el análisis del movimiento del cuerpo humano se requiere de la aplicación de determinados fundamentos de anatomía funcional como son las posiciones, planos, y ejes de referencia y de segmentos corporales [15], mostrados en la figura 2.4:

- Posición de referencia: comúnmente se utilizan dos posiciones de referencia: la posición anatómica, utiliza un método estandarizado para la posición que describe la anatomía y la biomecánica como el cuerpo humano de pie, con los pies juntos y en paralelo, las extremidades superiores colgando a lo largo del cuerpo y los antebrazos en supinación (palmas orientadas hacia adelante), con una mirada recta y horizontal, aunque esta descripción sitúa al cuerpo humano en posición de pie, sigue siendo válida en cualquier posición; y la posición neutral, consiste en las mismas posiciones, con la diferencia que en los antebrazos, que se encuentran en posición neutra de prosupinación (con los pulgares orientados hacia adelante). Esta diferencia hace que la posición neutra sirva como referencia para la medición angular de la movilidad articular y como una posición de partida para una serie de movimientos, actividades y ejercicios [15].
- Planos y ejes de referencia: el estudio del cuerpo humano se lo realiza de acuerdo a tres planos fundamentales del espacio [15]:
    - Plano frontal/eje sagitotransversal o antepostior: divide al cuerpo en dos partes, una parte anterior y una parte posterior.
    - Plano sagital/ eje frontotransversal o transversal: divide al cuerpo en dos partes, una parte derecha y otra izquierda.
    - Plano transversal/ eje frontosagital o longitudinal: divide al cuerpo en dos partes, una parte superior y otra parte inferior.

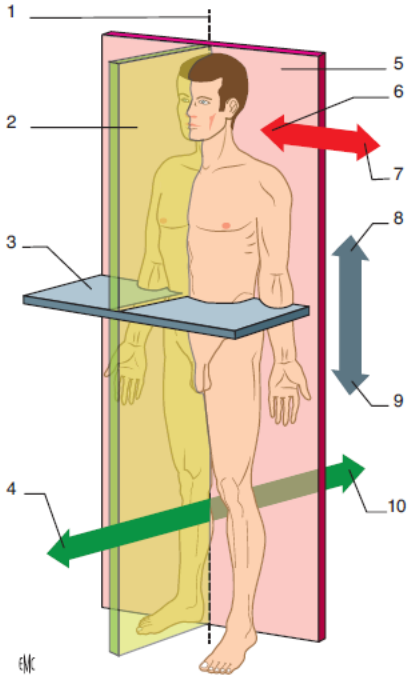


Fig 2.4 Planos y ejes de referencia y términos de orientación asociados: 1. Plano medial; 2. Plano sagital; 3. Plano transversal; 4. Ventral; 5. Plano frontal; 6. Medial; 7. Lateral; 8. Craneal; 9. Caudal; 10. Dorsal; según [15].

- Segmentos: el cuerpo humano se integra por segmentos unidos por articulaciones. El cuerpo humano consiste por los segmentos de la cabeza y el tronco, unidos por la región de la cabeza y el cuello, además de los segmentos de las extremidades superiores (brazo, antebrazo y mano, unidos por regiones del hombro, codo y muñeca) y las extremidades inferiores (muslo, pierna y pie, unidos por las regiones de la cadera, la rodilla y el tobillo) [15].
- Movimientos: una de las definiciones para el movimiento es el desplazamiento, la modificación continua de posición de un cuerpo o de una parte del mismo, en el espacio y el tiempo [15].

2.2.5.3 Cinemática

La cinemática se define como el estudio del movimiento de los cuerpos independientemente de las causas que lo producen. Para el estudio de la cinemática se requieren magnitudes físicas como el tiempo, la posición (conocer su localización en el espacio en un momento específico), la velocidad (la variación de la posición por unidad de tiempo) y la aceleración (la variación de la velocidad por

unidad de tiempo) [15].

- Cinemática lineal (en una dimensión).
    - Velocidad de un móvil.

En Ec. 2.1 [15], el desplazamiento lineal se refiere a aquel desplazamiento que está formado por una línea recta entre dos puntos. Se mide en metros y representa la rapidez del movimiento o del desplazamiento, ya sea lineal o angular. Se utiliza la fórmula de Velocidad media V_m en $^m/_s$:

$$V_m = \frac{\Delta x}{\Delta t} \qquad \text{Ec. 2.1}$$

donde

V_m= Velocidad media $[^m/_s]$

Δx= Variación de distancia $[m]$

$\Delta t =$ Variación de tiempo $[s]$

- Aceleración de un móvil.

Se utiliza la fórmula de la aceleración media, ya que cuando la velocidad media V_m varía, es posible conocer la rapidez con que lo hace conociendo su aceleración a_m, esta indica como varía V_m en un segundo. Se utiliza la fórmula de la Ec. 2.2 [15], de aceleración media a_m en $^m/_{s^2}$:

$$a_m = \frac{\Delta V_m}{\Delta t} \qquad \text{Ec. 2.2}$$

donde

a_m= Aceleración media $[^m/_{s^2}]$

ΔV_m = Variación de la velocidad media $[^m/_s]$

$\Delta t =$ Variación de tiempo $[s]$

- Cinemática angular (en dos-tres dimensiones):

Si se necesita saber la posición de un objeto que experimenta un movimiento de rotación en un momento t determinado, esto se define con su posición angular, tal como se visualiza en la figura 2.5:

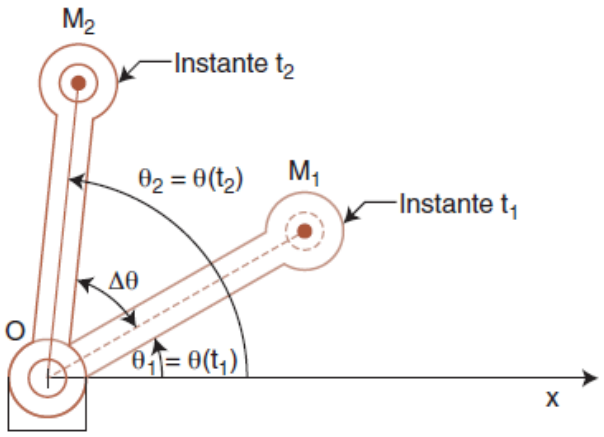


Fig 2.5 Movimiento de rotación de un objeto (no deformable). M1: posición1; M2: Posición2; $\theta 1$ y $\theta 2$: angulaciones del desplazamiento; $\Delta\theta$: diferencia de angulación; $t1$ y $t2$: tiempos de desplazamiento; O: origen del eje de rotación; según [15].

- Velocidad angular o velocidad de rotación ω:

  Definidos dos instantes $t1$ y $t2$ en la Ec. 2.3 [15], es posible definir una velocidad angular media ω_m, la cual se define como la derivada con respecto al tiempo de la posición angular y se expresa en $^{rad}/_{s}$:

$$\omega_m = \frac{\theta_2 - \theta_1}{t_2 - t_1} = \frac{\Delta\theta}{\Delta t} \qquad \text{Ec. 2.3}$$

donde

ω_m= Velocidad angular media $[^{rad}/_{s}]$

$\Delta\theta$ = Variación angular $[rad]$

Δt = Variación de tiempo $[s]$

Si el intervalo $(t_2 - t_1)$ es muy pequeño, se obtiene, en un tiempo t, la velocidad angular instantánea en Ec. 2.4 [15]:

$$\omega(t) = \lim_{\Delta_t \to 0} \frac{\Delta\theta}{\Delta t} = \frac{d\theta(t)}{dt} = \theta'(t) \qquad \text{Ec. 2.4}$$

donde

$\omega(t)$= Velocidad angular $[^{rad}/_{s}]$

$\Delta\theta$ = Variación angular $[rad]$

Δt = Variación de tiempo $[s]$

$\theta'(t)$= derivada de la posición angular con respecto al tiempo $[^{rad}/_{s}]$

- Aceleración angular α:

  Esta magnitud de obtiene a partir de la derivación de la velocidad angular, la cual se expresa en $^{rad}/_{s^2}$ en Ec. 2.5 [15]:

$$\alpha(t) = \frac{d\omega(t)}{dt} = \frac{d^2\theta(t)}{dt^2} = \theta''(t) \qquad \text{Ec. 2.5}$$

Dónde:

α_m= Aceleración angular media [$^{rad}/_{s^2}$]

$\theta''(t)$= segunda derivada de la posición angular con respecto al tiempo [$^{rad}/_{s^2}$]

2.2.5.4 Movimiento

Tomando como referencia a un cuerpo, el movimiento se define como el cambio de posición en el tiempo con relación a un punto de referencia considerándolo fijo [15]. En la figura 2.6 se puede explicar que, un objeto está en reposo o inmóvil en relación con un sistema de referencia si su posición no cambia con respecto a él en el tiempo; e inversamente, se considera como un objeto en movimiento en relación con un sistema de referencia si su posición cambia con respecto a él en el tiempo.

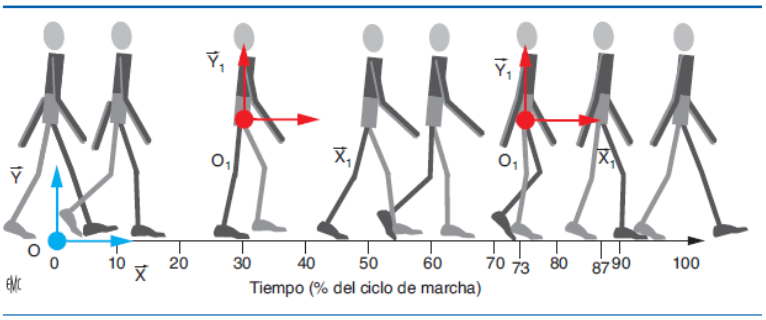


Fig 2.6 En un desplazamiento lineal el punto O_1 se desplaza en un eje horizontal, mas no en un eje vertical; según [15].

- Movimiento uniforme:

  Ocurre cuando un objeto experimenta un movimiento de traslación (rectilíneo o circular) en relación a un sistema de referencia, con ello, todos los puntos del sólido tendrán la misma velocidad. Un movimiento de traslación rectilíneo uniforme (Ec. 2.7 [15]) se produce sin aceleración (0 $^{m}/_{s^2}$) con una velocidad constante en el tiempo (Ec. 2.6[15]). Este movimiento también es conocido como MTRU.

$$a = 0$$

$$V = V_m = \frac{\Delta x}{\Delta t} = \frac{x - x_0}{t - t_0} = constante \quad \text{Ec. 2.6}$$

$$x = V.t + x_0 \quad \text{Ec. 2.7}$$

Dónde:

a= aceleración (m/s^2)

V= velocidad instantánea (m/s)

V_m= velocidad media (m/s)

t_0= instante inicial (s)

x_0= desplazamiento inicial (m)

- Movimiento uniformemente variado:

  Ocurre cuando el movimiento de la Ec. 2.10 o Ec. 2.11 [15], varía de manera uniforme cuando la aceleración de la Ec. 2.8 [15] de movimiento es constante, además de obtener la magnitud de la velocidad con la Ec. 2.9 [15]. Las ecuaciones del movimiento uniformemente variado son:

$$a = a_m = \frac{\Delta V}{\Delta t} = \frac{V - V_0}{t - t_0} = constante \quad \text{Ec. 2.8}$$

$$V = a.t + V_0 \quad \text{Ec. 2.9}$$

$$x = \frac{1}{2}.a.t^2 + V_0 + x_0 \quad \text{Ec. 2.10}$$

$$x - x_0 = \frac{V^2 - V_o^2}{2.a} \quad \text{Ec. 2.11}$$

donde

a = Aceleración $[^m/_{s^2}]$

x= desplazamiento $[m]$

a_m= Aceleración media $[^m/_{s^2}]$

ΔV = Variación de la velocidad $[^m/_s]$

$\Delta t =$ Variación de tiempo $[s]$

- Movimiento de rotación uniforme (MRU):

  La aceleración angular $\alpha(t)$ de la Ec. 2.12 [15] es nula y la velocidad angular $\omega(t)$ de la Ec. 2.13 [15] es constante, por lo que la ecuación de la posición angular Ec. 2.14 [15] y las demás ecuaciones en el MRU son:

$$\alpha(t) = \theta''(t) = 0 \qquad (rad/s^2) \qquad \text{Ec. 2.12}$$

$$\omega(t) = \omega_0 = constante \qquad \text{Ec. 2.13}$$

$$\theta(t) = \omega(t - t_0) + \theta_0 \qquad \text{Ec. 2.14}$$

donde

$\alpha(t)$= Aceleración angular con respecto al tiempo $[{}^{rad}/_{s^2}]$

$\theta''(t)$= segunda derivada de la posición angular con respecto al tiempo $[{}^{rad}/_{s^2}]$

$\omega(t)$= Velocidad angular con respecto al tiempo $[{}^{rad}/_{s}]$

$\theta(t)$= Posición angular con respecto al tiempo $[rad]$

$t_0, \omega_0\ y\ \omega_0$ son las condiciones iniciales de movimiento

- Movimiento de rotación uniformemente variado (MRUV):

  La aceleración angular $\alpha(t)$ en la Ec. 2.15 [15] permanece constante, por lo que, las ecuaciones horarias de un MRUV para la velocidad angular (Ec. 2.16 [15]) y su desplazamiento (Ec. 2.17 [15]) son:

$$\alpha(t) = \alpha_0 = constante \qquad \text{Ec. 2.15}$$

$$\omega(t) = \alpha_0(t - t_0) + \omega_0 \qquad \text{Ec. 2.16}$$

$$\theta(t) = \frac{1}{2}\alpha_0(t - t_0)^2 + \omega_0(t - t_0) + \theta_0 \qquad \text{Ec. 2.17}$$

donde

$\alpha(t)$= Aceleración angular con respecto al tiempo $[{}^{rad}/_{s^2}]$

$\omega(t)$= Velocidad angular con respecto al tiempo $[{}^{rad}/_{s}]$

$\theta(t)$= Posición angular con respecto al tiempo $[rad]$

$t_0, \alpha_0, \omega_0 y\ \theta_0$ son las condiciones iniciales de movimiento.

2.2.6 Tecnologías de inteligencia artificial.

La inteligencia artificial o artificial intelligence (AI) es uno de los campos de la informática la cual se centra en la creación de máquinas inteligentes con el objetico de emular inteligencia, comportamiento y el aprendizaje humano. Esta ha sido una parte activa de importantes avances en el aprendizaje automático, el procesamiento del lenguaje natural, la visión artificial y la robótica, debido a ello, ha experimentado avances notables en los últimos años, revolucionando varios sectores e industrias [16], tal como

se puede ver en la figura 2.7.

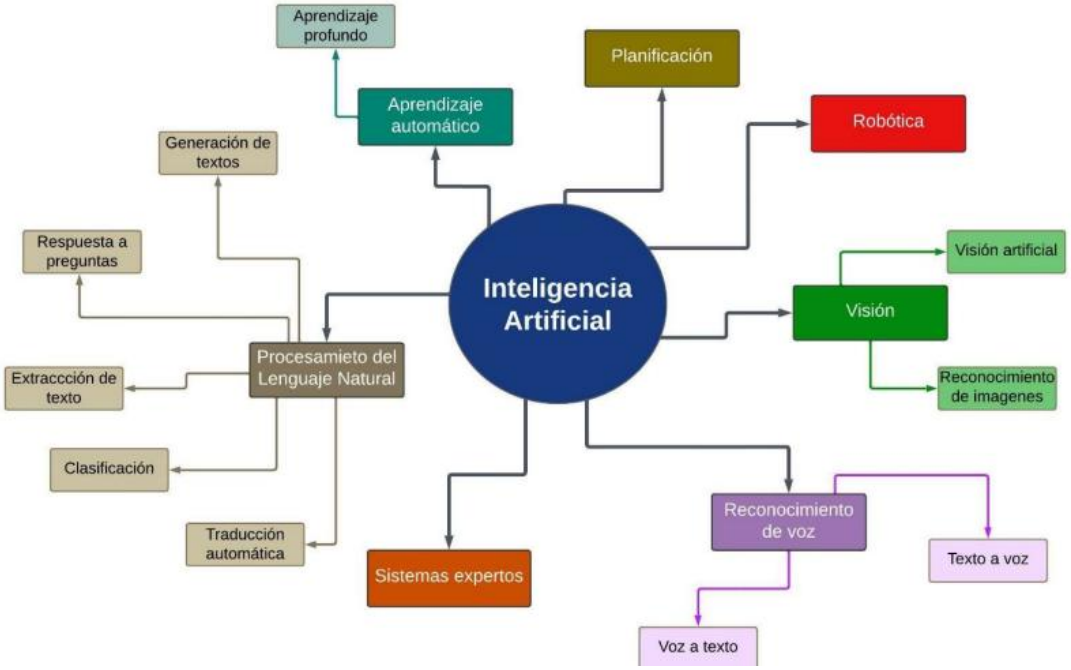


Fig 2.7 La inteligencia artificial abarca un gran número de disciplinas tecnológicas, en las cuales se van mejorando con el pasar del tiempo [16].

Estos avances han despertado un inmenso interés y entusiasmo en torno a las posibles aplicaciones de la IA en múltiples dominios del conocimiento como en la solución de problemas complejos dónde regularmente se requieren simulaciones y modelos para obtener una comprensión más profunda de ciertos fenómenos o comportamientos, dando como resultado, la exploración escenarios que pudiesen ser poco prácticos o imposibles de estudiar directamente, con una comprensión más profunda [16].

2.2.7 Visión Artificial: fundamentos y aplicaciones.

Según [17], la visión artificial es un campo de la inteligencia artificial la cual permite a los ordenadores y sistemas extraer información significativa a partir de imágenes digitales, videos y otras entradas visuales; y, a base de eso, tomar medidas o realizar recomendaciones en función de esa información.

La visión artificial funciona de manera muy similar a la visión humana, la visión artificial entrena a las máquinas para realizar funciones determinadas, con muy poco tiempo de procesamiento, con cámaras, datos y algoritmos en lugar de retinas, nervios ópticos y una corteza visual, como es el caso de la visión humana [17].

Además, en aspectos donde la visión humana es adecuada para la interpretación

cualitativa de una escena, la visión artificial se destaca en la medición cuantitativa convirtiendo a la misma escena en una escena estructurada, debido a su velocidad, precisión y repetibilidad [18].

2.2.7.1 Principios del procesamiento de Imágenes

El procesamiento de imágenes (digitales en sí), es el pilar en el cual se asientan cualquier sistema de visión artificial; este procesamiento comprende una sucesión de algoritmos que transforman datos de la imagen que se desea analizar en información útil para el análisis y la obtención de un resultado [19].

Las etapas fundamentales para el procesamiento de imágenes constan en: la adquisición de la imagen (muestreo), un preprocesamiento (filtros, contraste, reducción del ruido), segmentación (selección de las regiones de interés), la extracción de características y la interpretación [19], tal como se puede visualizar en la Figura 2.8.



Fig 2.8 Etapas del procesamiento de imágenes para un sistema de visión artificial.

Si se basa de estos conceptos al diseño del sistema de visión artificial para analizar el rendimiento de un velocista de escalada deportiva, hay que seguir un protocolo que cumplan con estos pasos, se debe obtener un video del entrenamiento del deportista con una excelente calidad de imagen, aplicar filtros en aspectos importantes como la detección de presas y en la silueta del escalador, entre otros aspectos [19].

2.2.7.2 Etapas de un sistema de visión artificial

Según [20], las etapas que se necesitan seguir para toda aplicación de procesamiento de imágenes son las mismas, solo dicha aplicación va a depender del producto deseado. Estas etapas son:

- **Adquisición o captura de imagen:** es el paso principal el cual busca convertir una imagen o un video digital en un o una serie de frames que pueda ser procesado por una computadora.

- **Preprocesamiento:** trata de aplicar técnicas de filtrado de imágenes que se aplican a los pixeles de un frame para optimizarla, con esto mejora las características de destacar cierta información; esto se logra mediante cálculos matemáticos como: transformada rápida de Fourier, Convolución, entre otros.
- **Segmentación:** consta en dividir a la imagen en secciones manipulables, por lo general en escala de grises; este es uno de los procesos con mayor dificultad debido a que de este paso depende la interpretación de resultados.
- **Extracción:** este es el proceso en el cual se reconoce a los elementos que se crearon en el paso de la segmentación, con ellos se obtiene información como características geométricas, tamaño, forma, etc.; esta información es utilizada para la identificación de los objetos deseados.
- **Interpretación:** en el último paso del procesamiento de imágenes, se realiza el reconocimiento o interpretación de resultados de una forma automatizada. Donde, las características de la imagen que se procesaron, son comparadas con una base de datos o algoritmos, y que finalmente nos brinda una salida de imagen procesada.

2.2.7.3 Aplicaciones de la visión artificial y la biomecánica en el deporte

De acuerdo a [21], la unión de la visión artificial y la biomecánica aplicada al deporte permitió avances significativos en el análisis y la mejora del rendimiento humano además de la prevención de lesiones. La existencia de algunos métodos de IA, verifican que poseen la eficacia para mejorar el rendimiento deportivo, tienen potencial para la predicción de lesiones, y pueden ser aplicadas específicamente para cada deporte. Las tecnologías de inteligencia artificial como son las redes neuronales convolucionales pueden alcanzar hasta un 94% de coincidencia con expertos internacionales en la evaluación de técnicas.

Aplicando estos conceptos a la escalada deportiva de velocidad, el uso de la inteligencia artificial puede analizar el seguimiento de la trayectoria del deportista, estimar la velocidad y la aceleración del deportista por segmentos, evaluar la sincronización de manos y pies, e incluso, comparar todos estos aspectos con un modelo de referencia. Y dar como resultado la detección de zonas a mejorar, cuantificar el rendimiento, entre otros.

2.2.7.4 Beneficios y limitaciones de la visión artificial en el deporte

Aplicar la visión artificial en el deporte conlleva beneficios como ser no invasivo, posee la capacidad de funcionar en condiciones menos controladas por lo que el deportista no tendrá la impresión de ser controlado, además genera métricas con bastante precisión y que puede generarlos con rapidez o en tiempo real [21].

Por otro lado, existen factores que pueden afectar a la precisión en el análisis de un sistema de visión artificial, como:

- Oclusión
- Iluminación
- Ángulos de cámara
- Movimientos muy rápidos o complejos

En el caso de la escalada deportiva de velocidad, se pueden generar sombras, múltiples movimientos simultáneos, cambios de ángulos de cámara, e incluso desplazamientos del deportista que generen complicaciones para el sistema.

También una de las limitaciones es que los algoritmos de estimación de postura que generalmente son de código abierto, no están diseñados para aplicaciones biomecánicas, por lo que los mismos algoritmos pueden generar resultados inconsistentes e imprecisos [22].

El diseño de uno de estos sistemas debe tener consideraciones de diseño bien definidas para poder lidiar con estos tipos de limitaciones, se puede definir tolerancias de error, protocolos de captura de imagen para analizar, e incluso generar condiciones de uso.

Capítulo III

Marco metodológico

3.1 Modelo de la Investigación

El trabajo de titulación se realiza a base de técnicas de investigación básica, las cuales ayudan a la realización de cada una de las etapas en las que se dividió dicho trabajo de titulación. Se aplica un método documental de investigación, para conocer y evidenciar de manera cualitativa, todos los fundamentos y conceptos necesarios para conocer acerca de la evaluación técnica de la escalada deportiva de velocidad y también aspectos biomecánicos del deportista a través de diversas fuentes bibliográficas como artículos científicos, libros, revistas, bibliotecas virtuales, etc.

A la par con esta investigación, Se emplea la investigación de campo para la recolección de información en el área donde se realiza la actividad, para la posterior investigación del diseño del sistema de análisis del performance de un deportista de escalada deportiva mediante visión artificial. Con la investigación descriptiva se aplica para describir los procedimientos realizados para la solución de la problemática [23].

Y finalmente, se involucra la investigación experimental la cual se basa en un protocolo de control mediante la presencia de variables, las cuales se manipulan y observan los resultados [24].

3.2 Diseño de la Investigación

3.2.1 Fase 1: Conceptualización de aspectos a considerar en el análisis del rendimiento del deportista.

- ***Actividad 1: Investigación de antecedentes y evaluación técnica de la escalada deportiva modalidad velocidad y estudio de casos:*** con esta actividad se busca

entender todos los aspectos relacionados a este deporte; se considera analizar definiciones, aspectos técnicos y reglamentos, parámetros que se analizan en la escalada deportiva de velocidad y estudiar casos de deportistas en competencias nacionales e internacionales.

- ***Actividad 2: Análisis biomecánico del deportista:*** se va a determinar qué aspectos biomecánicos son necesarios para estudiar el rendimiento del deportista al momento de realizar la actividad.
- ***Actividad 3: Elaboración de encuestas a deportistas y técnicos:*** con la colaboración de personas experimentadas en este deporte, se podrá obtener un punto de vista más claro de la problemática y también, plantear la mejor solución a esta.
- ***Actividad 4: Investigación de tecnologías de visión artificial para el deporte:*** esta actividad trata de investigar acerca de técnicas de visión artificial que se están aplicando a los deportes y posteriormente, determinar si estas se pueden aplicar al deporte de escalada deportiva de velocidad.
- ***Actividad 5: Especificaciones del sistema a diseñar:*** en este punto se especifican todos los criterios y restricciones del sistema a diseñar, tomando en cuenta toda la investigación previa realizada.

3.2.2 Fase 2: Diseño del sistema con visión artificial.

- ***Actividad 6: Planteamiento de posibles soluciones y selección de la mejor:*** con esta actividad se busca plantear varias soluciones para el diseño del sistema y escoger la mejor alternativa tomando en cuenta la investigación de las tecnologías de visión artificial, las entrevistas realizadas y las especificaciones del sistema.
- ***Actividad 7: Realización de cálculos necesarios para el sistema:*** en esta actividad de realiza las respectivas fórmulas y cálculos que se necesitan para el análisis de todos los aspectos que evalúan el rendimiento del deportista, para posteriormente ser añadidas en el diseño del sistema.
- ***Actividad 8: Diseño de la arquitectura del sistema****:* se separa al sistema por procedimientos y métodos con los cuales se adquiere la capacidad de llevar a cabo la creación del sistema.

- ***Actividad 9: Elaboración de diagramas de flujo y código de programación:*** ya una vez decidido todos los procedimientos que se van a realizar, se procede con el diseño del sistema, a través de la ayuda de diagramas de flujo para la creación del código de programación que necesita el sistema.

3.2.3 Fase 3: Implementación del prototipo del sistema.

- ***Actividad 10: Implementación del prototipo del sistema:*** en esta actividad se integra todo el código de programación creado y se verifica la funcionalidad de este, lo primordial en esta actividad es la detección del material audiovisual del deportista de escalada y la detección del deportista.
- ***Actividad 11: Integración de elementos:*** una vez que se pueda detectar al deportista se procede a seguir implementando el código de programación para la generación de datos de todos los aspectos requeridos.
- ***Actividad 12: Integración de funciones para el sistema:*** esta fase concluye con el análisis de los datos obtenidos por medio del sistema realizado y la impresión de los resultados.

3.2.4 Fase 4: Validación del sistema en circunstancias reales.

- ***Actividad 13: Realización de pruebas de funcionamiento en entornos controlados:*** se busca comprobar que el prototipo funciones correctamente, identificando cada uno de los aspectos y analizando los datos que obtiene.
- ***Actividad 14: Verificación del sistema en circunstancias reales:*** esta actividad busca comprobar la funcionalidad del prototipo del sistema creado bajo circunstancias de escalada reales, donde intervengan tanto el cuerpo técnico como el deportista.
- ***Actividad 15: Ajustes y/o modificaciones adicionales al sistema:*** en esta actividad se realiza un ajuste del sistema ocasionados por posibles factores externos, como la iluminación; además, se modifica el sistema si se ha determinado que los aspectos a medir tienen algún inconveniente.
- ***Actividad 16: Escritura y redacción del documento:*** se realiza la finalización del documento final del trabajo de titulación según las normas brindadas por la coordinación de la carrera.

Capítulo IV

RESULTADOS Y ANÁLISIS

En este capítulo se presentan los resultados alcanzados del diseño del sistema de visión artificial requerido. Este proceso se partió desde localización y selección un lugar estratégico que sirva de apoyo en la obtención de información y posteriormente la realización de pruebas de validación; se seleccionó a la provincia de Imbabura especialmente al cantón Ibarra debido a que allí existe una escuela deportiva de escalada donde han sobresalido hombres y mujeres deportistas que han sido consagrados como campeones nacionales, con ello, se obtuvo información mediante la realización de encuestas y posteriormente, se crearon especificaciones y alternativas de diseño para seleccionar de forma cuantitativa a la mejor solución. Y se finaliza con la realización de pruebas de funcionamiento y validación del sistema en condiciones reales.

4.1 Recolección y análisis de datos preliminares

En esta sección se describen los resultados de una encuesta dirigida a deportistas y entrenadores que practican la escalada deportiva de velocidad, especialmente, se contó con el apoyo de la escuela de escalada de Imbabura, dónde se procedió a encuestar a un grupo de 9 personas que pertenecían a un conjunto de entrenadores y deportistas de alto rendimiento y, con ello se obtuvo la información preliminar necesaria basada en sus conocimientos y experiencia, para el posterior diseño del sistema.

Con esta encuesta también se pretendió recopilar información que permita identificar

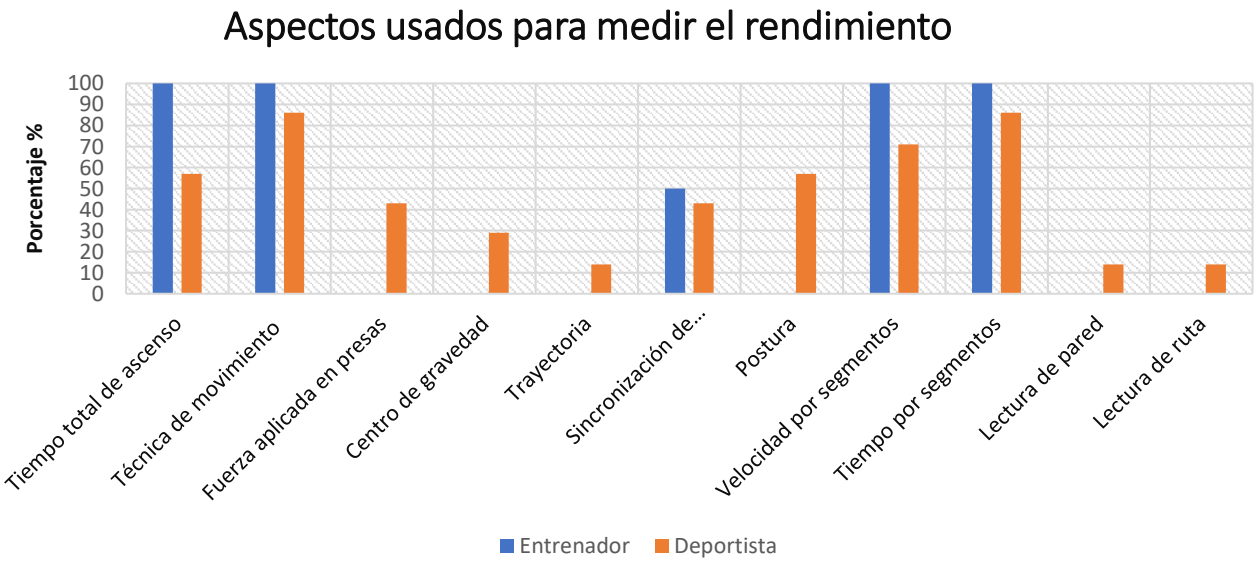
las carencias y necesidades que posee el usuario, tanto del deportista como del entrenador, al momento de evaluar el rendimiento y, también identificar las variables de interés que el sistema deberá medir y procesar.

Para la obtención de los resultados de la encuesta, se procedió a considerar dos factores importantes, el rol de la persona en el deporte, es decir, ser deportista o entrenador, y la experiencia que tienen en el deporte cada uno de los encuestados; con ello, estos resultados que se obtuvieron serán más concisos, debido a que son generados desde diferentes puntos de vista y, se los consideró en aspectos donde se pensó necesario.

4.1.1 Resultados y estudio de la encuesta

Primeramente, la encuesta permitió identificar los principales aspectos que se analizan para medir el rendimiento (performance) de un deportista, en el caso de la opinión entre entrenador y deportista sobresalen cuatro aspectos: el tiempo total de ascenso, la técnica de movimiento, la velocidad por segmentos y el tiempo por segmentos, tal como se muestra en la **figura 4.1.a.** Y para el caso de la experiencia que poseen los miembros se tienen resultados similares, los cuales se muestran en la **figura 4.1.b**. En ambos casos existieron aspectos como: el centro de gravedad y la postura del deportista, la trayectoria que el deportista recorre en la pared, etc., que también son utilizados, pero no por la mayoría de los encuestados, estos aspectos se analizaron posteriormente con el fin de determinar si deben ser incluidos al diseño o que serán útiles para obtener métricas que necesite el diseño.

(a)



(b)

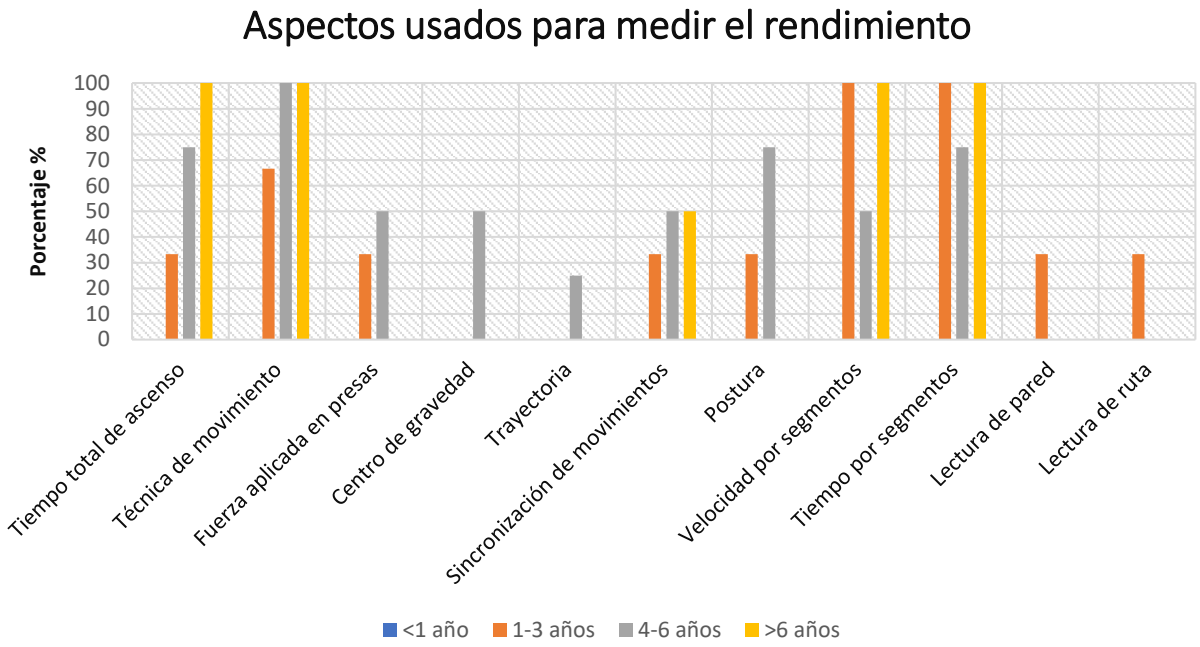


Fig 4.1 **Aspectos que se utilizan para medir el rendimiento de un deportista de escalada de velocidad: (a) resultados según el rol del encuestado y (b) resultados según la experiencia del encuestado.**

4.1.2 Obtención de material audiovisual

De los resultados de la encuesta mostrado en la **figura 4.2** también se obtuvo que, tanto deportistas como entrenadores, utilizan herramientas tecnológicas que posean cámaras convencionales para el análisis, es decir, se realiza la generación de videos, por lo que se procede a obtener material audiovisual mediante la misma metodología, proveniente de grabaciones de entrenamiento y/o posiblemente competencias.

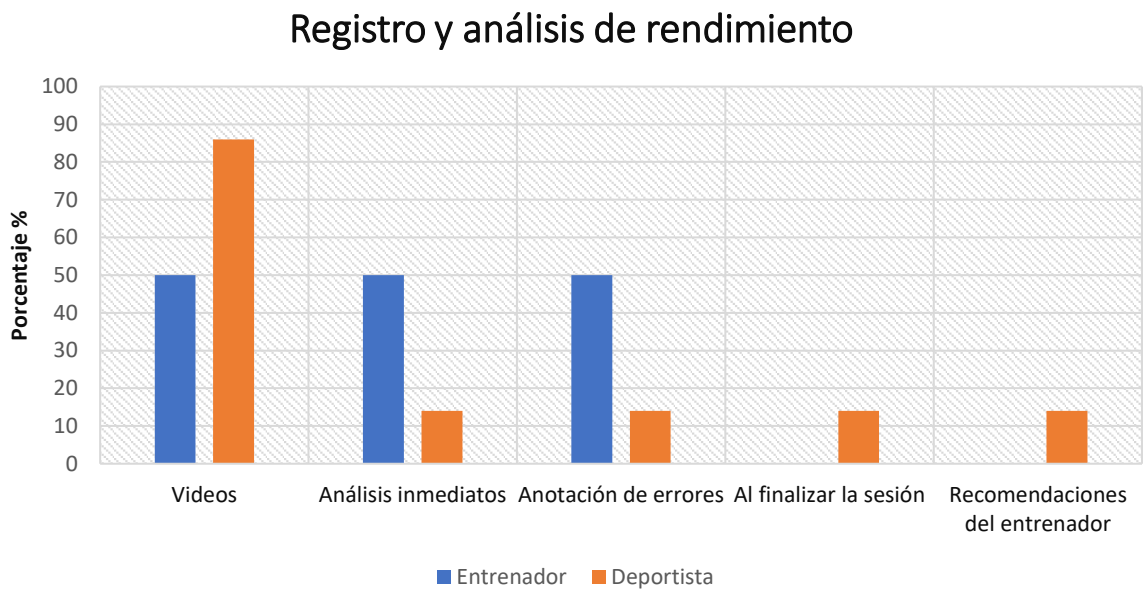


Fig 4.2 **Herramientas tecnológicas y métodos utilizados por deportistas y entrenadores para el análisis de rendimiento.**

Se considera que los videos generados por deportistas y entrenadores tratan de poseer las siguientes características:

- Un ángulo de grabación de 110°, lo que permite visualizar completamente a la pared de escalada.
- Una distancia aproximada de 7 metros al frente de la pared.
- Una resolución de imagen de 720p o 1080p, para visualizar correctamente los puntos clave del escalador y de la pared.

4.1.3 Parámetros que se considerarán en el análisis del rendimiento

Considerando varios factores como: los parámetros más utilizados por los deportistas y entrenadores concluidos en la encuesta realizada, los parámetros técnicos que se pueden analizar mediante video análisis [25], y los parámetros que un sistema de visión artificial puede analizar [26], se procede a concluir que los parámetros que se van a considerar para el análisis son los siguientes que se muestran en la **Tabla 4.1**:

Tabla 4.1 Descripción de los parámetros que se necesitarán previamente

N.º	Parámetro	Descripción
1	**Tiempo**	Servirá para realizar cálculos
2	**Trayectoria**	Analiza el desplazamiento del deportista en la pared mediante su centro de masa
3	**Técnica de movimiento**	Detección y cuantificación de agarres
4	**Biomecánica**	Abarca la detección del deportista y sus posturas
5	**Detección de fallos**	Detección de ocasiones en el que escalador cometa algún error y comience a descender.

4.1.4 Parámetros que se requieren conocer como resultado del análisis

Dado el objetivo del diseño del sistema de visión artificial, se concluyó que los parámetros óptimos con los que se podrá analizar a cada uno de los aspectos que utilizan deportistas y entrenadores para el análisis del performance del deportista serán los que se muestran en la **tabla 4.2**:

Tabla 4.2 Resultados del análisis del sistema que se diseñará

N.º	Parámetro	Descripción
1	**Tiempo**	Incorporará el tiempo total de ascenso y por tramos
2	**Tiempo entre presas**	Tiempo entre presas consecutivas tocadas por las manos.
3	**Número de movimientos**	Cuantificar el número de transiciones o agarres de presa a presa
4	**Velocidad**	Incorpora la velocidad del deportista a medida que realiza la escalada y, la velocidad de ascenso promedio.
5	**Trayectoria**	Analizará el desplazamiento del deportista en la pared mediante su centro de gravedad estimado, con una “ruta óptima” definida.
6	**Técnica de movimiento**	Comparación de agarres entre el deportista a analizar y el de referencia

7	**Comparación de métricas**	Comparación de métricas obtenidas entre el deportista de referencia y el deportista a analizar.

4.2 Especificaciones del sistema a diseñar

El sistema propuesto tiene el fin de ser utilizado como una herramienta de visión artificial que sea capaz de analizar el performance de un velocista de escalada deportiva, a partir del procesamiento de material audiovisual, con ello se presentan las especificaciones del sistema de visión artificial que se procederá a realizar abarcando requerimientos funcionales; y con ello, se indican varias restricciones operativas que serán necesarias para obtener resultados adecuados y precisos. Las descripciones que se presentan son las que se deberán efectuar, con el diseño y el prototipo del sistema, para cumplir con el objetivo final de validar el sistema bajo condiciones reales.

Este sistema busca ser eficiente, accesible y fácil de manipular, todo ello con la obtención de resultados correctos.

4.2.1 Requerimientos funcionales

- **Identificación de variables iniciales:** Diseñar un algoritmo robusto el cual permita identificar adecuadamente las variables iniciales que necesita el sistema para realizar el análisis del deportista.
- **Interfaz entendible y fácil de manipular:** Es importante que la interfaz que maneje el usuario sea intuitiva al momento de utilizarla y con accesos directos a la ejecución del sistema de visión artificial.
- **Visualización de resultados:** Una vez que el sistema concluya con el análisis del deportista, este debe mostrar instantáneamente los resultados obtenidos.
- **Resultados adecuados y con un nivel adecuado de precisión:** El sistema debe finalizar mostrando como resultado a los aspectos que se necesitan para evaluar el rendimiento del deportista además de que estos deban contar con un buen nivel de precisión.

4.2.2 Restricciones operativas

- **Obtención del material audiovisual:** Este tendrá que cumplir ciertas características como: tamaño, calidad, y visualización para poder ser “admisible” por el sistema.
- **Operabilidad del sistema:** El sistema deberá operar bajo ciertas condiciones,

tanto con el material audiovisual como con el funcionamiento del sistema, para asegurar la fiabilidad del análisis y de los resultados, debido a que el sistema poseerá ciertas limitaciones a causa de criterios de diseño.

- **Modalidad de grabación:** existen dos maneras en las que se puede generar el material audiovisual, una es grabar el video con un enfoque en el deportista y el otro con un enfoque en la pared de escalada, para el diseño de este sistema se utilizará la segunda opción y con ello, la pared de escalada suele estar conformada por dos rutas, por lo que también se necesitará que en el video generado solo se muestre una ruta, en la cual se encontrará el deportista.

4.3 Propuestas de solución

En esta sección se indican las propuestas de solución que se realizaron para dar resultado al sistema de análisis de performance del velocista de escalada deportiva, comenzando desde la realización de la metodología de caja negra y su descomposición funcional, siguiendo con la utilización de una matriz morfológica con la finalidad de obtener varias propuestas de solución y posteriormente, seleccionar a la mejor solución de manera cuantitativa utilizando la ponderación de criterios; todo esto se lo realiza teniendo en cuenta las especificaciones del sistema a diseñar y los aspectos que se necesitan como resultado del análisis.

4.3.1 Diagrama de caja negra

Las únicas entradas que el sistema a diseñar va a pedir al usuario serán los videos generados, uno servirá como referencia y el otro será en que se va a analizar; entonces, el sistema realizará el análisis pertinente y mostrará como resultado todos los aspectos analizados al usuario, tal como se muestra en la **figura 4.3**.

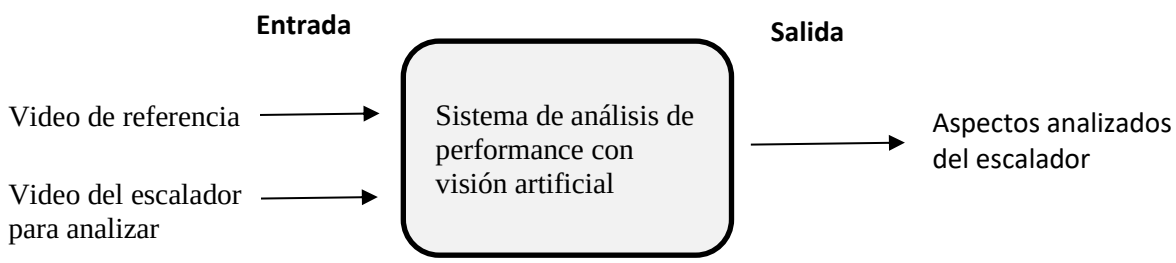


Fig 4.3 **Diagrama de caja negra para el sistema a realizar.**

4.3.2 Descomposición funcional

Una vez definido el sistema a realizar mediante el diagrama de caja negra planeado, se plantea la descomposición funcional del sistema, dónde se comienza con la realización de un diagrama el cual muestra de una forma más general, la estructura que llevará el sistema mediante funciones parciales y después se definen los componentes necesarios de cada función parcial.

4.3.2.1 Diagrama de descomposición funcional

Dado que necesitará extraer las mismas características o aspectos a ambos videos de entrada, se los aplicará el mismo procedimiento, esto abarca desde las características del material audiovisual que se ingresa, la aplicación de la visión artificial al sistema basado en las etapas del procesamiento de imágenes y la selección en lo correspondiente a la plataforma de implementación y visualización de resultados, en la **figura 4.4** se muestra una forma general del procedimiento mencionado.

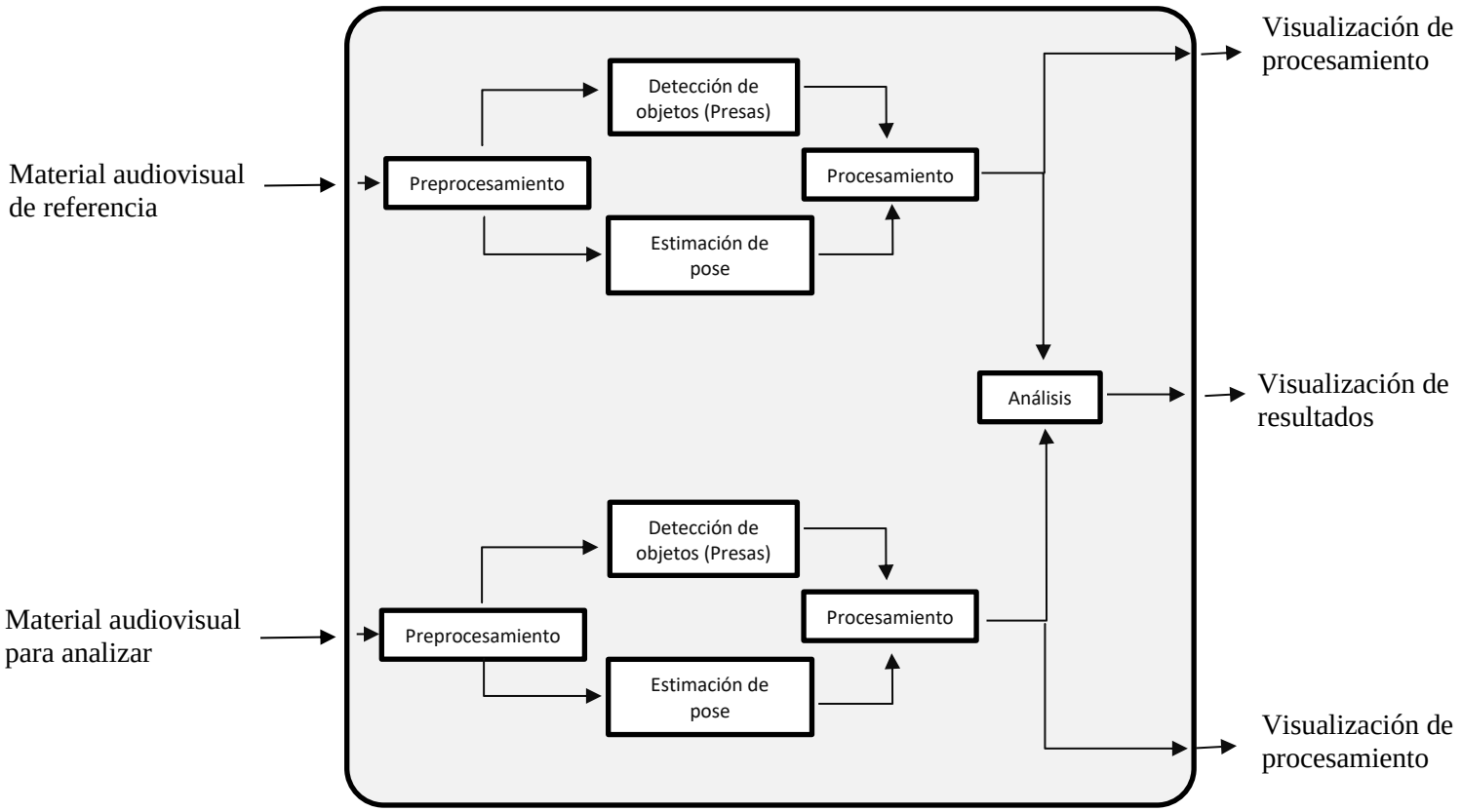


Fig 4.4 **Descomposición funcional del diagrama de caja negra basado en el principio de procesamiento de imágenes y los resultados requeridos.**

Describiendo al diagrama de la **figura 4.4** se tiene que: El sistema pedirá al usuario que ingrese dos videos los cuales deberán poseer características similares, tanto en su estructura como en su contenido, el primero que servirá como referencia, es decir, que el sistema realizará el procesamiento y extracción de datos y guardará los resultados de forma visual y cuantitativa, y se los comparará con el segundo video ingresado en el que se realizará el mismo procesamiento y extracción de datos; se realizará el análisis, comparando los resultados del video de referencia y del deportista en sí, y se concluirá con la visualización de los resultados de todos los aspectos definidos, además se contará con la opción de visualizar la obtención de métricas realizada a cada uno de los videos de los deportistas.

4.3.2.2 Componentes de descomposición funcional

Ahora se definen los componentes que se asignarán a cada una de las funciones parciales, basado en el proceso general que debe realizar el sistema; en la **tabla 4.3** se muestra a los componentes que se consideraron necesarios para el desarrollo del sistema a realizar.

Tabla 4.3 Descripción de las funciones parciales y enlistado de sus componentes.

Función parcial	Descripción	Componentes
Material audiovisual	Esta función parcial busca establecer las características del material audiovisual que va a utilizar el sistema.	Resolución Tamaño Orientación *(Los FPS y el formato se establecerá dependiendo de lo que sea mejor para el sistema)*
Preprocesamiento	Se refiere a la modificación inicial del video dándole las cualidades necesarias para procesos posteriores.	División en espacios de color Segmentación inicial Operación morfológica Segmentación 2 *(Reducción de ruido y normalización de brillo dependiendo de lo que necesite el sistema)*
Detección de objetos (Presas)	Se busca detectar y obtener métricas cuantificables de la pared para añadirlas al análisis.	Presas de la pared Seguimiento de presas tocadas

Estimación de pose	Se refiere a la obtención de métricas del deportista.	Detección de puntos importantes y silueta Seguimiento del deportista
Procesamiento y análisis	Busca la comparación de los resultados del deportista con el sujeto de referencia.	Tiempo entre presas Cálculo de aspectos
Plataforma de implementación	Se refiere a la obtención del medio por el cual se realizará el sistema y la interfaz que utilizará el usuario	Lenguaje de programación Librerías de Visión Artificial Librería para Interfaz Tipo de sistema
Visualización de resultados	Se centra la determinación de todos los métodos de visualización de resultados que se ofrecerán al usuario.	Video Gráficos Archivo

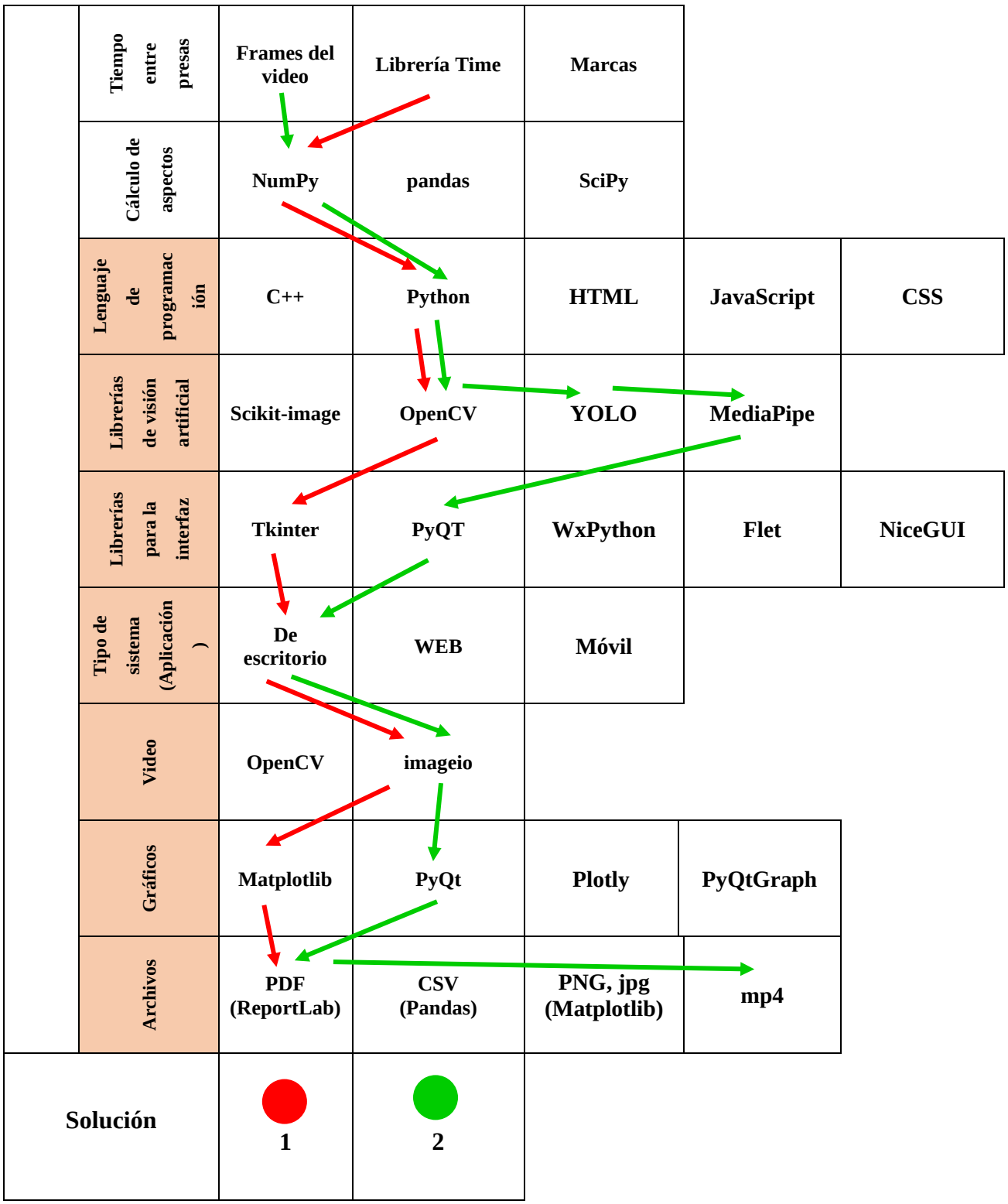
4.3.3 Matriz morfológica

Para la realización de propuestas de solución del sistema planeado, primeramente, se plantea un número de soluciones para cada componente de cada función parcial presentada, con la finalidad de "enlazar" las soluciones de los componentes y generar propuestas. En la **tabla 4.4** se muestra como fueron enlazados dichos componentes para proponer dos propuestas de solución.

Posteriormente se describieron a las propuestas de solución planteadas y, además, se detallan sus soluciones escogidas a razón de explicar su funcionamiento y su relación con las demás soluciones.

Tabla 4.4 Matriz morfológica para propuestas de soluciones.

Funciones parciales		S1	S2	S3	S4	S5
Material Audiovisual	Resolución	Baja definición (SD) 480p	Alta definición (HD) 720p	Alta definición completa (Full HD) 1080p	Ultra alta definición (UHD) 4K/2160p	
	Tamaño (píxeles)	SD (720x480)	HD (1280x720)	Full HD (1920x1080)	UHD (3840x2160)	
	Orientación	Vertical	Horizontal			
	División en espacios de color	RGB	HSV	YMK	BGR	Escala de grises
	Segmentación inicial	Histograma	Región de crecimiento	División de regiones	k-means	Watershed
	Operaciones morfológicas	Erosión	Dilatación	Extracción de borde	Llenado de región	
	Segmentación 2	Color	Área	Textura	Bounding Box	Bounding elipse
	Presas de la pared	Color (HSV)	YOLOv11	Segmentación por contornos	Detectron2	Mask R-CNN
	Seguimiento de presas tocadas	Detección de contacto por intersección	Detección por proximidad de muñecas			
	Detección de silueta y puntos importantes	MediaPipe	OpenPose	MoveNet	HRNet	
	Seguimiento del deportista	MediaPipe	YOLOv8	OpenPose	Optical Flow	DeepSort


Tiempo entre presas
Frames del video
Librería Time
Marcas
Cálculo de aspectos
NumPy
pandas
SciPy
Lenguaje de programación
C++
Python
HTML
JavaScript
CSS
Librerías de visión artificial
Scikit-image
OpenCV
YOLO
MediaPipe
Librerías para la interfaz
Tkinter
PyQT
WxPython
Flet
NiceGUI
Tipo de sistema (Aplicación)
De escritorio
WEB
Móvil
Video
OpenCV
imageio
Gráficos
Matplotlib
PyQt
Plotly
PyQtGraph
Archivos
PDF (ReportLab)
CSV (Pandas)
PNG, jpg (Matplotlib)
mp4
Solución
1
2

4.3.4 Propuesta de solución Nro. 1

La primera propuesta de solución se basa en la creación de un prototipo de una aplicación de escritorio utilizando el lenguaje de programación Python como lenguaje principal y la utilización de librerías como OpenCV para la parte de visión artificial y Tkinter para la parte de la interfaz de usuario. El material audiovisual que se necesitará ingresar al sistema debe poseer las características de: alta definición (HD) 720p, con un tamaño de 1280x720 pixeles y la orientación debe ser vertical. A el sistema se lo divide en subfunciones, basándose en los principios de procesamiento de imagen, los cuales se muestran en la **tabla 4.5** y con ellos se finaliza en la muestra de resultados en forma de video al momento de utilizar la interfaz, mediante la librería de PyQt, también mediante gráficos utilizando la librería Matplotlib y concluyendo con un reporte rápido del análisis en formato .pdf.

Tabla 4.5 Subfunciones del sistema que se diseñará en la propuesta Nro. 1

N.°	Subfunción	Solución(es)
1	División en espacios de color	HSV
2	Segmentación inicial	k-means, Watershed
3	Operaciones morfológicas	Llenado de región, extracción de borde
4	Segmentación 2	Color, Área
5	Presas de pared	YOLOv11
6	Seguimiento de presas tocadas	Detección de contacto por intersección
7	Detección de puntos importantes y silueta	Mediapipe
8	Seguimiento del deportista	Mediapipe, Optical Flow
9	Tiempo entre presas	Librería Time
10	Cálculo de aspectos	Numpy

4.3.5 Propuesta de solución Nro. 2

La segunda propuesta de solución también tiene como lenguaje de programación principal a Python para la creación de una aplicación de escritorio, así como la utilización de OpenCV, YOLO y MediaPipe para el aspecto de visión artificial y a la librería PyQt para la realización de la interfaz de usuario. Ahora, el material audiovisual que se presenta para esta propuesta debe ser: Alta definición completa (Full HD) 1080p, con un tamaño de 1920x1080 pixeles y con la orientación en vertical. También, esta propuesta mantiene algunas subfunciones de la propuesta Nro. 1, aún así, se obtendrán diferentes resultados en el diseño del sistema; las subfunciones de la propuesta Nro. 2 se muestran en la **tabla 4.6**, y con eso se trata de obtener resultados en forma de video en la interfaz, utilizando PyQt, gráficamente utilizando la misma librería, y con la generación de un reporte breve en formato .pdf a través de ReportLab y un video en formato .mp4 del análisis realizado utilizando imageio.

Tabla 4.6 Subfunciones del sistema que se diseñará en la propuesta Nro. 2

N.°	Subfunción	Solución(es)
1	División en espacios de color	RGB, HSV
2	Segmentación inicial	Histograma, División de regiones
3	Operaciones morfológicas	Extracción de borde
4	Segmentación 2	Bounding Box
5	Presas de pared	YOLOv11, Segmentación por contornos
6	Seguimiento de presas tocadas	Detección de contacto por intersección
7	Detección de puntos importantes y silueta	Mediapipe
8	Seguimiento del deportista	Mediapipe, YOLOv8
9	Tiempo entre presas	Frames del video
10	Cálculo de aspectos	Numpy

4.3.6 Selección de la mejor propuesta

Una vez que se han definido a las propuestas de solución, se procede a realizar la evaluación o ponderación de cada una de ellas, con el fin de encontrar la mejor alternativa a diseñar.

Primeramente, se elabora una lista de consideraciones o criterios de diseño, basados en la funcionalidad, la viabilidad y los aspectos técnicos, las cuales se muestran en la **tabla 4.7**; seguido de ello se determina de manera cuantitativa su orden de importancia y luego, a las propuestas de solución se las evaluará de acuerdo a cada uno de estas especificaciones de diseño.

Tabla 4.7 Criterios de diseño que se utilizarán en la evaluación

	Descripción
C1	Precisión
C2	Robustez
C3	Funcionalidad
C4	Diseño Integral
C5	Usabilidad
C6	Viabilidad técnica
C7	Implementable
C8	Estabilidad

4.3.6.1 Ponderación de criterios

Se evalúan a los criterios de diseño seleccionados de manera cuantitativa, teniendo en cuenta que se los compara entre ellos y se evalúan con mayor puntuación a aquellos que se consideren más importantes para el sistema a diseñar, esto se realiza con la finalidad de determinar qué criterios son más cruciales para el sistema, y con ello, darles mayor importancia en el transcurso del diseño del sistema; en la **tabla 4.8** se muestra el proceso de ponderación de criterios que se realizó.

Tabla 4.8 Ponderación de criterios

Criterios	C1	C2	C3	C4	C5	C6	C7	C8	E+1	Ponderación	%
C1	-	0,5	0	1	1	1	1	0,5	6	0,167	16,67
C2	0,5	-	0,5	1	1	1	0,5	0,5	6	0,167	16,67
C3	1	0,5	-	1	1	1	0,5	0,5	6,5	0,181	18,06
C4	0	0	0	-	0,5	0,5	0,5	0	2,5	0,069	6,94
C5	0	0	0	0,5	-	0,5	0,5	0	2,5	0,069	6,94
C6	0	0	0	0,5	0,5	-	0,5	0,5	3	0,083	8,33
C7	0	0,5	0,5	0,5	0,5	0,5	-	0,5	4	0,111	11,11
C8	0,5	0,5	0,5	1	1	0,5	0,5	-	5,5	0,153	15,28
									36	1	100

En la **tabla 4.9** se visualizan los resultados obtenidos de la ponderación de criterios, se determinó que la funcionalidad posee la ponderación más alta, por lo que se la considerará de más importancia y de más prioridad para el sistema, a esta le siguen la precisión y la robustez, las dos con la misma ponderación, debido que, al ser un sistema de análisis de rendimiento, se necesita obtener resultados que sean correctos y con un nivel adecuado de precisión. También se tendrá en cuenta la estabilidad del sistema y su implementabilidad. La viabilidad técnica, el diseño integral y la usabilidad serán consideradas como secundarias por obtener menor ponderación.

Tabla 4.9 Valoración resultante de la ponderación de criterios

	Criterio	Ponderación (%)
C3	Funcionalidad	18,06
C1	Precisión	16,67
C2	Robustez	16,67
C8	Estabilidad	15,28
C7	Implementable	11,11
C6	Viabilidad técnica	8,33
C4	Diseño Integral	6,94
C5	Usabilidad	6,94

4.3.6.2 Ponderación de propuestas

Una vez que se tienen los criterios de diseño ordenados cuantitativamente según su prioridad, se realiza la siguiente parte del análisis que es la ponderación de propuestas. Este proceso compara a las propuestas de solución planteadas en función de cada criterio de diseño.

Para el caso del criterio de funcionalidad se tiene que ambas propuestas de

solución darán resultados semejantes (ver **tabla 4.10**).

Tabla 4.10 Ponderación de las propuestas para el criterio C3.

Funcionalidad	Propuesta Nro. 1	Propuesta Nro. 2	E+1	Ponderación
Propuesta Nro. 1	-	0,5	1,5	0,500
Propuesta Nro. 2	0,5	-	1,5	0,500
		Suma	3	1
Propuesta Nro. 2 = Propuesta Nro. 1				

Para el criterio de precisión sobresale la propuesta de solución Nro. 2 (ver **tabla 4.11**).

Tabla 4.11 Ponderación de las propuestas para el criterio C1.

Precisión	Propuesta Nro. 1	Propuesta Nro. 2	E+1	Ponderación
Propuesta Nro. 1	-	0	1	0,333
Propuesta Nro. 2	1	-	2	0,667
		Suma	3	1
Propuesta Nro. 2 > Propuesta Nro. 1				

En el caso del criterio de robustez también se obtiene una ponderación similar (ver **tabla 4.12**).

Tabla 4.12 Ponderación de las propuestas para el criterio C2.

Robustez	Propuesta Nro. 1	Propuesta Nro. 2	E+1	Ponderación
Propuesta Nro. 1	-	0,5	1,5	0,500
Propuesta Nro. 2	0,5	-	1,5	0,500
		Suma	3	1
Propuesta Nro. 2 = Propuesta Nro. 1				

Para el criterio de estabilidad se tiene el resultado a favor de la propuesta de solución Nro. 1 (ver **tabla 4.13**).

Tabla 4.13 Ponderación de las propuestas para el criterio C8.

Estabilidad	Propuesta Nro. 1	Propuesta Nro. 2	E+1	Ponderación
Propuesta Nro. 1	-	1	2	0,667
Propuesta Nro. 2	0	-	1	0,333
		Suma	3	1
Propuesta Nro. 1 > Propuesta Nro. 2				

Para la implementabilidad se obtiene la misma ponderación para ambas propuestas, quiere decir que ambas alternativas son excelentes para ser

implementadas (ver **tabla 4.14**).

Tabla 4.14 Ponderación de las propuestas para el criterio C7.

Implementable	Propuesta Nro. 1	Propuesta Nro. 2	E+1	Ponderación
Propuesta Nro. 1	-	0,5	1,5	0,500
Propuesta Nro. 2	0,5	-	1,5	0,500
		Suma	3	1
Propuesta Nro. 2 = Propuesta Nro. 1				

En el criterio de la estabilidad técnica sobresale la propuesta de solución Nro2 (ver **tabla 4.15**).

Tabla 4.15 Ponderación de las propuestas para el criterio C6.

Viabilidad técnica	Propuesta Nro. 1	Propuesta Nro. 2	E+1	Ponderación
Propuesta Nro. 1	-	0	1	0,333
Propuesta Nro. 2	1	-	2	0,667
		Suma	3	1
Propuesta Nro. 2 > Propuesta Nro. 1				

En el caso del diseño integral también sobresale la propuesta de solución Nro2 (ver **tabla 4.16**).

Tabla 4.16 Ponderación de las propuestas para el criterio C4.

Diseño Integral	Propuesta Nro. 1	Propuesta Nro. 2	E+1	Ponderación
Propuesta Nro. 1	-	0	1	0,333
Propuesta Nro. 2	1	-	2	0,667
		Suma	3	1
Propuesta Nro. 2 > Propuesta Nro. 1				

Finalmente, en la **tabla 4.17** se muestra el resultado de la ponderación de soluciones con el caso de la usabilidad, la propuesta Nro. 2 posee mayor ponderación.

Tabla 4.17 Ponderación de las propuestas para el criterio C5.

Usabilidad	Propuesta Nro. 1	Propuesta Nro. 2	E+1	Ponderación
Propuesta Nro. 1	-	0	1	0,333
Propuesta Nro. 2	1	-	2	0,667
		Suma	3	1
Propuesta Nro. 2 > Propuesta Nro. 1				

4.3.7 Ponderación final

Se obtiene como resultado de la ponderación de propuestas a los siguientes datos (ver **tabla 4.18**), al sumar los resultados de los criterios de diseño de cada propuesta de solución se obtiene que para la propuesta Nro. 1 se obtuvo una suma de 0,461, mientras que para el caso de la propuesta Nro. 2 se obtuvo una suma de 0,539; por lo tanto, al comparar estos dos resultados se selecciona a la propuesta que contenga el valor mayor y con esto se concluye que, se utilizará a la propuesta de solución Nro. 2 para diseñar el sistema de visión artificial.

Tabla 4.18 Ponderación final

Conclusiones	C3	C1	C2	C8	C7	C6	C4	C5	SUMA	Ponderación
Propuesta Nro. 1	0,090	0,056	0,083	0,102	0,056	0,028	0,023	0,023	0,461	2
Propuesta Nro. 2	0,090	0,111	0,083	0,051	0,056	0,056	0,046	0,046	0,539	1

4.3.8 Descripción general de la solución

La evaluación de la mejor propuesta dio como resultado que la propuesta de solución Nro. 2 es la más factible a diseñar, ya que esta se destaca en su funcionalidad, precisión y robustez; esta propuesta permite continuar con el proceso de diseño.

La propuesta contiene una serie de pasos generales en los cuales se basará toda la codificación del sistema; primeramente, se utiliza el procedimiento de etapas de procesamiento de imágenes para extraer características importantes sobre la pared de escalada, ya sean de una imágenes referenciales o fotografías de la pared donde los deportistas realizan sus entrenamientos. Después, se realiza la extracción de aspectos o características importantes del deportista de referencia y del deportista a analizar, bajo las mismas etapas de procesamiento de imágenes, y estos aspectos o características los podrá visualizar el usuario en caso de necesitarlo, el siguiente paso es el de comparar y procesar resultados y, finalmente la muestra de estos resultados ya procesados en la interfaz, y en el caso que requiera el usuario, la impresión de un breve reporte en formato .pdf o la generación de un video de la interfaz en donde se visualiza el recorrido del deportista por la pared junto con los resultados.

Por otro lado, la interfaz se basará en mostrar una ventana principal en donde se encuentre el enunciado además de casilleros donde se deberá ingresar el video del deportista de referencia y del deportista, estos se podrán analizar y visualizar individualmente, y con el accionamiento de un botón se llevará a cabo el análisis con el

procedimiento ya planteado. En el accionamiento de este botón se creará una nueva ventana dónde se muestren los resultados procesados y con botones adicionales los cuales generarán ya sea, el reporte breve o el video.

4.4 Selección del protocolo o arquitectura

Para el desarrollo de la programación del sistema se consideraron varias alternativas de construcción de una aplicación, en particular, de escritorio; tomando en cuenta el tipo de prototipo que se va a realizar, el tipo de aplicación, el usuario final del sistema y las actualizaciones o el mantenimiento posterior que tendrá la aplicación; por ello, se concluyó que se realizará la aplicación de escritorio sin adoptar una arquitectura de software formal, el lugar de ello se la realizó con un enfoque de diseño basado en la construcción de toda la aplicación en un solo proyecto, con la realización de clases adecuadamente definidas, programación orientada a objetos y "a eventos", y de ser necesario, una organización o separación por módulos.

Con lo mencionado, se decidió agrupar en clases, distintos tipos de funciones que tengan relación entre sí, por ejemplo, en la **figura 4.5** se muestra el proceso con clases y funciones que se siguió para la creación de la interfaz, y con ello lograr una mejor organización en la estructura de la aplicación.

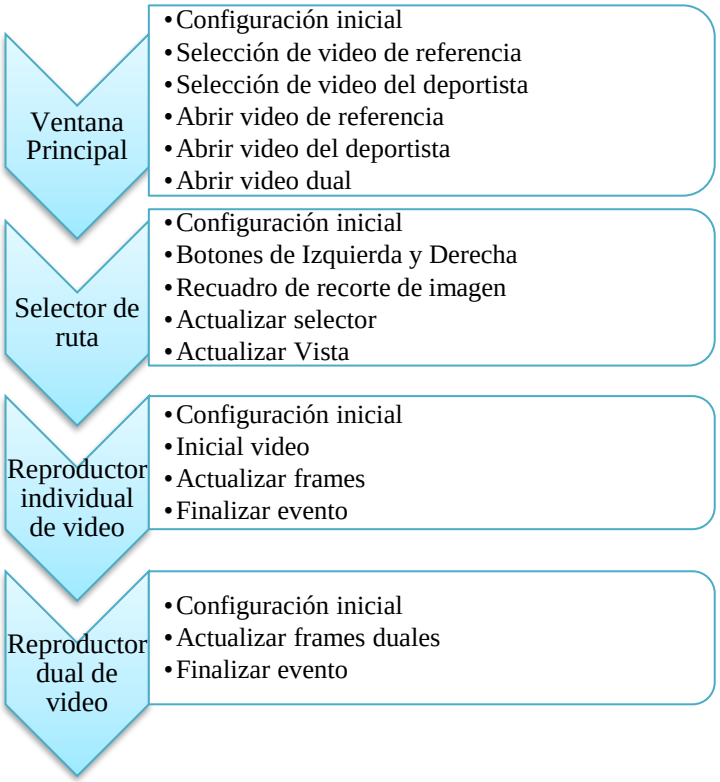


Fig 4.5 **Estructura en clases y funciones para la creación de la interfaz.**

4.4.1 Flujo de procesamiento de datos del sistema

Se requirió de la realización de diagramas donde se muestre todo el procedimiento de procesamiento de datos, con el fin de utilizar a este como referencia durante la programación de la aplicación y con ello, la correcta realización del sistema, en la **figura 4.6** se muestra un diagrama de flujo isométrico de procesamiento de datos que se realizó para esta aplicación.

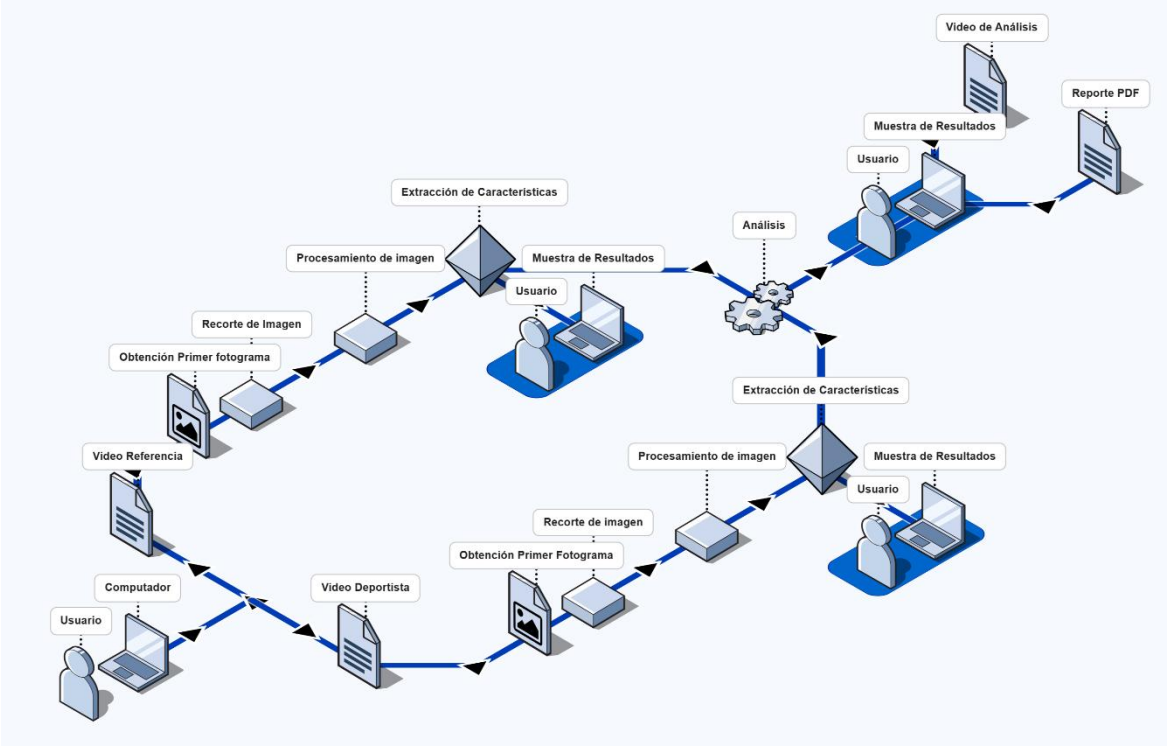


Fig 4.6 **Diagrama de flujo de procesamiento de datos del sistema.**

4.5 Desarrollo y diseño del sistema

En esta sección de muestra la realización del sistema en cuanto al diseño en la programación de la interfaz de usuario en la cual se buscó que sea interactiva, sencilla de usar y fácil de entender; también la utilización de las librerías de visión artificial seleccionadas previamente para la detección del deportista y con ello, crear su silueta y realizar su seguimiento en el transcurso de la escalada, con la finalidad de obtener las métricas iniciales necesarias para la comparación y análisis del performance. También se hizo énfasis en crear un programa robusto para la detección de presas de la pared de escalada, debido a que era de gran importancia para la obtención de las métricas mencionadas.

4.5.1 Interfaz

La **figura 4.7** muestra a la ventana principal de la interfaz de usuario creada, mediante la utilización de la librería PyQt en Python. El usuario deberá interactuar con la interfaz para que el sistema realice un análisis adecuado; en esta, se ingresan los videos con las características definidas previamente (**véase sección 4.3.5**), en los casilleros respectivos. Al presionar los botones de "Analizar" el sistema muestra al usuario el primer fotograma del video, esto tiene la finalidad de que el usuario seleccione el área de la trayectoria en la cual el usuario está realizando su entrenamiento, y de la misma manera deberá realizar el procedimiento para en laso del deportista de referencia.



Fig 4.7 **Ventana principal de la interfaz de usuario realizada.**

Una vez que haya realizado el reconocimiento del área dónde se encuentre el deportista, el sistema mostrará al usuario los resultados de la obtención de las métricas iniciales (**ver figura 4.8**), estas son:

- **Tiempo:** consta de tiempo general de escalada y contacto de presas.
- **Trayectoria:** se refiere al lugar de desplazamiento del deportista.
- **Técnica de movimiento:** hace referencia a la detección y cuantificación de agarres.
- **Biomecánica:** constituye el seguimiento de la silueta del deportista.

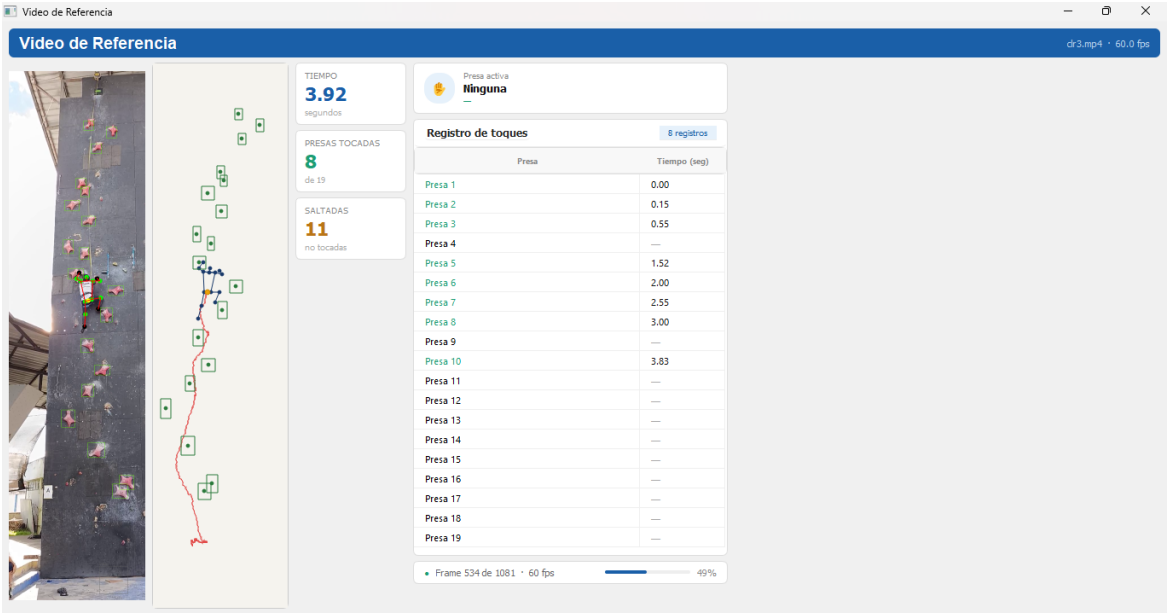


Fig 4.8 **Ventana asignada a la visualización de la obtención de métricas.**

Todas estas métricas serán utilizadas en la comparación del deportista con el sujeto de referencia y con ello, la obtención de resultados de manera gráfica hacia el usuario.

Finalmente, una vez que hayan revisado la obtención de métricas y parámetros de ambos videos, se podrá inicial el análisis de performance del deportista mediante el accionamiento de el botón de “ANÁLISIS GENERAL”, en la **figura 4.9** se muestra la ventana que se despliega, dónde se muestran los resultados obtenidos.

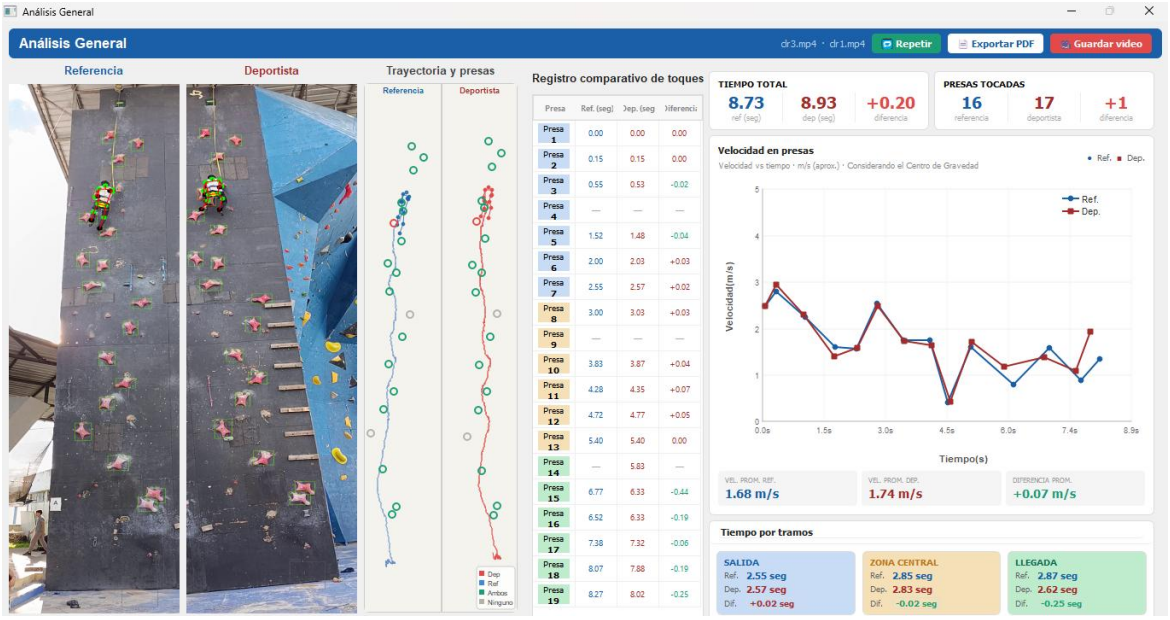


Fig 4.9 **Ventana que muestra el análisis general y la obtención de resultados.**

4.5.2 Implementación de la detección de pose y seguimiento del deportista

Se implementó un modelo de estimación de pose con el fin de identificar puntos anatómicos importantes del escalador, tales como: muñecas, tobillos y el punto medio de la cadera.

A través de la librería de MediaPipe y la utilización de su modelo de pose estimation, se asignó al código los puntos de referencia necesarios para el análisis, estos tuvieron importancia en la creación de la silueta del deportista que realiza la escalada y en la obtención de métricas que posteriormente sirvieron en la comparación y obtención de resultados.

Los puntos de referencia considerados se muestran en la **tabla 4.19**, en esta se indica a la parte del cuerpo y el número o índice a la que MediaPipe tiene asignado dentro de su librería.

Tabla 4.19 Puntos de referencia utilizados en el sistema

Parte del cuerpo	Índice
Cabeza	0
Hombro izquierdo	11
Hombro derecho	12
Codo Izquierdo	13
Codo derecho	14
Muñeca Izquierda	15
Muñeca derecha	16
Cadera (Izquierda)	23
Cadera (Derecha)	24
Rodilla izquierda	25
Rodilla derecha	26
Tobillo izquierdo	27
Tobillo derecho	28

En la codificación se crearon dos puntos adicionales que no existían dentro de la librería de MediaPipe, un punto se creó con la finalidad de posicionarse en la mitad de los hombros y ser conectado con el punto 0, correspondiente a la cabeza; el siguiente punto se posicionó en la mitad de la cadera, a este se lo utilizó como una aproximación al centro de gravedad de una persona, y se lo utilizó con el fin de graficar la trayectoria del escalador.

En la **figura 4.10** se muestra los resultados obtenidos de la detección de pose y seguimiento del deportista, el cual se basa en la muestra de la silueta del deportista y en graficación de la trayectoria de escalada.



Fig 4.10 **Detección de pose y seguimiento del escalador.**

4.5.3 Detección de presas

Para que no se presenten inconvenientes en la detección de presas, se decidió realizar el entrenamiento de la librería YOLOv11; con la finalidad de obtener una codificación robusta en la detección de presas a partir de la obtención de un numero específico de imágenes que contengan distintos casos en los cuales se pueda presentar una presa al momento de realizar la obtención del material audiovisual.

Con ello, se procedió a realizar un detector de objetos personalizado mediante la utilización de YOLOv11, previamente se creó un dataset con la utilización de Roboflow, es decir, que se utilizaron las imágenes de las presas para obtener información como forma y color, y con ello posteriormente ser detectadas, una vez obtenido el dataset se comenzó a entrenar a la librería a través de la plataforma de Google Colab; y finalmente, se realizaron pruebas de predicciones del modelo realizado, obteniendo resultados favorables (**ver figura 4.11**).

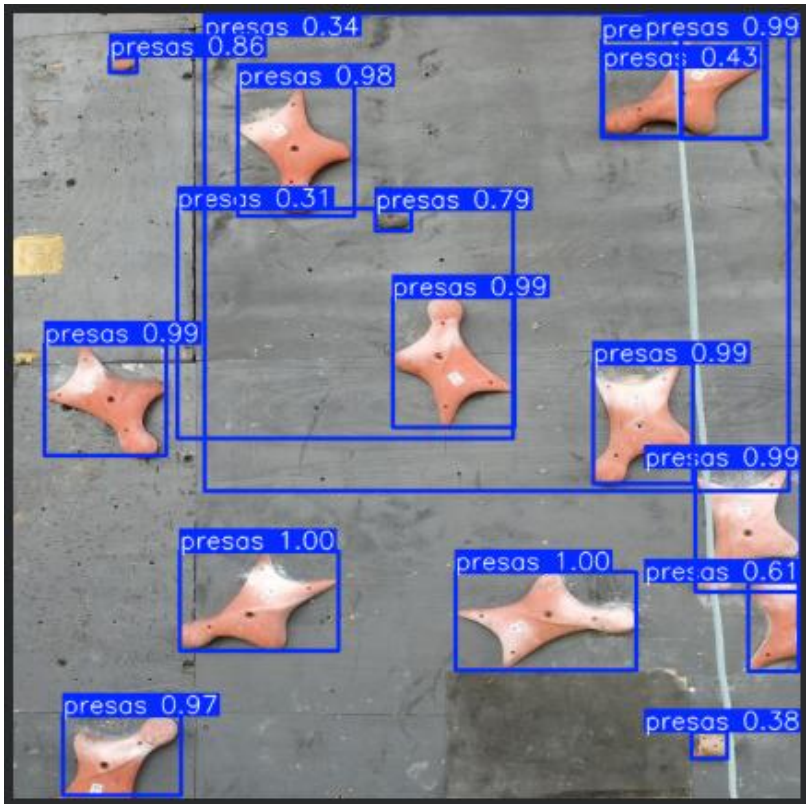


Fig 4.11 **Pruebas de predicciones del modelo creado para detección de presas.**

4.5.4 Obtención de métricas y resultados

Primeramente, la obtención de métricas se realizó mediante la combinación de la codificación de la silueta del escalador y la detección de las presas en la pared, esto permitió extraer información cinemática y métricas de tiempo tanto del escalador de referencia como del deportista a analizar.

Se utilizó metodologías de registro de tiempo para identificar el tiempo total de escalada y del agarre entre presa y presa, esto se realizó mediante la interacción de los puntos clave de la silueta del deportista con las presas de la pared, el registro de tiempo inicia cuando el sistema detecta que el escalador levanta el pie que se encuentra en el suelo y continua registrando la intersección de los puntos clave de las muñecas con las presas, además del tiempo que transcurre cuando el escalador suelta una presas y agarra otra; este registro finaliza cuando una de las muñecas interseca con un área correspondiente al botón que finaliza el conteo de tiempo en la pared de escalada (**ver figura 4.12**). También se aprovechó el registro de intersección de presas y los puntos clave de las muñecas para cuantificar el número de agarres de presas que realiza el escalador.

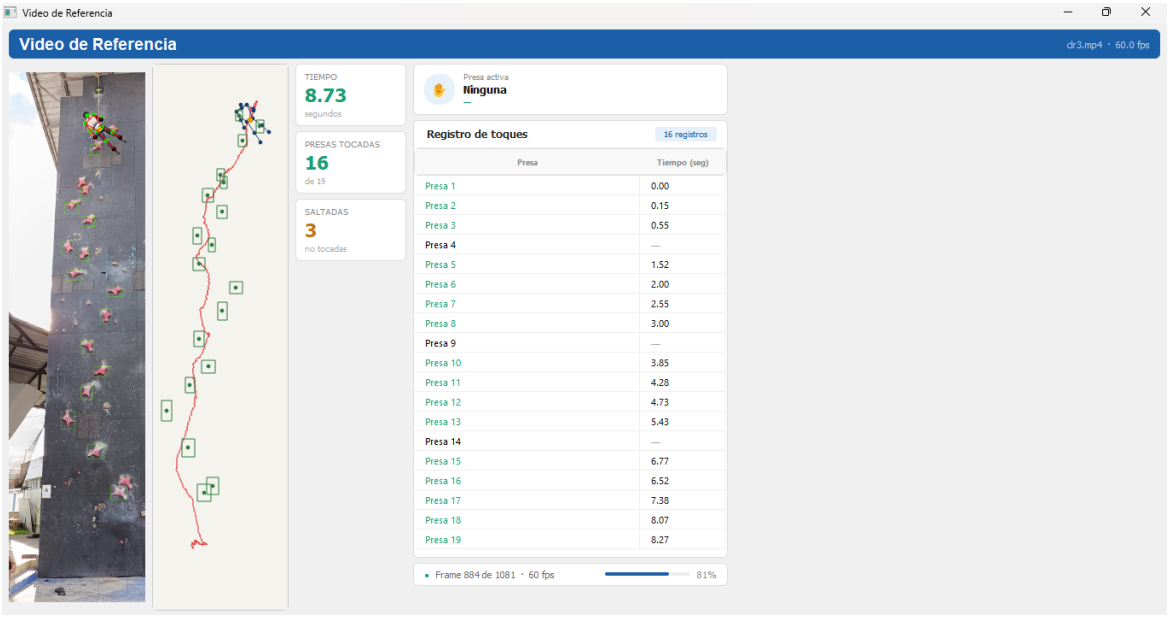


Fig 4.12 **Visualización de las métricas obtenidas.**

Bajo este mismo procedimiento se grafica mediante líneas a la trayectoria del escalador en la pared, utilizando el punto que se creó, el cual se ubicaba en el medio de la cadera, tal como se muestra en la **figura 4.13**.



Fig 4.13 **Graficación de la trayectoria del escalador.**

Con esto, se obtiene las métricas necesarias para realizar la comparación del deportista de referencia con el deportista a analizar; además de mostrar de manera gráfica al usuario los resultados conseguidos de la obtención de métricas que realizó el sistema (**ver figura 4.14**).

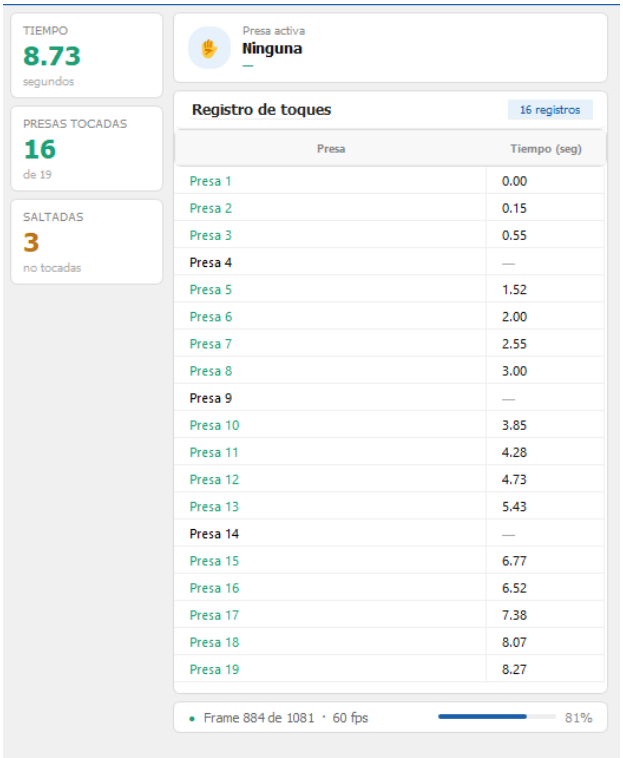


Fig 4.14 **Métricas obtenidas que se visualizan en la interfaz**

4.6 Análisis de datos

En esta sección se presenta la metodología que se utilizó para la comparación de métricas entre el deportista de referencia con el deportista a analizar y el procedimiento realizado para la obtención de resultados.

4.6.1 Comparación de métricas de rendimiento

El sistema creado registra las métricas en un tipo de matriz de cada uno de los dos escaladores en las cuales consta información de todas las presas que se encuentran en la pared, en ella se empareja la información obtenida del análisis de la obtención de métricas y posteriormente se continua con la comparación de estas métricas.

Se compara la información de estas dos matrices y se realizan los cálculos respectivos de acuerdo a qué tipo de resultado se quiere obtener y, todas las resultantes se los registra en una nueva matriz la cual servirá para mostrar a través de la interfaz toda la información que propuso mostrar al usuario.

En la **tabla 4.20** se muestra la manera en que la que se realizó en registro de resultantes en la matriz.

Tabla 4.20 Registro de datos

N.º	Parámetro	Descripción
1	**Tiempo**	Se registra ambos tiempos totales de ascenso y por tramos.
2	**Tiempo entre presas**	Para el tiempo entre presas consecutivas tocadas por las manos, se registra el menor valor de tiempo realizado.
3	**Número de movimientos**	Se registra el número menor de agarres.
4	**Velocidad**	La velocidad de agarre entre presa y presa y, la velocidad de ascenso promedio por tramos y de toda la pared se registra de los datos obtenidos del deportista a analizar, pero previamente se realiza una comparación y se mostrará un indicador en caso de que el valor haya sido favorable o no para el deportista a analizar.
5	**Trayectoria**	Se registran ambas trayectorias y se añaden indicadores para ser diferenciadas.
6	**Técnica de movimiento**	Se registran las presas que fueron utilizadas, con sus respectivos indicadores para los casos en donde, las presas hayan sido tocadas por ambos escaladores, las que solo utilizó el deportista de referencia y para las que solo utilizó el deportista a analizar.
7	**Comparación de métricas**	Se calculó de la variación de resultados entre el deportista de referencia y el deportista a analizar
8	**Detección de fallos**	Detección de momentos en el que escalador permanece inmóvil o con un intervalo de velocidad definido y, cuando el deportista suelte una presa, retroceda o se caiga, esto se logra mediante detección de interferencias en el proceso de registro de tiempo.

4.6.2 Interpretación de resultados del análisis

Una vez que se realizó la comparación de métricas de rendimiento y se obtuvo la

nueva matriz de datos, los cuales pertenecen a los resultados del análisis del sistema; se procede a realizar la codificación final que corresponde a la visualización de resultados hacia usuario a través de distintos métodos gráficos, esto se realizó con la finalidad de que, en la interfaz se muestren resultados intuitivos y fáciles de interpretar por parte del usuario.

Se consideraron varios aspectos para el diseño de la interfaz, dependiendo de qué tipo de datos se obtuvo y la mejor manera de visualizarlo (**ver figura 4.15**), dichos aspectos fueron:

- El registro de tiempo se lo mostró mediante tablas, ya que en este tipo de resultados se manejaba un gran número de datos y mostrarlo de esta manera iba a ser de fácil entendimiento para el usuario.
- En aspectos donde se manejó un solo tipo de datos, como registro de tiempo total de ascenso o número de movimientos o agarres, se asignó casilleros individuales.
- Para el caso de la trayectoria y la técnica de movimiento se mostró gráficamente a través de un recuadro que se ubicó a lado del video del deportista, este constó con la silueta de las presas y del deportista, la trayectoria se dibujó en el recuadro a medida que el escalador ascendía y se marcaron las presas que éste sostenía a medida que realizaba la escalada.
- Para la detección de pausas y fallos, se creó un indicador el cual se activaría en caso de que ocurra algunos de estos inconvenientes.
- Finalmente, para poder identificar que datos correspondían a cada uno de los deportistas, es decir, que datos fueron favorables y que datos tuvo en contra el deportista a analizar, se procedió a añadir colores a todos los datos que se iban a visualizar, correspondiendo el color verde para datos que correspondan al deportista a analizar y en rojo para el deportista de referencia.

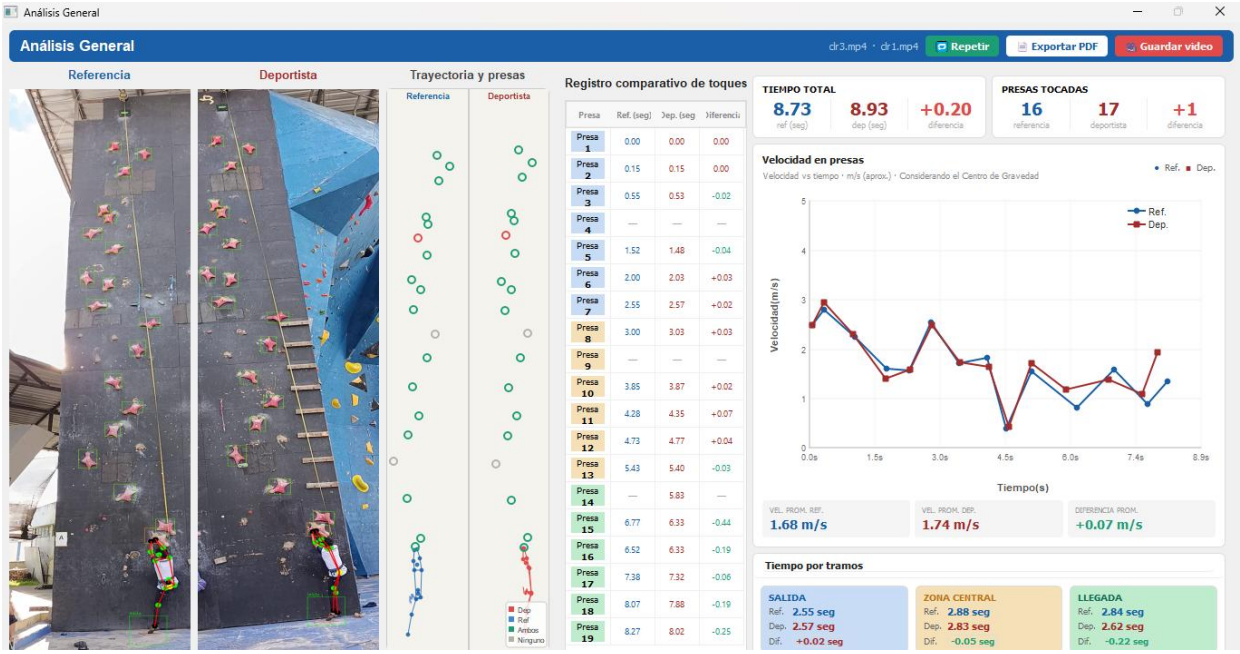


Fig 4.15 **Diseño final de la ventana que muestra los resultados del análisis al usuario.**

Adicional a esto, se añadieron dos botones que corresponden a la generación de los archivos por los cuales se podrá guardar el análisis realizado, un botón genera en reporte del análisis en formato PDF y el otro un video donde se muestra el ascenso del escalador y los resultados obtenidos.

4.7 Pruebas de funcionamiento y validación del sistema

En esta sección se evidencia la validación del sistema diseñado, para ello se procede a documentar el protocolo que se llevó a cabo para realización de las pruebas de funcionamiento; este protocolo verificará el correcto funcionamiento del sistema y los resultados que este proporciona cuando el material audiovisual obtenido por los usuarios cumple con el formato establecido y cuando no lo cumple.

Primeramente, el entorno donde se realizó las pruebas constó de un ordenador con sistema operativo Windows 11, procesador Intel Core I7 y 16GB de RAM; en esta etapa ya no se manejó al Entorno de Desarrollo Integrado (IDE), ni a el lenguaje de programación (Python) ni a ninguna de las librerías utilizadas en el diseño del sistema, se utilizó directamente el archivo ejecutable a fin de que este también sea validado; con ello, se empleó un conjunto de 4 pares de videos los cuales constan de diferentes características

como formato adecuado y formato no adecuado, que básicamente corresponden a casos dónde los videos fueron obtenidos por el tesista (entornos controlados) y casos dónde fueron obtenidos por deportistas y entrenadores (circunstancias reales).

Seguido de ello, se evaluó al sistema diseñado por etapas, en las cuales se analizó su correcto funcionamiento y su interacción con el material audiovisual ingresado; las etapas que se consideraron fueron:

- **Etapa de ejecución y portabilidad:** Se verificó que el archivo ejecutable del sistema funcione correctamente en un computador, sin la necesidad de instalar dependencias externas.
- **Etapa de verificación modular:** Se validó de forma individual cada componente del sistema: carga de video, detección de presas, estimación y seguimiento de pose, obtención de métricas, cálculo de resultados.
- **Etapa de exportación de resultados:** Se comprobó la correcta generación de las diferentes opciones por las cuales se pueden guardar los resultados del análisis obtenido, tanto el documento en pdf como la generación del video.
- **Etapa de verificación integrada:** se ejecutó todo el proceso de análisis, desde el ingreso del material audiovisual hasta la generación y exportación de resultados.
- **Etapa de pruebas de robustez:** Se evaluó el comportamiento del sistema bajo diferentes casos, es decir casos donde el material audiovisual cumple con el formato establecido y casos donde no los cumple.

A continuación, se muestran los resultados de la realización de pruebas y validación del sistema diseñado, primeramente, se realizaron pruebas en entornos controlados, con videos obtenidos por el tesista dentro de la institución donde los deportistas practican la disciplina de escalada de velocidad, seguido de ello, con videos obtenidos por deportistas y entrenadores de la institución; se analizaron los resultados obtenidos teniendo en cuenta las etapas mencionadas y con ello se valida el sistema diseñado.

4.7.1 Pruebas de funcionamiento con entornos controlados

Una vez finalizado el diseño del sistema, se procedió a transformarlo a un archivo ejecutable, es decir, se lo convirtió a un archivo que pueda ser instalado en cualquier ordenador.

Se verificó que este archivo no tuviera ningún tipo de problema durante su instalación,

realizando pruebas en un ordenador que no cuente con la IDE, Python ni alguna librería utilizada en el diseño del sistema.

Para las verificación de las etapas que corresponden al funcionamiento se obtuvo material audiovisual con especificaciones precisas las cuales permitían generar los videos adecuados para un sistema de visión artificial, estos se lograron bajo estrictos entornos como: buena calidad de imagen, buena iluminación, vestimenta adecuada del deportista, adecuado tamaño y orientación del video, entre otros., esto se ingresó a la aplicación del sistema instalada y se procedió a realizar el análisis, obteniendo resultados favorables, los cuales se muestran en la **figura 4.16**.

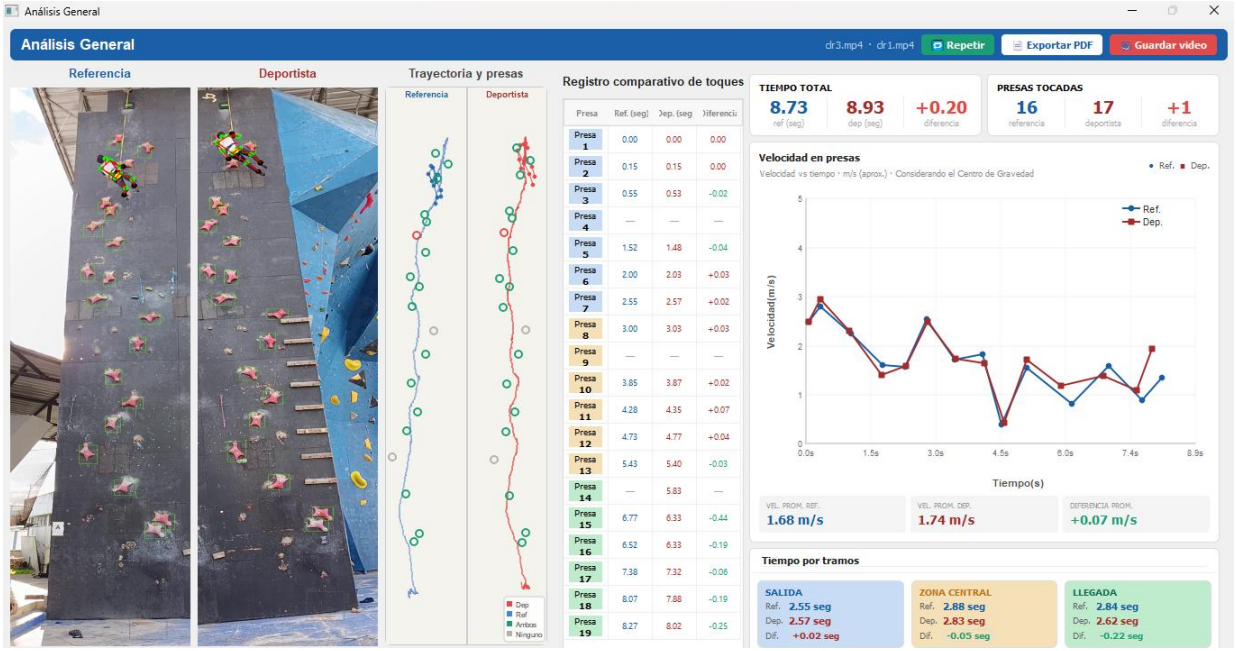
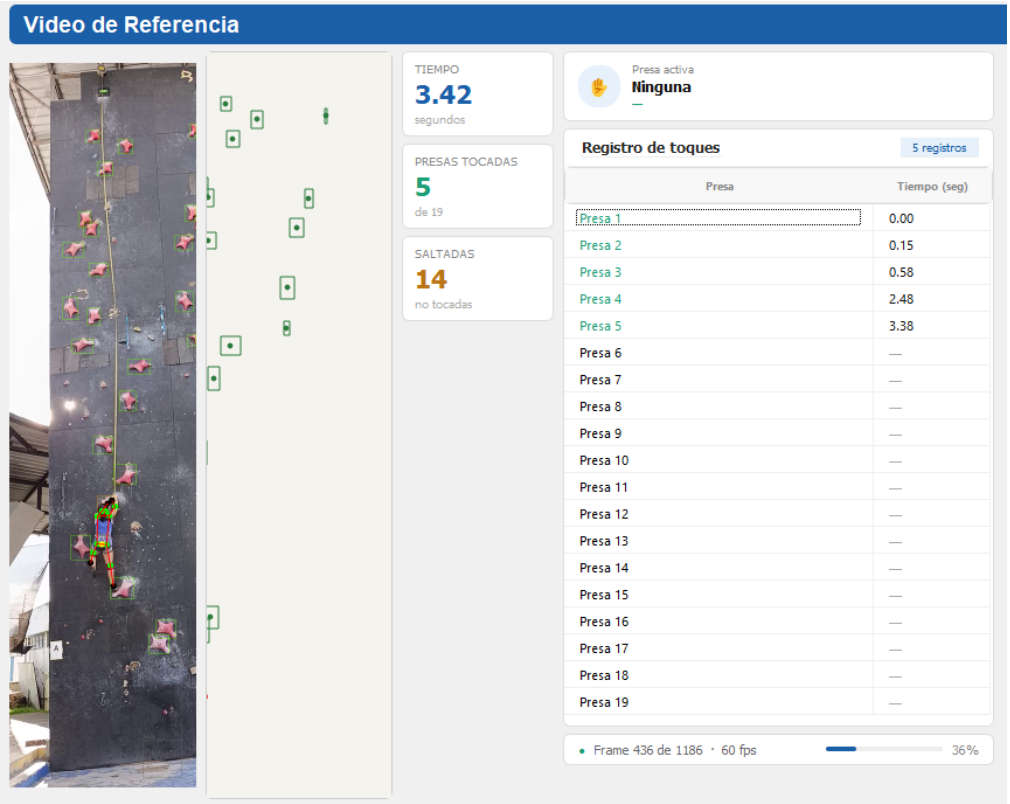


Fig 4.16 **Realización de pruebas de funcionamiento utilizando material audiovisual obtenido por el tesista.**

En la verificación de la etapa de pruebas de robustez se llegó a la conclusión de si el video obtenido para el análisis cumple con el formato establecido, el sistema no tendrá problemas para proceder a analizar, cabe mencionar que es recomendable que, en el video, solo aparezca el deportista en todo el transcurso de la escalada, ya que, si aparece en el video otra persona, a pesar de que el sistema es bastante robusto, puede generar una falsa detección y generar métricas incorrectas.

Por otra parte, se evidencia de que el sistema no es robusto cuando el video ingresado no cumple con el formato establecido (**ver figura 4.17**), ya que en las especificaciones del sistema a diseñar que se realizaron se detalla que este sistema solo será funcional bajo estrictas características que debe poseer el material audiovisual obtenido.

(a)



(b)

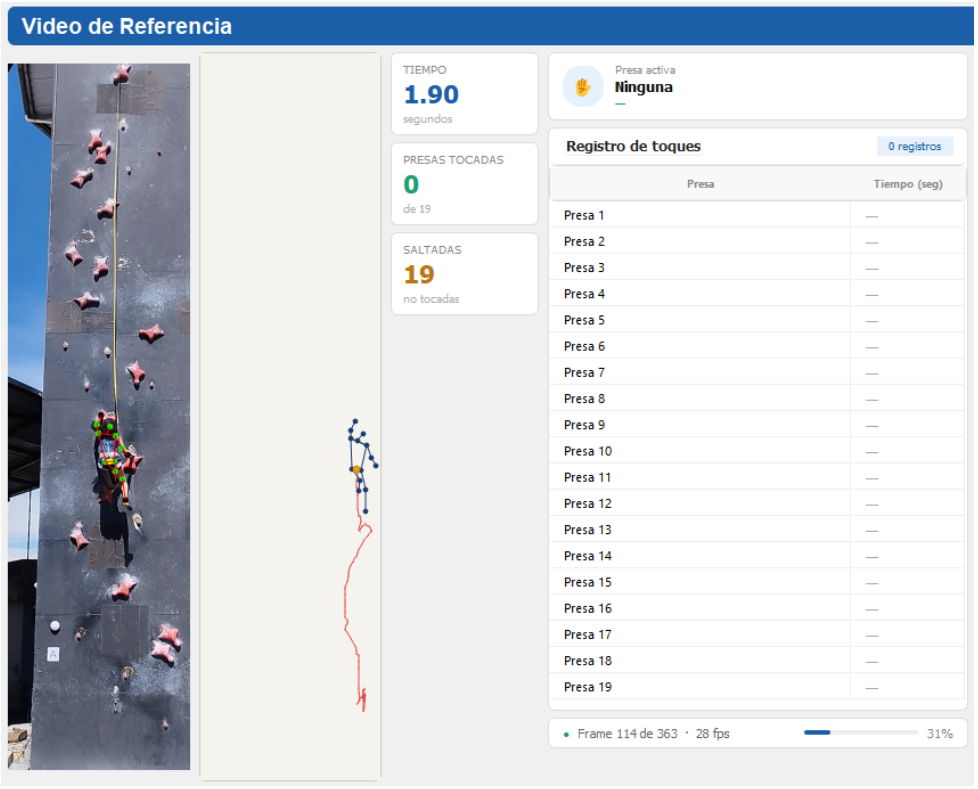


Fig 4.17 **Casos donde el sistema diseñado falla por el material audiovisual que no cumple con el formato establecido: (a) En video muestra presas de la otra ruta y (b) el video no muestra la pared completa y posee iluminación demasiado alta.**

4.7.2 Validación del sistema en circunstancias reales

Después de haber obtenido resultados correctos en los análisis personales realizados, se validó el sistema a través de material audiovisual obtenido por deportistas y entrenadores de esta disciplina, con la finalidad de conocer el comportamiento del programa, así como los resultados obtenidos, cuando ya está siendo utilizado por el usuario.

En varias pruebas realizadas, se determinó que el programa funciona correctamente, solo existieron inconvenientes en el tamaño del video que se ingresaba, puesto a que los diferentes dispositivos donde los usuarios obtenían el material audiovisual generaban diferentes tamaños de video, y al ingresar al programa, este generaba errores al momento de que el sistema comenzaba a analizar (**ver figura 4.18**). Una vez corregido este problema, el programa ya no presentaron problemas en las siguientes pruebas de validación con el resto de videos obtenidos por los usuarios (**ver figura 4.19**), y con esto, finalmente se concluye con el diseño del sistema de visión artificial para un velocista de escalada deportiva.

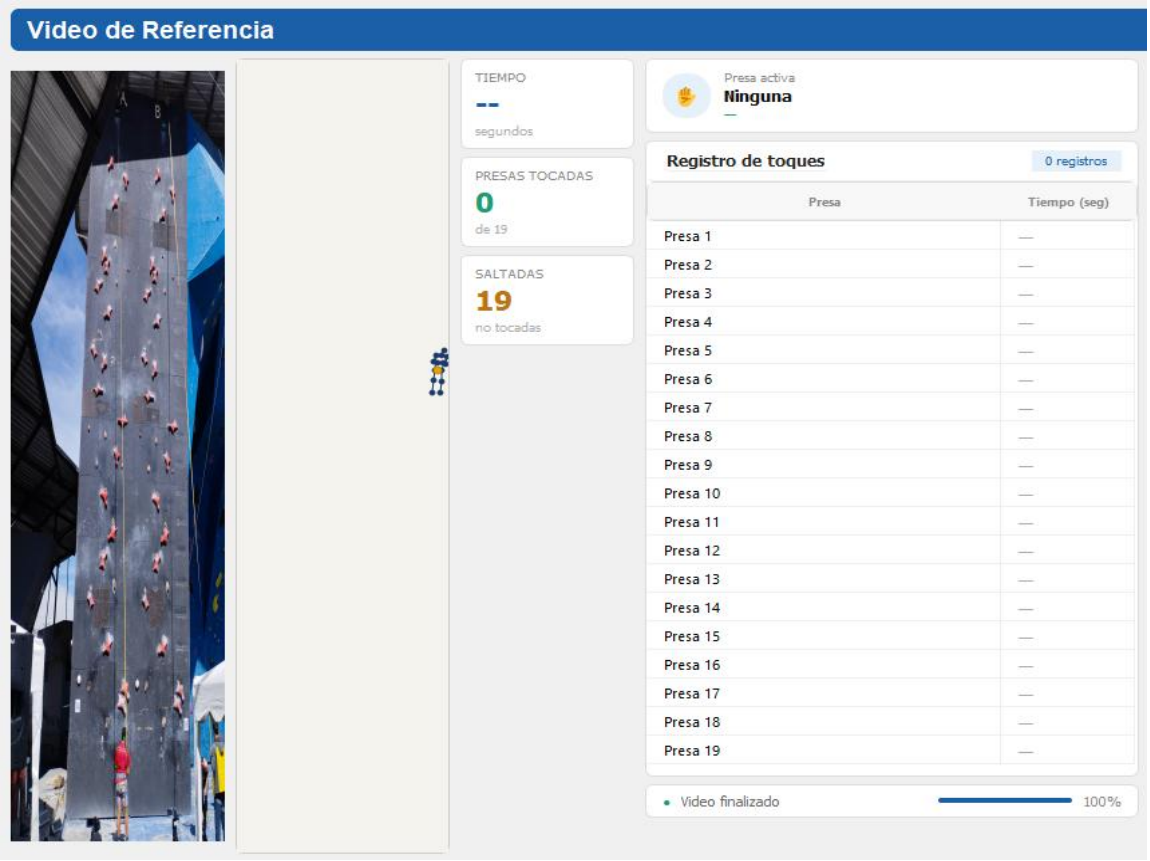


Fig 4.18 **Generación de error de análisis por diferente tamaño de video obtenido por el usuario.**

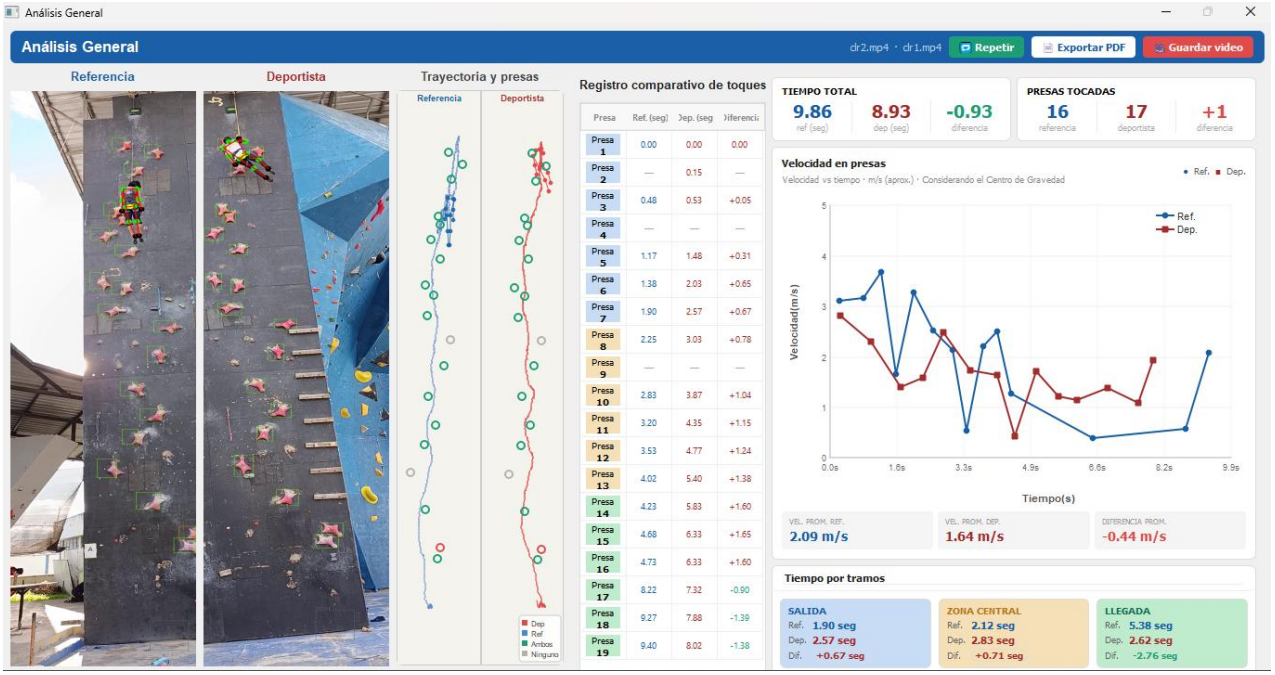


Fig 4.19 **Realización de pruebas para validación del sistema.**

Capítulo V

Conclusiones y recomendaciones

5.1 Conclusiones

Se logró identificar y sistematizar los principales aspectos que se involucran en el rendimiento de un velocista de escalada deportiva, entre ellos los tiempos (total, entre presas y por tramos), la trayectoria del deportista sobre la ruta, el número de presas tocadas y la velocidad. Esta conceptualización constituyó la base teórica que orientó al diseño del sistema, obtener justo los resultados que necesitaba el usuario para el análisis de su rendimiento.

El diseño del sistema utilizando visión artificial fue capaz de obtener datos objetivos del desempeño del escalador a partir de la obtención del material audiovisual, integrando un modelo de detección de presas a través de la utilización de YOLO y un modelo de estimación de pose como es MediaPipe. El diseño contempla con la obtención de las métricas para el análisis dual de video y la visualización de resultados hacia el usuario, garantizando que los componentes y módulos trabajen de manera coordinada para extraer y generar la información definida en la etapa de conceptualización.

Se implementó un prototipo funcional en forma de aplicación de escritorio desarrollada en el lenguaje de programación Python y con una interfaz diseñada utilizando PyQt, esto materializa el diseño propuesto y también integra a los módulos de detección de presas y estimación de silueta del deportista y la extracción de métricas, análisis y visualización y exportación de resultados. El prototipo brinda dos alternativas que

permiten guardar los resultados obtenidos del análisis, los cuales permiten una retroalimentación cuantitativa para la evaluación del rendimiento del deportista.

Se validó el sistema siguiendo un protocolo de pruebas aplicado a un número de casos determinado en donde se obtuvieron videos por parte del tesista y de los entrenadores y deportistas; con ello se pudo verificar el correcto funcionamiento del archivo ejecutable, el funcionamiento de cada módulo y la correcta obtención de resultados por parte del sistema. Los correctos resultados obtenidos evidencian que el sistema opera de manera estable y produce datos coherentes siempre y cuando se respete los resultados establecidos; quedando demostrada su aplicabilidad en condiciones o circunstancias reales de uso.

5.2 Recomendaciones

Se recomienda continuar optimizando el módulo de estimación de pose utilizando MediaPipe, especialmente en la estabilidad del punto medio de la cadera (hip_mid) y en la detección precisa de muñecas y pies. Durante el desarrollo se evidenció que pequeñas variaciones frame a frame pueden generar ruido en la trayectoria; por ello, sería conveniente intentar implementar nuevos filtros de suavizado para mejorar la cinemática sin perder información relevante.

También es recomendable fortalecer el sistema de detección de presas mediante la combinación de más técnicas como la de segmentación por color y análisis de contornos, especialmente para resolver el problema de presas rojas que en ciertos frames aparecen oscuras o negras.

Por último, se recomienda obtener el material audiovisual exactamente como indica las indicaciones del programa realizado, que se encuentra en la interfaz, en la ventana principal, debido a que, si el video ingresado tiene inconvenientes, el porcentaje de error en los resultados ya es evidente y considerable.

Bibliografía

[1] P. Legreneur, I. Rogowski, and T. Durif, “Kinematic analysis of the speed climbing event at the 2018 Youth Olympic Games,” *Comput Methods Biomech Biomed Engin*, vol. 22, no. sup1, pp. S264–S266, May 2019, doi: 10.1080/10255842.2020.1714907.

[2] L. Reveret, S. Chapelle, F. Quaine, and P. Legreneur, “3D Visualization of Body Motion in Speed Climbing,” *Front Psychol*, vol. 11, Sep. 2020, doi: 10.3389/fpsyg.2020.02188.

[3] P. Elias, V. Skvarlova, and P. Zezula, “SPEED21: Speed climbing motion dataset,” in *MMSports 2021 - Proceedings of the 4th International Workshop on Multimedia Content Analysis in Sports, co-located with ACM MM 2021*, 2021. doi: 10.1145/3475722.3482795.

[4] H. Iguma, A. Kawamura, and R. Kurazume, “A New 3D Motion and Force Measurement System for Sport Climbing,” in *2020 IEEE/SICE International Symposium on System Integration (SII)*, IEEE, Jan. 2020, pp. 1002–1007. doi: 10.1109/SII46433.2020.9026213.

[5] Á. Mira Abad, “Mejora de rendimiento deportivo mediante la detección de posturas por visión artificial,” Sep. 2023, Accessed: May 13, 2024. [Online]. Available: https://riunet.upv.es:443/handle/10251/196851

[6] K. Cha, E. Y. Lee, M. H. Heo, K. C. Shin, J. Son, and D. Kim, “Analysis of climbing postures and movements in sport climbing for realistic 3D climbing animations,” in *Procedia Engineering*, Elsevier Ltd, 2015, pp. 52–57. doi: 10.1016/j.proeng.2015.07.175.

[7] D. Pandurevic, P. Draga, A. Sutor, and K. Hochradel, “Analysis of Competition and Training Videos of Speed Climbing Athletes Using Feature and Human Body Keypoint Detection Algorithms,” *Sensors*, vol. 22, no. 6, Mar. 2022, doi: 10.3390/s22062251.

[8] Julien Pansiot, Rachel C. King, Douglas G. McIlwraith, Benny P. L. Lo, and Guang-Zhong Yang, *2008 5th International Summer School and Symposium on Medical Devices and Biosensors.* I E E E, 2008.

[9] D. Pandurevic, A. Sutor, and K. Hochradel, “Methods for quantitative evaluation of force and technique in competitive sport climbing,” in *Journal of Physics: Conference Series*, Institute of Physics Publishing, Nov. 2019. doi: 10.1088/1742-6596/1379/1/012014.

[10] D. Pandurevic, A. Sutor, and K. Hochradel, “Towards statistical analysis of predictive parameters in competitive speed climbing,” *Sports Engineering*, vol. 26, no. 1, Dec. 2023, doi: 10.1007/s12283-023-00431-7.

[11] R. Pardo and E. Danilo, “LA TÉCNICA EN LA ESCALADA DEPORTIVA: UNA REVISIÓN THE TECHNIQUE IN THE SPORT CLIMBING: A REVIEW,” 2015.

[12] Greg Presto, “La escalada deportiva en la Olympic Qualifier Series: todo lo que necesitas saber,” Olympics.com.

[13] ifsc-climbing.org, “Speed Climbing,” International Federation of Sport Climbing.

[14] M. Werlayne Stuart Soares Leite, “Biomecánica aplicada al deporte: contribuciones, perspectivas y desafíos,” *Lecturas: Educación física y deportes, ISSN-e 1514-3465, Nº. 170, 2012, 9 págs.*, no. 170, pp. 1–9, 2012, Accessed: Jul. 08, 2024. [Online]. Available:

https://dialnet.unirioja.es/servlet/articulo?codigo=4741932&info=resumen&idioma=ENG
[15] P. Balthazard, D. Currat, and F. Degache, “Fundamentos de biomecánica,” *EMC - Kinesiterapia - Medicina Física*, vol. 36, no. 4, pp. 1–8, Nov. 2015, doi: 10.1016/S1293-2965(15)74142-3.
[16] L. Márquez Benavides, E. L. Moreno Goytia, and L. F. Gonzalez Ramirez, “El uso de la inteligencia artificial en un entorno académico,” *Ciencia Nicolaita*, no. 89, Aug. 2023, doi: 10.35830/cn.vi89.721.
[17] IBM.com, “¿Qué es la visión artificial? | IBM.” Accessed: Jul. 12, 2024. [Online]. Available: https://www.ibm.com/es-es/topics/computer-vision
[18] cognex.com, “INTRODUCCIÓN A LA VISIÓN ARTIFICIAL Una guía para la automatización de procesos y mejorar la calidad”.
[19] R. C. . Gonzalez and R. E. . Woods, *Digital image processing*. Pearson, 2018.
[20] L. Llumiquinga and F. Rafael, “ESCUELA POLITÉCNICA NACIONAL FACULTAD DE INGENIERÍA MECÁNICA PROCESAMIENTO DE IMÁGENES MEDIANTE SOFTWARE LIBRE PYTHON PARA EL ANÁLISIS METALOGRÁFICO EN ACEROS DE BAJO CONTENIDO DE CARBONO PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO MECÁNICO FREDY RAFAEL LLULLUNA LLUMIQUINGA.”
[21] M. Souaifi *et al.*, “Artificial Intelligence in Sports Biomechanics: A Scoping Review on Wearable Technology, Motion Analysis, and Injury Prevention,” Aug. 01, 2025, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi: 10.3390/bioengineering12080887.
[22] L. Wade, L. Needham, P. McGuigan, and J. Bilzon, “Applications and limitations of current markerless motion capture methods for clinical gait biomechanics,” *PeerJ*, vol. 10, Feb. 2022, doi: 10.7717/peerj.12995.
[23] Cevallos Alicia, Polo Ernesto, Salgado Diana, and Orbea Marlon, *Métodos y técnicas de Investigación*, 1st ed. Guayaquil: Ediciones Grupo Compás, 2017.
[24] P. M. Vallejo, “Investigación experimental, diseño y contraste de medias.,” Madrid: Universidad Pontificia Comillas. Accessed: Oct. 06, 2025. [Online]. Available: http://students.aiu.edu/submissions/profiles/resources/onlineBook/A4t2k2_diseno%20investigacion.pdf
[25] S. Askari Hosseini and P. Wolf, “Performance indicators in speed climbing: insights from the literature supplemented by a video analysis and expert interviews,” 2023, *Frontiers Media SA*. doi: 10.3389/fspor.2023.1304403.
[26] B. T. Naik, M. F. Hashmi, and N. D. Bokde, “A Comprehensive Review of Computer Vision in Sports: Open Issues, Future Trends and Research Directions,” *Applied Sciences (Switzerland)*, vol. 12, no. 9, May 2022, doi: 10.3390/app12094429.

Anexos

A.1 Encuesta realizada a deportistas y entrenadores.

Encuesta: Sistema de análisis de performance de un velocista de escalada deportiva mediante visión artificial.

Estimado encuestado, esta encuesta es totalmente anónima, su propósito es recolectar información tales como: datos técnicos, experiencias y necesidades; que pueda llegar a ser importantes para la elaboración del proyecto presentado con el fin de optimizar el rendimiento en este deporte mediante herramientas de visión artificial; por ello, espero objetividad en sus respuestas.

Instrucciones:

Marca con ✓ o selecciona la opción que corresponda.

Los apartados con (*) son obligatorios.

1. Datos básicos (*)

Edad _____

Rol:

- Deportista
- Entrenador
- Otro (especifíca) _____

Experiencia en escalada deportiva de velocidad:

- < 1 año
- 1-3 años
- 4-6 años
- > 6 años

2. Entrenamiento actual (*)

¿Qué aspectos usa actualmente para medir el rendimiento? *(Marque todas las que aplique)*

- Tiempo total de ascenso
- Técnica de movimiento (patrón de pies y/o manos)
- Fuerza aplicada en presas
- Centro de gravedad
- Trayectoria
- Sincronización de movimientos
- Postura
- Velocidad por segmentos
- Tiempo por segmentos

- Otro _____

¿Qué herramientas tecnológicas utilizan? *(Marque todas las que aplique)*

- Cámaras convencionales
- Sensores portátiles
- Softwares de análisis
- Ninguna
- Otro _____

¿Qué aspectos del gesto técnico les cuesta más analizar con métodos tradicionales?

_____

¿Cómo registran y analizan los errores durante el entrenamiento? (ejemplo: videos post-ronda, feedback inmediato).

_____

3. Necesidades y desafíos

¿Qué información les gustaría obtener que actualmente no pueden medir? *(Marque todas las que aplique)*

- Ángulo exacto de las articulaciones
- Eficiencia de movimientos
- Otro _____

¿Ha probado sistemas de visión artificial para el análisis del performance (rendimiento)?

Sí ____ No ____

Si la respuesta es **No**, avance al siguiente numeral.

Si la respuesta es **Sí**, ¿qué beneficios y limitaciones encontraron?

_____

4. Experiencia del entrevistado

Como deportista: ¿Qué errores técnicos cometes frecuentemente y cómo los corriges?

_____

Como entrenador: ¿Qué patrones de movimiento identificas como claves para mejorar el tiempo?

_____

5. Sobre visión artificial (*)

¿Qué aspectos del performance le gustaría que analice un sistema de visión artificial? *(Marca máximo 3 opciones)*

- Tiempo
- Técnica de movimiento
- Fuerza aplicada
- Centro de gravedad
- Trayectoria
- Sincronización de movimientos
- Postura
- Velocidad por segmentos
- Tiempo por segmentos
- Otro _____

¿Prefiere un sistema en tiempo real (durante el entrenamiento) o análisis posterior?

- Tiempo Real
- Análisis posterior

¿Qué tipo de resultados le gustaría visualizar? *(Marca las que consideres necesarias)*

- Gráficos simples, como: velocidad vs. tiempo.
- Overlays en video, como: esqueleto biomecánico, trayectoria en la pared.
- Comparación de aspectos con otro deportista como referencia.

- Alertas sonoras/vibratorias durante el ascenso
- Informe detallado post-entrenamiento.
- Tiempo de escalada por presas.
- Otro _____

Del 1 al 5, ¿cuánto confiaría en un análisis generado por visión artificial?

- 1 (Nada)
- 2
- 3
- 4
- 5 (Totalmente)

6. Consentimiento y datos (*)

¿Autorizarías el uso de videos de sus entrenamientos para entrenar o desarrollar modelos de visión artificial aplicado a este deporte?

Sí ____ No ____

¿Qué condiciones impondrías respecto a la privacidad de los datos?

_____

"Agradezco inmensamente su tiempo y conocimientos compartidos. Sus respuestas son fundamentales para poder cumplir con este objetivo."

¡Gracias por su colaboración!

A.2 Enlace de archivos de evidencia.

https://drive.google.com/drive/folders/1s7BfkcdlOgD3O4eJRDIGnob95RydIptQ?usp=drive_link

A.3 Codificación realizada.

```
import sys
import cv2
import numpy as np
import mediapipe as mp
from ultralytics import YOLO
from PyQt5.QtWidgets import (
    QApplication, QMainWindow, QLabel, QWidget, QLineEdit, QVBoxLayout,
    QHBoxLayout, QPushButton, QFileDialog, QSizePolicy, QDialog,
    QFrame, QProgressBar, QTableWidget, QTableWidgetItem, QHeaderView
)
from PyQt5.QtGui import QIcon, QFont, QPixmap, QImage
from PyQt5.QtCore import Qt, QTimer

processed_reference_frames = []
processed_athlete_frames = []
# ---------------- CONFIGURACION DE POSE  ----------------

KEYPOINTS = {
    "cabeza": 0,
    "hombro_izq": 11, "hombro_der": 12,
    "codo_izq": 13, "codo_der": 14,
    "muñeca_izq": 15, "muñeca_der": 16,
    "cadera_izq": 23, "cadera_der": 24,
    "rodilla_izq": 25, "rodilla_der": 26,
    "tobillo_izq": 27, "tobillo_der": 28
}

BLACK_POINTS = {15, 16, 27, 28}

CONNECTIONS = [
    ("hombro_izq", "codo_izq"), ("codo_izq", "muñeca_izq"),
    ("hombro_der", "codo_der"), ("codo_der", "muñeca_der"),
    ("hombro_izq", "hombro_der"),
    ("cadera_izq", "mid_caderas"), ("cadera_der", "mid_caderas"),
    ("cadera_izq", "rodilla_izq"), ("rodilla_izq", "tobillo_izq"),
    ("cadera_der", "rodilla_der"), ("rodilla_der", "tobillo_der"),
    ("hombro_izq", "cadera_izq"), ("hombro_der", "cadera_der"),
    ("cabeza", "mid_hombros")
]


# Redimensiona un frame manteniendo proporción y agrega barras negras (relleno).

def resize_with_aspect_ratio(frame, target_width, target_height):
    h, w, _ = frame.shape
    aspect_ratio = w / h

    if target_width / target_height > aspect_ratio:
        # Ajuste por altura
        new_height = target_height
        new_width = int(aspect_ratio * new_height)
    else:
        # Ajuste por ancho
        new_width = target_width
        new_height = int(new_width / aspect_ratio)
```

```
    resized_frame = cv2.resize(frame, (new_width, new_height),
                               interpolation=cv2.INTER_AREA)

    # Creacion de fondo negro
    output = cv2.copyMakeBorder(
        resized_frame,
        top=(target_height - new_height) // 2,
        bottom=(target_height - new_height) // 2,
        left=(target_width - new_width) // 2,
        right=(target_width - new_width) // 2,
        borderType=cv2.BORDER_CONSTANT,
        value=(61, 4, 9)
    )

    return output


class PoseEstimator:
    """
    Pipeline hibrido YOLO + MediaPipe:
    - YOLO (cada N frames) actualiza el bounding box del deportista
    - MediaPipe siempre corre dentro de ese crop → sin fondo, sin otras personas
    - El crop sigue al deportista dinámicamente mientras sube
    """
    EMA_ALPHA      = 0.55
    MIN_VISIBILITY = 0.35
    MAX_INVALID    = 90
    YOLO_INTERVAL  = 5
    BBOX_PADDING   = 0.30  # más padding para que MediaPipe tenga más contexto

    def __init__(self):
        self._yolo = YOLO("yolov8n.pt")
        self._mp   = mp.solutions.pose
        self._pose = self._mp.Pose(
            static_image_mode=False,
            model_complexity=1,
            enable_segmentation=False,
            min_detection_confidence=0.4,
            min_tracking_confidence=0.4
        )
        self._last_landmarks = None
        self._smoothed       = {}
        self._frame_count    = 0
        self._invalid_streak = 0
        self._crop_box       = None  # (x1,y1,x2,y2) en píxeles, actualizado por YOLO
        self._ref_x_min      = 0.0
        self._ref_x_max      = 1.0
        self._ref_y_min      = 0.0
        self._ref_y_max      = 1.0

    def _preprocess(self, frame_rgb):
        hsv = cv2.cvtColor(frame_rgb, cv2.COLOR_RGB2HSV).astype('float32')
        v_mean = hsv[:, :, 2].mean()
        if v_mean > 10:
            scale = max(0.4, min(2.0, 120.0 / v_mean))
            hsv[:, :, 2] = np.clip(hsv[:, :, 2] * scale, 0, 255)
```

```
    return cv2.cvtColor(hsv.astype('uint8'), cv2.COLOR_HSV2RGB)

def _detect_bbox_raw(self, frame_rgb):
    """Deteccion del bbox con YOLO y lo retorna sin aplicarlo."""
    h, w = frame_rgb.shape[:2]
    frame_bgr = cv2.cvtColor(frame_rgb, cv2.COLOR_RGB2BGR)
    results   = self._yolo(frame_bgr, verbose=False, conf=0.25, classes=[0])
    best_box  = None
    best_conf = -1
    for r in results:
        if r.boxes is None:
            continue
        for box in r.boxes:
            x1n, y1n, x2n, y2n = box.xyxyn[0].tolist()
            cx = (x1n + x2n) / 2
            cy = (y1n + y2n) / 2
            if not (self._ref_x_min <= cx <= self._ref_x_max and
                    self._ref_y_min <= cy <= self._ref_y_max):
                continue
            conf = float(box.conf[0])
            if conf > best_conf:
                best_conf = conf
                pad_x = int((x2n - x1n) * self.BBOX_PADDING * w)
                pad_y = int((y2n - y1n) * self.BBOX_PADDING * h)
                bx1 = max(0, int(x1n * w) - pad_x)
                by1 = max(0, int(y1n * h) - pad_y)
                bx2 = min(w, int(x2n * w) + pad_x)
                by2 = min(h, int(y2n * h) + pad_y)
                best_box = (bx1, by1, bx2, by2)
    return best_box

def _update_crop_yolo(self, frame_rgb):
    box = self._detect_bbox_raw(frame_rgb)
    if box is not None:
        self._crop_box = box

def _run_mediapipe(self, frame_rgb):
    """
    Corre MediaPipe dentro del crop actual.
    Uso de un controlador PID para seguir al deportista sin oscilaciones.
    """
    h, w = frame_rgb.shape[:2]

    CROP_W = int(w * 0.40)
    CROP_H = int(h * 0.55)

    # Inicializar PID (si no existe)
    if not hasattr(self, '_pid'):
        self._pid = {
            'cx': 0, 'cy': 0,         # posición actual del centro del crop
            'ex': 0, 'ey': 0,         # error anterior
            'ix': 0, 'iy': 0,         # integral del error
            # Ganancias PID
            'Kp': 0.25,   # proporcional
            'Ki': 0.01,   # integral
            'Kd': 0.10,   # derivativo
        }
```

```
if self._crop_box is not None:
    x1, y1, x2, y2 = self._crop_box
    cx = (x1 + x2) // 2
    cy = (y1 + y2) // 2

    # Inicializa centro PID en primera detección
    if self._pid['cx'] == 0:
        self._pid['cx'] = cx
        self._pid['cy'] = cy

    x1 = max(0, self._pid['cx'] - CROP_W // 2)
    y1 = max(0, self._pid['cy'] - CROP_H // 2)
    x2 = min(w, x1 + CROP_W)
    y2 = min(h, y1 + CROP_H)

    crop = frame_rgb[y1:y2, x1:x2]
    if crop.size == 0:
        return None

    results = self._pose.process(crop)
    if results.pose_landmarks:
        lm = results.pose_landmarks.landmark
        crop_w = x2 - x1
        crop_h = y2 - y1
        landmarks = self._to_global(lm, x1, y1, crop_w, crop_h, w, h)

        # Setpoint: centro del cuerpo detectado en coords globales
        sp_x = int((lm[11].x + lm[12].x + lm[23].x + lm[24].x) / 4 * crop_w) + x1
        sp_y = int((lm[11].y + lm[12].y + lm[23].y + lm[24].y) / 4 * crop_h) + y1

        # PID
        pid = self._pid
        ex = sp_x - pid['cx']
        ey = sp_y - pid['cy']
        pid['ix'] = max(-50, min(50, pid['ix'] + ex))
        pid['iy'] = max(-50, min(50, pid['iy'] + ey))
        dx = ex - pid['ex']
        dy = ey - pid['ey']
        ux = pid['Kp'] * ex + pid['Ki'] * pid['ix'] + pid['Kd'] * dx
        uy = pid['Kp'] * ey + pid['Ki'] * pid['iy'] + pid['Kd'] * dy
        pid['cx'] = int(pid['cx'] + ux)
        pid['cy'] = int(pid['cy'] + uy)
        pid['ex'] = ex
        pid['ey'] = ey

        # Actualiza el crop_box con nuevo centro PID
        nx1 = max(0, pid['cx'] - CROP_W // 2)
        ny1 = max(0, pid['cy'] - CROP_H // 2)
        nx2 = min(w, nx1 + CROP_W)
        ny2 = min(h, ny1 + CROP_H)
        self._crop_box = (nx1, ny1, nx2, ny2)

        return landmarks

# Fallback: frame completo
results = self._pose.process(frame_rgb)
```

```
    if results.pose_landmarks:
        return results.pose_landmarks.landmark
    return None

def _to_global(self, lm, ox, oy, cw, ch, fw, fh):
    """Convierte landmarks de coords de crop a coords globales normalizadas."""
    class LM:
        def __init__(self, x, y, z, vis):
            self.x, self.y, self.z, self.visibility = x, y, z, vis
    out = []
    for p in lm:
        gx = (ox + p.x * cw) / fw
        gy = (oy + p.y * ch) / fh
        out.append(LM(gx, gy, p.z, p.visibility))
    return out

def _is_valid(self, landmarks):
    try:
        sho_y = (landmarks[11].y + landmarks[12].y) / 2
        hip_y = (landmarks[23].y + landmarks[24].y) / 2
        if sho_y >= hip_y:
            return False
        vis_count = sum(
            1 for idx in [11, 12, 23, 24]
            if landmarks[idx].visibility >= self.MIN_VISIBILITY
        )
        return vis_count >= 2
    except Exception:
        return False

def _ema_update(self, landmarks, w, h):
    updated = {}
    for idx in range(len(landmarks)):
        lm = landmarks[idx]
        if lm.visibility < self.MIN_VISIBILITY:
            if idx in self._smoothed:
                updated[idx] = self._smoothed[idx]
            continue
        nx, ny = lm.x * w, lm.y * h
        if idx in self._smoothed:
            px, py = self._smoothed[idx]
            nx = self.EMA_ALPHA * nx + (1 - self.EMA_ALPHA) * px
            ny = self.EMA_ALPHA * ny + (1 - self.EMA_ALPHA) * py
        updated[idx] = (nx, ny)
    self._smoothed = updated
    return updated

def draw_pose(self, frame):
    self._frame_count += 1
    h, w = frame.shape[:2]

    frame_pre = self._preprocess(frame)

    # YOLO cada N frames para actualizar el crop
    # Solo actualiza si aún no tenemos una detección confirmada
    # o si YOLO encuentra algo con alta confianza
    if self._frame_count % self.YOLO_INTERVAL == 0 or self._crop_box is None:
```

```
    new_box = self._detect_bbox_raw(frame_pre)
    if new_box is not None:
        # Solo actualiza el crop con YOLO si MediaPipe perdió la detección
        # o si es la primera vez — evita que YOLO mueva el crop cuando
        # el deportista está casi fuera del frame al llegar arriba
        if self._crop_box is None or self._invalid_streak > 10:
            self._crop_box = new_box

# MediaPipe siempre dentro del crop actual
landmarks = self._run_mediapipe(frame_pre)

valid_detection = False
smooth = {}
if landmarks is not None and self._is_valid(landmarks):
    self._last_landmarks = landmarks
    smooth = self._ema_update(landmarks, w, h)
    self._invalid_streak = 0
    valid_detection = True
else:
    self._invalid_streak += 1

if self._invalid_streak >= self.MAX_INVALID:
    self._last_landmarks = None
    self._smoothed       = {}
    self._invalid_streak = 0
    self._crop_box       = None
    return frame, None

if not valid_detection:
    smooth = self._smoothed if self._smoothed else {}
    if not smooth:
        return frame, None

if not smooth:
    return frame, None

landmarks = self._last_landmarks
points = {}
for name, idx in KEYPOINTS.items():
    if idx in smooth:
        x, y = int(smooth[idx][0]), int(smooth[idx][1])
        points[name] = (x, y)
        color = (0, 0, 0) if idx in BLACK_POINTS else (0, 255, 0)
        cv2.circle(frame, (x, y), 5, color, -1)

if "hombro_izq" in points and "hombro_der" in points:
    x = (points["hombro_izq"][0] + points["hombro_der"][0]) // 2
    y = (points["hombro_izq"][1] + points["hombro_der"][1]) // 2
    points["mid_hombros"] = (x, y)

if "cadera_izq" in points and "cadera_der" in points:
    x = (points["cadera_izq"][0] + points["cadera_der"][0]) // 2
    y = (points["cadera_izq"][1] + points["cadera_der"][1]) // 2
    points["mid_caderas"] = (x, y)
    cv2.circle(frame, (x, y), 5, (0, 255, 255), -1)

for a, b in CONNECTIONS:
```

```
            if a in points and b in points:
                cv2.line(frame, points[a], points[b], (0, 0, 255), 3)

        return frame, landmarks



POSE_ESTIMATOR = PoseEstimator()


# ------------------------------------------------------
#   DETECTOR DE PRESAS
# ------------------------------------------------------
class HoldDetector:
    """
    Deteccion de presas con YOLO para video estático.
    - last_known: conserva posición aunque esté ocluida (hasta 120 frames).
    - NMS manual: evita bboxes duplicados para la misma presa.
    - Umbral reducido para presas ya conocidas: menos pérdidas.
    - calibrate_from_video(): pre-puebla posiciones desde el frame central.
    """
    ASSOC_DIST = 30         # tolerancia (px) para asociar detección con presa conocida
    MAX_MISSING_FRAMES = 120
    KNOWN_CONF_THRESHOLD = 0.35  # asociar detección con presa ya conocida
(reproducción)
    NEW_CONF_THRESHOLD = 0.90    # muy alto para registrar presas nuevas (solo modo
abierto)
    NMS_IOU_THRESHOLD = 0.3  # IoU para fuionar detecciones duplicadas
    ASSOC_CONF = 0.35        # umbral YOLO para re-detectar presas durante la reproducción

    # --- Calibración por consenso temporal (multiples frames) ---
    CALIB_SAMPLES    = 30   # nº de frames muestreados a lo largo del video
    CALIB_CONF       = 0.70 # umbral YOLO moderado al calibrar (para no perder presas)
    CALIB_VOTE_RATIO  = 0.35 # fracción mínima de frames en que debe aparecer una presa
    CALIB_CLUSTER_IOU = 0.30 # IoU para agrupar la misma presa entre frames

    def __init__(self):
        self.model = YOLO("best2m20.pt")
        self.conf_threshold = 0.75
        self.last_known = {}
        self._next_id = 0

    # ------------------------------------------------------------------
    def _iou(self, b1, b2):
        ix1 = max(b1[0], b2[0]);
        iy1 = max(b1[1], b2[1])
        ix2 = min(b1[2], b2[2]);
        iy2 = min(b1[3], b2[3])
        inter = max(0, ix2 - ix1) * max(0, iy2 - iy1)
        if inter == 0:
            return 0.0
        a1 = (b1[2] - b1[0]) * (b1[3] - b1[1])
        a2 = (b2[2] - b2[0]) * (b2[3] - b2[1])
        return inter / (a1 + a2 - inter)

    def _nms(self, boxes):
        """NMS manual: elimina duplicados con IoU alto, conserva el de mayor conf."""
```

```
    if not boxes:
        return []
    boxes_s = sorted(boxes, key=lambda b: b[4], reverse=True)
    kept, suppressed = [], set()
    for i, b in enumerate(boxes_s):
        if i in suppressed:
            continue
        kept.append(b)
        for j in range(i + 1, len(boxes_s)):
            if j not in suppressed and self._iou(b, boxes_s[j]) > self.NMS_IOU_THRESHOLD:
                suppressed.add(j)
    return kept

def _raw_detections(self, frame, threshold=None):
    """
    Corre YOLO y retorna lista de (x1,y1,x2,y2,conf) después de NMS.
    """
    if threshold is None:
        threshold = self.conf_threshold
    results = self.model(frame, verbose=False)
    boxes = []
    fh, fw = frame.shape[:2]
    for box in results[0].boxes:
        conf = float(box.conf[0])
        if conf >= threshold:
            x1, y1, x2, y2 = map(int, box.xyxy[0])
            x1, x2 = max(0, x1), min(fw - 1, x2)
            y1, y2 = max(0, y1), min(fh - 1, y2)
            boxes.append((x1, y1, x2, y2, conf))
    return self._nms(boxes)

def _center(self, box):
    return (box[0] + box[2]) / 2, (box[1] + box[3]) / 2

def _dist(self, b1, b2):
    cx1, cy1 = self._center(b1)
    cx2, cy2 = self._center(b2)
    return ((cx1 - cx2) ** 2 + (cy1 - cy2) ** 2) ** 0.5

# ------------------------------------------------------------------
def _apply_crop(self, frame, crop):
    """Aplica el mismo recorte que usa el resto del pipeline."""
    if not crop:
        return frame
    if crop[0] == "manual":
        _, x, w_crop = crop
        return frame[:, x:x + w_crop]
    side, px = crop
    fh, fw, _ = frame.shape
    return frame[:, :fw - px] if side == "left" else frame[:, px:]

def calibrate_from_video(self, video_path, crop=None):
    """
    Determina el conjunto de presas por CONSENSO TEMPORAL.

    Muestrea muchos frames a lo largo del video, corre YOLO con umbral
    moderado en cada uno y agrupa las detecciones por posición. Solo se
```

conserva como presa real la que aparece en una fracción suficiente de los frames (votación). Así se detectan todas las presas reales aunque alguna detección individual sea de confianza media, y se descartan los falsos positivos que aparecen de forma esporádica.
"""
```
self.last_known = {}
self._next_id = 0

cap = cv2.VideoCapture(video_path)
total = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
if total <= 0:
    cap.release()
    return

lo, hi = int(total * 0.05), int(total * 0.95)
if hi <= lo:
    lo, hi = 0, max(1, total - 1)
step = max(1, (hi - lo) // max(1, self.CALIB_SAMPLES - 1))
sample_idxs = list(range(lo, hi + 1, step))

proc_w = 480
clusters = []
n_used = 0

for fi in sample_idxs:
    cap.set(cv2.CAP_PROP_POS_FRAMES, fi)
    ret, frame = cap.read()
    if not ret:
        continue
    frame = self._apply_crop(frame, crop)

    fh_o, fw_o = frame.shape[:2]
    scale = proc_w / fw_o
    small = cv2.resize(frame, (proc_w, int(fh_o * scale)),
                       interpolation=cv2.INTER_AREA)
    dets = self._raw_detections(small, threshold=self.CALIB_CONF)
    # a coords originales
    dets = [(int(x1 / scale), int(y1 / scale),
             int(x2 / scale), int(y2 / scale), conf)
            for (x1, y1, x2, y2, conf) in dets]
    n_used += 1

    for det in dets:
        placed = False
        for c in clusters:
            if self._iou(det, c['anchor']) > self.CALIB_CLUSTER_IOU:
                c['boxes'].append(det)
                c['frames'].add(fi)
                placed = True
                break
        if not placed:
            clusters.append({'anchor': det, 'boxes': [det], 'frames': {fi}})

cap.release()
if n_used == 0:
    return
```

```
    min_votes = max(2, int(round(n_used * self.CALIB_VOTE_RATIO)))

    def _med(vals):
        vals = sorted(vals)
        return vals[len(vals) // 2]

    kept = []
    for c in clusters:
        if len(c['frames']) < min_votes:
            continue
        bs = c['boxes']
        box = (_med([b[0] for b in bs]), _med([b[1] for b in bs]),
               _med([b[2] for b in bs]), _med([b[3] for b in bs]),
               max(b[4] for b in bs))
        kept.append(box)

    # NMS final por si dos clusters quedaron solapados
    kept = self._nms(kept)
    for (x1, y1, x2, y2, conf) in kept:
        self.last_known[self._next_id] = [x1, y1, x2, y2, conf, 0]
        self._next_id += 1

# ------------------------------------------------------------------
def _update_last_known(self, detections, allow_new=True, allow_remove=True,
                       update_pos=True):
    """
    Asocia detecciones con presas conocidas y actualiza su estado.

    allow_new=False   -> no crea presas nuevas (conjunto fijo tras calibrar;
                         evita falsos positivos durante la reproducción).
    allow_remove=False -> no elimina presas por oclusión prolongada (las
                         presas son estáticas: si el escalador las tapa,
                         se conservan aunque no se re-detecten un rato).
    update_pos=False  -> NO mueve las presas: conserva la posición de
                         calibración y solo marca detectada/ocluida. Evita
                         que las presas "caminen" por el jitter de YOLO.
    """
    det_known = [d for d in detections if d[4] >= self.KNOWN_CONF_THRESHOLD]
    det_new = [d for d in detections if d[4] >= self.NEW_CONF_THRESHOLD]

    used_det = set()
    used_known = set()

    for pid, entry in self.last_known.items():
        best_d, best_j = float('inf'), -1
        for j, det in enumerate(det_known):
            if j in used_det:
                continue
            d = self._dist(entry, det)
            if d < best_d:
                best_d, best_j = d, j
        if best_j >= 0 and best_d < self.ASSOC_DIST:
            x1, y1, x2, y2, conf = det_known[best_j]
            if update_pos:
                self.last_known[pid] = [x1, y1, x2, y2, conf, 0]
            else:
                # Presas estáticas: conservar posición, solo marca la detectada
```

```
                ox1, oy1, ox2, oy2, _, _ = self.last_known[pid]
                self.last_known[pid] = [ox1, oy1, ox2, oy2, conf, 0]
            used_det.add(best_j)
            used_known.add(pid)

        to_remove = []
        for pid in self.last_known:
            if pid not in used_known:
                self.last_known[pid][5] += 1
                if allow_remove and self.last_known[pid][5] > self.MAX_MISSING_FRAMES:
                    to_remove.append(pid)
        for pid in to_remove:
            del self.last_known[pid]

        if not allow_new:
            return

        # Agrega solo presas nuevas de alta confianza no asociadas
        known_centers = [self._center(e) for e in self.last_known.values()]
        for det in det_new:
            cx, cy = self._center(det)
            too_close = any(
                ((cx - kx) ** 2 + (cy - ky) ** 2) ** 0.5 < self.ASSOC_DIST
                for kx, ky in known_centers
            )
            if not too_close:
                x1, y1, x2, y2, conf = det
                self.last_known[self._next_id] = [x1, y1, x2, y2, conf, 0]
                known_centers.append((cx, cy))
                self._next_id += 1

HOLD_DETECTOR = HoldDetector()


# ----------------------------------------------------
#   TRACKER DE TOQUES DE PRESAS
# ----------------------------------------------------
class HoldTouchTracker:
    """
    Detecta cuándo las muñecas tocan una presa y registra el tiempo.
    - Presa 1: se registra automáticamente con tiempo 0.00 cuando el
      cronómetro arranca (el deportista la sostenía al inicio).
    - Toque válido: muñeca a menos de TOUCH_RADIUS px del centro de la
      presa durante MIN_TOUCH_FRAMES frames consecutivos.
    - Solo registra si el cronómetro está corriendo.
    """
    TOUCH_RADIUS     = 40   # px
    MIN_TOUCH_FRAMES = 10   # frames consecutivos (~0.33 seg a 30fps)

    def __init__(self, fps):
        self.fps = fps
        self.touch_log = {}  # {presa_num: tiempo_seg}
        self._contact_frames = {}  # {presa_num: frames_consecutivos}
        self.active_hold = None
        self._hold1_registered = False  # True cuando Presa 1 ya se registró

    def reset(self):
```

```
        self.touch_log = {}
        self._contact_frames = {}
        self.active_hold = None
        self._hold1_registered = False

    def register_hold1(self):
        """Registra la Presa 1 con tiempo 0.00 al iniciar el cronómetro."""
        if not self._hold1_registered:
            self.touch_log[1] = 0.00
            self._hold1_registered = True

    def _sorted_holds(self, hold_last_known):
        """Ordena presas por cy descendente → Presa 1 = más abajo."""
        items = sorted(hold_last_known.items(), key=lambda x: x[1][1], reverse=True)
        return [(i + 1, entry) for i, (_, entry) in enumerate(items)]

    def _dist(self, wx, wy, entry):
        cx = (entry[0] + entry[2]) / 2
        cy = (entry[1] + entry[3]) / 2
        return ((wx - cx) ** 2 + (wy - cy) ** 2) ** 0.5

    def update(self, landmarks, hold_last_known, frame_h, frame_w,
               elapsed_sec, timer_running):
        self.active_hold = None

        if not timer_running or landmarks is None or not hold_last_known:
            return self.active_hold, self.touch_log

        # Registrar Presa 1 con tiempo 0.00 al arrancar
        if not self._hold1_registered:
            self.register_hold1()

        wx_izq = int(landmarks[15].x * frame_w)
        wy_izq = int(landmarks[15].y * frame_h)
        wx_der = int(landmarks[16].x * frame_w)
        wy_der = int(landmarks[16].y * frame_h)

        sorted_holds = self._sorted_holds(hold_last_known)

        for presa_num, entry in sorted_holds:
            if presa_num == 1:
                continue  # Presa 1 ya registrada al inicio
            if presa_num in self.touch_log:
                continue  # ya registrada

            d_izq = self._dist(wx_izq, wy_izq, entry)
            d_der = self._dist(wx_der, wy_der, entry)
            touching = d_izq < self.TOUCH_RADIUS or d_der < self.TOUCH_RADIUS

            if touching:
                self._contact_frames[presa_num] = (
                        self._contact_frames.get(presa_num, 0) + 1
                )
                self.active_hold = presa_num
                if self._contact_frames[presa_num] >= self.MIN_TOUCH_FRAMES:
                    self.touch_log[presa_num] = round(elapsed_sec, 2)
            else:
```

```
            self._contact_frames[presa_num] = 0

        return self.active_hold, self.touch_log


class ClimbingTimer:
    """
    Cronómetro de escalada.

    Inicio: en el primer frame se detecta el tobillo izquierdo y se crea
            un área fija (bbox) alrededor de ese punto. El timer arranca
            cuando el tobillo sale de esa área.

    Fin (dos condiciones):
      1. Las muñecas suben por encima de las últimas presas conocidas.
      2. El deportista desciende durante 2 segundos consecutivos sin
         haber superado las últimas presas.
    """
    ANKLE_BOX_SIZE = 50  # mitad del lado del bbox del tobillo (px en frame original)
    # Validación para fijar la caja SOLO cuando el tobillo es confiable
    ANKLE_MIN_VIS = 0.5       # visibilidad mínima de tobillo y rodilla
    ANKLE_KNEE_MARGIN = 20    # px: el tobillo debe estar al menos esto por debajo de la
rodilla
    ANKLE_CONFIRM_FRAMES = 3  # frames válidos consecutivos antes de fijar la caja
    DESCENT_SECONDS = 2.0
    DESCENT_EMA_ALPHA = 0.4        # suavizado de la cadera (anti-jitter)
    DESCENT_MIN_DROP_RATIO = 0.05  # caída mínima vs punto más alto (fracción del alto
del frame)
    DESCENT_PEAK_TOL_RATIO = 0.01  # subida mínima para fijar un nuevo máximo (anti-
jitter)
    FALL_DROP_PX = 200            # caída (px en frame original) bajo el punto más alto = fallo

    def __init__(self, fps):
        self.fps = fps

        # Área fija del tobillo — se define en el primer frame
        self.ankle_box = None  # (x1, y1, x2, y2)
        self._ankle_set = False  # True cuando el área ya fue creada
        self._ankle_samples = []  # muestras (x,y) del tobillo para centrar la caja
        self._ankle_released = False  # True cuando el tobillo salió del área

        # Estado del cronómetro
        self.running = False
        self.finished = False
        self.start_frame = 0
        self.end_frame = 0
        self.current_frame = 0

        # Detección de descenso
        self._descent_frames = 0
        self._max_descent_frames = int(self.DESCENT_SECONDS * fps)
        self._prev_hip_y = None
        self._reached_top = False
        self.fault = False  # True si el fin fue por descenso sostenido (fallo)
        self._hip_ema = None           # cadera suavizada (EMA)
        self._hip_peak_y = None        # punto más alto alcanzado (menor y)
        self._hip_peak_frame = 0       # frame en que se alcanzó el punto más alto
```

```
        self._descent_start_frame = None

    def reset(self, fps):
        self.__init__(fps)

    def elapsed_seconds(self):
        if not self.running and not self.finished:
            return 0.0
        end = self.end_frame if self.finished else self.current_frame
        return round((end - self.start_frame) / self.fps, 2)

    def _point_in_box(self, x, y, box):
        x1, y1, x2, y2 = box
        return x1 <= x <= x2 and y1 <= y <= y2

    # ------------------------------------------------------------------
    def update(self, landmarks, hold_last_known, frame_h, frame_w):
        """
        Llamar cada frame.
        landmarks: landmarks de MediaPipe (puede ser None).
        Retorna tiempo transcurrido en segundos.
        """
        self.current_frame += 1

        if self.finished or landmarks is None:
            return self.elapsed_seconds()

        # Coordenadas del tobillo izquierdo en píxeles originales
        ankle_x = int(landmarks[27].x * frame_w)
        ankle_y = int(landmarks[27].y * frame_h)

        # ---- Fijar área SOLO con una detección confiable del tobillo ----
        if not self._ankle_set:
            ankle_lm = landmarks[27]   # tobillo izquierdo
            knee_lm  = landmarks[25]   # rodilla izquierda
            ankle_vis = getattr(ankle_lm, 'visibility', 1.0)
            knee_vis  = getattr(knee_lm,  'visibility', 1.0)
            knee_y = int(knee_lm.y * frame_h)

            # El tobillo debe ser visible y estar por DEBAJO de la rodilla
            # (y crece hacia abajo). Esto descarta el caso en que la pose
            # coloca mal el "tobillo" a la altura de la rodilla.
            valido = (ankle_vis >= self.ANKLE_MIN_VIS and
                      knee_vis  >= self.ANKLE_MIN_VIS and
                      ankle_y > knee_y + self.ANKLE_KNEE_MARGIN)

            if not valido:
                self._ankle_samples = []   # exigir frames válidos consecutivos
                return self.elapsed_seconds()

            self._ankle_samples.append((ankle_x, ankle_y))
            if len(self._ankle_samples) < self.ANKLE_CONFIRM_FRAMES:
                return self.elapsed_seconds()

            # Centra la caja en la mediana de las muestras confiables
            xs = sorted(s[0] for s in self._ankle_samples)
            ys = sorted(s[1] for s in self._ankle_samples)
```

```
        cx = xs[len(xs) // 2]
        cy = ys[len(ys) // 2]
        b = self.ANKLE_BOX_SIZE
        self.ankle_box = (cx - b, cy - b, cx + b, cy + b)
        self._ankle_set = True
        return self.elapsed_seconds()

    # Espera a que el tobillo salga del área
    if not self._ankle_released:
        if not self._point_in_box(ankle_x, ankle_y, self.ankle_box):
            self._ankle_released = True
            self.running = True
            self.start_frame = self.current_frame
        return self.elapsed_seconds()

    if not self.running:
        return self.elapsed_seconds()

    # Coordenadas de muñecas y cadera
    wrist_izq_y = int(landmarks[15].y * frame_h)
    wrist_der_y = int(landmarks[16].y * frame_h)
    hip_y = int((landmarks[23].y + landmarks[24].y) / 2 * frame_h)

    # FIN condición 1: muñecas por encima de las últimas presas
    if hold_last_known:
        top_hold_y = min(entry[1] for entry in hold_last_known.values())
        wrist_top_y = min(wrist_izq_y, wrist_der_y)
        if wrist_top_y < top_hold_y:
            self._reached_top = True
            self.running = False
            self.finished = True
            self.end_frame = self.current_frame
            return self.elapsed_seconds()

    # ---- FIN condición 2: FALLO por caída de la trayectoria ----
    # La trayectoria de la cadera se trata como una función: se guarda el
    # punto más alto alcanzado (peak) y se mide cuánto vuelve a BAJAR por
    # debajo de él. Si la caída supera FALL_DROP_PX se considera un fallo.
    # (Equivale a: una línea horizontal corta la trayectoria en dos alturas
    #  separadas verticalmente más de FALL_DROP_PX.) No exige tiempo
    #  sostenido: una caída rápida y grande dispara de inmediato.
    if self._hip_ema is None:
        self._hip_ema = float(hip_y)
    else:
        a = self.DESCENT_EMA_ALPHA
        self._hip_ema = a * hip_y + (1.0 - a) * self._hip_ema
    hip_s = self._hip_ema

    if self._hip_peak_y is None:
        self._hip_peak_y = hip_s
        self._hip_peak_frame = self.current_frame

    peak_tol = self.DESCENT_PEAK_TOL_RATIO * frame_h

    if hip_s < self._hip_peak_y - peak_tol:
        # La cadera subió de forma significativa -> nuevo punto más alto
        self._hip_peak_y = hip_s
```

```
            self._hip_peak_frame = self.current_frame
        elif (hip_s - self._hip_peak_y) >= self.FALL_DROP_PX:
            # Cayó más de FALL_DROP_PX bajo su punto más alto -> FALLO
            self.running = False
            self.finished = True
            self.fault = True
            self.end_frame = self._hip_peak_frame  # tiempo hasta el punto más alto
            return self.elapsed_seconds()

        self._prev_hip_y = hip_y
        return self.elapsed_seconds()


class CropSelectorDialog(QDialog):
    def __init__(self, video_path):
        super().__init__()

        self.roi_x = 0
        self.roi_width = None
        self.dragging = False
        self.last_mouse_x = 0
        self.manual_mode = False

        self.setWindowTitle("SELECCIONAR RUTA")
        self.setFixedSize(900, 700)
        self.crop_side = None
        self.crop_pixels = 600  # 250 para el vid y 500 para el esc, distancia fija de recorte

        layout = QVBoxLayout(self)

        title = QLabel("SELECCIONAR RUTA")
        title.setFont(QFont("Arial", 18, QFont.Bold))
        title.setAlignment(Qt.AlignCenter)
        layout.addWidget(title)

        btn_layout = QHBoxLayout()
        self.btn_left = QPushButton("IZQUIERDA")
        self.btn_right = QPushButton("DERECHA")
        btn_layout.addWidget(self.btn_left)
        btn_layout.addWidget(self.btn_right)
        layout.addLayout(btn_layout)

        self.preview = QLabel()
        self.preview.setAlignment(Qt.AlignCenter)
        self.preview.setStyleSheet("background:black;")
        layout.addWidget(self.preview, stretch=1)

        btn_accept = QPushButton("ACEPTAR")
        btn_accept.setFixedWidth(150)
        btn_accept.clicked.connect(self.accept)
        layout.addWidget(btn_accept, alignment=Qt.AlignRight)

        self.cap = cv2.VideoCapture(video_path)
        ret, frame = self.cap.read()
        self.original_frame = frame

        self.btn_left.clicked.connect(lambda: self.update_crop("left"))
```

```
        self.btn_right.clicked.connect(lambda: self.update_crop("right"))

        self.update_crop("left")

    def update_crop(self, side):
        self.crop_side = side
        self.manual_mode = False

        h, w, _ = self.original_frame.shape
        self.roi_width = w - self.crop_pixels

        if side == "left":
            self.roi_x = 0
        else:
            self.roi_x = self.crop_pixels

        self.update_preview()

    def update_preview(self):
        frame = self.original_frame.copy()
        h, w, _ = frame.shape

        # Dibujar rectángulo del ROI
        cv2.rectangle(
            frame,
            (self.roi_x, 0),
            (self.roi_x + self.roi_width, h),
            (0, 255, 0),
            3
        )

        frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        frame = resize_with_aspect_ratio(frame, 800, 500)

        h, w, ch = frame.shape
        img = QImage(frame.data, w, h, ch * w, QImage.Format_RGB888)
        self.preview.setPixmap(QPixmap.fromImage(img))

    def mousePressEvent(self, event):
        if event.buttons() == Qt.LeftButton:
            self.dragging = True
            self.last_mouse_x = event.x()
            self.manual_mode = True

    def mouseMoveEvent(self, event):
        if not self.dragging:
            return

        dx = event.x() - self.last_mouse_x
        self.last_mouse_x = event.x()

        h, w, _ = self.original_frame.shape
        self.roi_x = max(0, min(self.roi_x + dx, w - self.roi_width))

        self.update_preview()

    def mouseReleaseEvent(self, event):
```

```
        self.dragging = False

    def get_crop(self):
        return ("manual", self.roi_x, self.roi_width)

    def closeEvent(self, event):
        if hasattr(self, 'cap'):
            self.cap.release()
        event.accept()


# ----------------------------------------------------------
#   REPRODUCTOR DE VIDEO CON OPENCV
# ----------------------------------------------------------
class VideoWindow(QWidget):
    def __init__(self, video_path, title, crop=None, athlete_roi=None):
        super().__init__()

        self.crop = crop
        self.athlete_roi = athlete_roi
        self.setWindowTitle(title)
        self.setWindowFlag(Qt.Window)
        self.showMaximized()

        # --------- OPENCV ----------
        self.cap = cv2.VideoCapture(video_path)
        if not self.cap.isOpened():
            print("ERROR: No se pudo abrir el video")
            return

        self.fps = self.cap.get(cv2.CAP_PROP_FPS) or 30.0
        self.total_frames = int(self.cap.get(cv2.CAP_PROP_FRAME_COUNT))
        self.current_frame_idx = 0

        # Limpiar frames procesados de este carril para el nuevo video
        if title == "Video de Referencia":
            processed_reference_frames.clear()
        elif title == "Video del Deportista":
            processed_athlete_frames.clear()

        # Resetear pipeline híbrido YOLO+MediaPipe para este video
        POSE_ESTIMATOR._pose.close()
        POSE_ESTIMATOR._pose = POSE_ESTIMATOR._mp.Pose(
            static_image_mode=False,
            model_complexity=1,
            enable_segmentation=False,
            min_detection_confidence=0.4,
            min_tracking_confidence=0.4
        )
        POSE_ESTIMATOR._last_landmarks = None
        POSE_ESTIMATOR._smoothed       = {}
        POSE_ESTIMATOR._frame_count    = 0
        POSE_ESTIMATOR._invalid_streak = 0
        POSE_ESTIMATOR._crop_box = None
        if hasattr(POSE_ESTIMATOR, '_pid'):
            del POSE_ESTIMATOR._pid
        if athlete_roi:
```

```
            POSE_ESTIMATOR._ref_x_min = athlete_roi[0]
            POSE_ESTIMATOR._ref_x_max = athlete_roi[2]
            POSE_ESTIMATOR._ref_y_min = athlete_roi[1]
            POSE_ESTIMATOR._ref_y_max = 1.0
        else:
            POSE_ESTIMATOR._ref_x_min = 0.0
            POSE_ESTIMATOR._ref_x_max = 1.0
            POSE_ESTIMATOR._ref_y_min = 0.0
            POSE_ESTIMATOR._ref_y_max = 1.0

        # Resetear y calibrar presas para este video
        HOLD_DETECTOR.last_known = {}
        HOLD_DETECTOR._next_id = 0
        HOLD_DETECTOR.calibrate_from_video(video_path, crop)

        self._holds_snapshot = {pid: list(entry) for pid, entry in
HOLD_DETECTOR.last_known.items()}

        # Cronómetro y tracker de toques
        self.climb_timer = ClimbingTimer(self.fps)
        self.touch_tracker = HoldTouchTracker(self.fps)
        self._fault_shown = False  # el aviso de fallo se muestra una sola vez

        # Trayectoria acumulada de hip_mid en coords del panel
        self.hip_trajectory = []
        self._traj_offset_x = 0
        if hasattr(self, '_traj_offset_fixed'):
            del self._traj_offset_fixed
        if hasattr(self, 'hip_trajectory_transformed'):
            self.hip_trajectory_transformed = []
        if hasattr(self, 'skeleton_frames'):
            self.skeleton_frames = []
        # Últimas detecciones YOLO (pasadas desde update_frame al panel trayectoria)
        self.last_hold_boxes = []

        # --------- LAYOUT PRINCIPAL ----------
        root = QVBoxLayout(self)
        root.setContentsMargins(8, 8, 8, 8)
        root.setSpacing(6)

        # Barra de título
        title_bar = QFrame()
        title_bar.setStyleSheet("background-color: #1a5fa8; border-radius: 6px;")
        title_bar_layout = QHBoxLayout(title_bar)
        title_bar_layout.setContentsMargins(12, 6, 12, 6)
        lbl_title = QLabel(title)
        lbl_title.setFont(QFont("Arial", 14, QFont.Bold))
        lbl_title.setStyleSheet("color: white; background: transparent;")
        title_bar_layout.addWidget(lbl_title)
        title_bar_layout.addStretch()
        lbl_meta = QLabel(f"{video_path.split('/')[-1]}  ·  {self.fps:.1f} fps")
        lbl_meta.setStyleSheet("color: #b8d4f0; font-size: 11px; background: transparent;")
        title_bar_layout.addWidget(lbl_meta)
        root.addWidget(title_bar)

        # Fila principal: video | trayectoria | panel derecho
        main_row = QHBoxLayout()
```

```
        main_row.setSpacing(8)
        root.addLayout(main_row)

        # --- Video original (ancho fijo, sin resize interno) ---
        self.video_label = QLabel()
        self.video_label.setAlignment(Qt.AlignTop | Qt.AlignLeft)
        self.video_label.setStyleSheet("background-color: #111; border-radius: 6px;")
        self.video_label.setMinimumSize(160, 620)
        self.video_label.setFixedWidth(160)
        # self.video_label.setFixedHeight(650)
        self.video_label.setSizePolicy(QSizePolicy.Fixed, QSizePolicy.Fixed)
        main_row.addWidget(self.video_label)

        # --- Panel de trayectoria (mismo ancho fijo) ---
        self.traj_label = QLabel()
        self.traj_label.setAlignment(Qt.AlignTop | Qt.AlignLeft)
        self.traj_label.setStyleSheet(
            "background-color: #f5f3ee; border: 1px solid #ccc; border-radius: 6px;"
        )
        self.traj_label.setMinimumSize(160, 640)
        self.traj_label.setFixedWidth(160)
        self.traj_label.setFixedHeight(640)
        self.traj_label.setSizePolicy(QSizePolicy.Fixed, QSizePolicy.Fixed)
        main_row.addWidget(self.traj_label)
        self.traj_pixmap = None

        # --- Panel derecho: métricas (col izq) + tabla/activo/status (col der) ---
        right_panel = QHBoxLayout()
        right_panel.setSpacing(8)
        main_row.addLayout(right_panel)
        main_row.addStretch()

        # Columna izquierda: métricas en vertical
        metrics_col = QVBoxLayout()
        metrics_col.setSpacing(6)
        metrics_col.setContentsMargins(0, 0, 0, 0)

        def make_metric(label_text, value_text, value_color="#1a5fa8", sub=""):
            card = QFrame()
            card.setStyleSheet(
                "QFrame { background: white; border: 1px solid #ddd; border-radius: 6px; }"
            )
            card.setFixedWidth(130)
            cl = QVBoxLayout(card)
            cl.setContentsMargins(10, 8, 10, 8)
            cl.setSpacing(2)
            lbl = QLabel(label_text.upper())
            lbl.setStyleSheet("color: #888; font-size: 10px; background: transparent; border: none;")
            val = QLabel(value_text)
            val.setStyleSheet(
                f"color: {value_color}; font-size: 22px; font-weight: bold; "
                f"background: transparent; border: none;"
            )
            sub_lbl = QLabel(sub)
            sub_lbl.setStyleSheet("color: #aaa; font-size: 10px; background: transparent; border:
none;")
            cl.addWidget(lbl)
```

```
    cl.addWidget(val)
    cl.addWidget(sub_lbl)
    return card, val

card_t, self.lbl_tiempo = make_metric("Tiempo", "0.00", "#1a5fa8", "segundos")
card_p, self.lbl_presas = make_metric("Presas tocadas", "0", "#1d9e75", "de 19")
card_s, self.lbl_saltadas = make_metric("Saltadas", "0", "#ba7517", "no tocadas")
metrics_col.addWidget(card_t)
metrics_col.addWidget(card_p)
metrics_col.addWidget(card_s)
metrics_col.addStretch()
right_panel.addLayout(metrics_col)

# Columna derecha: presa activa + tabla + barra de estado
info_col = QVBoxLayout()
info_col.setSpacing(6)
info_col.setContentsMargins(0, 0, 0, 0)

# Indicador de presa activa
active_frame = QFrame()
active_frame.setStyleSheet(
    "QFrame { background: white; border: 1px solid #ddd; border-radius: 6px; }"
)
active_layout = QHBoxLayout(active_frame)
active_layout.setContentsMargins(12, 8, 12, 8)
active_layout.setSpacing(10)
icon_circle = QLabel("✋")
icon_circle.setStyleSheet(
    "background: #e6f1fb; border-radius: 16px; padding: 4px; "
    "font-size: 16px; border: none;"
)
icon_circle.setFixedSize(36, 36)
icon_circle.setAlignment(Qt.AlignCenter)
active_layout.addWidget(icon_circle)
active_text_col = QVBoxLayout()
active_text_col.setSpacing(1)
lbl_active_header = QLabel("Presa activa")
lbl_active_header.setStyleSheet(
    "color: #888; font-size: 10px; background: transparent; border: none;"
)
self.lbl_active_name = QLabel("Ninguna")
self.lbl_active_name.setStyleSheet(
    "color: #111; font-size: 13px; font-weight: bold; background: transparent; border: none;"
)
self.lbl_active_sub = QLabel("—")
self.lbl_active_sub.setStyleSheet(
    "color: #1d9e75; font-size: 10px; background: transparent; border: none;"
)
active_text_col.addWidget(lbl_active_header)
active_text_col.addWidget(self.lbl_active_name)
active_text_col.addWidget(self.lbl_active_sub)
active_layout.addLayout(active_text_col)
active_layout.addStretch()
info_col.addWidget(active_frame)

# Tabla de 19 presas
table_frame = QFrame()
```

```
        table_frame.setStyleSheet(
            "QFrame { background: white; border: 1px solid #ddd; border-radius: 6px; }"
        )
        table_frame_layout = QVBoxLayout(table_frame)
        table_frame_layout.setContentsMargins(0, 0, 0, 0)
        table_frame_layout.setSpacing(0)

        table_header_frame = QFrame()
        table_header_frame.setStyleSheet(
            "QFrame { background: transparent; border: none; border-bottom: 1px solid #eee; }"
        )
        th_layout = QHBoxLayout(table_header_frame)
        th_layout.setContentsMargins(12, 7, 12, 7)
        lbl_th = QLabel("Registro de toques")
        lbl_th.setStyleSheet(
            "font-size: 12px; font-weight: bold; color: #222; background: transparent;"
        )
        self.lbl_count_badge = QLabel("0 registros")
        self.lbl_count_badge.setStyleSheet(
            "font-size: 10px; background: #e6f1fb; color: #185fa5; "
            "border-radius: 10px; padding: 2px 8px;"
        )
        th_layout.addWidget(lbl_th)
        th_layout.addStretch()
        th_layout.addWidget(self.lbl_count_badge)
        table_frame_layout.addWidget(table_header_frame)

        self.touch_table = QTableWidget(19, 2)
        self.touch_table.setHorizontalHeaderLabels(["Presa", "Tiempo (seg)"])
        self.touch_table.horizontalHeader().setSectionResizeMode(0, QHeaderView.Stretch)
        self.touch_table.horizontalHeader().setSectionResizeMode(1,
QHeaderView.ResizeToContents)
        self.touch_table.verticalHeader().setVisible(False)
        self.touch_table.setEditTriggers(QTableWidget.NoEditTriggers)
        self.touch_table.setSelectionMode(QTableWidget.NoSelection)
        self.touch_table.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Fixed)
        self.touch_table.verticalHeader().setDefaultSectionSize(22)
        self.touch_table.setFixedHeight(60 + 19 * 22)
        self.touch_table.setVerticalScrollBarPolicy(Qt.ScrollBarAlwaysOff)
        self.touch_table.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOff)
        self.touch_table.setStyleSheet(
            "QTableWidget { border: none; font-size: 11px; background: white; "
            "gridline-color: #f0f0f0; }"
            "QHeaderView::section { background: #f8f8f8; color: #888; font-size: 10px; "
            "font-weight: bold; border: none; border-bottom: 1px solid #eee; padding: 4px 10px; }"
            "QTableWidget::item { padding: 4px 10px; border: none; }"
        )

        from PyQt5.QtGui import QColor
        from PyQt5.QtCore import Qt as _Qt
        for i in range(19):
            item_n = QTableWidgetItem(f"Presa {i + 1}")
            item_t = QTableWidgetItem("—")
            item_t.setForeground(QColor("#aaa"))
            item_n.setFlags(item_n.flags() & ~_Qt.ItemIsEditable)
            item_t.setFlags(item_t.flags() & ~_Qt.ItemIsEditable)
            self.touch_table.setItem(i, 0, item_n)
```

```
            self.touch_table.setItem(i, 1, item_t)

        table_frame_layout.addWidget(self.touch_table)
        info_col.addWidget(table_frame)

        # Barra de estado
        status_frame = QFrame()
        status_frame.setStyleSheet(
            "QFrame { background: white; border: 1px solid #ddd; border-radius: 6px; }"
        )
        status_layout = QHBoxLayout(status_frame)
        status_layout.setContentsMargins(12, 6, 12, 6)
        status_layout.setSpacing(8)
        dot = QLabel("●")
        dot.setStyleSheet("color: #1d9e75; font-size: 10px; background: transparent; border: none;")
        status_layout.addWidget(dot)
        self.lbl_status = QLabel("Frame 0 de 0")
        self.lbl_status.setStyleSheet(
            "color: #666; font-size: 11px; background: transparent; border: none;"
        )
        status_layout.addWidget(self.lbl_status)
        status_layout.addStretch()
        self.progress_bar = QProgressBar()
        self.progress_bar.setRange(0, 100)
        self.progress_bar.setValue(0)
        self.progress_bar.setTextVisible(False)
        self.progress_bar.setFixedWidth(100)
        self.progress_bar.setFixedHeight(4)
        self.progress_bar.setStyleSheet(
            "QProgressBar { background: #eee; border-radius: 2px; border: none; }"
            "QProgressBar::chunk { background: #1a5fa8; border-radius: 2px; }"
        )
        status_layout.addWidget(self.progress_bar)
        self.lbl_pct = QLabel("0%")
        self.lbl_pct.setStyleSheet(
            "color: #888; font-size: 11px; background: transparent; border: none;"
        )
        status_layout.addWidget(self.lbl_pct)
        info_col.addWidget(status_frame)
        info_col.addStretch()

        right_panel.addLayout(info_col)

        self._yolo_frame_counter = 0  # para correr YOLO cada N frames
        self._last_hold_boxes = []  # última detección YOLO válida

        # --------- TIMER ----------
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.update_frame)
        QTimer.singleShot(150, self.start_video)

    def start_video(self):
        self.timer.start(int(1000 / self.fps))

    def update_frame(self):
        ret, frame = self.cap.read()
        if not ret:
```

```
    self.timer.stop()
    self.lbl_status.setText("Video finalizado")
    return

self.current_frame_idx += 1

# Aplicar recorte
if self.crop:
    if self.crop[0] == "manual":
        _, x, w_crop = self.crop
        frame = frame[:, x:x + w_crop]
    else:
        side, px = self.crop
        fh, fw, _ = frame.shape
        if side == "left":
            frame = frame[:, :fw - px]
        else:
            frame = frame[:, px:]

frame_bgr = frame.copy()

# --- Reducir resolución para procesamiento ---
proc_w = 480
fh_orig, fw_orig = frame_bgr.shape[:2]
proc_scale = proc_w / fw_orig
proc_h = int(fh_orig * proc_scale)
frame_small = cv2.resize(frame_bgr, (proc_w, proc_h), interpolation=cv2.INTER_AREA)

# --- YOLO cada 3 frames ---
self._yolo_frame_counter += 1
if self._yolo_frame_counter % 3 == 0:
    detections_small = HOLD_DETECTOR._raw_detections(
        frame_small, threshold=HOLD_DETECTOR.ASSOC_CONF)
    # Escalar coords a resolución original
    detections_scaled = [
        (int(x1 / proc_scale), int(y1 / proc_scale),
         int(x2 / proc_scale), int(y2 / proc_scale), conf)
        for (x1, y1, x2, y2, conf) in detections_small
    ]
    HOLD_DETECTOR._update_last_known(
        detections_scaled, allow_new=False, allow_remove=False,
        update_pos=False)

# Dibuja desde last_known (verde=detectada, naranja=estimada/ocluida)
hold_boxes = []
for pid, entry in HOLD_DETECTOR.last_known.items():
    x1, y1, x2, y2, conf, missing = entry
    color = (0, 165, 255) if missing > 0 else (0, 255, 100)
    cv2.rectangle(frame_bgr, (x1, y1), (x2, y2), color, 1)
    # cv2.putText(frame_bgr, f"{conf:.2f}", (x1, max(y1 - 4, 10)),
    #             cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 1)
    hold_boxes.append((x1, y1, x2, y2))
self._last_hold_boxes = hold_boxes

# Pose sobre frame pequeño
frame_small_rgb = cv2.cvtColor(frame_small, cv2.COLOR_BGR2RGB)
_, landmarks = POSE_ESTIMATOR.draw_pose(frame_small_rgb)
```

```
# Dibuja pose sobre frame_bgr original escalando landmarks
if landmarks:
    points = {}
    for name, idx in KEYPOINTS.items():
        lm = landmarks[idx]
        x = int(lm.x * fw_orig)
        y = int(lm.y * fh_orig)
        points[name] = (x, y)
        color = (0, 0, 0) if idx in BLACK_POINTS else (0, 255, 0)
        cv2.circle(frame_bgr, (x, y), 8, color, -1)

    if "hombro_izq" in points and "hombro_der" in points:
        points["mid_hombros"] = (
            (points["hombro_izq"][0] + points["hombro_der"][0]) // 2,
            (points["hombro_izq"][1] + points["hombro_der"][1]) // 2
        )
    if "cadera_izq" in points and "cadera_der" in points:
        mx = (points["cadera_izq"][0] + points["cadera_der"][0]) // 2
        my = (points["cadera_izq"][1] + points["cadera_der"][1]) // 2
        points["mid_caderas"] = (mx, my)
        cv2.circle(frame_bgr, (mx, my), 8, (0, 255, 255), -1)

    for a, b in CONNECTIONS:
        if a in points and b in points:
            cv2.line(frame_bgr, points[a], points[b], (0, 0, 255), 2)

# Cronómetro
elapsed = self.climb_timer.update(
    landmarks, HOLD_DETECTOR.last_known, fh_orig, fw_orig
)

# Fallo por descenso sostenido: detener conteo y avisar una sola vez
if getattr(self.climb_timer, 'fault', False) and not self._fault_shown:
    self._fault_shown = True
    from PyQt5.QtWidgets import QMessageBox
    self.timer.stop()  # pausar la reproducción mientras se muestra el aviso
    QMessageBox.warning(
        self, "Fallo detectado",
        "Se ha detectado un descenso de manera "
        "continua por un tiempo prolongado.\n\n"
        "El conteo y el registro de tiempo se han detenido."
    )
    self.timer.start()  # reanudar (el tiempo queda congelado por finished=True)

# Tracker de toques
active_hold, touch_log = self.touch_tracker.update(
    landmarks, HOLD_DETECTOR.last_known, fh_orig, fw_orig,
    elapsed, self.climb_timer.running
)

# Dibujar área fija del tobillo
if self.climb_timer.ankle_box is not None:
    ax1, ay1, ax2, ay2 = self.climb_timer.ankle_box
    box_color = (100, 100, 100) if self.climb_timer._ankle_released else (0, 255, 0)
    cv2.rectangle(frame_bgr, (ax1, ay1), (ax2, ay2), box_color, 1)
    cv2.putText(frame_bgr, "inicio", (ax1, max(ay1 - 4, 10)),
```

```
            cv2.FONT_HERSHEY_SIMPLEX, 0.4, box_color, 1)

# Mostrar en QLabel
disp_w = self.video_label.width()
disp_h = self.video_label.height()
disp = cv2.resize(frame_bgr, (disp_w, disp_h), interpolation=cv2.INTER_AREA)
disp_rgb = cv2.cvtColor(disp, cv2.COLOR_BGR2RGB)
h_d, w_d = disp_rgb.shape[:2]
img = QImage(disp_rgb.data, w_d, h_d, w_d * 3, QImage.Format_RGB888)
self.video_label.setPixmap(QPixmap.fromImage(img))

# Guardar frames procesados
frame_rgb = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2RGB)
if self.windowTitle() == "Video de Referencia":
    if len(processed_reference_frames) < self.total_frames:
        processed_reference_frames.append(frame_rgb.copy())
if self.windowTitle() == "Video del Deportista":
    if len(processed_athlete_frames) < self.total_frames:
        processed_athlete_frames.append(frame_rgb.copy())

# Panel de trayectoria
self._update_traj_panel(frame_rgb, hold_boxes, landmarks)

# Actualizar widgets
from PyQt5.QtGui import QColor

# Tiempo
if self.climb_timer.finished:
    self.lbl_tiempo.setText(f"{elapsed:.2f}")
    self.lbl_tiempo.setStyleSheet(
        "color: #1d9e75; font-size: 22px; font-weight: bold; "
        "background: transparent; border: none;"
    )
elif self.climb_timer.running:
    self.lbl_tiempo.setText(f"{elapsed:.2f}")
else:
    self.lbl_tiempo.setText("--")

# Presas tocadas y saltadas
n_tocadas = len(touch_log)
self.lbl_presas.setText(str(n_tocadas))
self.lbl_saltadas.setText(str(19 - n_tocadas))

# Presa activa
if active_hold is not None:
    self.lbl_active_name.setText(f"Presa {active_hold}")
    self.lbl_active_sub.setText("en contacto")
else:
    self.lbl_active_name.setText("Ninguna")
    self.lbl_active_sub.setText("—")

# Tabla — actualizar solo filas nuevas
self.lbl_count_badge.setText(f"{n_tocadas} registros")
for presa_num, t_seg in touch_log.items():
    row = presa_num - 1
    item_t = self.touch_table.item(row, 1)
    if item_t and item_t.text() == "—":
```

```
            item_t.setText(f"{t_seg:.2f}")
            item_t.setForeground(QColor("#111"))
            item_n = self.touch_table.item(row, 0)
            if item_n:
                item_n.setForeground(QColor("#1d9e75"))

    # Resaltar presa activa
    if active_hold is not None:
        row = active_hold - 1
        item_n = self.touch_table.item(row, 0)
        if item_n:
            item_n.setBackground(QColor("#e6f1fb"))
        self.touch_table.scrollToItem(self.touch_table.item(row, 0))

    # Barra de estado
    pct = int(self.current_frame_idx / self.total_frames * 100) if self.total_frames else 0
    self.lbl_status.setText(
        f"Frame {self.current_frame_idx} de {self.total_frames}  ·  {self.fps:.0f} fps"
    )
    self.progress_bar.setValue(pct)
    self.lbl_pct.setText(f"{pct}%")

def _calc_traj_angle(self):
    return 0.0  # ya no se usa ángulo

def _update_traj_panel(self, frame_rgb, hold_boxes, landmarks):
    """
    Dibuja sobre el panel de trayectoria.
    Aplica un offset horizontal fijo para que la presa más alta
    quede centrada en el ancho del panel.
    """
    from PyQt5.QtGui import QPainter, QPen, QBrush, QColor
    import math

    pw = self.traj_label.width()
    ph = self.traj_label.height()
    if pw < 10 or ph < 10:
        return

    fh, fw = frame_rgb.shape[:2]
    scale = pw / fw

    def to_panel(x, y):
        return int(x * scale), int(y * scale)

    # Calcula offset horizontal una sola vez usando la presa más alta
    if not hasattr(self, '_traj_offset_x'):
        self._traj_offset_x = 0
    if hold_boxes and not hasattr(self, '_traj_offset_fixed'):
        # Presa más alta = menor y1
        top_box = min(hold_boxes, key=lambda b: b[1])
        cx_hold = int((top_box[0] + top_box[2]) / 2 * scale)
        self._traj_offset_x = (pw // 2) - cx_hold - 80  # ajustar este valor para mover izq/der
        self._traj_offset_fixed = True

    ox = self._traj_offset_x  # offset horizontal fijo
```

```
# Rotación fija horaria — modificar TRAJ_ROTATION_DEG para ajustar
TRAJ_ROTATION_DEG = 10
_rad = math.radians(TRAJ_ROTATION_DEG)
# Pivote = parte inferior central del panel
_cx_pivot = pw // 2
_cy_pivot = ph

def rotate_pt(x, y):
    dx, dy = x - _cx_pivot, y - _cy_pivot
    rx = dx * math.cos(_rad) - dy * math.sin(_rad)
    ry = dx * math.sin(_rad) + dy * math.cos(_rad)
    return int(_cx_pivot + rx), int(_cy_pivot + ry)

def rpt(x, y):
    px, py = to_panel(x, y)
    return rotate_pt(px + ox, py)

# Inicializar pixmap acumulado
if (self.traj_pixmap is None
        or self.traj_pixmap.width() != pw
        or self.traj_pixmap.height() != ph):
    self.traj_pixmap = QPixmap(pw, ph)
    self.traj_pixmap.fill(QColor("#f5f3ee"))

# Calcular hip_mid y añadir a trayectoria (raw y transformada)
if landmarks:
    hx = int((landmarks[23].x + landmarks[24].x) / 2 * fw)
    hy = int((landmarks[23].y + landmarks[24].y) / 2 * fh)
    new_pt = to_panel(hx, hy)
    self.hip_trajectory.append(new_pt)
    # Guarda también la coordenada transformada para DualVideoWindow
    if not hasattr(self, 'hip_trajectory_transformed'):
        self.hip_trajectory_transformed = []
    self.hip_trajectory_transformed.append(rotate_pt(new_pt[0] + ox, new_pt[1]))

# Redibujar trayectoria con offset
self.traj_pixmap.fill(QColor("#f5f3ee"))
if len(self.hip_trajectory) >= 2:
    p_redraw = QPainter(self.traj_pixmap)
    p_redraw.setRenderHint(QPainter.Antialiasing)
    pen_r = QPen(QColor("#e24b4a"), max(1, int(2 * scale)))
    p_redraw.setPen(pen_r)
    for i in range(1, len(self.hip_trajectory)):
        x1t, y1t = self.hip_trajectory[i - 1]
        x2t, y2t = self.hip_trajectory[i]
        rx1, ry1 = rotate_pt(x1t + ox, y1t)
        rx2, ry2 = rotate_pt(x2t + ox, y2t)
        p_redraw.drawLine(rx1, ry1, rx2, ry2)
    p_redraw.end()

# Componer display
display_pixmap = self.traj_pixmap.copy()
painter = QPainter(display_pixmap)
painter.setRenderHint(QPainter.Antialiasing)

# Silueta esqueleto con offset
if landmarks:
```

```
    def lpt(idx):
        if hasattr(landmarks[idx], 'x'):
            return rpt(int(landmarks[idx].x * fw), int(landmarks[idx].y * fh))
        else:
            return rpt(int(landmarks[idx][0] * fw), int(landmarks[idx][1] * fh))

    mhx = int((landmarks[11].x + landmarks[12].x) / 2 * fw) if hasattr(landmarks[11], 'x') \
          else int((landmarks[11][0] + landmarks[12][0]) / 2 * fw)
    mhy = int((landmarks[11].y + landmarks[12].y) / 2 * fh) if hasattr(landmarks[11], 'x') \
          else int((landmarks[11][1] + landmarks[12][1]) / 2 * fh)
    mid_hombros = rpt(mhx, mhy)

    pen_sil = QPen(QColor("#1a3a6b"), max(1, int(2 * scale)))
    painter.setPen(pen_sil)

    skel_lines  = []
    skel_points = []

    for a, b in [(11, 12), (11, 13), (13, 15), (12, 14), (14, 16),
                 (11, 23), (12, 24), (23, 24), (23, 25), (25, 27), (24, 26), (26, 28)]:
        ax, ay = lpt(a)
        bx, by = lpt(b)
        painter.drawLine(ax, ay, bx, by)
        skel_lines.append(((ax, ay), (bx, by)))
    head = lpt(0)
    painter.drawLine(head[0], head[1], mid_hombros[0], mid_hombros[1])
    painter.drawLine(mid_hombros[0], mid_hombros[1], lpt(11)[0], lpt(11)[1])
    painter.drawLine(mid_hombros[0], mid_hombros[1], lpt(12)[0], lpt(12)[1])

    r_pt = max(2, int(3 * scale))
    painter.setPen(QPen(QColor("#1a3a6b"), 1))
    painter.setBrush(QBrush(QColor("#1a3a6b")))
    for idx in [0, 11, 12, 13, 14, 15, 16, 23, 24, 25, 26, 27, 28]:
        px2, py2 = lpt(idx)
        painter.drawEllipse(px2 - r_pt, py2 - r_pt, r_pt * 2, r_pt * 2)
        skel_points.append((px2, py2))
    painter.drawEllipse(mid_hombros[0] - r_pt, mid_hombros[1] - r_pt, r_pt * 2, r_pt * 2)

    # Guarda esqueleto para pasarlo al DualVideoWindow
    self._last_skeleton = {'lines': skel_lines, 'points': skel_points}
    # Guarda esqueleto de cada frame para reproducción en DualVideoWindow
    if not hasattr(self, 'skeleton_frames'):
        self.skeleton_frames = []
    self.skeleton_frames.append({'lines': list(skel_lines), 'points': list(skel_points)})

# Punto hip_mid actual
if self.hip_trajectory:
    last_x, last_y = self.hip_trajectory[-1]
    cx2, cy2 = rotate_pt(last_x + ox, last_y)
    r_hip = max(3, int(5 * scale))
    painter.setPen(QPen(QColor("#cc8800"), 1))
    painter.setBrush(QBrush(QColor("#eaa00a")))
    painter.drawEllipse(cx2 - r_hip, cy2 - r_hip, r_hip * 2, r_hip * 2)

# Bounding boxes de presas con offset
pen_box = QPen(QColor("#2a7a3b"), max(1, int(1.5 * scale)))
painter.setPen(pen_box)
```

```
        painter.setBrush(Qt.NoBrush)
        r_center = max(2, int(4 * scale))
        for (x1, y1, x2, y2) in hold_boxes:
            px1, py1 = rpt(x1, y1)
            px2, py2 = rpt(x2, y2)
            painter.drawRect(px1, py1, px2 - px1, py2 - py1)
            cx_h = (px1 + px2) // 2
            cy_h = (py1 + py2) // 2
            painter.setBrush(QBrush(QColor("#2a7a3b")))
            painter.drawEllipse(cx_h - r_center, cy_h - r_center, r_center * 2, r_center * 2)
            painter.setBrush(Qt.NoBrush)

        painter.end()
        self.traj_label.setPixmap(display_pixmap)

    def update_hold_touch(self, presa_num, tiempo_seg, es_activa=False):
        """
        Actualiza la tabla cuando se registra un toque.
        presa_num: 1-19
        tiempo_seg: float con el tiempo
        es_activa: True si es la presa actualmente en contacto
        """
        from PyQt5.QtGui import QColor
        from PyQt5.QtWidgets import QTableWidgetItem

        row = presa_num - 1  # Presa 1 → fila 0
        item_t = QTableWidgetItem(f"{tiempo_seg:.2f}")
        item_t.setForeground(QColor("#111"))

        if es_activa:
            item_nombre = self.touch_table.item(row, 0)
            if item_nombre:
                item_nombre.setBackground(QColor("#e6f1fb"))
            item_t.setBackground(QColor("#e6f1fb"))
            self.lbl_active_name.setText(f"Presa {presa_num}")
            self.lbl_active_sub.setText("en contacto")

        self.touch_table.setItem(row, 1, item_t)
        self.touch_table.scrollToItem(self.touch_table.item(row, 0))

        tocadas = sum(
            1 for i in range(19)
            if self.touch_table.item(i, 1) and self.touch_table.item(i, 1).text() not in ("—", "")
        )
        self.lbl_presas.setText(str(tocadas))
        self.lbl_count_badge.setText(f"{tocadas} registros")

    def mark_skipped(self, presa_num):
        """Marca una presa como saltada en la tabla."""
        from PyQt5.QtGui import QColor
        row = presa_num - 1
        item_t = self.touch_table.item(row, 1)
        if item_t and item_t.text() == "—":
            item_t.setText("saltada")
            item_t.setForeground(QColor("#ba7517"))
            item_nombre = self.touch_table.item(row, 0)
            if item_nombre:
```

```
            item_nombre.setForeground(QColor("#ba7517"))
        self.lbl_saltadas.setText(
            str(int(self.lbl_saltadas.text() or "0") + 1)
        )

    def closeEvent(self, event):
        self.timer.stop()
        if hasattr(self, "cap"):
            self.cap.release()
        event.accept()


# ------------------------------------------------------
#  REPRODUCTOR DUAL (ANÁLISIS GENERAL)
# ------------------------------------------------------
class DualVideoWindow(QWidget):
    # Colores presas según estado
    COLOR_SOLO_DEP = "#e24b4a"
    COLOR_SOLO_REF = "#378add"
    COLOR_AMBOS = "#1d9e75"
    COLOR_NINGUNO = "#b4b2a9"

    def __init__(self, ref_path, dep_path, crop_ref=None, crop_dep=None,
                 touch_log_ref=None, touch_log_dep=None,
                 hip_traj_dep=None, hip_traj_ref=None,
                 tiempo_final_ref=None, tiempo_final_dep=None,
                 holds_dep=None,
                 holds_ref=None,
                 start_frame_ref=0, start_frame_dep=0, fps_ref=30.0, fps_dep=30.0,
                 skeleton_ref=None, skeleton_dep=None,
                 skeleton_frames_ref=None, skeleton_frames_dep=None,
                 traj_offset_dep=0, traj_offset_ref=0, traj_rot_deg=10):
        super().__init__()

        self.ref_path = ref_path
        self.dep_path = dep_path
        self.crop_ref = crop_ref
        self.crop_dep = crop_dep

        self.touch_log_ref = list(touch_log_ref) if touch_log_ref else []
        self.touch_log_dep = list(touch_log_dep) if touch_log_dep else []
        self.hip_traj_dep  = hip_traj_dep or []
        self.hip_traj_ref  = hip_traj_ref or []
        self.holds_dep     = holds_dep or {}
        self.holds_ref     = holds_ref or {}
        self.skeleton_ref  = skeleton_ref
        self.skeleton_dep  = skeleton_dep
        self.skeleton_frames_ref = skeleton_frames_ref or []
        self.skeleton_frames_dep = skeleton_frames_dep or []
        self.traj_offset_dep = traj_offset_dep
        self.traj_offset_ref = traj_offset_ref
        self.traj_rot_deg    = traj_rot_deg

        self.tiempo_final_ref = tiempo_final_ref
        self.tiempo_final_dep = tiempo_final_dep

        self.presas_ref = {int(p) for p, _ in self.touch_log_ref}
```

```
self.presas_dep = {int(p) for p, _ in self.touch_log_dep}

# Sincronización: 1 segundo antes del inicio de cada escalada
PRE_SECONDS = 1.0
self._ref_offset = max(0, start_frame_ref - int(PRE_SECONDS * fps_ref))
self._dep_offset = max(0, start_frame_dep - int(PRE_SECONDS * fps_dep))

self.left_index    = 0
self.right_index   = 0
self.left_finished = False
self.right_finished= False
self.traj_pixmap_dual = None
self.vel_pixmap = None

self.setWindowTitle("Análisis General")
self.setWindowFlag(Qt.Window)

# Adapta la pantalla descontando barra de tareas
screen = QApplication.primaryScreen().availableGeometry()
self.setMaximumSize(screen.width(), screen.height())
self.showMaximized()

# --------- LAYOUT PRINCIPAL ----------
root = QVBoxLayout(self)
root.setContentsMargins(8, 8, 8, 8)
root.setSpacing(6)

# Barra de título
title_bar = QFrame()
title_bar.setStyleSheet("background-color: #1a5fa8; border-radius: 6px;")
tb_layout = QHBoxLayout(title_bar)
tb_layout.setContentsMargins(12, 6, 12, 6)
tb_layout.setSpacing(8)
lbl_t = QLabel("Análisis General")
lbl_t.setFont(QFont("Arial", 13, QFont.Bold))
lbl_t.setStyleSheet("color: white; background: transparent;")
tb_layout.addWidget(lbl_t)
tb_layout.addStretch()
lbl_meta = QLabel(f"{ref_path.split('/')[-1]}  ·  {dep_path.split('/')[-1]}")
lbl_meta.setStyleSheet("color: #b8d4f0; font-size: 11px; background: transparent;")
tb_layout.addWidget(lbl_meta)

btn_repetir = QPushButton(" 🔁 Repetir")
btn_repetir.setStyleSheet(
    "QPushButton { background: #1d9e75; color: white; border: none; "
    "border-radius: 4px; padding: 4px 10px; font-size: 11px; font-weight: bold; }"
    "QPushButton:hover { background: #178a65; }"
)
btn_repetir.clicked.connect(self.repetir_videos)

btn_pdf = QPushButton(" 📄 Exportar PDF")
btn_pdf.setStyleSheet(
    "QPushButton { background: white; color: #1a5fa8; border: none; "
    "border-radius: 4px; padding: 4px 10px; font-size: 11px; font-weight: bold; }"
    "QPushButton:hover { background: #e6f1fb; }"
)
btn_pdf.clicked.connect(self.export_pdf)
```

```
btn_vid = QPushButton("🎥 Guardar video")
btn_vid.setStyleSheet(
    "QPushButton { background: #e24b4a; color: white; border: none; "
    "border-radius: 4px; padding: 4px 10px; font-size: 11px; font-weight: bold; }"
    "QPushButton:hover { background: #c73c3b; }"
)
btn_vid.clicked.connect(self.save_video)

tb_layout.addWidget(btn_repetir)
tb_layout.addWidget(btn_pdf)
tb_layout.addWidget(btn_vid)
root.addWidget(title_bar)

# Fila principal
# Alto de video — ajustar para cambiar altura de videos, trayectoria y tabla
_vid_h = 620
main_row = QHBoxLayout()
main_row.setSpacing(8)
root.addLayout(main_row)

# --- Video Referencia ---
left_block = QVBoxLayout()
left_block.setAlignment(Qt.AlignTop)
lbl_left = QLabel("Referencia")
lbl_left.setFont(QFont("Arial", 10, QFont.Bold))
lbl_left.setAlignment(Qt.AlignCenter)
lbl_left.setStyleSheet("color: #185fa5;")
left_block.addWidget(lbl_left)
self.left_label = QLabel()
self.left_label.setFixedWidth(200)
self.left_label.setFixedHeight(_vid_h)
self.left_label.setScaledContents(True)
left_block.addWidget(self.left_label)
main_row.addLayout(left_block)

# --- Video Deportista ---
right_block = QVBoxLayout()
right_block.setAlignment(Qt.AlignTop)
lbl_right = QLabel("Deportista")
lbl_right.setFont(QFont("Arial", 10, QFont.Bold))
lbl_right.setAlignment(Qt.AlignCenter)
lbl_right.setStyleSheet("color: #a32d2d;")
right_block.addWidget(lbl_right)
self.right_label = QLabel()
self.right_label.setFixedWidth(200)
self.right_label.setFixedHeight(_vid_h)
self.right_label.setScaledContents(True)
right_block.addWidget(self.right_label)
main_row.addLayout(right_block)

# Panel trayectoria
traj_col = QVBoxLayout()
traj_col.setSpacing(2)
traj_col.setAlignment(Qt.AlignTop)

lbl_traj = QLabel("Trayectoria y presas")
```

```
        lbl_traj.setFont(QFont("Arial", 10, QFont.Bold))
        lbl_traj.setAlignment(Qt.AlignCenter)
        lbl_traj.setStyleSheet("color: #444;")
        traj_col.addWidget(lbl_traj)
        traj_col.addSpacing(5)

        # Un solo recuadro con línea divisoria pintada internamente
        _traj_total_w = 180  # mismo ancho que antes
        self.traj_label_dual = QLabel()
        self.traj_label_dual.setFixedWidth(_traj_total_w)
        self.traj_label_dual.setFixedHeight(_vid_h)
        self.traj_label_dual.setStyleSheet(
            "background-color: #f5f3ee; border: 1px solid #ccc; border-radius: 6px;"
        )
        traj_col.addWidget(self.traj_label_dual)
        main_row.addLayout(traj_col)

        # Alias para compatibilidad
        self.traj_label_ref = self.traj_label_dual
        self.traj_label_dep = self.traj_label_dual

        # Panel derecho: dos columnas
        self.right_panel_widget = QWidget()
        right_panel = QHBoxLayout(self.right_panel_widget)
        right_panel.setSpacing(6)
        main_row.addWidget(self.right_panel_widget)
        main_row.addStretch()

        # Columna izquierda: tabla anclada arriba, se expande hacia abajo
        col_left = QVBoxLayout()
        col_left.setSpacing(6)
        col_left.setAlignment(Qt.AlignTop)
        right_panel.addLayout(col_left)

        # Título de la tabla con mismo formato que los títulos de video
        lbl_table_title = QLabel("Registro comparativo de toques")
        lbl_table_title.setFont(QFont("Arial", 10, QFont.Bold))
        lbl_table_title.setAlignment(Qt.AlignCenter)
        lbl_table_title.setStyleSheet("color: #222;")
        col_left.addWidget(lbl_table_title)
        col_left.addSpacing(5)

        # Tabla comparativa — alto propio ajustable
        _table_h = 615  # ajustar este valor para cambiar el alto de la tabla
        table_frame = QFrame()
        table_frame.setStyleSheet(
            "QFrame { background: white; border: 1px solid #ddd; }"
        )
        tf_layout = QVBoxLayout(table_frame)
        tf_layout.setContentsMargins(0, 0, 0, 0)
        tf_layout.setSpacing(0)

        self.comp_table = QTableWidget(19, 4)
        self.comp_table.setHorizontalHeaderLabels(["Presa", "Ref. (seg)", "Dep. (seg)",
"Diferencia"])
        self.comp_table.horizontalHeader().setSectionResizeMode(QHeaderView.Stretch)
        self.comp_table.verticalHeader().setVisible(False)
```

```
        self.comp_table.setEditTriggers(QTableWidget.NoEditTriggers)
        self.comp_table.setSelectionMode(QTableWidget.NoSelection)
        self.comp_table.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Fixed)
        self.comp_table.verticalHeader().setDefaultSectionSize(max(22, (_table_h - 28) // 19))
        self.comp_table.setFixedHeight(_table_h)
        self.comp_table.setVerticalScrollBarPolicy(Qt.ScrollBarAlwaysOff)
        self.comp_table.setHorizontalScrollBarPolicy(Qt.ScrollBarAlwaysOff)
        self.comp_table.setStyleSheet(
            "QTableWidget { border: none; font-size: 10px; gridline-color: #f0f0f0; }"
            "QHeaderView::section { background: #f8f8f8; color: #888; font-size: 9px; font-weight:
bold; "
            "border: none; border-bottom: 1px solid #eee; padding: 3px 6px; }"
            "QTableWidget::item { padding: 3px 6px; border: none; }"
        )

        from PyQt5.QtGui import QColor
        # Colores de grupo
        COLOR_G1 = QColor("#c8dcf5")  # Salida P1-P7: azul más oscuro
        COLOR_G2 = QColor("#f5e0b8")  # Zona central P8-P13: amarillo más oscuro
        COLOR_G3 = QColor("#c0ecce")  # Llegada P14-P19: verde más oscuro

        for i in range(19):
            presa = i + 1
            if presa <= 7:
                bg = COLOR_G1
                bg_hex = "#c8dcf5"
            elif presa <= 13:
                bg = COLOR_G2
                bg_hex = "#f5e0b8"
            else:
                bg = COLOR_G3
                bg_hex = "#c0ecce"

            # Todas las presas: "Presa" arriba, número abajo en negrita
            item_n = QTableWidgetItem()
            item_n.setFlags(item_n.flags() & ~Qt.ItemIsEditable)
            item_n.setBackground(bg)
            lbl_presa = QLabel(f"<div style='text-align:center;line-
height:1.1'>Presa<br><b>{presa}</b></div>")
            lbl_presa.setAlignment(Qt.AlignCenter)
            # Usar el color de fondo exacto en el stylesheet del QLabel
            lbl_presa.setStyleSheet(f"background-color: {bg_hex}; font-size: 10px; border: none;")
            self.comp_table.setItem(i, 0, item_n)
            self.comp_table.setCellWidget(i, 0, lbl_presa)
            for col, txt in enumerate(["—", "—", "—"], start=1):
                it = QTableWidgetItem(txt)
                it.setForeground(QColor("#aaa"))
                it.setFlags(it.flags() & ~Qt.ItemIsEditable)
                it.setTextAlignment(Qt.AlignCenter)
                it.setBackground(bg)
                self.comp_table.setItem(i, col, it)

        tf_layout.addWidget(self.comp_table)
        col_left.addWidget(table_frame)

        # Recuadro TIEMPO POR TRAMOS
        tramos_frame = QFrame()
```

```
        tramos_frame.setStyleSheet(
            "QFrame { background: white; border: 1px solid #ddd; border-radius: 6px; }"
        )
        tramos_frame.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Maximum)
        tr_layout = QVBoxLayout(tramos_frame)
        tr_layout.setContentsMargins(0, 0, 0, 0)
        tr_layout.setSpacing(0)

        tr_header = QFrame()
        tr_header.setStyleSheet(
            "QFrame { background: transparent; border: none; border-bottom: 1px solid #eee; }"
        )
        tr_h_layout = QHBoxLayout(tr_header)
        tr_h_layout.setContentsMargins(10, 6, 10, 6)
        lbl_tr = QLabel("Tiempo por tramos")
        lbl_tr.setStyleSheet("font-size: 11px; font-weight: bold; color: #222; background:
transparent;")
        tr_h_layout.addWidget(lbl_tr)
        tr_layout.addWidget(tr_header)

        tr_grid = QHBoxLayout()
        tr_grid.setContentsMargins(8, 8, 8, 8)
        tr_grid.setSpacing(8)

        tramos_def = [
            ("Salida",       1,  7,  "#185fa5", "#c8dcf5"),
            ("Zona central", 8,  13, "#ba7517", "#f5e0b8"),
            ("Llegada",      14, 19, "#0f6e56", "#c0ecce"),
        ]

        self._tramo_widgets = []
        for nombre, p_ini, p_fin, color, bg_color in tramos_def:
            card = QFrame()
            card.setStyleSheet(
                f"QFrame {{ background: {bg_color}; border: 1px solid #ddd; border-radius: 6px; }}"
            )
            card.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Maximum)
            cl = QVBoxLayout(card)
            cl.setContentsMargins(8, 6, 8, 6)
            cl.setSpacing(3)

            lbl_name = QLabel(nombre.upper())
            lbl_name.setStyleSheet(
                f"font-size: 10px; font-weight: bold; color: {color}; background: transparent; border:
none;"
            )
            cl.addWidget(lbl_name)

            # Lista compacta: Ref, Dep, Diferencia con prefijo
            entries = [
                ("Ref.",  "#185fa5"),
                ("Dep.",  "#a32d2d"),
                ("Dif.",  color),
            ]
            lbl_vals = []
            for prefix, c in entries:
                row_item = QHBoxLayout()
```

```
        row_item.setSpacing(4)
        lbl_pre = QLabel(prefix)
        lbl_pre.setStyleSheet(
            "font-size: 9px; color: #666; background: transparent; border: none;"
        )
        lbl_pre.setFixedWidth(22)
        lbl_v = QLabel("--")
        lbl_v.setStyleSheet(
            f"font-size: 11px; font-weight: bold; color: {c}; "
            f"background: transparent; border: none; font-variant-numeric: tabular-nums;"
        )
        row_item.addWidget(lbl_pre)
        row_item.addWidget(lbl_v)
        row_item.addStretch()
        cl.addLayout(row_item)
        lbl_vals.append(lbl_v)

    tr_grid.addWidget(card)
    self._tramo_widgets.append((p_ini, p_fin, lbl_vals[0], lbl_vals[1], lbl_vals[2]))

tr_layout.addLayout(tr_grid)
col_left.addStretch()

# Columna derecha: métricas (lado a lado) + gráfica de velocidad + tiempo por tramos
col_right = QVBoxLayout()
col_right.setSpacing(6)
right_panel.addLayout(col_right)

# Tiempo total | Presas tocadas — lado a lado
metrics_row = QHBoxLayout()
metrics_row.setSpacing(6)
metrics_row.addWidget(self._make_metric_card(
    "Tiempo total",
    "lbl_tiempo_ref", "ref (seg)", "#185fa5",
    "lbl_tiempo_dep", "dep (seg)", "#a32d2d",
    "lbl_tiempo_diff", "diferencia"
))
metrics_row.addWidget(self._make_metric_card(
    "Presas tocadas",
    "lbl_presas_ref", "referencia", "#185fa5",
    "lbl_presas_dep", "deportista", "#a32d2d",
    "lbl_presas_diff", "diferencia"
))
col_right.addLayout(metrics_row)

vel_frame = QFrame()
vel_frame.setStyleSheet(
    "QFrame { background: white; border: 1px solid #ddd; border-radius: 6px; }"
)
vel_layout = QVBoxLayout(vel_frame)
vel_layout.setContentsMargins(10, 10, 10, 10)
vel_layout.setSpacing(6)

#Encabezado gráfica
vel_header = QHBoxLayout()
vel_title_col = QVBoxLayout()
lbl_vel_title = QLabel("Velocidad en presas")
```

```
        lbl_vel_title.setStyleSheet(
            "font-size: 11px; font-weight: bold; color: #222; background: transparent; border: none;")
        lbl_vel_sub = QLabel("Velocidad vs tiempo · m/s (aprox.) · Considerando el Centro de
Gravedad")
        lbl_vel_sub.setStyleSheet("font-size: 9px; color: #888; background: transparent; border:
none;")
        vel_title_col.addWidget(lbl_vel_title)
        vel_title_col.addWidget(lbl_vel_sub)
        vel_header.addLayout(vel_title_col)
        vel_header.addStretch()

        # Leyenda gráfica
        for color, estilo, texto in [
            ("#185fa5", "●", "Ref."),
            ("#a32d2d", "■", "Dep."),
        ]:
            dot = QLabel(estilo)
            dot.setStyleSheet(f"color: {color}; font-size: 10px; background: transparent; border:
none;")
            lbl = QLabel(texto)
            lbl.setStyleSheet("font-size: 10px; color: #666; background: transparent; border: none;")
            vel_header.addWidget(dot)
            vel_header.addWidget(lbl)
        vel_layout.addLayout(vel_header)

        # Canvas de la gráfica (dibujada con QPainter)
        self.vel_canvas = QLabel()
        self.vel_canvas.setMinimumHeight(340)
        self.vel_canvas.setMinimumWidth(500)
        self.vel_canvas.setStyleSheet("background: transparent; border: none;")
        self.vel_canvas.setSizePolicy(QSizePolicy.Expanding, QSizePolicy.Fixed)
        vel_layout.addWidget(self.vel_canvas)

        # Tarjetas de promedios
        prom_row = QHBoxLayout()
        prom_row.setSpacing(5)
        for attr, label, color in [
            ("lbl_vel_ref", "Vel. prom. ref.", "#185fa5"),
            ("lbl_vel_dep", "Vel. prom. dep.", "#a32d2d"),
            ("lbl_vel_diff", "Diferencia prom.", "#a32d2d"),
        ]:
            card = QFrame()
            card.setStyleSheet(
                "QFrame { background: #f5f5f5; border-radius: 4px; border: none; }"
            )
            cl = QVBoxLayout(card)
            cl.setContentsMargins(8, 6, 8, 6)
            cl.setSpacing(2)
            lbl = QLabel(label.upper())
            lbl.setStyleSheet("font-size: 8px; color: #888; background: transparent;")
            val = QLabel("--")
            val.setStyleSheet(f"font-size: 14px; font-weight: bold; color: {color}; background:
transparent;")
            cl.addWidget(lbl)
            cl.addWidget(val)
            setattr(self, attr, val)
            prom_row.addWidget(card)
```

```
        vel_layout.addLayout(prom_row)
        col_right.addWidget(vel_frame)
        col_right.addWidget(tramos_frame)
        col_right.addStretch()

        # TIMER
        self.timer = QTimer(self)
        self.timer.timeout.connect(self.update_dual_frame)
        self.timer.start(80)

        # Poblar datos y dibujar
        self._populate_data()
        QTimer.singleShot(400, self._draw_traj_panel)
        QTimer.singleShot(500, self._draw_vel_chart)

    # ---------------------------------------------------------------
    def _make_metric_card(self, title,
                          attr1, sub1, color1,
                          attr2, sub2, color2,
                          attr3, sub3):
        card = QFrame()
        card.setStyleSheet(
            "QFrame { background: white; border: 1px solid #ddd; border-radius: 6px; }"
        )
        cl = QVBoxLayout(card)
        cl.setContentsMargins(10, 8, 10, 8)
        cl.setSpacing(4)
        lbl_t = QLabel(title.upper())
        lbl_t.setStyleSheet("font-size: 10px; color: #000; font-weight: bold; background: transparent;
border: none;")
        cl.addWidget(lbl_t)

        row = QHBoxLayout()
        row.setSpacing(8)
        for attr, sub, color in [(attr1, sub1, color1), (attr2, sub2, color2), (attr3, sub3, "#444")]:
            col = QVBoxLayout()
            col.setSpacing(1)
            val = QLabel("--")
            val.setStyleSheet(
                f"font-size: 19px; font-weight: bold; color: {color}; "
                f"background: transparent; border: none; font-variant-numeric: tabular-nums;"
            )
            val.setAlignment(Qt.AlignCenter)
            sub_lbl = QLabel(sub)
            sub_lbl.setStyleSheet("font-size: 9px; color: #aaa; background: transparent; border: none;")
            sub_lbl.setAlignment(Qt.AlignCenter)
            col.addWidget(val)
            col.addWidget(sub_lbl)
            setattr(self, attr, val)
            row.addLayout(col)

            if attr != attr3:
                sep = QFrame()
                sep.setFrameShape(QFrame.VLine)
                sep.setStyleSheet("color: #eee; background: #eee; border: none;")
                sep.setFixedWidth(1)
                row.addWidget(sep)
```

```
    cl.addLayout(row)
    return card

# ------------------------------------------------------------------
def _populate_data(self):
    from PyQt5.QtGui import QColor

    map_ref = {int(p): t for p, t in self.touch_log_ref}
    map_dep = {int(p): t for p, t in self.touch_log_dep}

    # Tiempo total — usar el tiempo final del cronómetro (fuente de verdad)
    t_ref = self.tiempo_final_ref
    t_dep = self.tiempo_final_dep
    self.lbl_tiempo_ref.setText(f"{t_ref:.2f}" if t_ref is not None else "--")
    self.lbl_tiempo_dep.setText(f"{t_dep:.2f}" if t_dep is not None else "--")
    if t_ref is not None and t_dep is not None:
        diff_t = t_dep - t_ref
        sign = "+" if diff_t > 0 else ""
        self.lbl_tiempo_diff.setText(f"{sign}{diff_t:.2f}")
        self.lbl_tiempo_diff.setStyleSheet(
            f"font-size: 19px; font-weight: bold; background: transparent; border: none; "
            f"color: {'#e24b4a' if diff_t > 0 else '#1d9e75'};"
        )

    # Presas tocadas
    n_ref = len(self.presas_ref)
    n_dep = len(self.presas_dep)
    self.lbl_presas_ref.setText(str(n_ref) if self.touch_log_ref else "--")
    self.lbl_presas_dep.setText(str(n_dep) if self.touch_log_dep else "--")
    if self.touch_log_ref or self.touch_log_dep:
        diff_p = n_dep - n_ref
        sign = "+" if diff_p > 0 else ""
        self.lbl_presas_diff.setText(f"{sign}{diff_p}")
        self.lbl_presas_diff.setStyleSheet(
            f"font-size: 19px; font-weight: bold; background: transparent; border: none; "
            f"color: {'#e24b4a' if diff_p > 0 else '#1d9e75'};"
        )

    # Tabla comparativa
    for i in range(19):
        presa = i + 1
        t_r = map_ref.get(presa)
        t_d = map_dep.get(presa)

        item_r = QTableWidgetItem(f"{t_r:.2f}" if t_r is not None else "—")
        item_r.setForeground(QColor("#185fa5") if t_r is not None else QColor("#aaa"))
        item_r.setTextAlignment(Qt.AlignCenter)

        item_d = QTableWidgetItem(f"{t_d:.2f}" if t_d is not None else "—")
        item_d.setForeground(QColor("#a32d2d") if t_d is not None else QColor("#aaa"))
        item_d.setTextAlignment(Qt.AlignCenter)

        if t_r is not None and t_d is not None:
            diff = t_d - t_r   # dep - ref: negativo = dep más rápido
            sign = "+" if diff > 0 else ""
            item_diff = QTableWidgetItem(f"{sign}{diff:.2f}")
```

```
            item_diff.setForeground(QColor("#1d9e75") if diff < 0 else QColor("#a32d2d"))
        else:
            item_diff = QTableWidgetItem("—")
            item_diff.setForeground(QColor("#aaa"))
        item_diff.setTextAlignment(Qt.AlignCenter)

        for it in [item_r, item_d, item_diff]:
            it.setFlags(it.flags() & ~Qt.ItemIsEditable)

        # Preservar color de fondo según grupo
        presa = i + 1
        if presa <= 7:
            bg = QColor("#e8f0fb")
        elif presa <= 13:
            bg = QColor("#fdf3e3")
        else:
            bg = QColor("#eafbf0")
        for it in [item_r, item_d, item_diff]:
            it.setBackground(bg)

        self.comp_table.setItem(i, 1, item_r)
        self.comp_table.setItem(i, 2, item_d)
        self.comp_table.setItem(i, 3, item_diff)

    # Tiempo por tramos
    # Tramo 1 (Salida P1-P7):   t_fin_tramo - t_ini_tramo
    # Tramo 2 (Central P8-P13): t_fin_tramo - t_fin_tramo_anterior (P7)
    # Tramo 3 (Llegada P14-P19): t_fin_tramo - t_fin_tramo_anterior (P13)
    if hasattr(self, '_tramo_widgets'):
        tramos_calc = [
            (1,  7,  0),
            (8,  13, 7),
            (14, 19, 13),
        ]
        for (p_ini, p_fin, p_prev), (_, _, lbl_ref, lbl_dep, lbl_diff) in \
                zip(tramos_calc, self._tramo_widgets):

            # Última presa tocada dentro del tramo
            presas_r_in = [map_ref[p] for p in range(p_ini, p_fin + 1) if p in map_ref]
            presas_d_in = [map_dep[p] for p in range(p_ini, p_fin + 1) if p in map_dep]

            # Tiempo de referencia del tramo anterior
            if p_prev == 0:
                t_prev_r = 0.0
                t_prev_d = 0.0
            else:
                # Última presa tocada del tramo anterior
                prev_presas_r = [map_ref[p] for p in range(1, p_prev + 1) if p in map_ref]
                prev_presas_d = [map_dep[p] for p in range(1, p_prev + 1) if p in map_dep]
                t_prev_r = prev_presas_r[-1] if prev_presas_r else None
                t_prev_d = prev_presas_d[-1] if prev_presas_d else None

            # Calcular duración del tramo
            if presas_r_in and t_prev_r is not None:
                t_tramo_r = presas_r_in[-1] - t_prev_r
                lbl_ref.setText(f"{t_tramo_r:.2f} seg")
            else:
```

```
            lbl_ref.setText("--")
            t_tramo_r = None

        if presas_d_in and t_prev_d is not None:
            t_tramo_d = presas_d_in[-1] - t_prev_d
            lbl_dep.setText(f"{t_tramo_d:.2f} seg")
        else:
            lbl_dep.setText("--")
            t_tramo_d = None

        if t_tramo_r is not None and t_tramo_d is not None:
            diff_tr = t_tramo_d - t_tramo_r
            sign  = "+" if diff_tr > 0 else ""
            color = "#a32d2d" if diff_tr > 0 else "#1d9e75"
            lbl_diff.setText(f"{sign}{diff_tr:.2f} seg")
            lbl_diff.setStyleSheet(
                f"font-size: 10px; font-weight: bold; color: {color}; background: transparent;"
            )
        else:
            lbl_diff.setText("--")

# ------------------------------------------------------------------
def _compute_traj_alignment(self):
    """
    Alinea la trayectoria/silueta de ref con la del dep usando
    las posiciones de P1 (presa más baja) y la última presa como anclas.

    Las presas del dep ya están en el espacio del panel dual (con offset y rotación).
    Las presas del ref están en espacio del frame original (480px) y necesitan
    convertirse al mismo espacio para poder calcular el desplazamiento.

    Devuelve (dx, dy, scale_y, ref_p1_y).
    """
    import math as _m
    src_w = 160.0
    SCALE = 0.90

    if len(self.hip_traj_dep) < 2 or len(self.hip_traj_ref) < 2:
        return 0, 0, 1.0, (self.hip_traj_ref[0][1] if self.hip_traj_ref else 0)

    # --- Posición de P1 y última presa del DEP en espacio del panel dual ---
    pw = self.traj_label_dual.width()
    ph = self.traj_label_dual.height()

    ox   = self.traj_offset_dep
    _rad = _m.radians(self.traj_rot_deg)
    _cx_pivot = int(pw / 2)
    _cy_pivot = ph

    def rotate_pt(x, y):
        dx2, dy2 = x - _cx_pivot, y - _cy_pivot
        rx = dx2 * _m.cos(_rad) - dy2 * _m.sin(_rad)
        ry = dx2 * _m.sin(_rad) + dy2 * _m.cos(_rad)
        return _cx_pivot + rx, _cy_pivot + ry

    def holds_to_panel(holds):
        """Convierte presas del frame 480px al espacio del panel dual."""
```

```
    frame_scale = src_w / 480.0
    pts = []
    for entry in holds.values():
        cx_raw = (entry[0] + entry[2]) / 2 * frame_scale
        cy_raw = (entry[1] + entry[3]) / 2 * frame_scale
        cx_raw += ox
        px_d = cx_raw * pw / src_w * SCALE
        py_d = cy_raw * pw / src_w * SCALE
        rx, ry = rotate_pt(px_d, py_d)
        pts.append((rx, ry))
    return pts

dep_hold_pts = sorted(holds_to_panel(self.holds_dep), key=lambda p: p[1], reverse=True)
ref_hold_pts_raw = sorted(self.holds_ref.values(), key=lambda e: e[1], reverse=True) \
                if self.holds_ref else []

# P1 dep = presa más baja (mayor Y en panel)
# Última presa dep = presa más alta (menor Y)
if dep_hold_pts:
    dep_p1   = dep_hold_pts[0]   # presa 1: más abajo
    dep_last = dep_hold_pts[-1]  # última presa: más arriba
else:
    # Fallback: primer y último punto de trayectoria
    dep_p1   = self.hip_traj_dep[0]
    dep_last = self.hip_traj_dep[-1]

# P1 ref y última presa ref en espacio del frame ref (480px → panel ref)
# Estos están en coordenadas del VideoWindow de ref (sin rotación del dual)
# Los usamos solo para calcular la escala relativa entre ambas paredes
if ref_hold_pts_raw:
    def ref_hold_to_panel(entry):
        frame_scale_ref = src_w / 480.0
        cx = (entry[0] + entry[2]) / 2 * frame_scale_ref
        cy = (entry[1] + entry[3]) / 2 * frame_scale_ref
        # Aplicar offset del ref
        traj_offset_ref = getattr(self, 'traj_offset_ref', 0)
        cx += traj_offset_ref
        return cx * pw / src_w * SCALE, cy * pw / src_w * SCALE

    ref_p1   = ref_hold_to_panel(ref_hold_pts_raw[0])
    ref_last = ref_hold_to_panel(ref_hold_pts_raw[-1])
else:
    ref_p1   = self.hip_traj_ref[0]
    ref_last = self.hip_traj_ref[-1]

# Calcular escala: distancia vertical entre P1 y última presa
h_dep = abs(dep_p1[1] - dep_last[1])
h_ref = abs(ref_p1[1] - ref_last[1])
scale_y = (h_dep / h_ref) if h_ref > 0.001 else 1.0
scale_y = max(0.3, min(3.0, scale_y))

# Traslación: alinear P1 de ref con P1 de dep
# Primero se transforma ref_p1 al mismo espacio que hip_traj_ref
# hip_traj_ref ya tiene el offset+rotación del VideoWindow de ref aplicado
# ref_p1 también está en ese espacio (sin rotación del dual)
# Usamos el primer punto de hip_traj_ref como P1 de ref
ref_traj_p1 = self.hip_traj_ref[0]
```

```
    dx = dep_p1[0] - ref_traj_p1[0]
    dy = dep_p1[1] - ref_traj_p1[1]

    return dx, dy, scale_y, ref_traj_p1[1]

def _apply_traj_align(self, pts, dx, dy, scale_y, ref_start_y):
    result = []
    for x, y in pts:
        nx = x + dx
        ny = ref_start_y + (y - ref_start_y) * scale_y + dy
        result.append((nx, ny))
    return result

def _apply_skeleton_align(self, skel, dx, dy, scale_y, ref_start_y):
    """
    Alinea el esqueleto: traslada dx,dy y escala Y desde ref_start_y.
    La escala se aplica igual que a la trayectoria para que coincidan.
    """
    if not skel:
        return skel
    def ax(x): return x + dx
    def ay(y): return ref_start_y + (y - ref_start_y) * scale_y + dy
    new_lines  = [((ax(x1), ay(y1)), (ax(x2), ay(y2)))
                  for (x1,y1),(x2,y2) in skel.get('lines', [])]
    new_points = [(ax(x), ay(y)) for x, y in skel.get('points', [])]
    return {'lines': new_lines, 'points': new_points}

def _draw_traj_panel(self):
    """
    Dibuja el panel de trayectoria en vivo usando el progreso actual.
    El render se delega en _render_traj_pixmap para poder reutilizarlo al
    exportar el video (silueta y trayectoria animadas por frame).
    """
    pw = self.traj_label_dual.width()
    ph = self.traj_label_dual.height()
    if pw < 10 or ph < 10:
        QTimer.singleShot(200, self._draw_traj_panel)
        return

    n_ref = self.left_index  + self._ref_offset
    n_dep = self.right_index + self._dep_offset
    pixmap = self._render_traj_pixmap(
        pw, ph, n_ref, n_dep, self.skeleton_ref, self.skeleton_dep
    )
    self.traj_pixmap_dual = pixmap
    self.traj_label_dual.setPixmap(pixmap)

def _render_traj_pixmap(self, pw, ph, n_ref, n_dep, skeleton_ref, skeleton_dep):
    """
    Renderiza el panel de trayectoria y presas a un QPixmap para el
    progreso dado: n_ref/n_dep puntos de trayectoria y las siluetas
    skeleton_ref/skeleton_dep, sin depender del estado de reproduccion.
    """
    from PyQt5.QtGui import QPainter, QPen, QBrush, QColor, QFont as QF

    TITLE_H = 16
```

```
half    = pw // 2
body_h  = ph - TITLE_H
SRC_W   = 160.0
SRC_H   = 620.0
TRAJ_ROT = self.traj_rot_deg

n_ref = max(0, min(n_ref, len(self.hip_traj_ref)))
n_dep = max(0, min(n_dep, len(self.hip_traj_dep)))

def make_tp(panel_x0, panel_w):
    sx = panel_w / SRC_W
    sy = body_h  / SRC_H
    def tp(x, y):
        return int(x * sx + panel_x0), int(y * sy + TITLE_H)
    return tp, sx, sy

def holds_to_src(raw_cx, raw_cy, offset_x):
    import math
    px = raw_cx * (SRC_W / 480.0) + offset_x
    py = raw_cy * (SRC_W / 480.0)
    _r  = math.radians(TRAJ_ROT)
    _cx = SRC_W / 2;  _cy = SRC_H
    ddx, ddy = px - _cx, py - _cy
    rpx = _cx + ddx*math.cos(_r) - ddy*math.sin(_r)
    rpy = _cy + ddx*math.sin(_r) + ddy*math.cos(_r)
    return rpx, rpy

pixmap = QPixmap(pw, ph)
pixmap.fill(QColor("#f5f3ee"))
painter = QPainter(pixmap)
painter.setRenderHint(QPainter.Antialiasing)
lw = max(1, int(pw / 120))

# IZQUIERDA: REFERENCIA
tp_ref, sx_r, sy_r = make_tp(0, half)
traj_offset_ref = getattr(self, 'traj_offset_ref', 0)
ref_holds = getattr(self, 'holds_ref', {}) or {}

painter.save()
painter.setClipRect(0, TITLE_H, half, body_h)

if len(self.hip_traj_ref[:n_ref]) >= 2:
    painter.setPen(QPen(QColor(100, 140, 200, 200), lw))
    traj = self.hip_traj_ref[:n_ref]
    for i in range(1, len(traj)):
        painter.drawLine(*tp_ref(*traj[i-1]), *tp_ref(*traj[i]))

if skeleton_ref:
    c_r = QColor(24, 95, 165, 200); cf_r = QColor(24, 95, 165, 80)
    painter.setPen(QPen(c_r, max(1, lw))); painter.setBrush(QBrush(cf_r))
    for (x1,y1),(x2,y2) in skeleton_ref.get('lines', []):
        painter.drawLine(*tp_ref(x1,y1), *tp_ref(x2,y2))
    painter.setBrush(QBrush(c_r))
    for x,y in skeleton_ref.get('points', []):
        r = max(2, int(3*sx_r)); px_,py_ = tp_ref(x,y)
        painter.drawEllipse(px_-r, py_-r, r*2, r*2)
```

```
        if ref_holds:
            for pn, (pid, e) in enumerate(sorted(ref_holds.items(), key=lambda x: x[1][1],
reverse=True), 1):
                rpx, rpy = holds_to_src((e[0]+e[2])/2, (e[1]+e[3])/2, traj_offset_ref)
                cx, cy = tp_ref(rpx, rpy)
                in_d = pn in self.presas_dep; in_r = pn in self.presas_ref
                col = QColor(self.COLOR_NINGUNO)
                if in_d and in_r: col = QColor(self.COLOR_AMBOS)
                elif in_r: col = QColor(self.COLOR_SOLO_REF)
                elif in_d: col = QColor(self.COLOR_SOLO_DEP)
                r = max(4, int(5*sx_r))
                painter.setBrush(Qt.NoBrush); painter.setPen(QPen(col, 2))
                painter.drawEllipse(cx-r, cy-r, r*2, r*2)

        painter.restore()
        painter.setPen(QColor("#185fa5")); painter.setFont(QF("Arial", 7, QF.Bold))
        painter.drawText(0, 0, half, TITLE_H, Qt.AlignCenter, "Referencia")

        painter.setPen(QPen(QColor("#aaa"), 1)); painter.drawLine(half, 0, half, ph)

        # DERECHA: DEPORTISTA
        tp_dep, sx_d, sy_d = make_tp(half, half)

        painter.save()
        painter.setClipRect(half, TITLE_H, half, body_h)

        if len(self.hip_traj_dep[:n_dep]) >= 2:
            painter.setPen(QPen(QColor(self.COLOR_SOLO_DEP), lw))
            traj = self.hip_traj_dep[:n_dep]
            for i in range(1, len(traj)):
                painter.drawLine(*tp_dep(*traj[i-1]), *tp_dep(*traj[i]))

        if skeleton_dep:
            c_d = QColor(self.COLOR_SOLO_DEP); cf_d = QColor(self.COLOR_SOLO_DEP);
cf_d.setAlpha(100)
            painter.setPen(QPen(c_d, max(1, lw))); painter.setBrush(QBrush(cf_d))
            for (x1,y1),(x2,y2) in skeleton_dep.get('lines', []):
                painter.drawLine(*tp_dep(x1,y1), *tp_dep(x2,y2))
            painter.setBrush(QBrush(c_d))
            for x,y in skeleton_dep.get('points', []):
                r = max(2, int(3*sx_d)); px_,py_ = tp_dep(x,y)
                painter.drawEllipse(px_-r, py_-r, r*2, r*2)

        if self.holds_dep:
            ox_d = self.traj_offset_dep
            for pn, (pid, e) in enumerate(sorted(self.holds_dep.items(), key=lambda x: x[1][1],
reverse=True), 1):
                rpx, rpy = holds_to_src((e[0]+e[2])/2, (e[1]+e[3])/2, ox_d)
                cx, cy = tp_dep(rpx, rpy)
                in_d = pn in self.presas_dep; in_r = pn in self.presas_ref
                col = QColor(self.COLOR_NINGUNO)
                if in_d and in_r: col = QColor(self.COLOR_AMBOS)
                elif in_d: col = QColor(self.COLOR_SOLO_DEP)
                elif in_r: col = QColor(self.COLOR_SOLO_REF)
                r = max(4, int(5*sx_d))
                painter.setBrush(Qt.NoBrush); painter.setPen(QPen(col, 2))
                painter.drawEllipse(cx-r, cy-r, r*2, r*2)
```

```
        painter.restore()
        painter.setPen(QColor("#a32d2d")); painter.setFont(QF("Arial", 7, QF.Bold))
        painter.drawText(half, 0, half, TITLE_H, Qt.AlignCenter, "Deportista")

        leg_items = [(self.COLOR_SOLO_DEP,"Dep"),(self.COLOR_SOLO_REF,"Ref"),
                (self.COLOR_AMBOS,"Ambos"),(self.COLOR_NINGUNO,"Ninguno")]
        leg_font = QF("Arial", 6); painter.setFont(leg_font)
        fm = painter.fontMetrics()
        sq, pad, row_h = 6, 3, 11
        max_tw = max(fm.horizontalAdvance(t) for _,t in leg_items)
        bw = sq+4+max_tw+pad*2; bh = row_h*len(leg_items)+pad*2
        bx = pw-bw-4; by = ph-bh-4
        painter.setPen(QPen(QColor("#ccc"),1));
painter.setBrush(QBrush(QColor(255,255,255,210)))
        painter.drawRoundedRect(bx, by, bw, bh, 3, 3)
        for i,(col,txt) in enumerate(leg_items):
            ry=by+pad+i*row_h; sx2=bx+pad; sy2=ry+(row_h-sq)//2
            painter.setPen(Qt.NoPen); painter.setBrush(QBrush(QColor(col)))
            painter.drawRect(sx2,sy2,sq,sq)
            painter.setPen(QColor("#333"))
            painter.drawText(sx2+sq+4,ry,max_tw+pad,row_h,Qt.AlignLeft|Qt.AlignVCenter,txt)

        painter.end()
        return pixmap

    def refresh_traj_holds(self, hold_boxes_dep):
        """Actualiza los centros de presas con color según estado. Llamar desde
update_dual_frame."""
        from PyQt5.QtGui import QPainter, QPen, QBrush, QColor
        if self.traj_pixmap_dual is None:
            return
        pixmap = self.traj_pixmap_dual.copy()
        painter = QPainter(pixmap)
        painter.setRenderHint(QPainter.Antialiasing)
        pw = pixmap.width()
        scale = pw / 180
        r = max(4, int(5 * scale))
        for idx, (x1, y1, x2, y2) in enumerate(hold_boxes_dep):
            presa = idx + 1
            cx = int(((x1 + x2) / 2) * scale)
            cy = int(((y1 + y2) / 2) * scale)
            in_dep = presa in self.presas_dep
            in_ref = presa in self.presas_ref
            if in_dep and in_ref:
                color = self.COLOR_AMBOS
            elif in_dep:
                color = self.COLOR_SOLO_DEP
            elif in_ref:
                color = self.COLOR_SOLO_REF
            else:
                color = self.COLOR_NINGUNO
            painter.setBrush(QBrush(QColor(color)))
            painter.setPen(QPen(QColor(color), 1))
            painter.drawEllipse(cx - r, cy - r, r * 2, r * 2)
        painter.end()
        self.traj_label_dual.setPixmap(pixmap)
```

```
# ----------------------------------------------------------------
# Calibración aproximada: distancia vertical real entre Presa 1 (más baja)
# y la última presa de la ruta. La pared estándar mide 15m pero el encuadre
# incluye piso/techo extra, así que esta es una aproximación editable.
WALL_HEIGHT_M = 14.0 #distancia aproximada de los puntos mencionados

def _px_to_m_factor(self):
    """
    Calcula el factor de conversión px → metros usando la distancia
    vertical en píxeles entre la presa más baja y la más alta detectadas,
    mapeada a WALL_HEIGHT_M.
    """
    if not self.holds_dep:
        return None
    ys = [entry[1] for entry in self.holds_dep.values()] + \
         [entry[3] for entry in self.holds_dep.values()]
    if not ys:
        return None
    height_px = max(ys) - min(ys)
    if height_px <= 0:
        return None
    return self.WALL_HEIGHT_M / height_px

def _draw_vel_chart(self):
    """
    Dibuja la gráfica de velocidad entre presas con QPainter.
    Eje X: tiempo (segundos), de 0 hasta el final de la escalada de cada uno.
    Eje Y: velocidad en m/s, fijo de 0 a 5.
    """
    from PyQt5.QtGui import QPainter, QPen, QBrush, QColor, QFont as QF

    map_ref = {int(p): t for p, t in self.touch_log_ref}
    map_dep = {int(p): t for p, t in self.touch_log_dep}
    ambos   = sorted(self.presas_ref & self.presas_dep)

    px_to_m = self._px_to_m_factor()
    if px_to_m is None:
        return

    # Distancia vertical en px entre cada par de presas consecutivas (tocadas
    # por ambos), usando las posiciones reales detectadas en holds_dep.
    sorted_holds = sorted(self.holds_dep.items(),
                          key=lambda e: e[1][1], reverse=True)
    # presa_num (1-19) -> centro y en px
    presa_cy = {}
    for i, (pid, entry) in enumerate(sorted_holds, start=1):
        presa_cy[i] = (entry[1] + entry[3]) / 2

    # tramos: (t_mid_ref, vel_ref_ms, t_mid_dep, vel_dep_ms)
    tramos = []
    for i in range(len(ambos) - 1):
        p1, p2 = ambos[i], ambos[i + 1]
        if p1 not in presa_cy or p2 not in presa_cy:
            continue
        dist_m = abs(presa_cy[p1] - presa_cy[p2]) * px_to_m
```

```
        dt_ref = map_ref[p2] - map_ref[p1]
        dt_dep = map_dep[p2] - map_dep[p1]
        if dt_ref <= 0 or dt_dep <= 0 or dist_m <= 0:
            continue

        vel_ref_ms = dist_m / dt_ref
        vel_dep_ms = dist_m / dt_dep
        t_mid_ref  = (map_ref[p1] + map_ref[p2]) / 2
        t_mid_dep  = (map_dep[p1] + map_dep[p2]) / 2

        tramos.append((t_mid_ref, round(vel_ref_ms, 3),
                       t_mid_dep, round(vel_dep_ms, 3)))

    # Actualizar promedios
    if tramos:
        vel_ref_list = [t[1] for t in tramos]
        vel_dep_list = [t[3] for t in tramos]
        avg_r = sum(vel_ref_list) / len(vel_ref_list)
        avg_d = sum(vel_dep_list) / len(vel_dep_list)
        diff  = avg_d - avg_r
        sign  = "+" if diff > 0 else ""
        self.lbl_vel_ref.setText(f"{avg_r:.2f} m/s")
        self.lbl_vel_dep.setText(f"{avg_d:.2f} m/s")
        self.lbl_vel_diff.setText(f"{sign}{diff:.2f} m/s")
        self.lbl_vel_diff.setStyleSheet(
            f"font-size: 14px; font-weight: bold; background: transparent; "
            f"color: {'#1d9e75' if diff >= 0 else '#e24b4a'};"
        )

    if not tramos:
        return

    pw = self.vel_canvas.width() or 400
    ph = self.vel_canvas.height() or 320

    pixmap = QPixmap(pw, ph)
    pixmap.fill(QColor("white"))
    painter = QPainter(pixmap)
    painter.setRenderHint(QPainter.Antialiasing)

    # Márgenes
    ml = 52
    mr = 16
    mt = 16
    mb = 52
    cw = pw - ml - mr
    ch = ph - mt - mb

    # Eje Y fijo: 0 a 5 m/s
    Y_MIN, Y_MAX = 0.0, 5.0

    # Eje X: tiempo, de 0 hasta el máximo tiempo final disponible
    t_end_candidates = [t for t in
                        [self.tiempo_final_ref, self.tiempo_final_dep] if t]
    if t_end_candidates:
        X_MAX = max(t_end_candidates)
    else:
```

```
    X_MAX = max(tr[0] for tr in tramos + [(0,)] ) if tramos else 1.0
X_MAX = max(X_MAX, 1.0)

def px_t(t):
    t = max(0.0, min(X_MAX, t))
    return ml + int(t / X_MAX * cw)

def py_pos(v):
    v = max(Y_MIN, min(Y_MAX, v))
    return mt + ch - int((v - Y_MIN) / (Y_MAX - Y_MIN) * ch)

font_s    = QF("Arial", 7)
font_m    = QF("Arial", 8)
font_axis = QF("Arial", 8, QF.Bold)

# Grid horizontal (Y: cada 1 m/s)
painter.setPen(QPen(QColor("#f0f0f0"), 1))
for step in range(6):  # 0,1,2,3,4,5
    y = py_pos(step)
    painter.drawLine(ml, y, ml + cw, y)

#  Grid vertical (X: divisiones de tiempo)
n_xticks = 6
painter.setPen(QPen(QColor("#f5f5f5"), 1))
for i in range(n_xticks + 1):
    t = X_MAX * i / n_xticks
    x = px_t(t)
    painter.drawLine(x, mt, x, mt + ch)

#  Ejes
painter.setPen(QPen(QColor("#bbb"), 1))
painter.drawLine(ml, mt, ml, mt + ch)
painter.drawLine(ml, mt + ch, ml + cw, mt + ch)

# Etiquetas eje Y (0 a 5 m/s)
painter.setFont(font_s)
painter.setPen(QColor("#666"))
for step in range(6):
    y = py_pos(step)
    painter.drawText(0, y - 6, ml - 4, 14, Qt.AlignRight | Qt.AlignVCenter,
                     str(step))

# Leyenda eje Y (rotada)
painter.save()
painter.setFont(font_axis)
painter.setPen(QColor("#555"))
painter.translate(10, mt + ch // 2)
painter.rotate(-90)
painter.drawText(-50, -4, 120, 14, Qt.AlignCenter, "Velocidad(m/s)")
painter.restore()

# Etiquetas eje X (tiempo)
painter.setFont(font_s)
painter.setPen(QColor("#666"))
for i in range(n_xticks + 1):
    t = X_MAX * i / n_xticks
    x = px_t(t)
```

```
    painter.drawText(x - 18, mt + ch + 4, 36, 14,
                     Qt.AlignCenter, f"{t:.1f}s")

# Leyenda eje X
painter.setFont(font_axis)
painter.setPen(QColor("#555"))
painter.drawText(ml + cw // 2 - 25, ph - 14, 90, 12,
                 Qt.AlignCenter, "Tiempo(s)")

# Línea referencia continua (azul)
pts_ref = sorted([(t, v) for t, v, _, _ in tramos])
pen_ref = QPen(QColor("#185fa5"), 2, Qt.SolidLine)
painter.setPen(pen_ref)
for i in range(len(pts_ref) - 1):
    painter.drawLine(px_t(pts_ref[i][0]),   py_pos(pts_ref[i][1]),
                     px_t(pts_ref[i+1][0]), py_pos(pts_ref[i+1][1]))
painter.setBrush(QBrush(QColor("#185fa5")))
painter.setPen(QPen(QColor("#185fa5"), 1))
for t, v in pts_ref:
    painter.drawEllipse(px_t(t) - 3, py_pos(v) - 3, 6, 6)

#  Línea deportista continua (roja)
pts_dep = sorted([(t, v) for _, _, t, v in tramos])
pen_dep = QPen(QColor("#a32d2d"), 2, Qt.SolidLine)
painter.setPen(pen_dep)
for i in range(len(pts_dep) - 1):
    painter.drawLine(px_t(pts_dep[i][0]),   py_pos(pts_dep[i][1]),
                     px_t(pts_dep[i+1][0]), py_pos(pts_dep[i+1][1]))
painter.setBrush(QBrush(QColor("#a32d2d")))
painter.setPen(QPen(QColor("#a32d2d"), 1))
for t, v in pts_dep:
    r2 = 3
    painter.drawRect(px_t(t) - r2, py_pos(v) - r2, r2 * 2, r2 * 2)

# Leyenda superior derecha
painter.setFont(font_m)
lx = ml + cw - 80
ly = mt + 6
painter.setPen(QPen(QColor("#185fa5"), 2))
painter.drawLine(lx, ly + 5, lx + 18, ly + 5)
painter.setBrush(QBrush(QColor("#185fa5")))
painter.setPen(QPen(QColor("#185fa5"), 1))
painter.drawEllipse(lx + 6, ly + 2, 6, 6)
painter.setPen(QColor("#333"))
painter.drawText(lx + 22, ly, 50, 12, Qt.AlignLeft | Qt.AlignVCenter, "Ref.")
painter.setPen(QPen(QColor("#a32d2d"), 2))
painter.drawLine(lx, ly + 20, lx + 18, ly + 20)
painter.setBrush(QBrush(QColor("#a32d2d")))
painter.setPen(QPen(QColor("#a32d2d"), 1))
painter.drawRect(lx + 6, ly + 17, 6, 6)
painter.setPen(QColor("#333"))
painter.drawText(lx + 22, ly + 14, 50, 12, Qt.AlignLeft | Qt.AlignVCenter, "Dep.")

painter.end()
self.vel_pixmap = pixmap
self.vel_canvas.setPixmap(pixmap)
```

```
# ------------------------------------------------------------------
def update_dual_frame(self):
    if len(processed_reference_frames) == 0 or len(processed_athlete_frames) == 0:
        return

    lw = self.left_label.width()
    lh = self.left_label.height()
    rw = self.right_label.width()
    rh = self.right_label.height()

    # Video referencia — con offset de sincronización
    ref_idx = self.left_index + self._ref_offset
    if ref_idx < len(processed_reference_frames):
        f1 = processed_reference_frames[ref_idx]
        h1, w1 = f1.shape[:2]
        f1s = cv2.resize(f1, (lw, lh), interpolation=cv2.INTER_AREA)
        img1 = QImage(f1s.data, lw, lh, lw * 3, QImage.Format_RGB888)
        self.left_label.setPixmap(QPixmap.fromImage(img1))
        self.left_index += 1
    else:
        self.left_finished = True

    # Video deportista — con offset de sincronización
    dep_idx = self.right_index + self._dep_offset
    if dep_idx < len(processed_athlete_frames):
        f2 = processed_athlete_frames[dep_idx]
        f2s = cv2.resize(f2, (rw, rh), interpolation=cv2.INTER_AREA)
        img2 = QImage(f2s.data, rw, rh, rw * 3, QImage.Format_RGB888)
        self.right_label.setPixmap(QPixmap.fromImage(img2))
        self.right_index += 1
    else:
        self.right_finished = True

    # Actualizar siluetas usando el frame correspondiente al índice actual
    if self.skeleton_frames_ref and self.left_index > 0:
        idx_r = min(self.left_index - 1 + self._ref_offset, len(self.skeleton_frames_ref) - 1)
        self.skeleton_ref = self.skeleton_frames_ref[max(0, idx_r)]
    if self.skeleton_frames_dep and self.right_index > 0:
        idx_d = min(self.right_index - 1 + self._dep_offset, len(self.skeleton_frames_dep) - 1)
        self.skeleton_dep = self.skeleton_frames_dep[max(0, idx_d)]
    self._draw_traj_panel()

    if self.left_finished and self.right_finished:
        self.timer.stop()

# ------------------------------------------------------------------
def repetir_videos(self):
    """Reinicia la reproducción de los videos y el panel de trayectoria una vez."""
    self.left_index    = 0
    self.right_index   = 0
    self.left_finished = False
    self.right_finished= False
    if not self.timer.isActive():
        self.timer.start(80)

def export_pdf(self):
    """Exporta panel trayectoria + métricas + tabla a PDF."""
```

```
        from PyQt5.QtWidgets import QFileDialog, QMessageBox
        path, _ = QFileDialog.getSaveFileName(self, "Guardar PDF", "analisis.pdf", "PDF (*.pdf)")
        if not path:
            return
        try:
            from reportlab.lib.pagesizes import A4
            from reportlab.platypus import SimpleDocTemplate, Table, TableStyle, Paragraph, Spacer,
Image as RLImage
            from reportlab.lib import colors
            from reportlab.lib.styles import getSampleStyleSheet
            import tempfile

            doc = SimpleDocTemplate(path, pagesize=A4)
            styles = getSampleStyleSheet()
            story = []

            story.append(Paragraph("Análisis General - Escalada de Velocidad", styles["Title"]))
            story.append(Spacer(1, 12))

            # Panel trayectoria como imagen
            if self.traj_pixmap_dual:
                tmp = tempfile.NamedTemporaryFile(suffix=".png", delete=False)
                self.traj_pixmap_dual.save(tmp.name)
                story.append(RLImage(tmp.name, width=180, height=320))
                story.append(Spacer(1, 10))

            # Métricas
            map_ref = {p: t for p, t in self.touch_log_ref}
            map_dep = {p: t for p, t in self.touch_log_dep}
            # Tiempo total: usar el tiempo final del cronómetro (misma fuente de
            # verdad que el panel en pantalla).
            t_ref = self.tiempo_final_ref
            t_dep = self.tiempo_final_dep

            data_met = [
                ["", "Referencia", "Deportista", "Diferencia"],
                ["Tiempo total (seg)",
                 f"{t_ref:.2f}" if t_ref is not None else "--",
                 f"{t_dep:.2f}" if t_dep is not None else "--",
                 f"{t_dep - t_ref:+.2f}" if (t_ref is not None and t_dep is not None) else "--"],
                ["Presas tocadas",
                 str(len(self.presas_ref)),
                 str(len(self.presas_dep)),
                 f"{len(self.presas_dep) - len(self.presas_ref):+d}"],
            ]
            tbl_met = Table(data_met, colWidths=[130, 90, 90, 90])
            tbl_met.setStyle(TableStyle([
                ("BACKGROUND", (0, 0), (-1, 0), colors.HexColor("#1a5fa8")),
                ("TEXTCOLOR", (0, 0), (-1, 0), colors.white),
                ("FONTNAME", (0, 0), (-1, 0), "Helvetica-Bold"),
                ("ALIGN", (1, 0), (-1, -1), "CENTER"),
                ("GRID", (0, 0), (-1, -1), 0.5, colors.lightgrey),
                ("ROWBACKGROUNDS", (0, 1), (-1, -1), [colors.white, colors.HexColor("#f5f5f5")]),
            ]))
            story.append(tbl_met)
            story.append(Spacer(1, 12))
```

```
# Tiempo por tramos (mismos colores que la tabla de toques)
if hasattr(self, '_tramo_widgets') and self._tramo_widgets:
    tramo_meta = [
        ("Salida (P1-P7)",        "#c8dcf5"),
        ("Zona central (P8-P13)", "#f5e0b8"),
        ("Llegada (P14-P19)",     "#c0ecce"),
    ]
    data_tramos = [["Tramo", "Ref. (seg)", "Dep. (seg)", "Diferencia"]]
    for (p_ini, p_fin, lbl_r, lbl_d, lbl_df), (nombre, _bg) in \
            zip(self._tramo_widgets, tramo_meta):
        data_tramos.append([nombre, lbl_r.text(), lbl_d.text(), lbl_df.text()])
    tbl_tramos = Table(data_tramos, colWidths=[150, 80, 80, 90])
    style_tramos = [
        ("BACKGROUND", (0, 0), (-1, 0), colors.HexColor("#1a5fa8")),
        ("TEXTCOLOR", (0, 0), (-1, 0), colors.white),
        ("FONTNAME", (0, 0), (-1, 0), "Helvetica-Bold"),
        ("ALIGN", (1, 0), (-1, -1), "CENTER"),
        ("GRID", (0, 0), (-1, -1), 0.5, colors.lightgrey),
    ]
    for r, (nombre, bg) in enumerate(tramo_meta, start=1):
        style_tramos.append(("BACKGROUND", (0, r), (-1, r), colors.HexColor(bg)))
    tbl_tramos.setStyle(TableStyle(style_tramos))
    story.append(Paragraph("Tiempo por tramos", styles["Heading2"]))
    story.append(tbl_tramos)
    story.append(Spacer(1, 12))

# Gráfica de velocidad entre presas
vel_pm = getattr(self, 'vel_pixmap', None)
if vel_pm is None:
    vel_pm = self.vel_canvas.pixmap()
if vel_pm is not None and not vel_pm.isNull():
    tmpv = tempfile.NamedTemporaryFile(suffix=".png", delete=False)
    vel_pm.save(tmpv.name)
    vw = vel_pm.width() or 500
    vh = vel_pm.height() or 320
    disp_w = 440.0
    disp_h = disp_w * vh / vw
    story.append(Paragraph("Velocidad entre presas", styles["Heading2"]))
    story.append(RLImage(tmpv.name, width=disp_w, height=disp_h))
    story.append(Spacer(1, 12))

# Tabla comparativa
data_tbl = [["Presa", "Ref. (seg)", "Dep. (seg)", "Diferencia"]]
for i in range(19):
    p = i + 1
    t_r = map_ref.get(p)
    t_d = map_dep.get(p)
    if t_r is not None and t_d is not None:
        diff = t_d - t_r   # dep - ref (misma convención que en pantalla)
        diff_str = f"{diff:+.2f}"
    else:
        diff_str = "—"
    data_tbl.append([
        f"Presa {p}",
        f"{t_r:.2f}" if t_r is not None else "—",
        f"{t_d:.2f}" if t_d is not None else "—",
        diff_str,
```

```
            ])
        tbl = Table(data_tbl, colWidths=[100, 100, 100, 100])
        tbl.setStyle(TableStyle([
            ("BACKGROUND", (0, 0), (-1, 0), colors.HexColor("#1a5fa8")),
            ("TEXTCOLOR", (0, 0), (-1, 0), colors.white),
            ("FONTNAME", (0, 0), (-1, 0), "Helvetica-Bold"),
            ("ALIGN", (1, 0), (-1, -1), "CENTER"),
            ("GRID", (0, 0), (-1, -1), 0.5, colors.lightgrey),
            # Color por tramo (mismo esquema que el recuadro de tiempo por tramos)
            ("BACKGROUND", (0, 1),  (-1, 7),  colors.HexColor("#c8dcf5")),  # Salida P1-P7
            ("BACKGROUND", (0, 8),  (-1, 13), colors.HexColor("#f5e0b8")),  # Central P8-P13
            ("BACKGROUND", (0, 14), (-1, 19), colors.HexColor("#c0ecce")),  # Llegada P14-P19
        ]))
        story.append(Paragraph("Registro comparativo de toques", styles["Heading2"]))
        story.append(tbl)
        doc.build(story)
        QMessageBox.information(self, "PDF guardado", f"Archivo guardado en:\n{path}")
    except ImportError:
        QMessageBox.warning(self, "Error", "Instala reportlab:\npip install reportlab")
    except Exception as e:
        QMessageBox.critical(self, "Error al exportar", str(e))

def save_video(self):
    """
    Graba el análisis dual como video MP4 H.264 usando imageio-ffmpeg.
    """
    from PyQt5.QtWidgets import QFileDialog, QMessageBox

    path, _ = QFileDialog.getSaveFileName(
        self, "Guardar video", "analisis.mp4", "Video (*.mp4)"
    )
    if not path:
        return

    if not processed_reference_frames or not processed_athlete_frames:
        QMessageBox.warning(self, "Sin datos",
                            "Primero analiza los dos videos antes de guardar.")
        return

    try:
        import numpy as np
        import imageio.v2 as iio

        fps    = 12

        # Offsets de sincronizacion: empezar 1 seg antes de que
        # arranque el cronometro de cada video (igual que en vivo)
        ref_start = self._ref_offset
        dep_start = self._dep_offset
        n_frames  = max(len(processed_reference_frames) - ref_start,
                        len(processed_athlete_frames)  - dep_start)
        n_frames  = max(1, n_frames)

        from PyQt5.QtCore import QPoint
        from PyQt5.QtGui import QImage as _QImage

        def qimage_to_numpy(qimg):
```

```
    w, h = qimg.width(), qimg.height()
    rows = []
    for y in range(h):
        ptr = qimg.scanLine(y)
        ptr.setsize(w * 3)
        rows.append(np.frombuffer(ptr, dtype=np.uint8).copy())
    return np.stack(rows, axis=0).reshape(h, w, 3)

# Capturar la ventana completa una sola vez: sirve de fondo estatico
# (titulo, tablas, graficas, metricas) conservando el tamano y la
# posicion exacta de cada recuadro tal como se ven en la ventana.
screen    = QApplication.primaryScreen()
full_grab = screen.grabWindow(self.winId())
full_img  = full_grab.toImage().convertToFormat(_QImage.Format_RGB888)
full_arr  = qimage_to_numpy(full_img)

scale_x = full_img.width()  / self.width()
scale_y = full_img.height() / self.height()

# H.264 requiere dimensiones pares
out_h = full_arr.shape[0] - (full_arr.shape[0] % 2)
out_w = full_arr.shape[1] - (full_arr.shape[1] % 2)
full_arr = full_arr[:out_h, :out_w, :]

def widget_rect_px(widget):
    tl = widget.mapTo(self, QPoint(0, 0))
    x = int(tl.x() * scale_x)
    y = int(tl.y() * scale_y)
    w = int(widget.width()  * scale_x)
    h = int(widget.height() * scale_y)
    w = max(0, min(w, out_w - x))
    h = max(0, min(h, out_h - y))
    return x, y, w, h

lx, ly, lw, lh = widget_rect_px(self.left_label)   # video referencia
rx, ry, rw, rh = widget_rect_px(self.right_label)  # video deportista
tx, ty, tw, th = widget_rect_px(self.traj_label_dual)

# Grosor del borde del panel de trayectoria que se conserva del grab
traj_margin = max(1, int(round(scale_x)))

traj_pw = self.traj_label_dual.width()
traj_ph = self.traj_label_dual.height()
len_skel_ref = len(self.skeleton_frames_ref)
len_skel_dep = len(self.skeleton_frames_dep)

def render_traj_np(n_ref, n_dep, skel_ref, skel_dep):
    pm  = self._render_traj_pixmap(traj_pw, traj_ph,
                                   n_ref, n_dep, skel_ref, skel_dep)
    qim = pm.toImage().convertToFormat(_QImage.Format_RGB888)
    arr = qimage_to_numpy(qim)
    return cv2.resize(arr, (tw, th), interpolation=cv2.INTER_AREA)

writer = iio.get_writer(
    path,
    format="ffmpeg",
    mode="I",
```

```
            fps=fps,
            codec="libx264",
            ffmpeg_log_level="quiet",
            output_params=["-crf", "23", "-preset", "fast",
                           "-pix_fmt", "yuv420p"],
            ffmpeg_params=None,
            macro_block_size=None,
        )

        len_ref = len(processed_reference_frames)
        len_dep = len(processed_athlete_frames)

        for i in range(n_frames):
            ri = ref_start + i
            di = dep_start + i

            f1 = processed_reference_frames[ri] if ri < len_ref \
                 else processed_reference_frames[-1]
            f2 = processed_athlete_frames[di] if di < len_dep \
                 else processed_athlete_frames[-1]

            frame = full_arr.copy()

            # Videos en su tamano/posicion exactos (como en la ventana)
            if lw > 0 and lh > 0:
                frame[ly:ly + lh, lx:lx + lw, :] = \
                    cv2.resize(f1, (lw, lh), interpolation=cv2.INTER_AREA)
            if rw > 0 and rh > 0:
                frame[ry:ry + rh, rx:rx + rw, :] = \
                    cv2.resize(f2, (rw, rh), interpolation=cv2.INTER_AREA)

            # Trayectoria y silueta progresivas, sincronizadas con cada video.
            # Se conserva el borde del panel copiando solo su interior.
            n_ref = min(ri + 1, len(self.hip_traj_ref))
            n_dep = min(di + 1, len(self.hip_traj_dep))
            skel_ref = self.skeleton_frames_ref[min(ri, len_skel_ref - 1)] \
                       if len_skel_ref else self.skeleton_ref
            skel_dep = self.skeleton_frames_dep[min(di, len_skel_dep - 1)] \
                       if len_skel_dep else self.skeleton_dep

            if tw > 0 and th > 0:
                traj_np = render_traj_np(n_ref, n_dep, skel_ref, skel_dep)
                m = traj_margin
                if tw > 2 * m and th > 2 * m:
                    frame[ty + m:ty + th - m, tx + m:tx + tw - m, :] = \
                        traj_np[m:th - m, m:tw - m, :]
                else:
                    frame[ty:ty + th, tx:tx + tw, :] = traj_np

            writer.append_data(frame)

        writer.close()
        QMessageBox.information(self, "Video guardado",
                                f"Video guardado en:\n{path}")

    except ImportError:
        QMessageBox.warning(self, "Librería faltante",
```

```
                "Instala las dependencias:\n"
                "pip install imageio imageio-ffmpeg")
        except Exception as e:
            QMessageBox.critical(self, "Error al guardar video", str(e))

    def closeEvent(self, event):
        self.timer.stop()
        event.accept()


# ----------------------------------------------------
#   VENTANA PRINCIPAL UTN
# ----------------------------------------------------
class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()

        self.crop_ref = None
        self.crop_dep = None
        self.setWindowTitle("Sistema de Análisis UTN")
        self.setWindowIcon(QIcon("UTN.png"))
        self.setWindowFlag(Qt.Window)

        central_widget = QWidget()
        self.setCentralWidget(central_widget)

        main_layout = QVBoxLayout()

        # INSTRUCCIONES
        lbl_instructions = QLabel(
            "Indicaciones:\n"
            "1. El video de referencia y del deportista a analizar deben ser en alta definición y
orientado verticalmente.\n"
            "2. Los videos ingresados deben mostrar la pared de escalada completa enfocando la ruta
donde se encuentre el deportista.\n"
            "3. Al presionar los botones de ANALIZAR, la aplicación le permitirá seleccionar el área
de la trayectoria donde se encuentra el deportista. \n"
            "4. Al aceptar el recorte se mostrarán los datos que extrae el sistema del video ingresado, si
considera que dichos datos son erróneos, vuelva a ingresar el video.\n"
            "5. El botón de ANÁLISIS GENERAL compara al video del deportista con el video de
referencia, por lo cuál debe ingresar los videos adecuadamente al sistema.\n"
            "6. En la ventana de análisis general tendrá la opción de guardar el análisis realizado en
forma de video o mediante un reporte breve en formato PDF."
        )

        lbl_instructions.setFont(QFont("Arial", 8))
        lbl_instructions.setAlignment(Qt.AlignLeft)
        lbl_instructions.setStyleSheet(
            "color: #2e302e;"
            "background-color: rgba(255,255,255,0.6);"
            "padding: 6px;")
        lbl_instructions.setWordWrap(True)

        # IMAGEN UTN
        img = QLabel()
        pix = QPixmap("UTN.png")
        img.setPixmap(pix)
```

```
        # img.setScaledContents(True)
        img.setMaximumHeight(250)
        img.setAlignment(Qt.AlignCenter)
        main_layout.addWidget(img)

        # LÍNEAS
        lineas_texto = ["UNIVERSIDAD TÉCNICA DEL NORTE",
                        "Facultad de Ingeniería en Ciencias Aplicadas",
                        "Ingeniería en Mecatrónica \n",
                        'Tema: "Sistema de análisis de performance de un velocista de escalada deportiva
mediante visión artificial"',
                        "Autor: Edisson Vargas"]

        for i, texto in enumerate(lineas_texto):
            lbl = QLabel(texto)

            font = QFont("Times new roman")

            # Personalización por etiqueta
            if i == 0:
                font.setPointSize(26)
                font.setBold(True)

            elif i == 1:
                font.setPointSize(18)
                font.setBold(True)

            elif i == 2:
                font.setPointSize(19)
                font.setBold(True)
                # font.setItalic(True)

            elif i == 3:
                font.setPointSize(16)

            elif i == 4:
                font.setPointSize(14)

            lbl.setFont(font)
            lbl.setAlignment(Qt.AlignCenter)
            main_layout.addWidget(lbl)

        # ---------------- VIDEO REFERENCIA ----------------
        ref_layout = QHBoxLayout()

        self.ref_input = QLineEdit()
        self.ref_input.setPlaceholderText("Seleccione el video de referencia...")
        ref_layout.addWidget(self.ref_input)

        btn_sel_ref = QPushButton("Seleccionar")
        btn_sel_ref.clicked.connect(self.select_reference_video)
        ref_layout.addWidget(btn_sel_ref)

        lbl_ref = QLabel("REFERENCIA")
        lbl_ref.setFont(QFont("Arial", 14))
        ref_layout.addWidget(lbl_ref)
```

```
    btn_anal_ref = QPushButton("ANALIZAR")
    btn_anal_ref.clicked.connect(self.open_reference_view)
    ref_layout.addWidget(btn_anal_ref)

    main_layout.addLayout(ref_layout)

    # ---------------- VIDEO DEPORTISTA ----------------
    dep_layout = QHBoxLayout()

    self.dep_input = QLineEdit()
    self.dep_input.setPlaceholderText("Seleccione el video del deportista...")
    dep_layout.addWidget(self.dep_input)

    btn_sel_dep = QPushButton("Seleccionar")
    btn_sel_dep.clicked.connect(self.select_athlete_video)
    dep_layout.addWidget(btn_sel_dep)

    lbl_dep = QLabel("DEPORTISTA")
    lbl_dep.setFont(QFont("Arial", 14))
    dep_layout.addWidget(lbl_dep)

    btn_anal_dep = QPushButton("ANALIZAR")
    btn_anal_dep.clicked.connect(self.open_athlete_view)
    dep_layout.addWidget(btn_anal_dep)

    main_layout.addLayout(dep_layout)

    # ---------------- ANÁLISIS GENERAL ----------------
    btn_general = QPushButton("ANÁLISIS GENERAL")
    btn_general.setFont(QFont("Arial", 12))
    btn_general.setFixedHeight(30)
    btn_general.setFixedWidth(250)
    btn_general.clicked.connect(self.open_general_view)

    btn_layout = QHBoxLayout()
    btn_layout.addStretch()
    btn_layout.addWidget(btn_general)
    btn_layout.addStretch()

    main_layout.addLayout(btn_layout)
    main_layout.addSpacing(30)
    central_widget.setLayout(main_layout)

    # las dos siguientes lineas es parte de las INSTRUCCIONES
    main_layout.addSpacing(6)  # Empuja el contenido hacia arriba
    main_layout.addWidget(lbl_instructions)
    self.showMaximized()

# Métodos para seleccionar videos
def select_reference_video(self):
    path, _ = QFileDialog.getOpenFileName(self, "Seleccionar Video")
    if path:
        self.ref_input.setText(path)

def select_athlete_video(self):
    path, _ = QFileDialog.getOpenFileName(self, "Seleccionar Video")
    if path:
```

```
        self.dep_input.setText(path)

    # Abrir reproductores
    def open_reference_view(self):
        path = self.ref_input.text().strip()
        if not path:
            return

        dlg = CropSelectorDialog(path)
        if dlg.exec_() == QDialog.Accepted:
            self.crop_ref = dlg.get_crop()
            athlete_roi = dlg.get_athlete_roi() if hasattr(dlg, 'get_athlete_roi') else None
            self.win_ref = VideoWindow(
                path,
                "Video de Referencia",
                self.crop_ref,
                athlete_roi=athlete_roi
            )
            self.win_ref.show()

    def open_athlete_view(self):
        path = self.dep_input.text().strip()
        if not path:
            return

        dlg = CropSelectorDialog(path)
        if dlg.exec_() == QDialog.Accepted:
            self.crop_dep = dlg.get_crop()
            athlete_roi = dlg.get_athlete_roi() if hasattr(dlg, 'get_athlete_roi') else None
            self.win_dep = VideoWindow(
                path,
                "Video del Deportista",
                self.crop_dep,
                athlete_roi=athlete_roi
            )
            self.win_dep.show()

    def open_general_view(self):
        ref = self.ref_input.text().strip()
        dep = self.dep_input.text().strip()

        if not ref or not dep:
            return

        # Recoger datos de los VideoWindows si ya se analizaron
        touch_log_ref = []
        touch_log_dep = []
        hip_traj_dep = []
        hip_traj_ref = []
        tiempo_final_ref = None
        tiempo_final_dep = None

        if hasattr(self, 'win_ref') and hasattr(self.win_ref, 'touch_tracker'):
            touch_log_ref    = list(self.win_ref.touch_tracker.touch_log.items())
            tiempo_final_ref = self.win_ref.climb_timer.elapsed_seconds()
            hip_traj_ref     = list(getattr(self.win_ref, 'hip_trajectory_transformed',
                                            self.win_ref.hip_trajectory))
```

```
    start_frame_ref  = self.win_ref.climb_timer.start_frame or 0
    fps_ref          = self.win_ref.fps
    skeleton_ref     = getattr(self.win_ref, '_last_skeleton', None)
    skeleton_frames_ref = list(getattr(self.win_ref, 'skeleton_frames', []))
    holds_ref        = dict(getattr(self.win_ref, '_holds_snapshot', {}))
    traj_offset_ref  = getattr(self.win_ref, '_traj_offset_x', 0)
else:
    start_frame_ref  = 0
    fps_ref          = 30.0
    skeleton_ref     = None
    skeleton_frames_ref = []
    holds_ref        = {}
    traj_offset_ref  = 0

if hasattr(self, 'win_dep') and hasattr(self.win_dep, 'touch_tracker'):
    touch_log_dep    = list(self.win_dep.touch_tracker.touch_log.items())
    hip_traj_dep     = list(getattr(self.win_dep, 'hip_trajectory_transformed',
                                    self.win_dep.hip_trajectory))
    tiempo_final_dep = self.win_dep.climb_timer.elapsed_seconds()
    start_frame_dep  = self.win_dep.climb_timer.start_frame or 0
    fps_dep          = self.win_dep.fps
    skeleton_dep     = getattr(self.win_dep, '_last_skeleton', None)
    skeleton_frames_dep = list(getattr(self.win_dep, 'skeleton_frames', []))
    traj_offset_dep  = getattr(self.win_dep, '_traj_offset_x', 0)
    traj_rot_deg     = 10
else:
    start_frame_dep  = 0
    fps_dep          = 30.0
    skeleton_dep     = None
    skeleton_frames_dep = []
    traj_offset_dep  = 0
    traj_rot_deg     = 10

if hasattr(self, 'win_dep') and getattr(self.win_dep, '_holds_snapshot', None):
    holds_dep = {pid: list(entry)
                 for pid, entry in self.win_dep._holds_snapshot.items()}
else:
    holds_dep = {pid: list(entry)
                 for pid, entry in HOLD_DETECTOR.last_known.items()}

self.win_general = DualVideoWindow(
    ref_path=ref,
    dep_path=dep,
    crop_ref=self.crop_ref,
    crop_dep=self.crop_dep,
    touch_log_ref=touch_log_ref,
    touch_log_dep=touch_log_dep,
    hip_traj_dep=hip_traj_dep,
    hip_traj_ref=hip_traj_ref,
    tiempo_final_ref=tiempo_final_ref,
    tiempo_final_dep=tiempo_final_dep,
    holds_dep=holds_dep,
    holds_ref=holds_ref,
    start_frame_ref=start_frame_ref,
    start_frame_dep=start_frame_dep,
    fps_ref=fps_ref,
    fps_dep=fps_dep,
```

```
            skeleton_ref=skeleton_ref,
            skeleton_dep=skeleton_dep,
            skeleton_frames_ref=skeleton_frames_ref,
            skeleton_frames_dep=skeleton_frames_dep,
            traj_offset_dep=traj_offset_dep,
            traj_offset_ref=traj_offset_ref,
            traj_rot_deg=traj_rot_deg
        )
        self.win_general.show()


# ------------------------------------------------------
# MAIN
# ------------------------------------------------------
def main():
    app = QApplication(sys.argv)
    window = MainWindow()
    sys.exit(app.exec_())


if __name__ == "__main__":
    main()
```