UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Desarrollo de un sistema para detección remota del Nistagmo basado en análisis de vibraciones

TESIS

Que para obtener el título de

Ingeniera Mecánica

P R E S E N T A

Leticia Alejandra Arias Coss

DIRECTOR DE TESIS

Dr. César Abraham Luna Estrada

Ciudad Universitaria, Cd. Mx., 2026

Agradecimientos

A todas las personas que se han cruzado en mi camino, porque directa o indirectamente han contribuido a formar la persona que soy hoy.

En la UNAM encontré una segunda casa desde el instante en que entré a prepa 1, y, más adelante, en la FI. Viví experiencias, oportunidades y emociones de todo tipo, que no hubiera sido igual sin mis compañeros y amistades tan especiales, con quienes nos apoyamos en lo personal, académico y profesional a lo largo de este camino.

Aunque no los nombre, a mis amigos por todo el cariño y ayuda en el proceso, incluyendo a mis vecinos, con su internet curse la primera mitad de la carrera. Mis padrinos Charlie y Rebe, por brindarme un espacio tranquilo y cafesito. Al cuarto piso por ser un espacio y una comunidad increíbles y más cafesito. Christian Arellano por responder mis dudas y explicar a detalle temas de oftalmología.

A todos los profesores que me han dado clase o ayudado, la mayoría han sido maravillosos. En especial a la maestra Cecilia, mi otra mamá; Roberto por conectarme con las matemáticas; Rocío con la física; Lázaro Morales por transmitir su pasión a la biomecánica; mi tutora Ale Garza y Fer Velázquez por contestar mis dudas y guiarme durante toda la carrera. Juan Velázquez por el apoyo en todos los ámbitos y ser como mi papá de la facultad. Finalmente, mi director de tesis, César a quien conocí en primer semestre y me ayuda a cerrar mi ciclo como estudiante, incluyendo crisis existenciales.

A mi familia, por todo lo vivido y su compañía, en espacial a mis tíos Coss Galván, por apoyarme, cuidarme y, en momentos, ser como segundas madres y padres.

Peggy por darme sus huesitos, Negra por cada sábado, Bella, Katherine, Lily y todas mis mascotas, por compartir su vida conmigo. Un tiempo solo éramos Harry Potter y yo contra el mundo, siempre a mi lado con amor puro.

También a ese ser especial que me acompaña y ayuda en cada paso para seguir creciendo, eres magia en la vida.

A mi hermano Eduardo, por estar ahí, sigue creciendo y cumpliendo tus metas, siempre te cuidaré. A mis abuelitos por ser parte esencial de mi vida, dándome todo con solo verlos y en cada abrazo, abuelo no veras este logro que compartimos, pero siempre estarás conmigo, abuela gracias por todo. A mis padres, Alfredo por el apoyo hasta los primeros semestres, Leticia por quedarte desde el inicio, ahora y siempre.

Eduardo, Leticia y abuelitos, esto también es suyo. Gracias por el amor incondicional.

ÍNDICE

CAPÍTULO 1

INTRODUCCIÓN

Capítulo 1 Introducción

1.1 Antecedentes

1.1.1 Sistemas del cuerpo humano

Los sistemas del cuerpo humano están formados por grupos de órganos o tejidos que trabajan de manera conjunta para producir y mantener las funciones vitales, como se observa en la Figura 1 [1].

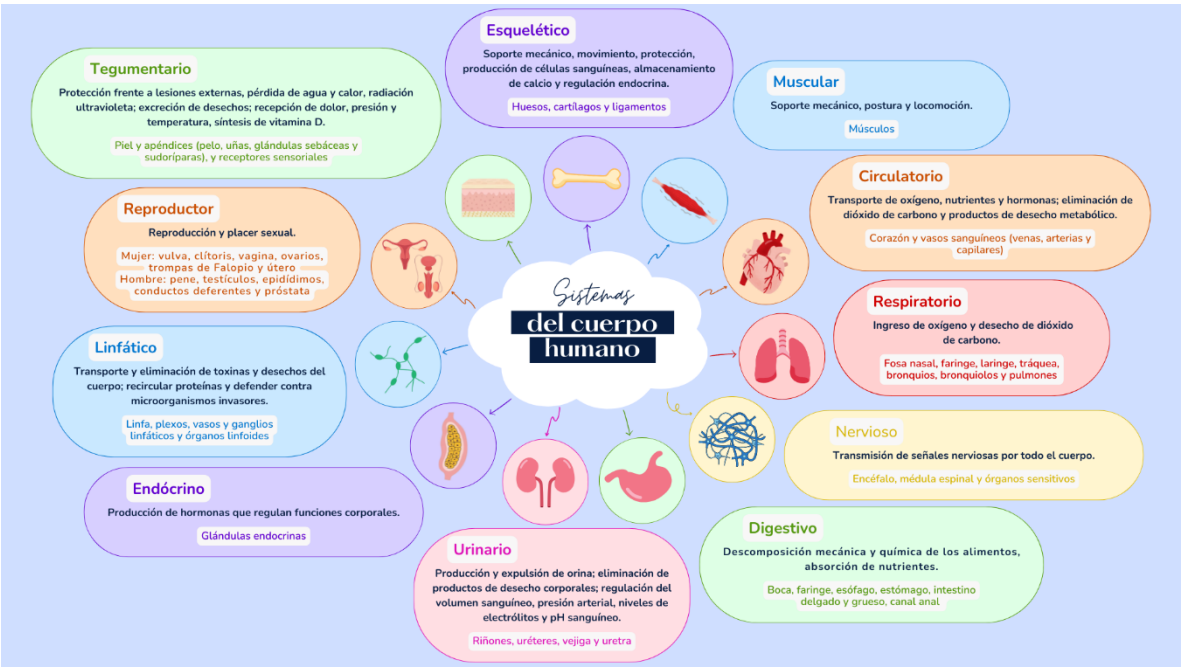

Figura 1. *Sistemas del cuerpo humano con sus órganos o tejidos. Elaboración propia.*

1.1.2 Aplicaciones médicas de la vibración

La Figura 2 sintetiza las aplicaciones de la **vibración** en los distintos sistemas del cuerpo humano, las cuales abarcan desde la prevención y mejora de lesiones en el cuerpo [2–4] hasta la rehabilitación [5–8], la disminución de síntomas [9–11], el diagnóstico de enfermedades [12, 13] e incluso el placer [14, 15].

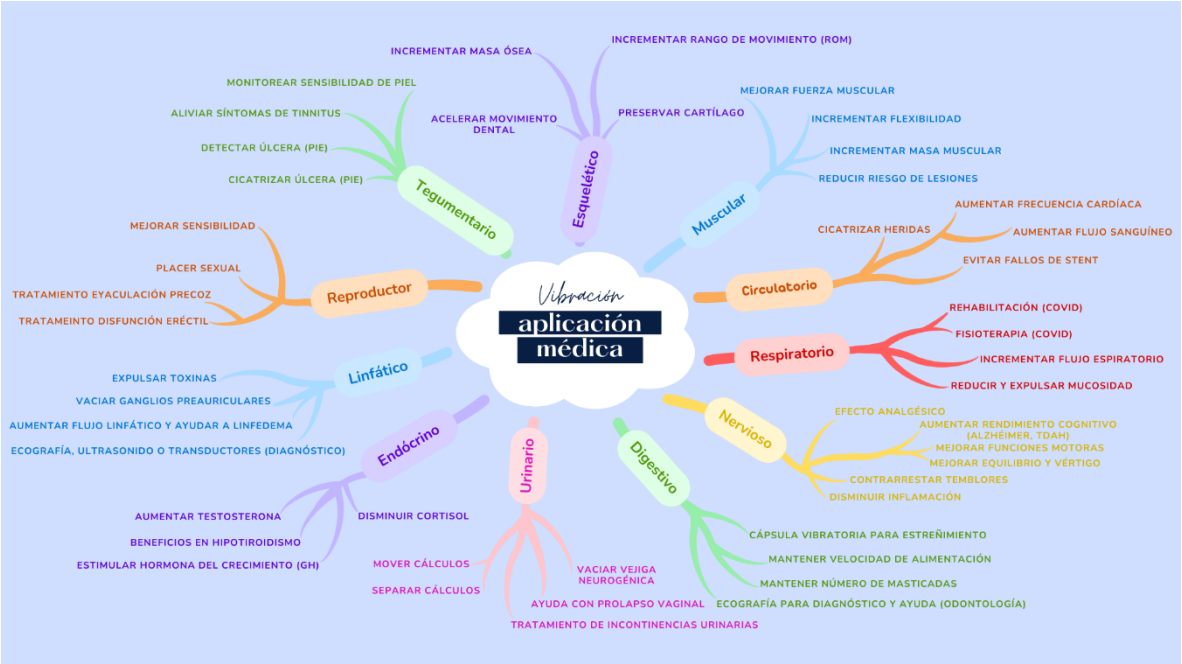

Figura 2. *Aplicaciones médicas de la vibración en sistemas del cuerpo humano. Elaboración propia.*

En el ámbito del **sistema nervioso**, una de las aplicaciones se orienta a mejorar el equilibrio y reducir el vértigo, el cual puede estar asociado a una condición denominada **nistagmo**, generalmente clasificada como oftalmológica [16].

1.1.3 Oftalmología

Dentro del sistema nervioso se encuentra el sistema sensorial, conformado por los cinco sentidos: vista, oído, olfato, gusto y tacto. En el caso de la vista, existen diversas patologías.

Algunos de los padecimientos más frecuentes se deben principalmente al mal uso de lentes de contacto, un cuerpo ajeno, virus, bacterias o alergias. Aunque también pueden originarse por hongos o traumatismos, estos últimos suelen considerarse emergencias [17].

Signos y síntomas clínicos generales

Los **signos** son objetivos y detectables al observar, en este caso, el ojo y sus alrededores; pueden cuantificarse [18]. Los **síntomas**, aunque algunos se pueden ver y medir, en su mayoría son subjetivos y corresponden a lo que experimenta y describe el paciente [19].

Cada patología presenta sus propios signos y síntomas; algunos pueden ser compartidos, y pueden manifestarse solo algunos o la totalidad de ellos. El enfoque principal en esta sección se centra en aquellos que son visibles.

Patologías oftálmicas

La mayoría de las patologías tiene signos y síntomas no visibles, por lo que se enlistan aquellas que presentan manifestaciones externas en la Figura 3, siendo ojo rojo el signo más frecuente [17, 20].

Figura 3. *Clasificación de patologías y afecciones oftalmológicas. Elaboración propia.*

De todas ellas, las cuatro con mayor número de signos y síntomas son la dacriocistitis, el coloboma, el retinoblastoma y el nistagmo. Este último puede presentarse como signo de otro padecimiento ocular o de manera independiente.

Panorama nacional de oftalmología

En 2019 se estimó que 11 millones de personas en México vivían con ceguera o discapacidad visual. La pérdida de visión constituye un problema de salud pública, siendo la mayoría de los casos prevenibles [21].

México se ubicó entre los 20 países con mayor número de personas con discapacidad visual y ceguera. La Sociedad Mexicana de Oftalmología reportó que hay aproximadamente 2.2 millones personas con deficiencia visual y 415 mil con ceguera. [22, 23].

Para 2020, se calculó que el 44 % de la población mexicana presentaba algún tipo de discapacidad visual. La OMS estimó que hasta el 80 % de los casos de ceguera y deficiencia visual son prevenibles [24].

1.1.4 Nistagmo

¿Qué es?

El nistagmo es un movimiento u oscilación, rítmico e involuntario de uno o ambos ojos [25, 26].

De acuerdo con un especialista oftalmólogo, la distribución por grupo etario varía según la causa del nistagmo. Los casos de origen neurológico central son más frecuentes en adultos, mientras que aquellos asociados a baja visión temprana se observan principalmente en población pediátrica. De manera general, el nistagmo se presenta con mayor frecuencia en menores de edad; por ejemplo, en condiciones de desarrollo u otras situaciones similares.

El nistagmo puede originarse por distintos factores, agrupados principalmente en **oculares** y **neurológicos**, como se muestra en la Figura 4 [26, 27].

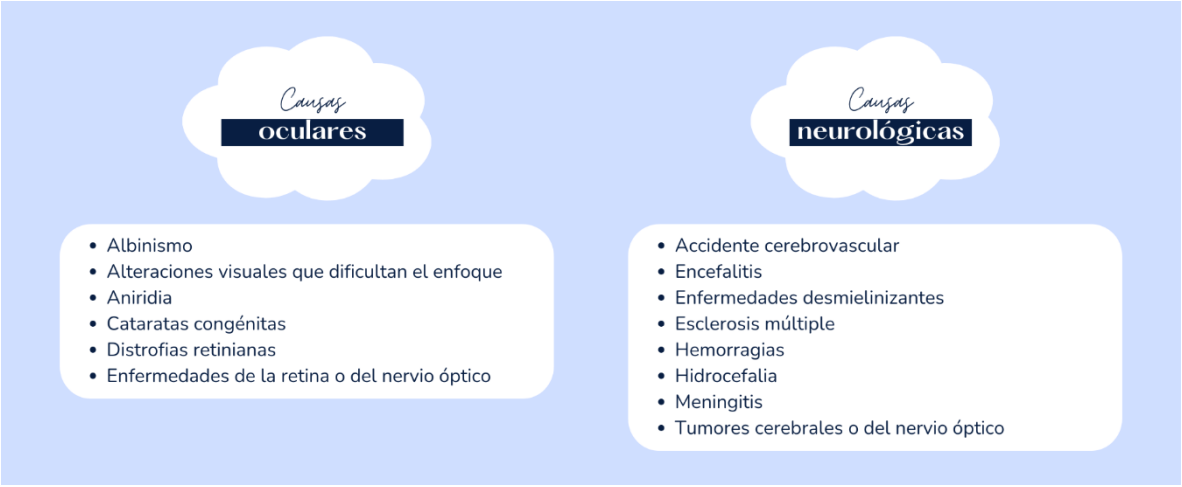

Figura 4. *Causas funcionales del nistagmo. Elaboración propia.*

Además, la literatura describe otros factores asociados a su aparición, como los genéticos, hereditarios, problemas del oído interno, por ejemplo, la enfermedad de *Ménière*, el uso de medicamentos (litio o anticonvulsivos) y el consumo de drogas o alcohol [26].

Existen diversos criterios o características para la clasificación del nistagmo, entre los que se incluyen el tiempo de aparición, las causas funcionales, la dirección y el

sentido del movimiento, así como parámetros dinámicos como la amplitud, la frecuencia y la velocidad de las fases. Estos criterios se describen a continuación.

Tiempo de aparición

Algunos autores lo identifican según el momento de la vida en que se presenta. Puede ser **congénito**, cuando aparece al nacer o en los primeros seis meses de vida, o **adquirido**, cuando se presenta después de ese periodo [25–29].

Causas funcionales

El nistagmo puede clasificarse como **fisiológico**, cuando es una respuesta natural a estímulos externos, o **patológico**, cuando resulta de una lesión o enfermedad, como se muestra en la Figura 5. También puede presentarse como **idiopático**, cuando su causa es desconocida [25, 27].

Figura 5. *Causas fisiológicas y patológicas del nistagmo. Elaboración propia.*

Dirección y sentido

El movimiento ocular puede presentarse de manera similar o distinta en ambos ojos [25, 27, 29].

Según su dirección, puede distinguirse como:

- **Horizontal:** con desplazamiento alternante hacia la derecha e izquierda
- **Vertical:** con desplazamiento hacia arriba y abajo

- **Rotatorio o torsional:** movimiento circular alrededor de su eje, en sentido horario o antihorario
- **Mixto:** combinación de los anteriores. Puede presentarse en forma oblicua, con un ángulo diagonal, o circular, cuando describe un movimiento completo (horizontal, vertical y torsional) en sentido horario o antihorario

En la Figura 6 se ilustran los diferentes tipos de movimiento.

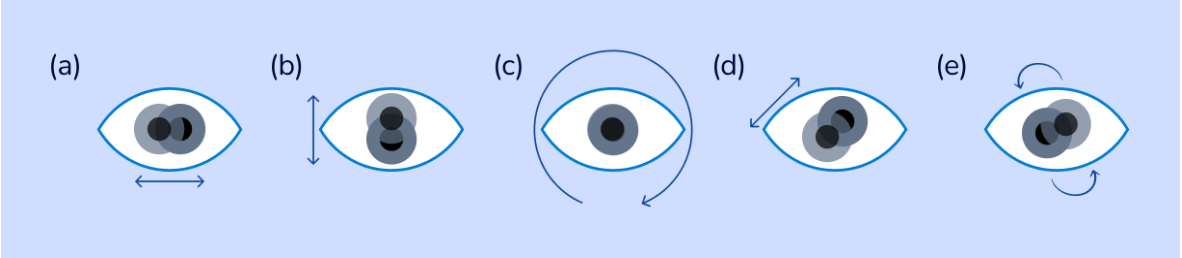

Figura 6. *Dirección y sentido del nistagmo: (a) horizontal, (b) vertical, (c) rotatorio o torsional, (d) oblícuo, (e) circular. Los ejemplos son ilustrativos, la dirección y el sentido pueden variar en últimos tres casos. Elaboración propia.*

Amplitud

La **amplitud** se refiere a la distancia o desplazamiento que realiza el ojo en cada movimiento. De acuerdo con distintos autores, existen diversos rangos para caracterizar el nistagmo [27].

Frecuencia

La **frecuencia** corresponde al número de oscilaciones o ciclos que realiza el ojo en una unidad de tiempo. Generalmente se mide en Hertz (Hz) cuando se considera el segundo (*ciclos/segundo*), aunque también puede expresarse en ciclos por minuto (*ciclos/minuto*), según lo señalado por diversos autores [25, 27, 29].

Velocidad de fases

La velocidad de fases describe la relación entre los movimientos lentos y rápidos del ojo, como se muestra en la Figura 7. Según lo señalado en la literatura [27, 29], el nistagmo puede agruparse en:

- **Pendular:** fases simétricas (ida y vuelta con la misma velocidad)
- **Resorte:** fases asimétricas
    - *Fase lenta:* desplazamiento progresivo en una dirección
    - *Fase rápida:* retorno veloz
- **Mixto:** combinación de ambas formas

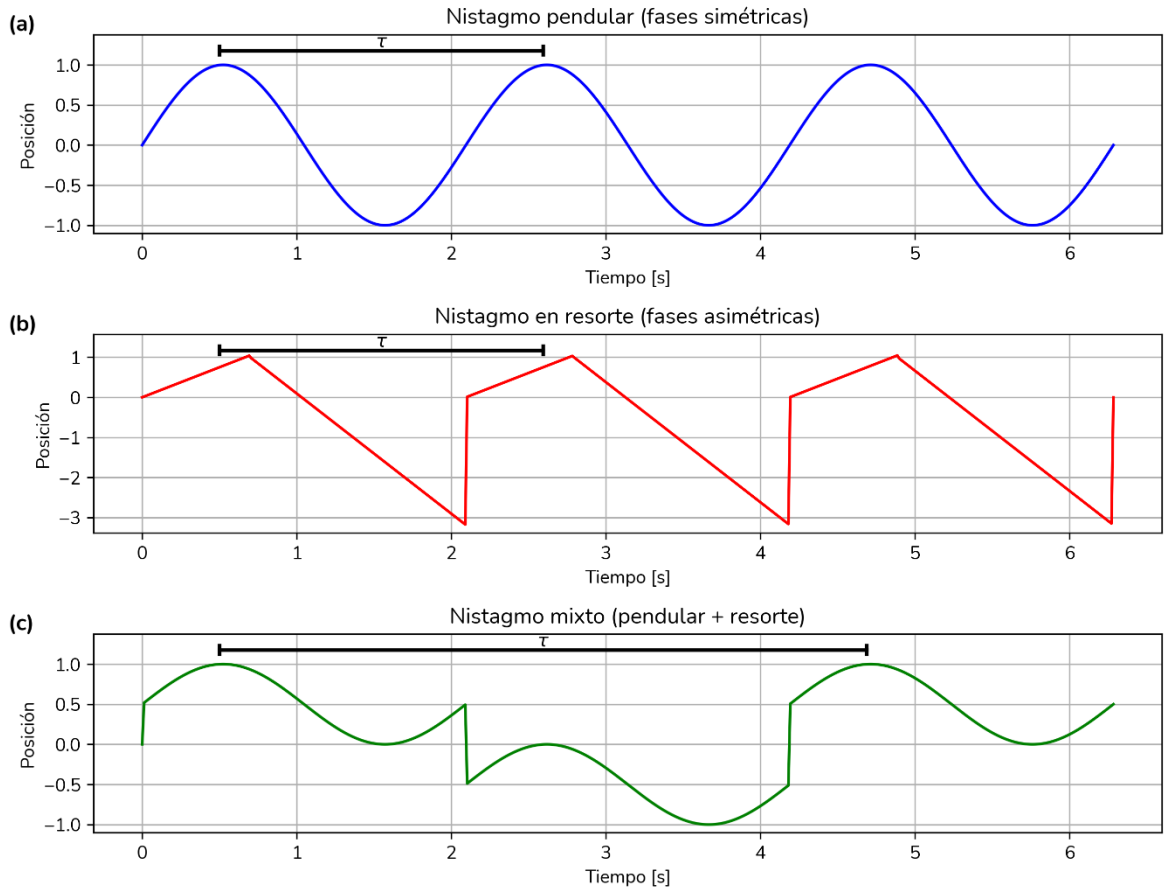

Figura 7. *Ejemplificación de formas de nistagmo: (a) pendular con fases simétricas, (b) resorte con fases asimétricas y (c) mixto, combinación de ambos patrones. Elaboración propia.*

Signos y síntomas

Además del movimiento ocular involuntario característico, se han descrito otros signos asociados al nistagmo, reportados en la literatura especializada. Entre los **signos oculares** destacan el estrabismo, que implica la desviación de uno o ambos ojos, y la tortícolis, inclinación o giro de la cabeza; mientras que, en el ámbito **neurológico**, pueden presentarse vómitos, pérdida del equilibrio, ataxia (deterioro de la coordinación) y parálisis de nervios oculomotores [27].

El nistagmo puede acompañarse de distintos síntomas según su origen. Entre los **síntomas oculares** se incluyen problemas visuales, fotofobia (sensibilidad a la luz), ambliopía u "ojo vago", y dificultad para ver en la oscuridad. A nivel **neurológico**, son comunes la oscilopsia (sensación de que el entorno se mueve), las náuseas y los mareos [25–27].

Diagnóstico

El nistagmo es un padecimiento visible, por lo que puede identificarse a simple vista observando el movimiento ocular. Para su caracterización se consideran parámetros como dirección y sentido, amplitud, frecuencia y velocidad [25].

Además, existen posiciones oculares de interés: el **punto cero o posición de bloqueo**, donde el nistagmo disminuye o desaparece, y la **zona neutra**, en la que el nistagmo en resorte cambia de dirección y se vuelve pendular o desaparece [25, 30].

La tortícolis es un signo asociado, ya que el paciente busca situar los ojos en la posición que minimiza el movimiento [25, 27].

El abordaje diagnóstico del nistagmo incluye un **examen ocular integral** que evalúa la función visual y permite descartar patologías asociadas. Este examen considera la historia médica, la agudeza visual, la evaluación de pupilas, la alineación y movimiento de los ojos, la presión ocular, así como la revisión de párpados, córnea, iris y cristalino para descartar cataratas o cicatrices. También puede incluir la dilatación pupilar para examinar la retina y el nervio óptico [31].

Además, pueden emplearse **pruebas clínicas** específicas para evaluar la respuesta ocular [26, 32], algunas de las cuales permiten registrar con precisión los movimientos oculares**:**

- Giro durante 30 segundos seguido de fijación en un punto para detectar la respuesta ocular.
- Tambor frente a los ojos: cilindro con perforaciones verticales y dibujo interior, o con franjas en blanco y negro o dibujos adicionales (optocinético).
- Electronistagmografía (ENG): registra movimientos oculares mediante electrodos que miden la actividad eléctrica de los músculos oculares.
- Videonistagmografía (VNG): utiliza gafas infrarrojas para registrar en tiempo real los movimientos oculares; análisis visual y preciso.
- Electrofisiológicas:
    - Electroretinograma (ERG): mide la respuesta eléctrica de la retina ante estímulos luminosos.
    - *Visually Evoked Potentials* (PEV) o *Visually Evoked Responses* (VER): registran la actividad cortical generada ante un estímulo visual.

El diagnóstico puede complementarse con **estudios de laboratorio** o **pruebas especializadas** que permiten identificar su origen y descartar otras patologías asociadas. Entre ellas se incluyen la tomografía computarizada (TC), la resonancia magnética (RM) y el análisis de sangre o pruebas genéticas [26, 29, 30, 32, 33].

El abordaje diagnóstico puede involucrar tanto oftalmólogos como neurólogos u otorrinolaringólogos [29, 34].

Tratamiento

El nistagmo no tiene cura, pero es posible mejorar la agudeza visual, disminuir la amplitud y la frecuencia del movimiento, así como corregir o reducir la tortícolis.

Entre las **alternativas médicas** utilizadas se incluyen el uso de fármacos, anteojos o lentes que mejoran la claridad visual y reducen la velocidad de los movimientos, así como prismas oculares. También pueden emplearse inyecciones, por ejemplo, de toxina botulínica en los músculos horizontales para relajarlos, aunque pueden provocar visión doble o caída de párpados. En algunos casos, el tratamiento se orienta a la resolución de la causa subyacente, como la suspensión del consumo de alcohol o drogas en el nistagmo adquirido [27, 29].

Asimismo, puede recurrirse a **cirugías** que reposicionan los músculos responsables del movimiento ocular, con lo que se reduce la necesidad de girar la cabeza [29].

1.1.5 Telemedicina

La **telemedicina** es un concepto vigente desde hace algunos años. No se limita al diagnóstico, tratamiento y seguimiento de pacientes; también facilita la conexión entre hospitales en tiempo real, lo que permite compartir ideas, presentar casos específicos o solicitar la colaboración inmediata de especialistas [35].

Durante la pandemia (punto clave), la telemedicina se consolidó como alternativa para brindar atención médica sin necesidad de acudir presencialmente a hospitales, reduciendo el riesgo de contagio y manteniendo el acceso a servicios básicos de salud. Su uso ha disminuido posteriormente, aunque se buscan soluciones para garantizar su implementación efectiva [36].

Categorías

La **teleconsulta** implica atención médica con interacción entre paciente y uno o más profesionales de la salud mediante Tecnologías de la Información y la Comunicación (TIC) [35]. Puede ser interactiva, cuando la comunicación es remota en tiempo real entre profesional y usuario, o no interactiva, cuando ocurre de forma asincrónica. La teleexperticia se da entre dos profesionales de la salud, donde uno atiende presencialmente y otro a distancia, también están incluidas las juntas médicas e interconsultas [37].

Por otro lado, la **teleatención** se especializa en evaluar y monitorear tratamientos, enfermedades o trastornos utilizando TIC [35]. El telemonitoreo corresponde a un método sincrónico o asincrónico donde personal de salud y usuario recopilan y transmiten datos clínicos a distancia [37].

Finalmente, la **telecirugía** consiste en un servicio de intervención quirúrgica en el que al menos un cirujano utiliza dispositivos teleelectrónicos o robóticos especializados para manipular herramientas quirúrgicas a distancia [35].

Beneficios

La telemedicina ofrece múltiples ventajas y ha demostrado aportaciones significativas en el ámbito de la salud pública y la atención médica remota [35, 38, 39]:

- Acceso a servicios en zonas rurales con recursos limitados o inexistentes
- Ahorro económico en traslados
- Evita el deterioro de pacientes en condiciones críticas durante el traslado
- Reducción del tiempo de atención y posibilidad de respuesta inmediata
- Disponibilidad de dispositivos de monitoreo domiciliario a bajo costo
- Disminución del riesgo de contagio de enfermedades
- Mayor accesibilidad a personas con movilidad reducida
- Ajuste y seguimiento continuo del tratamiento
- Reducción de la carga en servicios hospitalarios
- A nivel internacional, contribuye a reducir la desigualdad social y fortalecer los sistemas de salud

Retos

A pesar de sus aportaciones, la telemedicina enfrenta limitaciones y desafíos en su implementación y desarrollo [35–37, 39–42]:

- Factores organizacionales: falta de capacitación del personal
- Ausencia de protocolos claros, con poco control, validación y fiabilidad
- Carencia de equipo adecuado
- Dificultades tecnológicas
- Falta de experiencia del usuario en servicios virtuales
- Algunos dispositivos de monitoreo domiciliario tienen precios elevados
- Factores económicos
- No recomendada para el diagnóstico de enfermedades complejas que requieren exploración física
- Consideraciones éticas y legales: privacidad y seguridad

De no resolverse, podrían perderse los beneficios y acentuarse las limitaciones existentes [43]. Se recomienda priorizar su uso en atención primaria que no requiera intervención urgente, evaluando después la necesidad de atención presencial [36].

La oftalmología es una de las especialidades que ha incorporado la telemedicina, con resultados positivos [35, 40].

1.1.6 Teleoftalmología

La telemedicina ha mejorado el acceso a consultas especializadas, permitiendo diagnósticos, tratamientos y seguimientos más precisos. Dependiendo de la región, existen entre 30 y 52 especialistas por cada millón de habitantes, principalmente en zonas urbanas. La **teleoftalmología** rompe esta barrera geográfica, ofreciendo una alternativa accesible e innovadora que facilita la detección temprana de enfermedades, la referencia oportuna de urgencias y el manejo remoto de patologías no graves, garantizando seguimiento periódico [39].

Diversos artículos demuestran los beneficios en pacientes con enfermedades frecuentes como la retinopatía diabética y el glaucoma, confirmando su utilidad en el diagnóstico oportuno y un direccionamiento adecuado [37, 39].

Actualmente existen aplicaciones para realizar diagnósticos, en su mayoría sobre padecimientos estáticos [41]. Sin embargo, para una evaluación integral, en

algunos casos aún se requieren pruebas específicas realizadas por especialistas de la salud o estudios de laboratorio con equipo especializado [39].

1.2 Planteamiento del problema

El nistagmo es una afección dinámica que, de acuerdo con el especialista, generalmente constituye un signo de diversas enfermedades, con frecuente implicación neurológica. A pesar del tratamiento, pueden persistir secuelas, por lo que se busca detener su progresión y evitar la pérdida visual por completo. Tras su diagnóstico inicial, se determina el tipo presente mediante un diagnóstico topográfico para localizar el origen del problema, seguido de un diagnóstico etiológico para identificar su causa. En ausencia de una causa subyacente, se clasifica como nistagmo esencial, de carácter crónico.

Saber el tipo de nistagmo y la causa, si existe, es clave para tratar la enfermedad principal e intentar mejorar esta afección. Para caracterizarlo, sé que requiere principalmente de cuatro parámetros: dirección y sentido, amplitud, frecuencia y velocidad de fases.

Aunque el diagnóstico se realiza de manera presencial, su caracterización generalmente requiere estudios específicos, como los ya mencionados. En entornos de telemedicina, aumenta la complejidad desde el diagnóstico inicial, ya que, si bien es posible realizar parte del examen ocular de manera remota, existen limitaciones, entre las que destacan la calidad de la conexión y la disminución en la resolución de las imágenes o vídeos compartidos [39, 40]; así como los retos específicos compartidos por el especialista al diagnosticar nistagmo: encontrar una iluminación adecuada que ilumine el centro el ojo sin molestar al paciente, maniobras de seguimiento contemplando el campo visual y estímulos que tomen en cuenta el trayecto completo ocular, así como poder grabarlo.

La mayoría de las aplicaciones oftalmológicas actuales se enfocan en diagnósticos estáticos [41]. Al ser el nistagmo dinámico, las herramientas disponibles en línea resultan limitadas. Un ejemplo es el **tambor optocinético digital**, implementado en algunas aplicaciones móviles (*OptoDrum* o *OKN Drum Pro*), que permiten inducir el reflejo optocinético de manera remota, como vemos en la Figura 8. No obstante, estas soluciones se limitan a la simulación de la prueba y no incluyen un análisis automático de los resultados.

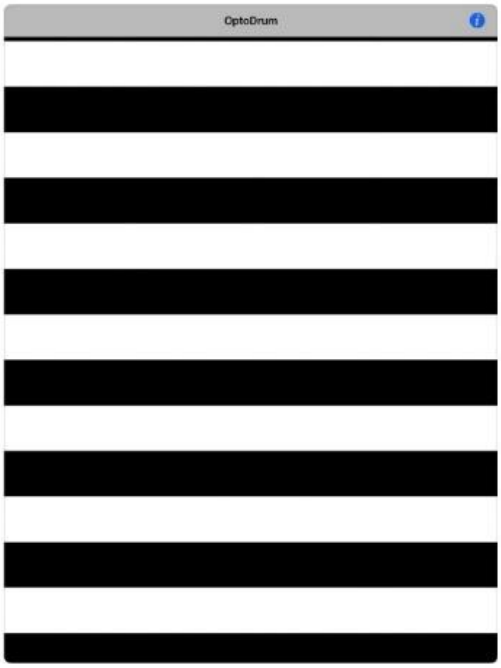

Figura 8. *Ejemplo de tambor optocinético digital* (OptoDrum, 2019). *El paciente sigue con la mirada las franjas en movimiento para inducir el reflejo optocinético que permite evaluar la presencia y características del nistagmo. Elaboración propia.*

Ante esto, surge la oportunidad de un nuevo proyecto que consta de dos partes:

1. Ayudar con la detección de nistagmo de forma objetiva.
2. Ayudar con la caracterización del movimiento.

Con la finalidad de documentar, llevar un seguimiento de pacientes personalizado y generar una nueva base de datos accesible. De pláticas con el especialista, se destaca que podría ayudar a identificar causas del nistagmo haciendo estudios estadísticos a grandes escalas.

1.3 Análisis del espacio de soluciones: Diagramas de polaridad

A partir de las limitaciones identificadas en los sistemas actuales para identificar, caracterizar y encontrar la causa del nistagmo, se realiza un análisis del espacio de soluciones mediante diagramas de polaridad. Para ello se definen las funciones que debe cumplir un sistema para la identificación, caracterización y encontrar la causa del nistagmo, así como las especificaciones de los sistemas actualmente utilizados (Figura 9).

Figura 9. *Funciones del sistema, especificaciones de sistemas actuales y atributos de valor identificados. Elaboración propia.*

Como referencia, se consideran los métodos empleados en la práctica clínica, tales como el examen ocular, tambor de nistagmo u optocinético, prueba de rotación, ENG, ERG, VNG, resonancia magnética (RM), tomografía computarizada (TC), pruebas genéticas, PEV/VER.

Las funciones del sistema se determinan con base en los requerimientos clínicos necesarios para realizar el proceso diagnóstico en todas sus etapas. Considerando los elementos indispensables para analizar el comportamiento del nistagmo y contribuir a la orientación terapéutica. Actualmente no existe un sistema que cubra de manera integral estos requerimientos, por lo que en la práctica se emplean combinaciones de estudios; sin embargo, esta fragmentación puede limitar la eficiencia y objetividad del proceso diagnóstico.

Las especificaciones técnicas se establecen a partir del análisis de las características observadas en los métodos clínicos actuales, considerando aspectos como la duración de las pruebas, el costo de los estudios, el tiempo de entrega de resultados y la necesidad de asistencia presencial especializada. A partir de este análisis, se definen las especificaciones deseables (resaltadas en azul en la Figura 9), alineadas con los requerimientos clínicos y los criterios de accesibilidad relevantes para el paciente. Asimismo, se contempla la posibilidad de implementación remota como un criterio de diseño orientado a ampliar la

accesibilidad y priorizar la obtención de resultados directos y objetivos que mejoren la eficiencia del proceso diagnóstico.

Con la información recopilada y los criterios previamente definidos, se elaboran diagramas de polaridad que permiten visualizar de manera comparativa las características más relevantes de los sistemas analizados. Esta representación facilita el contraste entre fortalezas y limitaciones de las soluciones existentes, así como la identificación de áreas de oportunidad para la propuesta desarrollada (Figura 10).

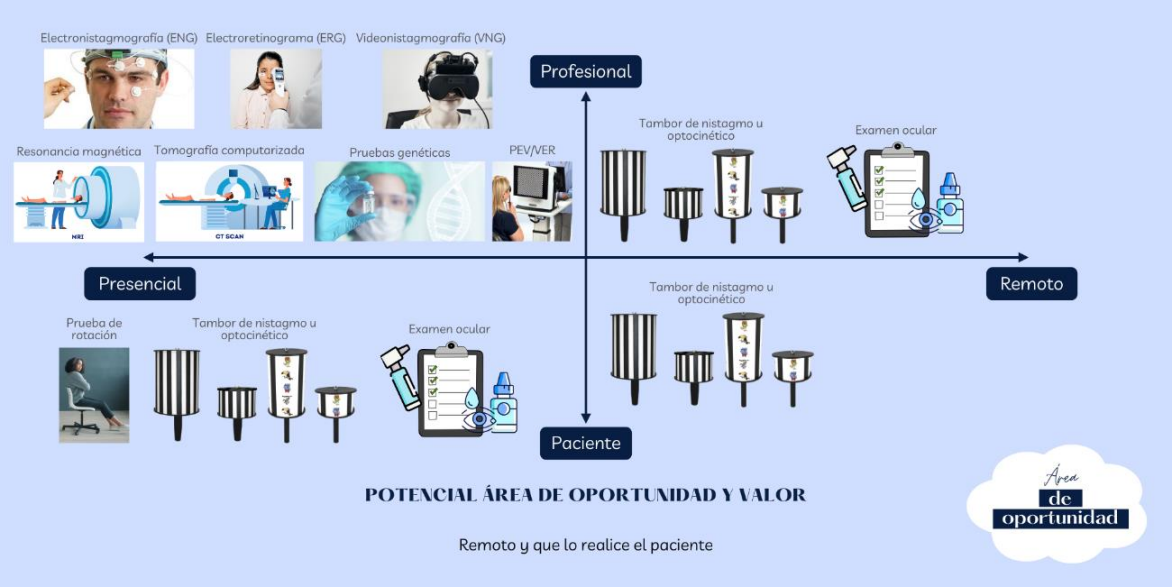

Figura 10. *Ejemplo de diagrama de polaridad de sistemas actuales utilizados en el diagnóstico de nistagmo. Elaboración propia.*

Posteriormente, al contrastar las especificaciones deseables con las limitaciones identificadas en los sistemas actuales, se determinaron los atributos de mayor valor para la propuesta de solución (Figura 9). Entre ellos, destacan la posibilidad de implementación remota y su ejecución por parte del propio paciente como elementos clave para ampliar la accesibilidad y reducir la dependencia de evaluación presencial especializada. De esta manera, se cubren las deficiencias detectadas y se aporta mayor valor en términos clínicos y prácticos, considerando que los resultados generados requieren siempre interpretación y validación médica especializada.

1.4 Justificación

Una alternativa para la detección objetiva del nistagmo contribuye al apoyo diagnóstico y al control oportuno, ya que una identificación temprana permite prevenir su progresión, atender sus efectos y mejorar la calidad de vida del paciente.

Los métodos actuales suelen depender de equipos especializados, personal capacitado o condiciones muy controladas, lo que limita su accesibilidad en ciertos contextos, tanto presenciales, como remotos.

A diferencia de *ConVNG* [44], el sistema propuesto en este trabajo busca una alternativa más ligera y accesible, sin requerir redes neuronales ni grandes volúmenes de datos para iniciar con la detección. Su enfoque se basa en métodos físicos y matemáticos para el análisis directo del movimiento, utilizando la Transformada Rápida de Fourier (FFT) para caracterizar la vibración ocular, permitiendo una evaluación controlada y adaptable a distintos entornos clínicos o experimentales, siempre de manera objetiva.

A diferencia de ConVNG [44], el sistema propuesto en este trabajo busca una alternativa más ligera y accesible, sin requerir redes neuronales ni grandes volúmenes de datos para iniciar con la detección, dado que los enfoques basados en aprendizaje profundo suelen requerir bases de datos extensas y especializadas, las cuales no son ampliamente accesibles en el caso del nistagmo. Su enfoque se basa en métodos físicos y matemáticos para el análisis directo del movimiento, utilizando la Transformada Rápida de Fourier (FFT) para caracterizar la vibración ocular, permitiendo una evaluación controlada y adaptable a distintos entornos clínicos o experimentales, siempre de manera objetiva.

Su implementación futura permitiría ampliar el acceso a herramientas de apoyo diagnóstico y facilitar el seguimiento y documentación objetiva del movimiento ocular en distintos entornos clínicos y experimentales.

1.5 Objetivo general

Evaluar una propuesta de diseño de un sistema para confirmar si el movimiento ocular es rítmico con el propósito de identificar nistagmo de manera objetiva.

1.5.1 Objetivos específicos

- Implementar el preprocesamiento y procesamiento de señales para detectar y documentar el movimiento ocular.
- Implementar dos métodos de análisis para la detección de movimiento ocular rítmico.
- Evaluar y comparar el desempeño de ambos métodos en la identificación de la ritmicidad ocular bajo las mismas condiciones de prueba.

1.6 Alcances

Este trabajo se centra en la primera etapa del proyecto, enfocada en determinar si el movimiento ocular presenta un comportamiento rítmico.

- No se analiza en tiempo real, solo vídeos pre-grabados.
- Se analiza un rostro a la vez.
- Se analiza uno o dos ojos (preferible).
- Como resultado se indica si el movimiento del ojo es rítmico o arrítmico.

1.7 Restricciones

Programa

- El lenguaje utilizado es Python.
- Se detectan iris y pupila para su seguimiento.

Vídeo

- El formato de vídeo debe ser MP4 o MOV.
- Duración mínima de 3 segundos.
- La cámara se mantiene fija.
- No se permite cambiar el zoom.
- Debe verse, como mínimo, desde las cejas hasta la nariz; máximo el rostro completo.
- La zona de los ojos debe estar libre de cabello u objetos.
- Iluminación estable, sin cambios de intensidad ni reflejos en los ojos.
- Ambiente tranquilo, libre de distracciones.

Usuario

- Debe mantener la mirada en un punto fijo.
- No debe moverse.
- No debe parpadear.
- No debe hablar.
- No debe gesticular ni hacer expresiones.

1.8 Requerimientos y especificaciones objetivo

Preproceso

1) El módulo convertir_video convertirá el vídeo a formato h264 si no se puede leer. La conversión se analizará a través de la codificación en formato [h264].

2) El módulo rotación rotará el vídeo para que se visualice como fue grabado en ángulos de 90°, 180° o 270°. La rotación se medirá a través de la orientación del vídeo en grados [°].

3) El módulo roi detectará ojos e iris en un solo rostro. La detección de ojos e iris se medirá a través de la generación de recuadros, del cálculo de centro en coordenadas cartesianas [px] y radio del iris en píxeles [px].

4) El módulo roi identificará si el modo es ambos ojos o solo uno. La identificación del modo se analizará a través del área relativa de los ojos detectados en píxeles cuadrados [px²], clasificando como binocular o monocular.

5) El módulo corregir_movimiento estabilizará el vídeo fijando la zona de ojos la mayor parte del tiempo posible. La estabilización del vídeo se medirá a través del desplazamiento corregido en píxeles [px].

6) El módulo seguimiento identificará la posición inicial del iris al inicio o en los primeros momentos del vídeo. La identificación de la posición inicial del iris se medirá a través de la distancia (coordenadas cartesianas) en píxeles [px] y en función del tiempo en segundos [s].

7) El módulo seguimiento seguirá la trayectoria del iris la mayor parte del tiempo posible. El seguimiento de la trayectoria del iris se medirá a través de la distancia

(coordenadas cartesianas) en píxeles [px] y en función del tiempo en segundos [s].

8) El módulo seguimiento estabilizará la trayectoria del iris corrigiendo el movimiento global de los ojos mediante puntos de referencia alrededor de la región ocular. La estabilización de la trayectoria del iris se medirá a través de las coordenadas corregidas en píxeles [px] y en función del tiempo en segundos [s].

Proceso

9) El módulo análisis determinará la distancia al punto inicial (d) [px] para todas las coordenadas (x, y). La determinación de la distancia d se calculará a través de la distancia al punto inicial en píxeles [px] y en función del tiempo en segundos [s].

10) El módulo análisis graficará la señal para todas las señales (x, y, d) en distancia [px] contra tiempo [s]. La gráfica de la señal se analizará a través de la trayectoria del iris: distancia en píxeles [px] y tiempo en segundos [s].

11) El módulo análisis seleccionará la señal principal que domine. La selección de la señal que domina (señal principal) se analizará a través del rango de movimiento en píxeles [px].

12) El módulo análisis normalizará la señal principal en un rango de 0-1. La normalización de la señal principal se calculará a través de la amplitud relativa en valores adimensionales [1].

13) El módulo análisis graficará la señal principal en amplitud adimensional contra tiempo [s]. La gráfica de la señal principal se analizará a través de la trayectoria de la señal principal normalizada en amplitud relativa en valores adimensionales [1] y tiempo en segundos [s].

14) El módulo análisis básico identificará las crestas de la señal normalizada con mayor valor en amplitud adimensional en función del tiempo [s]. La identificación de crestas se medirá a través de la señal normalizada en amplitud relativa en valores adimensionales [1] y tiempo en segundos [s].

15) El módulo análisis básico graficará las crestas con mayor valor en amplitud adimensional contra tiempo [s]. La gráfica de las crestas se analizará a través de

la trayectoria de la señal principal normalizada y la posición de las crestas en amplitud relativa en valores adimensionales [1] y tiempo en segundos [s].

16) El módulo análisis básico determinará los intervalos entre crestas según su duración en segundos [s]. La determinación de los intervalos entre crestas se calculará a través del tiempo transcurrido entre crestas en segundos [s].

17) El módulo análisis básico determinará la variabilidad de intervalos entre todas las crestas identificadas. La determinación de variabilidad de intervalos se calculará a través del coeficiente de variación: desviación estándar de intervalos entre promedio de intervalos en valores adimensionales [1].

18) El módulo análisis básico clasificará el movimiento ocular como rítmico si el coeficiente de variación es menor a 0.25. La clasificación de si el movimiento ocular es rítmico se analizará a través del coeficiente de variación en valores adimensionales [1].

19) El módulo análisis FFT transformará de la señal principal al dominio de la frecuencia [Hz]. La transformada de la señal principal al dominio de la frecuencia se calculará a través de la transformada rápida de Fourier en Hertz [Hz].

20) El módulo análisis FFT determinará la media del espectro de la señal principal. La determinación de la media del espectro se calculará a través del promedio de amplitudes en el dominio de la frecuencia en valores adimensionales [1].

21) El módulo análisis FFT determinará el espectro de frecuencias de la señal principal en amplitud adimensional en función de la frecuencia [Hz]. La determinación del espectro de frecuencias se calculará a través de la distribución de amplitudes en valores adimensionales [1] y frecuencia en Hertz [Hz].

22) El módulo análisis FFT normalizará el espectro de frecuencias de la señal principal en un rango de amplitud de 0-1 en función de la frecuencia [Hz]. La normalización del espectro de frecuencias se calculará a través del rango relativo de amplitudes en valores adimensionales [1] y frecuencia en Hertz [Hz].

23) El módulo análisis FFT graficará el espectro de frecuencias de la señal principal en amplitud adimensional en función de la frecuencia [Hz]. La gráfica del

espectro de frecuencias se analizará a través de amplitud en valores adimensionales [1] y frecuencia en Hertz [Hz].

24) El módulo análisis FFT graficará el espectro de frecuencias de la señal principal normalizada en un rango de amplitud adimensional en función de la frecuencia [Hz]. La gráfica del espectro de frecuencias se analizará a través del espectro de frecuencias normalizado en amplitud relativa en valores adimensionales [1] y frecuencia en Hertz [Hz].

25) El módulo análisis FFT identificará la frecuencia dominante en amplitud y frecuencia [Hz]. La identificación de la frecuencia principal se medirá a través de la cresta de máxima amplitud en valores adimensionales [1] y frecuencia en Hertz [Hz].

26) El módulo análisis FFT clasificará el movimiento ocular como rítmico si la amplitud de la frecuencia dominante entre la media del espectro es mayor a 5. La clasificación de si el movimiento ocular es rítmico se analizará a través de la relación amplitud dominante entre la media del espectro en valores adimensionales [1].

Los resultados se almacenan en diferentes formatos según su tipo: los vídeos generados en archivos MP4, los datos en CSV, las gráficas en PNG, y el registro final del preproceso y proceso con resultados en un archivo de texto LOG.

Los requerimientos y especificaciones previamente definidos se integraron en una matriz de requerimientos–especificaciones con el objetivo de visualizar la relación entre cada función del sistema y la métrica asociada que permite evaluar su cumplimiento. Esta matriz permite identificar correspondencias entre ambos elementos.

		Especificaciones							
		1	2	3	4	5	6	7	8
	Requerimientos	Convertir a formato h264	Girar vídeo en ángulos de 90 [°]	Medir y calcular en pixeles [px] o identificar coordenadas	Calcular área en pixeles cuadrados [px2]	Determinar modo monocular o binocular	En función del tiempo en segundos [s] o calcularlo	Transmormar a valor adimencional [1] o evaluarlo	En función de la frecuencia en Hertz [Hz] o calcularla
1	Asegurar que el vídeo es legible	•							
2	Detectar rotación incial del vídeo		•						
3	Detectar ojo e iris			•					
4	Identificar uno o dos ojos				•	•			
5	Estabilizar vídeo fijando la zona de ojos			•					
6	Identificar posición inicial del iris			•			•		
7	Seguir la trayectoria del iris			•			•		
8	Estabilizar trayectoria del iris con respecto a las zonas de anclaje			•			•		
9	Determinar la distancia al punto inicial (d) para cada coordenada (x, y)			•			•		
10	Gráficar la trayectoria ocular para las tres señales (x, y y d)			•			•		
11	Seleccionar señal principal			•					
12	Normalizar señal principal en rango 0-1							•	
13	Graficar señal principal normalizada						•	•	
14	Identificar crestas de la señal principal normalizada						•	•	
15	Graficar crestas con mayor amplitud						•	•	
16	Determinarduración de intervalos entre crestas						•		
17	Determinar variabilidad de intervalos entre crestas							•	
18	Clasificar el movimiento ocular rítmico con Coeficiente de Variación (CV<0.25)							•	
19	Transformar la señal principal al dominio de la frecuencia								•
20	Determinar la media del espectro de la señal principal							•	
21	Determinar el espectro de frecuencias de la señal principal							•	•
22	Normalizar el espectro de frecuencias de la señal principal en rango 0-1							•	•
23	Graficar el espectro de frecuencias de la señal principal							•	•
24	Graficar el espectro de frecuencias de la señal principal normalizada							•	•
25	Identificar la frecuencia dominante							•	•
26	Clasificar el movimiento ocular rítmico con relación amplitud/media espectro (amplitud>5)							•	

Figura 11. *Matriz de requerimientos-especificaciones. Elaboración propia.*

CAPÍTULO 2

MARCO TEÓRICO

Capítulo 1 Marco teórico

2.1 Señales

Una señal transmite información sobre el estado o comportamiento de un sistema físico y se representa como función de una o más variables independientes, como tiempo o espacio. Puede ser **continua**, cuando está definida en todos los instantes del tiempo, o **discreta**, cuando se define solo en ciertos instantes.

La diferencia entre señal **analógica** y **digital** se establece en que la primera es continua tanto en tiempo como en amplitud, mientras que la segunda es discreta en ambas dimensiones y se representa como una secuencia de números [45].

Sistemas de procesamiento digitales

Un sistema digital es aquel en el que la entrada y la salida son señales digitales. El procesamiento digital se ocupa de la transformación de señales discretas en amplitud y tiempo mediante algoritmos. Matemáticamente, un sistema de tiempo discreto transforma una secuencia de entrada $x[n]$ en una secuencia de salida $y[n]$:

$$y[n] = T\{x[n]\} \quad (1)$$

Las señales discretas pueden provenir del muestreo de una señal continua o generarse directamente en un proceso discreto. Aunque sistemas de tiempo continuo y discreto comparten propiedades, los segundos requieren su propio marco. Estos sistemas permiten simular procesos analógicos o realizar transformaciones imposibles con hardware continuo, ofreciendo flexibilidad para aplicaciones avanzadas [45].

Muestreo y frecuencia de muestreo (*Nyquist*)

El muestreo periódico consiste en registrar una señal continua en intervalos regulares de tiempo T, generando una secuencia discreta

$$x[n] = x_c(nT) \quad (2)$$

x_c señal continua original

T período de muestreo

$f_s = \frac{1}{T}$ frecuencia de muestreo

Este proceso se implementa en la práctica con un convertidor Analógico-Digital (A/D) aunque surgen limitaciones como la cuantización y la tasa de muestreo disponible. Una señal continua no siempre puede reconstruirse solo a partir de sus muestras, ya que diferentes señales pueden producir la misma secuencia; para evitar esta ambigüedad es necesario imponer restricciones.

En el dominio de la frecuencia, el muestreo de una señal continua puede verse como la multiplicación de la señal por un tren de impulsos, generando una secuencia discreta. Este proceso puede analizarse tanto en el dominio del tiempo como en el de la frecuencia. En este último, el espectro de la señal continua se repite periódicamente en múltiplos de la frecuencia de muestreo Ω_S [45].

Si la frecuencia de muestreo es suficientemente alta ($\Omega_S > 2\Omega_N$, donde $\Omega_s = 2\pi f_s$ es la frecuencia angular de muestreo y $\Omega_N = 2\pi f_N$ corresponde a la frecuencia angular máxima de la señal), los espectros no se solapan y la señal puede reconstruirse exactamente aplicando un filtro pasa-bajas ideal.

Si la condición no se cumple ($\Omega_S < 2\Omega_N$), los espectros se superponen, generando ***aliasing***, es decir, la aparición de componentes de baja frecuencia que no existen en la señal original, lo que impide su correcta reconstrucción.

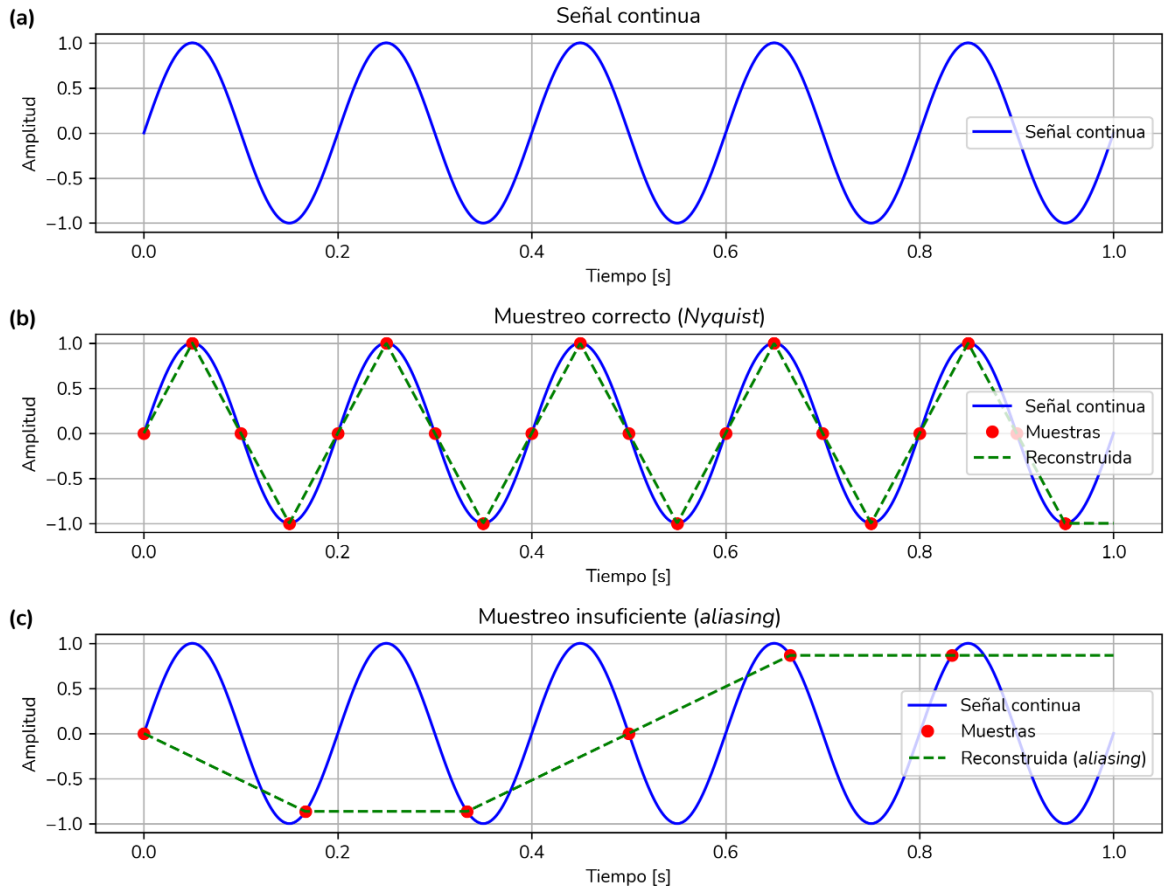

Figura 12. *Ejemplo de muestreo de una señal: (a) señal original, (b) muestreo suficiente cumpliendo el criterio de Nyquist, (c) muestreo insuficiente que produce aliasing. Elaboración propia.*

El **Teorema de Muestreo de *Nyquist-Shannon***, establece que la frecuencia de muestreo debe ser al menos el doble de la frecuencia más alta de la señal para evitar aliasing. Si se cumple el criterio de *Nyquist* ($\Omega_S > 2\Omega_N$), una señal continúa limitada en banda puede reconstruirse exactamente a partir de sus muestras [46].

Esto se logra mediante la **interpolación sinc**, donde sinc(·) es la función seno cardinal, definida como $\mathrm{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$ y cada muestra $x[n]$ se expande con la función de interpolación $h_r(t)$, la cual corresponde a la función sinc evaluada en $\frac{t}{T}$

$$h_r(t) = \frac{\sin\frac{\pi t}{T}}{\frac{\pi t}{T}} \qquad (3)$$

y la suma de todas ellas permite reproducir la señal original $x_c(t)$.

Este principio es la base de los convertidores Digital-Analógico (D/A), que permiten obtener nuevamente la señal continua a partir de su representación discreta sin pérdida de información (si no hay *aliasing*) [46].

2.1.1 Señales biomédicas

Las señales biomédicas surgen de los procesos fisiológicos del cuerpo (cardíacos, nerviosos, musculoesqueléticos, entre otros) y reflejan su estado o alteraciones. Pueden ser eléctricas, bioquímicas o físicas, y permiten detectar tanto funciones normales como procesos patológicos. Representadas como funciones en el tiempo, son fundamentales para el análisis y monitoreo clínico. Ejemplos representativos: electrocardiograma (ECG), electroencefalograma (EEG) y electromiograma (EMG).

Los principales objetivos del análisis de señales biomédicas son obtener información de fenómenos fisiológicos, apoyar en el diagnóstico de patologías, facilitar monitoreo continuo o periódico, realizar terapia y control mediante retroalimentación, y evaluar objetivamente desempeño o tratamientos.

El diagnóstico clínico depende de la interpretación del especialista; las nuevas tecnologías permiten complementarlo con métodos automáticos y objetivos [46].

Naturaleza fisiológica y ruido inherente

Los registros de señales biomédicas pueden verse afectados por ruido, artefactos y limitaciones técnicas. Estos elementos pueden ocultar o distorsionar la información fisiológica relevante, lo que hace necesario aplicar métodos de preprocesamiento y corrección antes del análisis. Ejemplos comunes de artefactos: actividad muscular, interferencia de la red eléctrica y movimiento del paciente [46].

Tipos de ruido

El **ruido aleatorio** es impredecible, como el ruido térmico o electrónico (se origina en los componentes del sistema), que afecta a todos los sistemas de adquisición. El **ruido estructurado** proviene de fuentes repetitivas o periódicas, como la interferencia de la red eléctrica (proviene del entorno externo) 50/60 Hz. Por último, la **interferencia fisiológica** corresponde a señales propias del cuerpo que se superponen a la señal de interés, como el movimiento ocular en un EEG o la actividad muscular en un ECG.

El Procesamiento Digital de Señales (DSP) es una disciplina con múltiples aplicaciones, incluida la medicina. Su desarrollo se consolidó con la llegada de los microprocesadores y algoritmos como la Transformada Rápida de Fourier (FFT), fundamentales en el análisis de señales biomédicas [45].

2.2 Preprocesamiento

El preprocesamiento busca mejorar la calidad de las señales antes de su análisis, eliminando ruido y tendencia, y adaptando la señal a condiciones comparables [46].

Los procesos **estacionarios** mantienen propiedades estadísticas constantes en el tiempo, mientras que los **no estacionarios** presentan variaciones temporales, algo frecuente en biomedicina. Los **cicloestacionarios** muestran patrones periódicos, como ocurre en la respiración.

2.2.1 Filtrado

Los métodos de filtrado permiten reducir interferencias y resaltar la información de interés. La selección depende del tipo de señal, el ruido presente y el objetivo del análisis.

En el dominio del tiempo, se utilizan métodos como la **media móvil**, que atenúa fluctuaciones lentas o ruido de alta frecuencia; los **filtros basados en derivadas**, que eliminan tendencias o cambios rápidos; y el **promedio sincronizado**, útil en señales periódicas, pues alinea varias repeticiones y las promedia [46].

En el dominio de la frecuencia, se aplican filtros **pasa-bajos, pasa-altos** o **pasa-banda** para eliminar componentes no deseados según el rango de frecuencia, además de filtros ***Wiener***, **no lineales** y **adaptativos**, que ajustan sus coeficientes en tiempo real o se adaptan a la estadística del ruido [45, 46].

2.2.2 Normalización (min–max, media–varianza)

La normalización ajusta las señales a una escala común, reduciendo variabilidad entre sujetos y condiciones de registro. En aplicaciones oculares, se ha propuesto normalizar señales ópticas para minimizar diferencias debidas a reflectividad, tamaño pupilar o posición ocular [47].

Los métodos más utilizados son **min–max**, que reescala valores a un rango fijo, por ejemplo [0,1], y **media–varianza (*z-score*)**, que centra la señal en media cero y varianza unitaria, permitiendo comparar amplitudes independientemente de la escala original.

2.3 Procesamiento

El procesamiento de señales biomédicas permite identificar eventos fisiológicos de interés, caracterizar la ritmicidad de los movimientos y, en aplicaciones clínicas, aportar criterios objetivos para el diagnóstico asistido por computadora [46].

2.3.1 Detección de picos

La detección de eventos es fundamental para localizar fenómenos clínicos relevantes. Estos suelen manifestarse como picos o transitorios en la señal [46].

Los principales métodos incluyen el uso de **derivadas**, que permiten identificar cambios bruscos en la pendiente para localizar máximos o mínimos significativos; **filtros adaptativos**, que resaltan componentes de interés y reducen ruido; **umbral y ventanas temporales**, que aplican límites para discriminar eventos reales de fluctuaciones; **correlación cruzada o *matched filters***, que comparan la señal con una forma de onda esperada; y el **análisis morfológico**, que estudia la forma de onda para detectar eventos clínicos relevantes [45, 46].

Intervalos y variabilidad

Tras detectar picos, se analizan intervalos y variaciones entre ellos.

Las métricas comunes incluyen los intervalos entre picos sucesivos, la amplitud relativa de las ondas y la duración de los ciclos. Entre los indicadores adicionales se consideran el número de cruces por cero, el análisis de derivadas y la variabilidad de amplitud y frecuencia.

Estas métricas permiten distinguir entre movimientos rítmicos (intervalos regulares) e irregulares (alta variación).

Análisis fractal y no lineal

Para señales complejas se aplican métodos no lineales como la **dimensión fractal**, que cuantifica la irregularidad de la señal, y la **entropía**, que mide el grado de desorden o complejidad.

Estos enfoques complementan los métodos clásicos y son útiles cuando los patrones patológicos no pueden diferenciarse solo con técnicas lineales.

En conjunto, la detección de picos y el análisis de intervalos y variabilidad constituyen una etapa esencial en el estudio de señales biomédicas. Estas herramientas permiten identificar ritmicidad, caracterizar la complejidad de los movimientos y, en aplicaciones clínicas, aportar criterios objetivos para el diagnóstico asistido por computadora [46].

2.3.2 Espectro de frecuencias

El análisis espectral permite estudiar señales discretas en el dominio de la frecuencia mediante la **Transformada Discreta de Fourier (DFT)** y su versión optimizada, **la Transformada Rápida de Fourier (FFT)**. Estas herramientas son fundamentales para caracterizar la periodicidad y los componentes dominantes en señales biomédicas, son la base en procesamiento de audio e imagen y establecer criterios objetivos para el diagnóstico asistido por computadora [45].

Transformadas de Fourier

Para una señal de N muestras, la DFT se define como:

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{\frac{-j2\pi kn}{N}}, k = 0,1, \dots, N-1 \quad (4)$$

Su inversa

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[n] e^{\frac{j2\pi kn}{N}} \quad (5)$$

La DFT establece una correspondencia entre el dominio temporal y el frecuencial; N muestras en el tiempo se corresponden con N coeficientes espectrales igualmente espaciados en frecuencia.

Puede interpretarse como la descomposición de la señal en una suma de senos y cosenos discretos permitiendo analizar la distribución de energía en distintas frecuencias.

El cálculo directo de la DFT requiere N^2 operaciones, lo que resulta poco eficiente para secuencias largas. La FFT reduce esta complejidad a $N \log_2 N$ al descomponer el cálculo en etapas sucesivas más simples, reutilizando resultados intermedios y evitando operaciones redundantes mediante simetrías. Esta reducción del costo

computacional hace viable el análisis de señales largas o de grandes volúmenes de datos en tiempos compatibles con aplicaciones prácticas.

En la práctica, la información obtenida en el dominio de la frecuencia depende fuertemente de la duración de la señal analizada y de la frecuencia de muestreo. La resolución espectral está determinada por $\Delta f = f_s/N$, por lo que señales cortas o muestreadas a baja frecuencia pueden dificultar la separación de componentes cercanas en frecuencia. Además, el análisis de señales finitas introduce efectos de fuga espectral (*spectral leakage*), que pueden distorsionar la estimación de la energía en frecuencia y requieren el uso de ventanas adecuadas.

Las **señales periódicas** muestran picos definidos en frecuencias fundamentales y sus armónicos, mientras que las **no periódicas** distribuyen su energía de forma más ancha y difusa. El **filtrado en frecuencia** permite eliminar bandas de ruido mediante multiplicación selectiva y transformación inversa. La **Densidad Espectral de Potencia (PSD)** cuantifica cómo se distribuye la energía en la señal, siendo un criterio clave en el análisis biomédico.

Los principales algoritmos son ***Radix-2* en tiempo (DIT)**, que divide la secuencia de entrada en pares (muestras pares e impares) aplicando la DFT a cada subconjunto y combinando los resultados, y ***Radix-2* en frecuencia (DIF)**, que divide los coeficientes espectrales en dos grupos reorganizando los cálculos de salida. Se representan mediante estructuras tipo mariposa (*butterfly*). La FFT es una factorización de la matriz DFT en productos de matrices más simples y constituye el núcleo de la mayoría de sistemas modernos de análisis de señales por su eficiencia en tiempo real [45].

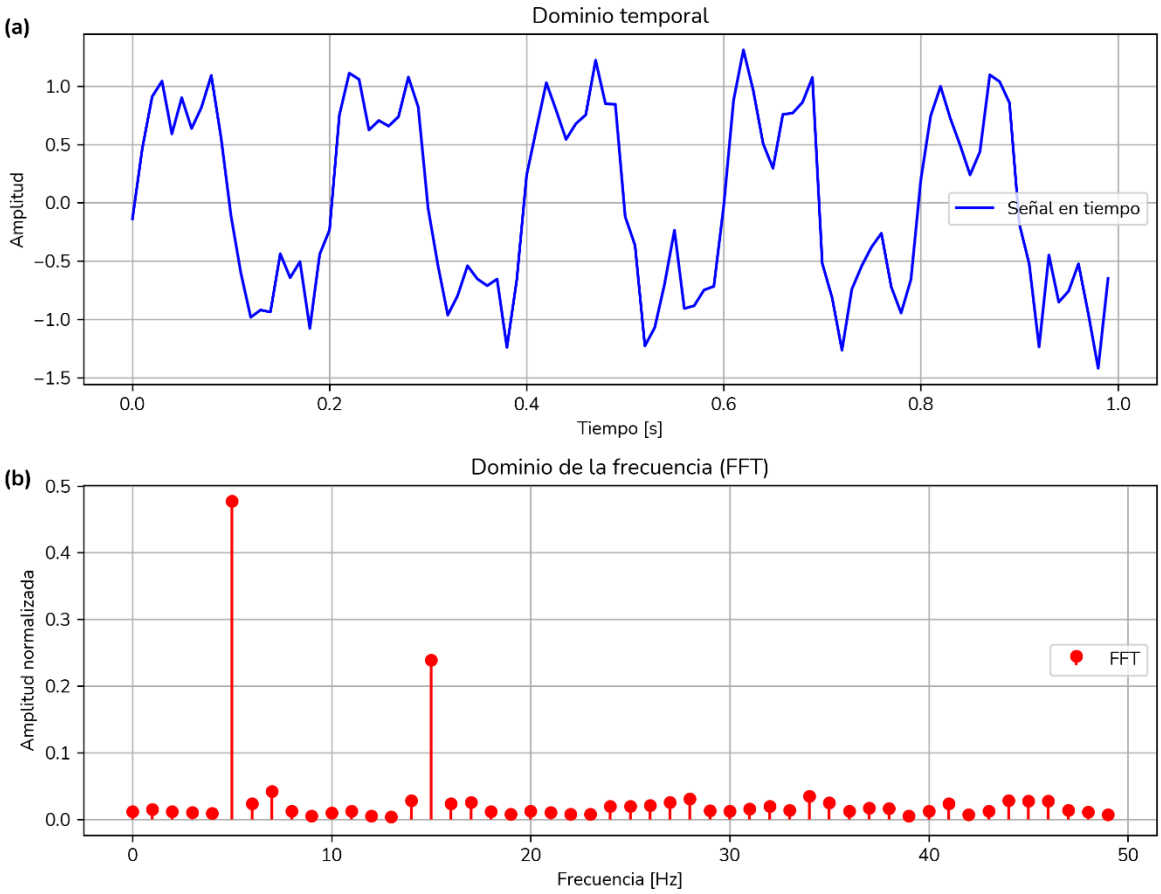

Figura 13. *Representación de una señal: (a) en el dominio temporal y (b) su espectro de frecuencias mediante la FFT. Elaboración propia.*

2.3.3 Señales biomédicas

Muchas señales fisiológicas son **periódicas** o **cuasiperiódicas**. La Transformada de Fourier revela las frecuencias presentes y sus amplitudes. Una señal periódica presenta un espectro discreto (líneas en frecuencias específicas), mientras que una señal no periódica muestra un espectro continuo [46].

Además, se emplean distintos métodos de análisis, como el **periodograma clásico** que es un método sencillo basado en la FFT, aunque limitado por fugas espectrales y resolución restringida. Las **ventanas** se utilizan para reducir los efectos indeseados de la señal finita. Los métodos **paramétricos (AR, MA, ARMA)** generan espectros más suaves, útiles en registros cortos o ruidosos.

La resolución espectral depende de la duración de la señal y de la frecuencia de muestreo. Si no se cumple el criterio de *Nyquist*, aparece *aliasing* (distorsión del

espectro). Además, el ruido fisiológico y los artefactos pueden enmascarar las frecuencias de interés, lo que exige un filtrado o preprocesamiento adecuado.

Aplicaciones

El **ECG** presenta una frecuencia dominante asociada a la frecuencia cardíaca y los armónicos describen la morfología del latido. En el **EEG**, las bandas alfa, beta, delta y theta permiten identificar estados fisiológicos o patológicos.

El **EMG** muestra un espectro ancho que refleja la complejidad de la contracción muscular y la fatiga, mientras que las señales oculares permiten analizar oscilaciones periódicas como vibraciones de baja frecuencia, útiles para caracterizar movimientos rítmicos.

2.4 Vibraciones

La vibración es el movimiento oscilatorio de un sistema respecto a una posición de equilibrio. Se caracteriza por parámetros como amplitud, frecuencia, periodo y amortiguamiento. Puede clasificarse como **libre**, cuando ocurre tras una perturbación en las condiciones iniciales del sistema; **forzada**, cuando es causada por una excitación externa; **determinística o aleatoria**, según la previsibilidad del estímulo.

El análisis de vibraciones permite predecir la respuesta de sistemas físicos y constituye la base para estudiar estabilidad, resonancia y transferencia de energía, con aplicaciones desde la ingeniería mecánica hasta la biomedicina [48].

2.4.1 Modelado básico

El modelo fundamental es el sistema masa–rigidez–amortiguamiento (*Figura 14*) [48, 49].

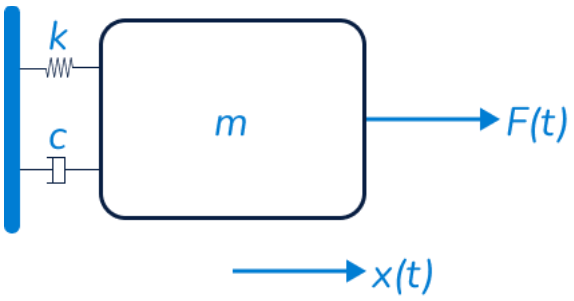

Figura 14. *Modelo masa–rigidez–amortiguamiento. Elaboración propia a partir de la literatura revisada.*

El comportamiento del sistema se describe mediante la ecuación diferencial:

$$m\ddot{u} + c\dot{u} + ku = F(t) \tag{6}$$

m masa: propiedad inductiva (inercial)

c amortiguador: propiedad resistiva (disipación)

k rigidez: propiedad asociada a la capacitiva (almacenamiento)

$\ddot{u}$ aceleración

$\dot{u}$ velocidad

u desplazamiento

$F(t)$ fuerza externa aplicada

Vibración libre (SDOF)

Su comportamiento depende de la frecuencia natural del sistema y del nivel de amortiguamiento [48, 49].

Cuando $F(t) = 0$:

$$m\ddot{u} + c\dot{u} + ku = 0 \tag{7}$$

Donde

ω_n frecuencia natural

ζ razón de amortiguamiento

El sistema **no amortiguado** oscila indefinidamente con frecuencia natural $\omega_n = \sqrt{\frac{k}{m}}$. En el caso **amortiguado subcrítico** ($\zeta < 1$) la oscilación ocurre con frecuencia amortiguada $\omega_d = \omega_n\sqrt{1-\zeta^2}$. Por su parte, los sistemas **críticos** ($\zeta = 1$) **y sobreamortiguados** ($\zeta > 1$) no presentan oscilaciones, sino que retornan al equilibrio con distinta velocidad.

El decremento logarítmico cuantifica la reducción de amplitud ciclo a ciclo en sistemas amortiguados.

Vibración forzada

La respuesta total se compone de estado: uno **transitorio**, que decae con el tiempo, y otro **estacionario**, que persiste mientras la fuerza actúe [48, 49].

Cuando $F(t) \neq 0$, asumiendo excitación armónica

$$m\ddot{u} + c\dot{u} + ku = F_0 \sin(\omega t) \tag{8}$$

La salida también es sinusoidal, con la misma frecuencia ω, pero distinta amplitud y fase. En este contexto, se define el **factor de magnificación** (o de ganancia de amplitud) que es la razón entre la amplitud de la respuesta en régimen estacionario y la amplitud estática producida por la misma fuerza, y el **ángulo de fase** que es el desfase entre la excitación armónica y la respuesta del sistema en régimen estacionario [48, 49].

Resonancia

Condición dinámica de estabilidad marginal, donde la frecuencia de excitación coincide con la natural ($\omega \approx \omega_n$), se caracteriza por que la amplitud crece significativamente con el tiempo. El amortiguamiento controla, pero no elimina el fenómeno.

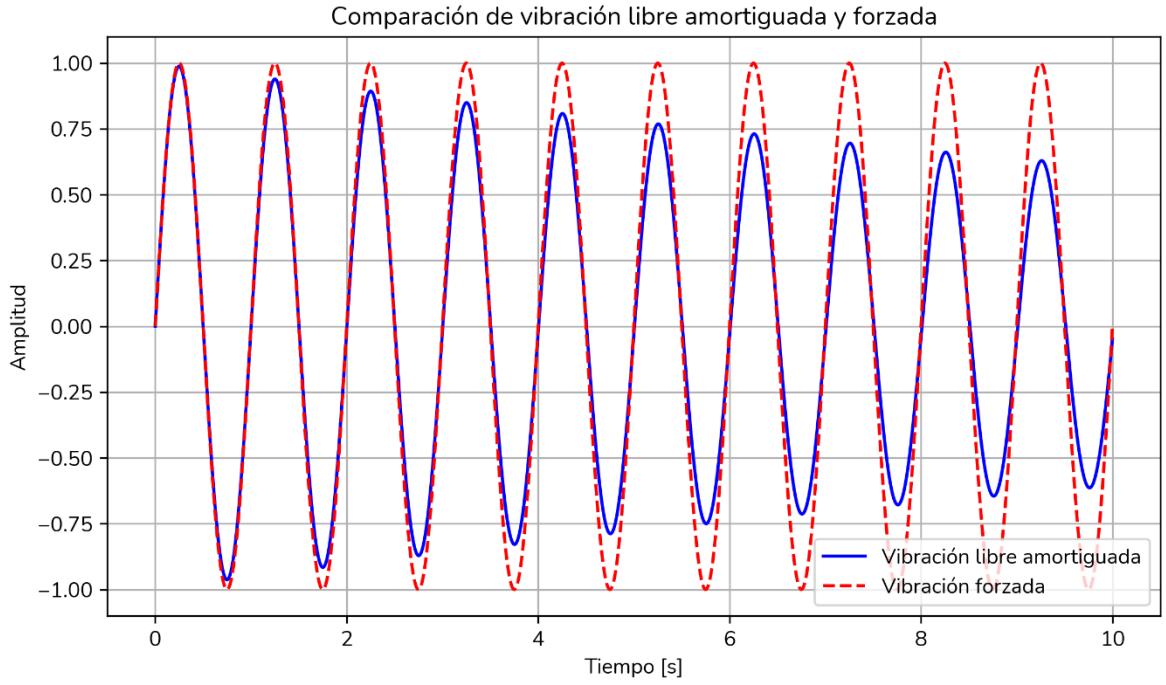

Figura 15. *Comparación de vibración libre y vibración forzada. Elaboración propia.*

Las **fuerzas periódicas complejas** se analizan como suma de senos y cosenos (series de Fourier), mientras que, en **impulsos o escalones**, la salida puede calcularse mediante la función de respuesta al impulso o integrales de convolución.

2.4.2 Sistemas de múltiples grados de libertad (MDOF)

En sistemas con varios grados de libertad (matricialmente) [48, 49]:

$$\{M\}\{\ddot{u}\} + \{C\}\{\dot{u}\} + \{K\}\{u\} = \{F(t)\} \quad (9)$$

$\{M\}$ matiz de masa

$\{C\}$ matriz de amortiguamiento

$\{K\}$ matriz de rigidez

$\{\ddot{u}\}$ vector de aceleración

$\{\dot{u}\}$ vector de velocidad

$\{u\}$ vector de desplazamiento

$\{F(t)\}$ vector de fuerzas externas

Estos sistemas poseen varias **frecuencias naturales** (ω_i) y **formas modales** (ϕ_i); cada modo describe una forma espacial de movimiento. Además, las formas modales (*Eigen* vectores) son normales entre sí.

2.4.3 Analogía con nistagmo

Los conceptos de vibraciones mecánicas se extienden al análisis de movimientos oculares rítmicos [48, 49]:

- El movimiento ocular rítmico puede modelarse como una vibración periódica.
- La frecuencia natural equivale a la frecuencia dominante detectada en FFT.
- El amortiguamiento se asocia a la atenuación fisiológica de la señal (ej. disminución de amplitud).
- La resonancia se relaciona con estímulos que amplifican el movimiento ocular.

2.5 Ritmo en oftalmología

Los movimientos oculares rítmicos pueden describirse como señales periódicas representadas mediante ondas, donde la repetición de picos y valles refleja la ritmicidad en el tiempo [46].

El análisis del ritmo se basa en la relación entre el patrón temporal y su contenido espectral. En el **dominio temporal**, la detección de picos facilita el cálculo de intervalos y la estimación de la variabilidad, mientras que en el **dominio de la frecuencia**, la FFT permite identificar componentes dominantes que caracterizan la periodicidad de la señal [46].

Estas técnicas se aplican en teleoftalmología, mediante dispositivos modernos que registran y analizan los movimientos oculares de forma remota. Esto habilita diagnósticos más accesibles y objetivos basados en el ritmo de la señal ocular [46].

2.5.1 Herramientas computacionales

Algunos autores destacan el uso de *MATLAB* y *Python*; este último, con librerías como *SciPy* [45, 46], es de código abierto y cuenta con bibliotecas especializadas para visualización de datos y procesamiento matricial.

Estas herramientas permiten implementar algoritmos de filtrado, detección de picos y transformada rápida de Fourier (FFT), así como simular, procesar y visualizar señales en entornos digitales.

CAPÍTULO 3

METODOLOGÍA

Capítulo 3 Metodología

3.1 Desarrollo del sistema

3.1.1 Software

Python es el lenguaje de programación seleccionado por aumentar su popularidad y versatilidad en los últimos años. Es multiparadigma, admite programación orientada a objetos, funcional y procedural, cuenta con librerías como *OpenCV*, *MediaPipe*, *NumPy*, *SciPy*, y *Matplotlib* para análisis de datos, imágenes y vídeo, útil al trabajar con rostro. Su facilidad de adaptación permite integrar interfaces gráficas, bases de datos y procesos de inteligencia artificial, *machine learning*, automatización y desarrollo web [50–52].

Entorno de desarrollo

El sistema se implementa en *Visual Studio Code* (*VS Code*), un Entorno de Desarrollo Integrado (IDE) de editor de código abierto que permite personalizar y agilizar las tareas mediante extensiones, integrando Inteligencia Artificial (IA). Ofrece herramientas de depuración, control de versiones, historial local y opciones de navegación con teclado. Su integración con GitHub facilita el trabajo colaborativo y mejora la compatibilidad con distintos lenguajes, accesible desde cualquier dispositivo mediante repositorios remotos o en el navegador [53].

3.1.2 Proceso de desarrollo

El sistema se desarrolla de forma incremental e iterativa. Cada versión incorpora correcciones y nuevas funciones, evaluadas con diferentes vídeos, lo que permite refinar el desempeño e identificar las necesidades del sistema.

Fase 1. Módulos independientes

Se inicia con un módulo simple de detección de movimiento; posteriormente se integran seguimiento, análisis y rotación, trabajando de manera independiente.

Los primeros vídeos incluyen a una persona, que en estas primeras iteraciones se trata de mí, en movimiento, seguidos de grabaciones obtenidas de internet (aeropuerto, persona pateando un balón de fútbol y péndulos). Finalmente se graba

una pelota en movimiento oscilatorio, variando colores para confirmar el funcionamiento correcto.

Fase 2. Módulos integrados

Se definen los módulos e integran en el programa principal `main.py`. Se crea la carpeta de preproceso y el módulo correspondiente con sus submódulos en orden, añadiendo uno de estabilización. Posteriormente se incorporan submódulos auxiliares para optimizar el código y evitar duplicidad.

En esta fase el sistema se automatiza completamente, implementando vídeos de nistagmo obtenidos de distintos sitios web y redes sociales, así como de ojos sanos en un ambiente más controlado. Se prueban diferentes métodos y, en algunos casos, se aplican estrategias alternativas para evitar fallos. Con cada nueva función se ajustan los parámetros, permitiendo procesar diversos vídeos bajo condiciones similares.

3.2 Organización

3.2.1 Flujo del sistema

La estructura modular permite que cada archivo de *Python* (.py) corresponda a un módulo o submódulo, lo que facilita la organización y el flujo de trabajo, como se muestra en la Figura 17. En este flujo se incluyen submódulos auxiliares, como la región de interés (ROI), entendida como el área del vídeo donde se analiza el movimiento, y el display, que corresponde a la ventana de visualización del vídeo procesado.

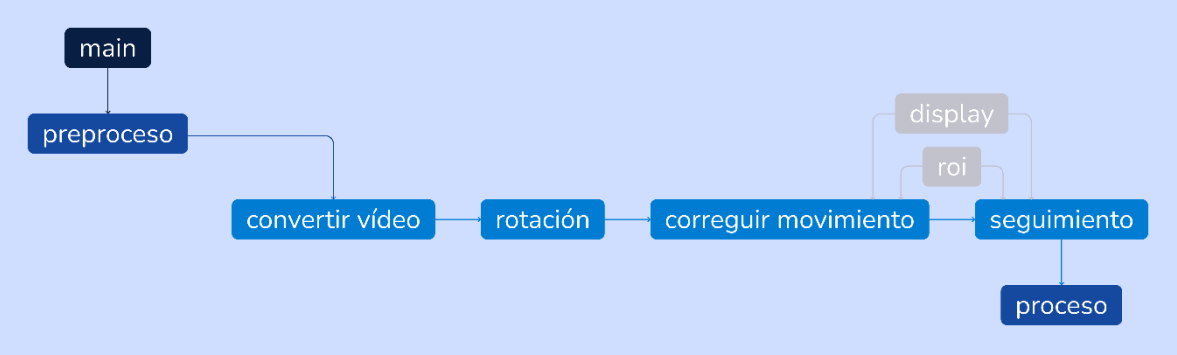

Figura 16. *Flujo de módulos del sistema. Elaboración propia.*

3.2.2 Estructura

El proyecto se organiza en tres carpetas principales: **src**, que contiene los archivos de programación; **videos**, donde se almacenan los vídeos originales y archivos de salida del preproceso (conversión, rotación y estabilización); y **datos**, que guarda los archivos de salida generados por el preproceso (seguimiento), proceso (análisis) y programa principal (main).

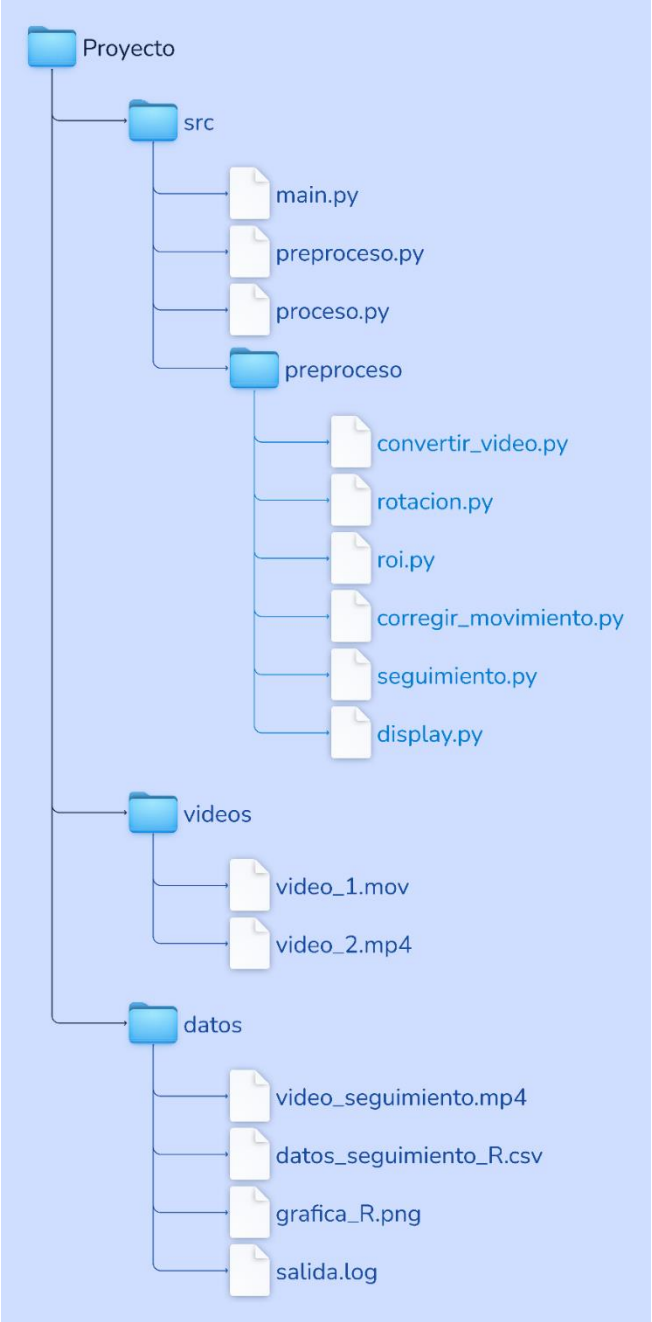

Figura 17. *Estructura de directorios del proyecto. Elaboración propia.*

3.3 Funcionalidad

3.3.1 Programa principal

En el programa principal (`main.py`) se define la ruta del vídeo original, que se envía al módulo de preproceso, este genera el modo de seguimiento, el cual transfiere al módulo de proceso.

Todos los módulos y submódulos especializados muestran mensajes en consola al inicio y al final de su ejecución, algunos agregan información relevante. Se implementa una clase que registra los mensajes dentro de un archivo de texto (`salida.log`), útil para verificar el flujo de ejecución y resultados.

3.3.2 Módulo de preproceso

El módulo `preproceso.py` recibe la ruta del vídeo original como entrada y ejecuta los cuatro submódulos especializados.

Al finalizar la corrección de movimiento se define el modo, que se pasa al seguimiento y constituye la salida del preproceso.

```
modo = lado_fijo if lado_fijo in ("left", "right") else "both"
```

Los submódulos `corregir_movimiento.py` y `seguimiento.py` pueden ejecutarse en tiempo real visualizados en una ventana. Esta función se activa o desactiva con una variable lógica y permite interrumpir la ejecución presionando *Esc*, mostrando un mensaje que indica la cancelación.

```
ventana = True  # Visualización en tiempo real
```

Submódulo especializado `convertir_vídeo.py`

Recibe la ruta del vídeo original y verifica si el archivo puede leerse con *OpenCV*. Si el vídeo es compatible, devuelve la misma ruta; en caso contrario, realiza la conversión a H.264 mediante *ffmpeg*, generando un nuevo archivo de vídeo (`_h264.mp4`). Finalmente, valida que el archivo convertido sea legible.

```
def asegurar_h264(ruta_video):
    cap = cv2.VideoCapture(ruta_video)
    ok, _ = cap.read()
    cap.release()
```

```
    if ok:
        return ruta_video  # Legible, no convierte

    # Conversión automática con ffmpeg
    nuevo_archivo = base + "_h264.mp4"
    comando = [
        "ffmpeg", "-y", "-i", ruta_video,
        "-c:v", "libx264", "-pix_fmt", "yuv420p",
        nuevo_archivo
    ]
    subprocess.run(comando, check=True)
    return nuevo_archivo
```

Devuelve la ruta del vídeo listo para usarse, los mensajes indican si es necesaria la conversión con el nombre del nuevo archivo.

Submódulo especializado `rotación.py`

Recibe la ruta del vídeo legible y corrige, en caso necesario, la orientación. Primero comprueba si existe un archivo corregido previamente; de ser así, lo utiliza. Si no, consulta los metadatos del vídeo para identificar un ángulo de rotación (90°, 180° o 270°). Cuando hay rotación recorre el vídeo cuadro por cuadro, aplica la corrección y genera un nuevo archivo de vídeo (`_rotado.mp4`) con la orientación adecuada. Elimina los metadatos de rotación para evitar giros adicionales en pruebas posteriores.

```
def rotacion(ruta_video):
    rotacion = obtener_rotacion(ruta_video)
    cap = cv2.VideoCapture(ruta_video)

    if rotacion == 0:
        return ruta_video  # Sin corrección

    out = cv2.VideoWriter(ruta_salida, fourcc, fps, (out_w, out_h))
```

```
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        frame_rotado = corregir_orientacion(frame, rotacion)
        out.write(frame_rotado)
```

Devuelve la ruta del vídeo con la orientación correcta, los mensajes indican si se usa un archivo corregido, o la rotación y si es necesaria la corrección con el nombre del nuevo archivo.

Submódulo especializado `corregir_movimiento.py`

Recibe la ruta del vídeo legible y con orientación correcta, así como la indicación de si debe verse en tiempo real. Estabiliza el movimiento global tomando como referencia la región periocular (párpados y comisuras), sin usar el iris.

```
cap = cv2.VideoCapture(ruta_video)
ok, first = cap.read()


multi0 = detectar_ojos_e_iris_multi(first)
L0 = multi0.get("left")
R0 = multi0.get("right")
tpl_gray, bbox_ref, ref_tl = _init_periocular_template(first, L0, R0)
```

Si se detectan movimientos bruscos o falsos positivos, se descartan mediante heurísticas de anti-salto[1], histéresis[2] y límites de reubicación[3]. Los desplazamientos validados se aplican como traslaciones, generando un nuevo archivo de vídeo estabilizado (`_estabilizado.mp4`). Además, se guardan archivos de texto con datos de depuración (`_estabilizado_debug.mp4`) y los desplazamientos por cuadro (`_estabilizado_deltas.csv`).

[1] Reglas simples usadas para evitar correcciones bruscas causadas por errores de detección entre cuadros consecutivos.
[2] Técnica tipo *Schmitt* aplicada al desplazamiento temporal (dx, dy), que introduce una zona muerta y doble umbral para evitar microoscilaciones y cambios bruscos entre cuadros consecutivos.
[3] Restricciones que fijan el desplazamiento máximo permitido por cuadro para prevenir correcciones excesivas.

```
dx = ref_tl[0] - cur_tl[0]
dy = ref_tl[1] - cur_tl[1]
dx, dy = _smooth_and_clamp(dx, dy)  # Suavizado y límite
dx = _apply_hysteresis(dx, "x")
dy = _apply_hysteresis(dy, "y")
stab = _translate(frame, dx, dy)    # Aplicar traslación
```

Devuelve la ruta del vídeo corregido y el modo de seguimiento (ojo izquierdo, derecho o ambos), los mensajes indican si hubo problemas al detectar el ojo o aplicar la corrección, y el nombre de los archivos generados.

Submódulo especializado `seguimiento.py`

Recibe la ruta del vídeo estabilizado y el modo de seguimiento, así como la indicación de si debe verse en tiempo real. Rastrea el movimiento del iris cuadro a cuadro.

Detecta el ojo y el iris en los cuadros iniciales, crea *trackers* CSRT[4] y establece puntos de anclaje en cuatro zonas alrededor del ojo. Mediante flujo óptico y transformaciones afines ajusta la posición detectada para compensar pequeños movimientos de cámara o ruido.

```
datos_multi = detectar_ojos_e_iris_multi(frame0)


if modo in ("both", "left") and datos_multi["left"]:
    # Ajustar caja del iris
    ib = _sanitize_bbox(frame0.shape, datos_multi["left"]["iris_bbox"])
    # Crear tracker CSRT
    tracker_L = _crear_tracker_csrt()
    # Inicializar seguimiento del ojo izquierdo
    tracker_L.init(frame0, tuple(int(v) for v in ib))
```

Cuando el seguimiento es válido, registra las coordenadas relativas de cada iris con respecto a su posición inicial y genera un vídeo de depuración con sobreposiciones gráficas (rectángulos, puntos y métricas).

[4] Algoritmos de seguimiento (*Channel and Spatial Reliability Tracker*) que permiten localizar un objeto de forma robusta a lo largo de cuadros consecutivos, incluso ante cambios moderados de iluminación o escala.

```
# Actualizar transformación por flujo óptico
M_L, ninl_L, rmse_L = _update_transform(prev_gray, gray, anchors_L)
# Obtener parámetros de movimiento
s, th, tx, ty = _sim_params(M_L)
# Corregir movimiento relativo
x_est, y_est = _apply_inv_affine(M_L, cx_L, cy_L)
# Guardar trayectoria
trayectoria_L.append((t_frame, x_est - origin_L[0], y_est - origin_L[1]))
```

Genera un archivo de vídeo con el seguimiento (`video_seguimiento.mp4`) y archivos de datos (`datos_seguimiento_L.csv`, `datos_seguimiento_R.csv`) que contienen las trayectorias en función del tiempo. Los mensajes indican el modo seleccionado, problemas al detectar ojo o iris, advertencias por ROI inválido y el nombre de los archivos generados.

Submódulo auxiliar `roi.py`

Recibe cada cuadro del vídeo y detecta la región de interés (ROI) correspondiente al ojo e iris. Utiliza *MediaPipe FaceMesh* para obtener puntos de referencia y, con ellos, delimitar el contorno ocular e iris, además de calcular su centro y radio. También define el modo de análisis, que puede ser binocular cuando ambos ojos son válidos o monocular si solo hay uno.

```
# Inicializar modelo FaceMesh
with mp_face.FaceMesh(refine_landmarks=True, max_num_faces=1) as mesh:
    rgb = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2RGB)  # Convertir a RGB
    res = mesh.process(rgb)  # Procesar el cuadro

    if res.multi_face_landmarks:
        lm = res.multi_face_landmarks[0].landmark    # Obtener landmarks faciales
        # Coordenadas del ojo e iris
        left_eye = [ (lm[i].x * w, lm[i].y * h) for i in LEFT_EYE_IDX ]
        left_iris = [ (lm[i].x * w, lm[i].y * h) for i in LEFT_IRIS_IDX ]
```

Si no se detecta el rostro completo, recurre a la caché de detecciones previas o emplea transformadas de Hough[5] en casos de *close-up* para localizar la pupila. Incluye memoria temporal para tolerar pérdidas breves y una función de reinicio del estado cuando es necesario.

```
# Detección alternativa por Hough
def _fallback_pupil_hough(frame_bgr, prev=None):
    g = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2GRAY)
    g = cv2.equalizeHist(g)
    g = cv2.medianBlur(g, 5)
    circles = cv2.HoughCircles(g, cv2.HOUGH_GRADIENT, dp=1.2, minDist=30,
                               param1=80, param2=18, minRadius=5, maxRadius=120)
```

Devuelve una estructura de datos por ojo (recuadro, centro y radio del iris, modo de análisis) e indicadores del método de detección usado.

Submódulo auxiliar `display.py`

Recibe cada cuadro procesado y lo muestra en tiempo real en una ventana de *OpenCV*, ajustando la visualización a la resolución del monitor. Utiliza parámetros de visualización como el nombre de la ventana, el factor de escala y la opción de pantalla completa. Calcula el tamaño máximo permitido, escala la imagen sin deformarla, la centra sobre un fondo negro (*letterbox*) y permite configurarla en pantalla completa o en reducción proporcional.

3.3.3 Módulo de proceso

El módulo `proceso.py` recibe el modo y analiza el movimiento ocular para categorizarlo.

Localiza los archivos generados en el seguimiento y organiza la información de tiempo y coordenadas (x, y). A partir de ellas calcula la distancia al punto inicial (d) y genera gráficas del movimiento ocular (x, y, d) respecto al tiempo.

[5] Técnica que permite detectar formas geométricas específicas (p. ej., círculos), útil para localizar la pupila en acercamientos.

Calcula el rango de movimiento en los ejes x y y (diferencia entre el valor máximo y mínimo de cada señal). Si uno de los ejes es 1.7 veces mayor que el otro, se selecciona como señal principal; si ninguno predomina, se considera un movimiento combinado. Esta señal se normaliza.

El factor 1.7 se definió de manera experimental como criterio de predominancia entre ejes. Este valor permitió diferenciar adecuadamente movimientos predominantemente unidireccionales de aquellos con componente combinado, evitando clasificaciones incorrectas cuando las magnitudes de ambos ejes eran similares en la base de datos actual.

```
def seleccionar_senal_principal(x, y, distancia):
    # Rango de movimiento (variación) en cada eje
    rango_x = np.max(x) - np.min(x)
    rango_y = np.max(y) - np.min(y)

    if rango_x > 1.7 * rango_y:
        return x, "x"
    elif rango_y > 1.7 * rango_x:
        return y, "y"
    else:
        return distancia, "distancia"
```

Genera las gráficas de las tres señales (desplazamientos_L.png, desplazamientos_R.png) y la señal principal (desplazamientoPrincipal_L.png, desplazamientoPrincipal_R.png).

Análisis de ritmicidad básico

Identifica los puntos máximos de la señal mediante el criterio de prominencia mínima utilizando la función *find_peaks*. Si no se detectan al menos tres picos, se indica que no hay información suficiente para evaluar la ritmicidad.

```
peaks, _ = find_peaks(senal, prominence=0.1)  # Picos (prominencia mínima 0.1)
```

A partir de los picos detectados se calculan los intervalos de tiempo entre ellos, se obtiene la media y la desviación estándar de estos valores y se determina el coeficiente de variación (CV).

```
intervalos = np.diff(tiempos_picos)  # Intervalos de tiempo
prom = np.mean(intervalos)  # Promedio o media
std = np.std(intervalos)  # Desviación estándar

# Coeficiente de variación | Usar valor alto si el promedio es cero
cv = (std / prom if prom > 0 else 1e9)
```

El coeficiente de variación se eligió porque permite evaluar la regularidad temporal de los intervalos entre picos de forma normalizada, independientemente de la escala de la señal. Esto lo hace adecuado para comparar señales con distintas amplitudes y velocidades.

En la literatura, un CV bajo (≈0.1 en estudios de sprint [54] y marcha [55]) se asocia con baja variabilidad temporal. Dado el enfoque distinto, se evaluaron valores entre 0.10 y 0.30, seleccionando 0.25 como umbral experimental para la base de datos actual.

Si el CV es bajo ($CV < 0.25$), los intervalos entre picos son regulares y se considera que el movimiento es rítmico. En caso contrario, se clasifica como arrítmico.

```
if cv < 0.25:
    print("✅ Movimiento RÍTMICO (nistagmo detectado)")
    return True
else:
    print("❌ Movimiento ARÍTMICO (sin nistagmo)")
    return False
```

Genera la gráfica donde los picos detectados aparecen resaltados sobre la señal principal (`picos_L.png`, `picos_R.png`). Los resultados incluyen la media de intervalos, la desviación estándar, el CV y la conclusión sobre la ritmicidad.

Análisis de ritmicidad FFT

Centra la señal en cero restando su media y posteriormente se aplica la Transformada Rápida de Fourier (FFT) con la función *scipy.fft*. El espectro obtenido permite identificar las frecuencias presentes y sus respectivas amplitudes.

Se utiliza la mediana del intervalo temporal por su robustez ante variaciones o valores atípicos en la adquisición de vídeo, evitando sesgos que podrían afectar el promedio.

```
# Frecuencia de muestreo a partir del intervalo medio de tiempo
fs = 1.0 / np.median(np.diff(t))

# Número total de muestras de la señal
N = len(senal)

# Centrar señal eliminando su componente DC (promedio)
senal_centrada = senal - np.mean(senal)

# Transformada rápida de Fourier (FFT)
yf = fft(senal_centrada)

# Frecuencias correspondientes (mitad positiva del espectro)
xf = fftfreq(N, 1 / fs)[: N // 2]

# Amplitud normalizada del espectro
amplitud = 2.0 / N * np.abs(yf[: N // 2])
```

La frecuencia dominante se define como aquella cuya amplitud es máxima (excluyendo la componente en 0 Hz).

```
# índice de la frecuencia con mayor amplitud, ignorando DC (0 Hz)
idx_max = (np.argmax(amplitud[1:]) + 1)

# Frecuencia dominante
freq_dom = xf[idx_max]

# Amplitud dominante
amp_dom = amplitud[idx_max]
```

La amplitud se eligió porque permite cuantificar la presencia de una componente periódica en el dominio de la frecuencia, comparando la energía de la frecuencia principal frente al contenido espectral medio. Esto facilita distinguir señales con un patrón rítmico definido de aquellas con comportamiento no periódico, asociado a ruido o variaciones irregulares.

En la literatura se reporta una amplitud de 9.59, respecto a ángulos para la vibración no controlada en mano, en que caso de *Párkinson*. Dado que es un enfoque diferente, se prueban diferentes valores ($3 < amplitud\ real > 5$), hasta encontrar un valor adecuado para la base de datos actual.

En la literatura se reportan amplitudes angulares de referencia (ej. 9.59° en temblor por *Párkinson* [56]), lo que respalda el uso de la amplitud como indicador de magnitud oscilatoria. Dado el uso de una magnitud espectral relativa en este trabajo, se evaluaron distintos valores, seleccionando 5 como criterio experimental para la base de datos actual.

Si la amplitud dominante es al menos cinco veces mayor que el valor medio del espectro ($amplitud\ real > 5$), el movimiento se considera rítmico. En caso contrario, se clasifica como arrítmico.

```
if amp_rel > 5:
    print("✅ Movimiento RÍTMICO (nistagmo detectado)")
    return True
else:
    print("❌ Movimiento ARÍTMICO (sin nistagmo)")
    return False
```

Genera las gráficas con el espectro de frecuencias y amplitud real y otra con la amplitud normalizada (`fft_L.png`, `fft_R.png`). Los resultados incluyen la frecuencia dominante con su amplitud, la relación entre amplitud dominante y media del espectro, y la conclusión sobre la ritmicidad.

Como ya se ha mencionado, los umbrales utilizados para los análisis de ritmicidad básico ($CV < 0.25$) y FFT ($amplitud\ real > 5$) se definieron de manera empírica, tras evaluar distintos valores hasta lograr una separación consistente entre movimientos rítmicos y no rítmicos en la base de datos analizada.

CAPÍTULO 4

VALIDACIÓN DEL SISTEMA

Capítulo 4 Validación del sistema

4.1 Pruebas

La validación consiste en catalogar el movimiento ocular como rítmico si es nistagmo o arrítmico en un ojo sano. Una vez definido el sistema, se prueban vídeos para las iteraciones finales, lo que contribuye a determinar sus límites y las condiciones de vídeo necesarias.

4.1.1 Vídeos

Aunque se establecen ciertas restricciones necesarias para evitar errores en el diagnóstico, el sistema mantiene flexibilidad ante distintas condiciones de grabación, ya que se probaron diferentes vídeos, la mayoría obtenidos sin un entorno controlado. Algunos de ellos fueron recortados debido a movimientos bruscos de usuario, zoom excesivo o desplazamiento de cámara, conservando solo los segmentos útiles para el análisis.

De las dieciséis pruebas realizadas, solo un vídeo corresponde a un ojo sano que no presenta nistagmo (**vídeo 0**), pero muestra un movimiento de usuario perceptible. La Tabla 1 incluye únicamente los vídeos con **observaciones relevantes** que influyen en la detección y validación del sistema; la tabla completa se presenta en el Anexo A.

Tabla 1. *Características principales de los vídeos.*

Vídeo	Ojo(s)	Duración [s]	Observaciones
4	Izquierdo	1	Se ve un poco del ojo derecho abierto y el usuario se acerca
6	Izquierdo	4	Hay zoom y el usuario cierra un poco el ojo
8	Ambos	8	Se ve la cara completa y el usuario está un poco lejos
10	Izquierdo	4	Usuario cierra el ojo al final
12	Ambos	6	Cara muy cerca, se ven ambos ojos, pero casi no se ve nariz ni otros puntos clave de del rostro y parpadea dos veces

(Tabla 1 continúa en la siguiente página)

(Continuación de la Tabla 1)

13	Ambos	3	Cara muy cerca, se ven ambos ojos, pero casi no se ve nariz ni otros puntos clave de del rostro y parpadea una vez
14	Ambos	3	Cara muy cerca, se ven ambos ojos, pero casi no se ve nariz ni otros puntos clave de del rostro
15	Ambos	1	Se ve la cara completa y el usuario está lejos

En la mayoría se observa un **reflejo ocular** notable, leve **movimiento del usuario** y, en ciertos casos, **de la cámara**. En otros, la **zona de ojos** es limitada debido a un acercamiento excesivo, lo que afecta parcialmente la detección.

4.1.1 Salidas del sistema

La Tabla 2 y Tabla 3 muestran las salidas generadas por el sistema para dos casos representativos: el **vídeo 0**, correspondiente a un ojo derecho sano, y el **vídeo 2**, con ambos ojos que presentan nistagmo. En ambos se incluyen las etapas intermedias y finales obtenidas en cada módulo/submódulo del sistema.

Tabla 2. *Archivos de salida generados por el sistema del caso 0.*

Archivo	Tipo	(sub)módulo asociado
video_0.mov	Vídeo MOV	—
video_0_rotado.mp4	Vídeo MP4	Rotación
video_0_rotado_estabilizado.mp4	Vídeo MP4	Corrección
video_0_rotado_estabilizado_debug.mp4	Vídeo MP4	Corrección
video_0_rotado_estabilizado_deltas.csv	Texto CSV	Corrección
datos_seguimiento_R.csv	Texto CSV	Seguimiento
fft_R.png	Imagen PNG	Proceso
grafica_senalPrincipal_R.png	Imagen PNG	Proceso
graficas_xyd_R.png	Imagen PNG	Proceso
picos_R.png	Imagen PNG	Proceso
salida.log	Texto LOG	Principal
video_seguimiento_R.mp4	Vídeo MP4	Seguimiento

Tabla 3. *Archivos de salida generados por el sistema del caso 2.*

Archivo	Tipo	(sub)módulo asociado
video_2.mp4	Vídeo MP4	—
video_2_estabilizado.mp4	Vídeo MP4	Corrección
video_2_estabilizado_debug.mp4	Vídeo MP4	Corrección
video_2_estabilizado_deltas.csv	Texto CSV	Corrección
datos_seguimiento_L.csv	Texto CSV	Seguimiento
datos_seguimiento_R.csv	Texto CSV	Seguimiento
fft_L.png	Imagen PNG	Proceso
fft_R.png	Imagen PNG	Proceso
grafica_senalPrincipal_L.png	Imagen PNG	Proceso
grafica_senalPrincipal_R.png	Imagen PNG	Proceso
graficas_xyd_L.png	Imagen PNG	Proceso
graficas_xyd_R.png	Imagen PNG	Proceso
picos_L.png	Imagen PNG	Proceso
picos_R.png	Imagen PNG	Proceso
salida.log	Texto LOG	Principal
video_seguimiento.mp4	Vídeo MP4	Seguimiento

A continuación, se muestran dos cuadros de referencia de los vídeos originales utilizados en las pruebas, que sirven como base de comparación con los resultados del sistema.

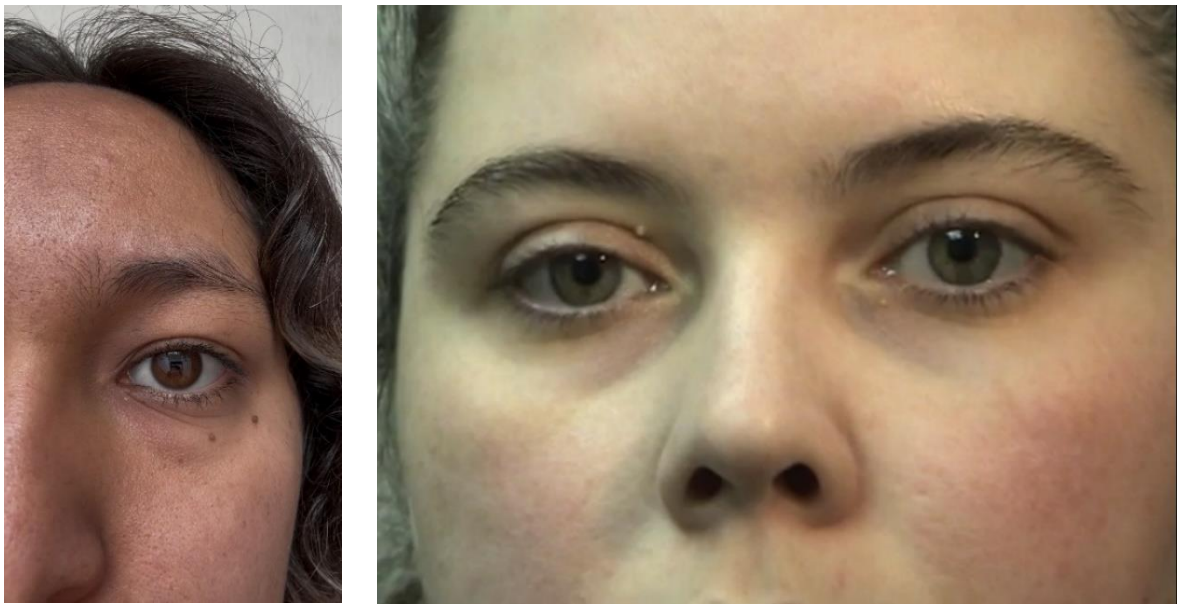

Figura 18. *Cuadros de referencia de los vídeos originales correspondientes a los casos 0 (izquierda) y 2 (derecha).*

Se encierra el ojo (o ambos, según el caso) en un recuadro verde que delimita la zona de anclaje mantenida fija durante todo el vídeo, corrigiendo el movimiento presente en la grabación.

Figura 19. *Proceso de estabilización en los casos 0 (izquierda) y 2 (derecha).*

El ojo izquierdo se marca con punto azul y recuadro verde, y el derecho con punto rojo y recuadro amarillo.

Se emplean cuatro zonas de anclaje con trece puntos cada una, que se muestran en verde cuando se mantienen estables y en rojo cuando pierden coincidencia con el punto de referencia. Estas zonas sirven como referencia fija respecto a la cual se evalúa el movimiento ocular, considerándose confiable el seguimiento cuando al menos dieciséis puntos permanecen válidos.

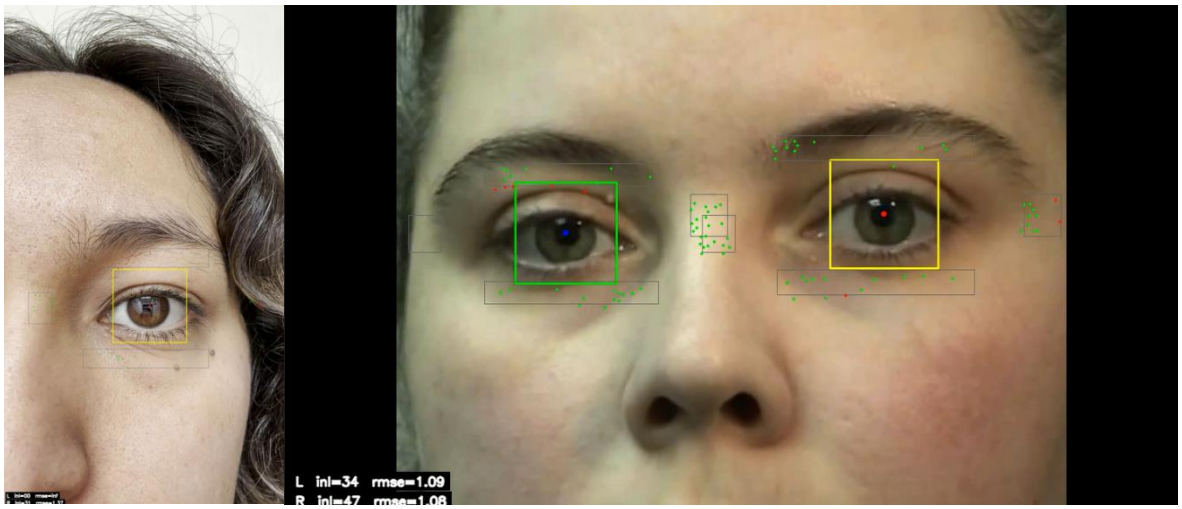

Figura 20. *Seguimiento ocular de los casos 0 (izquierda) y 2 (derecha).*

En la Figura *20* se observa un seguimiento correcto para el caso 0, con una adecuada identificación del iris y la pupila del ojo derecho, a pesar del reflejo considerable. La zona de anclaje del canto externo derecho presenta un fallo al situarse sobre el cabello.

En cambio, el *caso 2* representa un seguimiento casi ideal, con una correcta identificación del iris y la pupila de ambos ojos. El anclaje presenta puntos estables y, aunque en el canto externo del ojo izquierdo algunos no se visualizan, se cumple con el mínimo requerido para considerarse válido.

La **Figura 20** muestra dos ejemplos de fallos en la identificación, estos casos se incluyen para ilustrar las condiciones en las que el seguimiento no resulta válido.

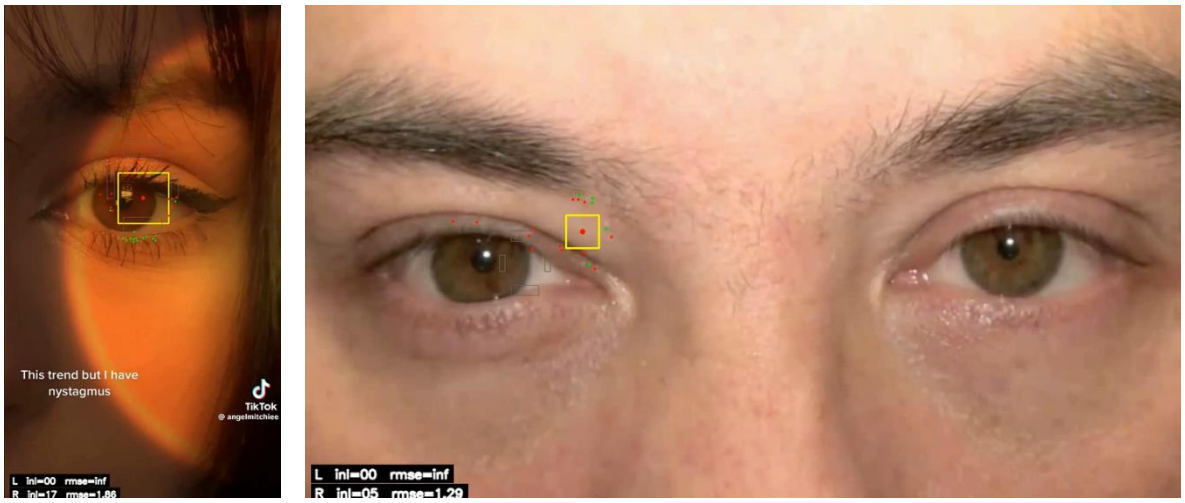

Figura 21. *Casos 11 (izquierda) y 13 (derecha) con fallos en la identificación.*

En el caso 11, la detección se encuentra desplazada hacia un lado del iris en lugar del centro de la pupila, correspondiente al ojo derecho con un reflejo considerable; además, las zonas de anclaje están incorrectamente ubicadas. En el caso 13, la identificación del ojo izquierdo y sus zonas de anclaje es totalmente fallida, mientras que el ojo derecho no se detecta.

Los datos del seguimiento ocular se obtienen por ojo, registrando su posición en las coordenadas *x* y *y* junto con el tiempo correspondiente. Las siguientes tablas muestran los primeros 10 datos obtenidos.

Tabla 4. *Coordenadas del ojo derecho en el caso 0.*

tiempo	x	y
0.033	0	0
0.067	0.6	-0.1
0.1	0.4	-0.2
0.133	0.4	-0.2
0.167	-0.4	-0.2
0.2	0.3	-0.1
0.233	1.1	-0.3
0.267	0.8	0.3
0.3	1.6	-0.2
0.333	2.4	0.2

Tabla 5. *Coordenadas de los ojos izquierdo (izquierda) y derecho (derecha) en el caso 2.*

tiempo	x	y
0.033	0	0
0.067	0	0
0.1	-0.9	0.8
0.133	-3.4	1.3
0.167	-5.2	1.8
0.2	-6.7	1.4
0.233	-3.7	1.4
0.267	-1.4	1.9
0.3	-0.2	2.1
0.333	-0.2	2

tiempo	x	y
0.033	0	0
0.067	1	1
0.1	0.1	2
0.133	-2.5	2.5
0.167	-10	4
0.2	-9.6	2.5
0.233	-5.7	2.5
0.267	-1.3	3.2
0.3	-0.5	3
0.333	-1	3.5

A partir de los datos obtenidos en el seguimiento, se generan las gráficas de desplazamiento en los ejes *x* y *y*, así como la distancia recorrida obtenida en función del tiempo para cada caso.

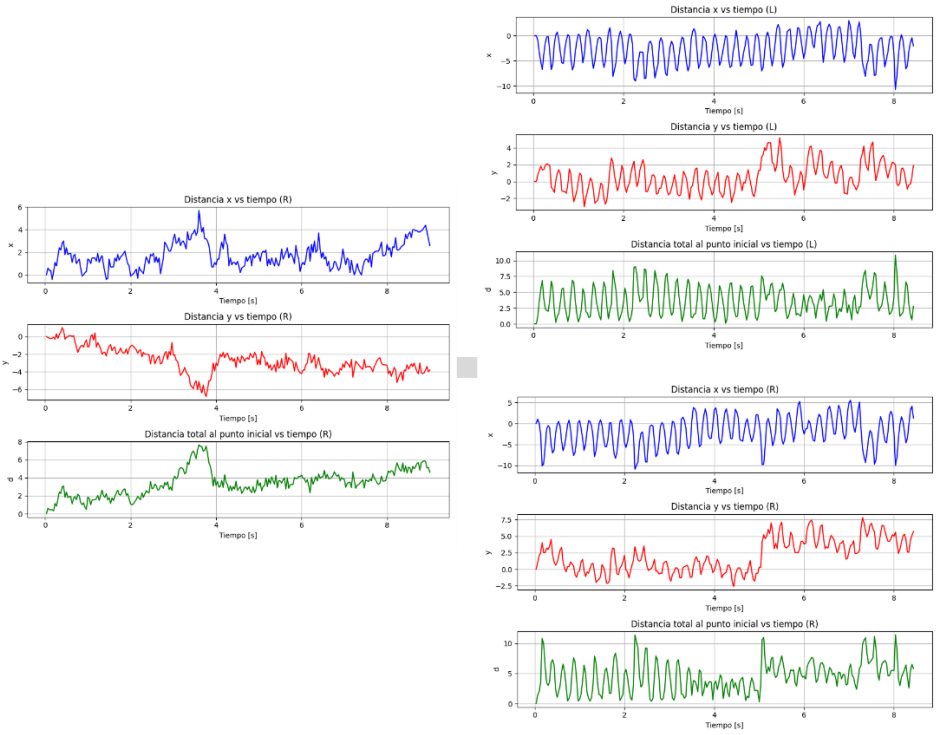

Figura 22. *Gráficas de desplazamiento de los casos 0 (izquierda) y 2 (derecha).*

Se selecciona la señal principal y se normaliza con el fin de facilitar la comparación entre casos.

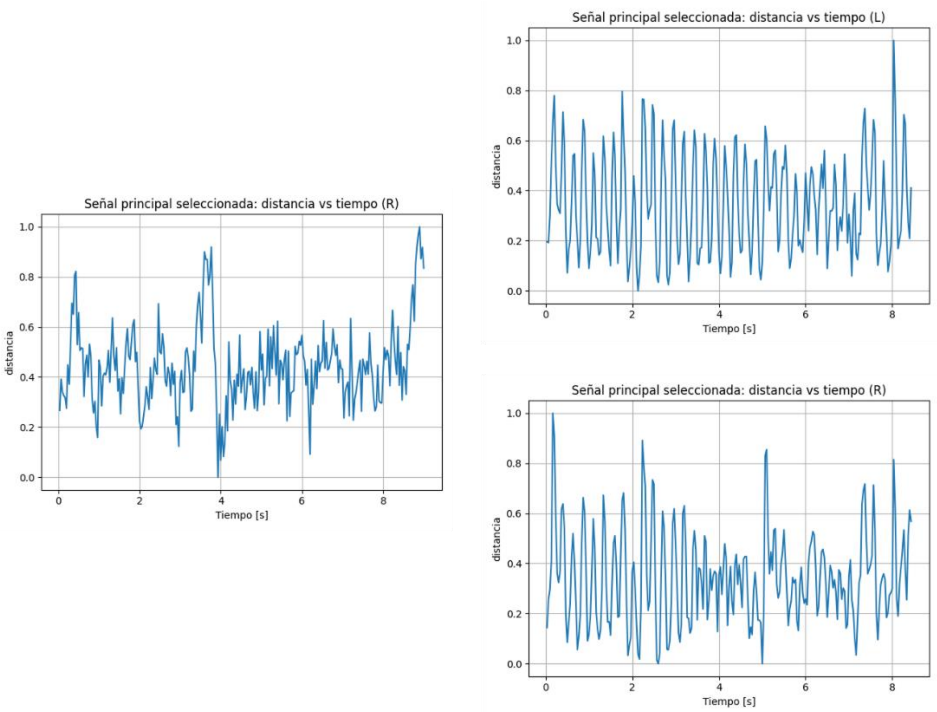

Figura 23. *Señal principal normalizada por ojo de los casos 0 (izquierda) y 2 (derecha).*

El análisis básico identifica los picos correspondientes a los movimientos oculares detectados.

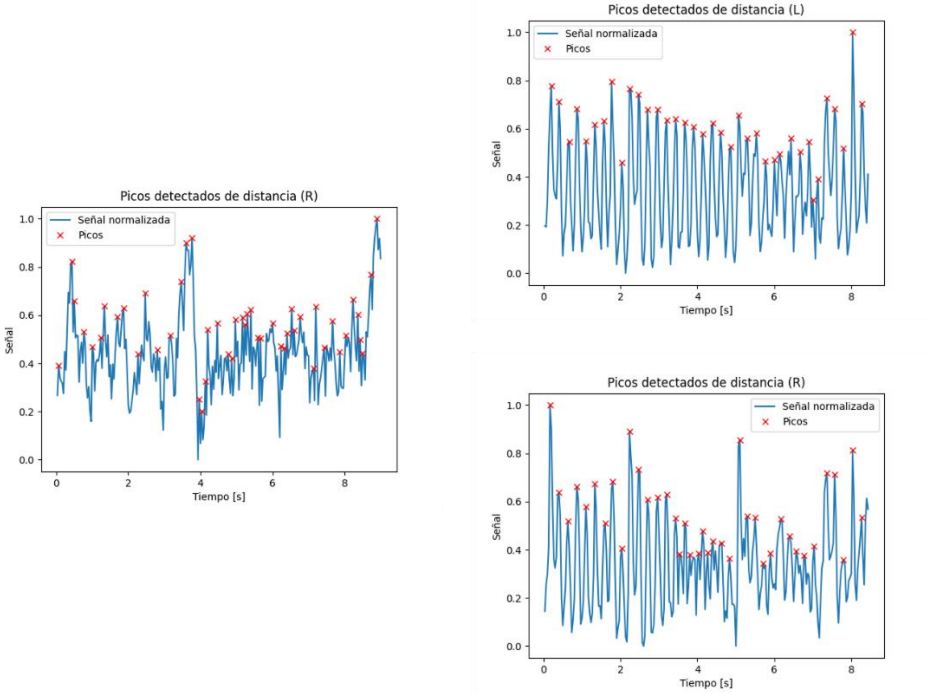

Figura 24. *Detección de picos por ojo para los casos 0 (izquierda) y 2 (derecha).*

Se aplica la Transformada Rápida de Fourier (FFT) para identificar la frecuencia dominante del movimiento ocular.

Figura 25. *Espectro de frecuencias por análisis FFT en los casos 0 (izquierda) y 2 (derecha).*

Los siguientes fragmentos corresponden a la sección del archivo final de salida.

Resultados obtenidos para el ojo derecho del caso 0:

```
📊 Movimiento en ambos ejes → se usará la distancia al punto inicial.

--- Análisis básico con picos (R) ---
Media.Int = 0.184s, Desv.std = 0.100s, Coef.Var = 0.55
❌ Movimiento ARÍTMICO (sin nistagmo)

--- Análisis Transformada de Fourier FFT (R) ---
Frecuencia dominante: 1.57 Hz con amplitud 0.068
Relación amplitud dominante / media espectro = 4.55
❌ Movimiento ARÍTMICO (sin nistagmo)
```

Resultados obtenidos para ambos ojos del caso 2:

```
📊 Movimiento en ambos ejes → se usará la distancia al punto inicial.

--- Análisis básico con picos (L) ---
Media.Int = 0.224s, Desv.std = 0.038s, Coef.Var = 0.17
✅ Movimiento RÍTMICO (nistagmo detectado)

--- Análisis Transformada de Fourier FFT (L) ---
Frecuencia dominante: 4.31 Hz con amplitud 0.147
Relación amplitud dominante / media espectro = 9.68
✅ Movimiento RÍTMICO (nistagmo detectado)

📊 Movimiento en ambos ejes → se usará la distancia al punto inicial.

--- Análisis básico con picos (R) ---
Media.Int = 0.213s, Desv.std = 0.049s, Coef.Var = 0.23
✅ Movimiento RÍTMICO (nistagmo detectado)
```

```
--- Análisis Transformada de Fourier FFT) (R) ---
Frecuencia dominante: 4.43 Hz con amplitud 0.109
Relación amplitud dominante / media espectro = 6.84
✅ Movimiento RÍTMICO (nistagmo detectado)
```

4.2 Resultados

En algunos casos se pierde el seguimiento al parpadear o no se ancla correctamente al centro del iris o la pupila. Se descartan los casos en los que el sistema realiza el seguimiento sobre el párpado u otra región distinta al iris o pupila. Por ello, el **vídeo 13** se excluye completamente al no detectar correctamente ningún ojo.

También queda descartado el **vídeo 11**, ya que las zonas de anclaje se fijan sobre el ojo y el parpado móvil, los cuales se desplazan junto con el movimiento ocular.

Los vídeos con duración de un segundo no ofrecen el tiempo suficiente para realizar un análisis completo ni ser catalogados como rítmicos por ningún método, por lo que se descartan los vídeos **4**, **9** y **15**.

Los resultados se obtienen mediante dos métodos de análisis de ritmicidad, denominados **básico**, que evalúa la regularidad temporal de los intervalos entre picos mediante el coeficiente de variación (CV), y **FFT**, que analiza la señal en el dominio de la frecuencia mediante la Transformada Rápida de Fourier (FFT).

Tabla 6. *Resultados de detección del movimiento mediante los métodos básico y FFT.*

Vídeo	¿Rítmico?	¿Acertó?	
		Básico	FFT
0	No	✓	✓
1	Sí	✗	✓
2	Sí	✓	✓
3	Sí	✗	✓
5	Sí	✗	✓
6	Sí	✓	✗
7	Sí	✓	✗
8	Sí	✗	✓
10	Sí	✓	✓
12	Sí	✗	✓
14	Sí	✗	✓

Para el criterio de validación del método básico, cuando no hay suficientes picos para el análisis, el **resultado** se considera **positivo** si el movimiento no es rítmico y **negativo** si lo es.

Para el criterio de validación general, se considera un **resultado positivo** cuando:

1. En vídeos de **un ojo**, este se detecta y clasifica correctamente;
2. en vídeos de **dos ojos**, ambos se detectan y clasifican correctamente; o
3. en vídeos de **dos ojos**, al menos uno se detecta y ese mismo se clasifica correctamente.

A partir de estos resultados se construyó la matriz de confusión y se calcularon métricas de desempeño adicionales.

Tabla 7. *Matriz de confusión de los métodos evaluados.*

Matriz de confusión		
Métrica	Básico	FFT
True Positive (TP)	4	8
True Negative (TN)	1	1
False Positive (FP)	0	0
False Negative (FN)	6	2

Tabla 8. *Métricas de desempeño comparativas entre el método básico y FFT.*

Métricas de desempeño (%)			
Métrica	Básico	FFT	Diferencia
Acurracy	45.45	81.82	36.36
Precision	100	100	0
Recall	40	80	40
F1-score	57.14	88.89	31.75

4.3 Análisis de resultados

El **método básico** tiene la limitante de que, si no encuentra al menos tres picos, muestra una advertencia y no genera resultado, lo cual puede generar conflicto si en los ojos sanos no se presenta ningún tipo de movimiento.

En cambio, el **método FFT** es más robusto y concreto: no depende del número de picos, analiza toda la señal y siempre produce un resultado positivo o negativo. Además, identifica la frecuencia dominante, es menos sensible al ruido o variaciones de amplitud, permite cuantificar la fuerza del ritmo mediante la relación de amplitud dominante sobre el espectro y facilita distinguir componentes rítmicos sutiles que no son evidentes en la forma de la señal.

Las condiciones del vídeo y del usuario influyen directamente en el desempeño del sistema. En la identificación de ojo(s), no se logra una detección correcta en los casos donde la zona ocular no está completamente visible (3 de 9 vídeos). Cuando se analiza un solo ojo, puede identificarse erróneamente como el contrario (2 de 7). Se observa una tendencia a fallas de detección cuando la región ocular está parcialmente oculta, lo que lleva a descartar los datos de seguimiento en los vídeos **12** y **14**. Otro motivo de descarte ocurre cuando un ojo aparece parcialmente en la imagen por algunos instantes, ya que el programa lo detecta de forma incorrecta. Por ello, se recomienda grabar ambos ojos para evitar este tipo de errores.

Factores como reflejos en el ojo, zoom o acercamientos excesivos, movimiento de la cámara y del usuario también afectan la detección del movimiento. En los vídeos con una duración de tres segundos (**3**, **7** y **14**), la FFT acierta en dos casos, mientras que el método básico solo en uno, lo que permite establecer un tiempo mínimo de grabación.

En general, la FFT muestra mayor consistencia ante variaciones en las condiciones de vídeo, mientras que el método básico resulta más sensible a ellas. De las once pruebas seleccionadas, seis fueron clasificadas correctamente únicamente por el método FFT, tres coincidieron en el resultado con ambos métodos y dos fueron acertadas solo por el método básico.

Las pruebas con mejor desempeño corresponden al **vídeo 0**, de un ojo sano, y al **vídeo 2**, con presencia de nistagmo en ambos ojos. El vídeo 0 fue el único realizado en condiciones controladas; aun así, presenta leve movimiento del usuario, lo que permite validar que el sistema clasifica correctamente el movimiento arrítmico. Aunque los demás vídeos no fueron grabados de forma controlada, resultaron útiles para establecer los límites del sistema, mientras que el vídeo 2, con mejores condiciones, demuestra que un usuario puede realizar una grabación adecuada de manera autónoma.

En términos de desempeño global, la **FFT** mostró una mejora respecto al método básico, alcanzando un accuracy de 81.82% frente a 45.45%. Esta mejora también se reflejó en el F1-score de 88.89% frente a 57.14%, métrica más representativa en bases de datos desbalanceadas. La diferencia fue de **36.36%** en **accuracy** y **31.75%** en **F1-score**, dentro de la base evaluada.

CAPÍTULO 5

CONCLUSIONES

Capítulo 5 Conclusiones

5.1 Conclusiones

Los vídeos controlados se realizaron con ojos sanos, incluyendo algunos con movimiento ocular voluntario probados en las primeras etapas. En los casos sin movimiento ocular se observó un leve desplazamiento del usuario; uno de estos vídeos fue seleccionado para validar que el sistema clasifica correctamente los movimientos arrítmicos.

Los vídeos con nistagmo, en cambio, no se grabaron en condiciones controladas, ya que su finalidad original era únicamente ilustrar el padecimiento. Algunos se recortaron para conservar las tomas más estables, lo que dio lugar a fragmentos de apenas un segundo de duración. Este escenario representó un reto que impulsó la incorporación de nuevas funciones en el preproceso, con el objetivo de obtener señales lo más libres de ruido posible.

Una vez completado el desarrollo del preproceso, se realizaron diversas pruebas para definir los parámetros utilizados en la base de datos actual, buscando procesar la mayor cantidad de vídeos de forma adecuada. Lo que inicialmente parecía una limitante terminó contribuyendo a establecer las características finales del sistema.

El desarrollo del preproceso y del proceso se llevó a cabo de manera paralela. Aunque esta estrategia no fue la más eficiente, permitió refinar progresivamente el sistema hasta lograr un preproceso estable, capaz de proporcionar señales más limpias y mejorar así el desempeño de ambas etapas.

Durante esta fase conjunta se buscó filtrar los datos obtenidos del preproceso en el proceso, ya que la estabilización inicial era muy básica. Al intentar perfeccionarla surgió un “efecto rebote”, lo que motivó la implementación de un anclaje en el seguimiento. De esta forma, el movimiento ocular se obtuvo de manera relativa a zonas fijas, manteniendo separadas las funciones de ambas partes del sistema y evitando interferencias entre ellas.

El desarrollo del proceso se realizó de forma iterativa y experimental; para definir los parámetros se evaluaron distintos vídeos, desde péndulos en etapas tempranas hasta personas con y sin nistagmo.

El método FFT surgió como una alternativa al análisis básico, dado que este último presentaba un desempeño cercano al 50% (45.45% de accuracy y 57.14% de F1-score) y no analizaba los casos sin movimiento ocular, limitándose a emitir una advertencia.

Con los resultados finales, el método FFT demostró mayor precisión y consistencia en la detección de movimientos rítmicos que el método básico. En términos cuantitativos, mejoró la clasificación en 36.36% (accuracy) y 31.75% (F1-score).

Dado el tamaño y desbalance de la muestra, estos resultados se limitan al conjunto de datos evaluado. El análisis incluyó un único caso de ojo sano, en el cual no se registraron falsos positivos; sin embargo, este resultado no es suficiente para establecer generalizaciones, por lo que se requiere validación adicional con una muestra ampliada.

El vídeo 0, correspondiente a un ojo sano, permitió validar que el preproceso identifica y trabaja correctamente con un solo ojo, y que el proceso clasifica de manera adecuada los movimientos arrítmicos. Por su parte, el vídeo 2, que muestra ambos ojos con nistagmo, validó la capacidad del sistema para identificar y procesar ambos ojos, clasificando correctamente el movimiento rítmico. Estos dos casos representan las pruebas con mejor desempeño y validan el funcionamiento integral del sistema desarrollado.

En conjunto, el sistema desarrollado cumple con el objetivo general de confirmar si el movimiento ocular es rítmico para identificar nistagmo. Asimismo, satisface los objetivos específicos al ser un proceso automatizado, sin intervención de un profesional, y enfocado en ser una herramienta accesible y adaptable a entornos remotos.

En conjunto, se cumplió el objetivo general de evaluar una propuesta de diseño de un sistema para confirmar si el movimiento ocular es rítmico con el propósito de identificar nistagmo de manera objetiva. Asimismo, se alcanzaron los objetivos específicos mediante la implementación del preprocesamiento y procesamiento de

señales para detectar y documentar el movimiento ocular, el desarrollo de dos métodos de análisis para la detección de movimiento ocular rítmico y la evaluación comparativa de su desempeño bajo las mismas condiciones de prueba.

CAPÍTULO 6

TRABAJO A FUTURO

Capítulo 6 Trabajo a futuro

6.1 Propuesta de implementación

La aplicación se plantea como una herramienta móvil disponible en dispositivos con cámara integrada, aprovechando la resolución que estos ofrecen frente a otros equipos electrónicos. De este modo se elimina la necesidad de utilizar accesorios adicionales y se garantiza un uso accesible y práctico.

6.1.1 Software

Frontend

Flutter es un *framework* de código abierto que permite desarrollar aplicaciones móviles, web y de escritorio a partir de un mismo código base, con soporte para *Android, iOS, Windows, MacOS* y *Linux*. Utiliza el lenguaje *Dart* y combina Programación Orientada a Objetos (POO) y declarativa. Ofrece una interfaz de usuario (UI) que mantiene la misma apariencia en diferentes plataformas, compila en código máquina para mejorar el rendimiento y cuenta con una amplia comunidad de código abierto, especialmente enfocada en aplicaciones móviles [57, 58].

Backend

FastAPI es un *framework* para crear APIs con *Python*. Se basa en Programación Orientada a Pbjetos (POO) y declarativa, proporciona buen rendimiento y facilita una codificación rápida con soporte para un desarrollo ágil [59].

Además, se propone integrar un servicio de almacenamiento en la nube (como *Firebase* o AWS S3) para guardar de forma segura los registros y resultados generados por el sistema, garantizando la disponibilidad y respaldo de la información.

6.1.2 Organización y diseño

La aplicación se organiza mediante un menú superior que agrupa las secciones principales: inicio, perfil, resultados, página principal y análisis. En la sección de inicio se gestiona el acceso y registro de usuario; el perfil muestra información personal y resultados recientes; los resultados concentran los análisis previos; la

página principal explica qué es el nistagmo y qué puede esperar el usuario de la aplicación; y el módulo de análisis ofrece instrucciones breves, grabación de vídeo y procesamiento automático.

La interfaz presenta un diseño simple y secuencial que guía al usuario desde el inicio de sesión hasta la obtención del resultado final. Incluye pantallas de bienvenida, acceso a la cámara frontal, previsualización y confirmación del material grabado. Durante la grabación se emiten tres señales de audio: la primera indica 10 segundos de preparación, la segunda marca el inicio de la captura y la tercera confirma su finalización.

El diseño busca transmitir confianza, serenidad y formalidad, además de mejorar la experiencia de usuario. Se emplea una paleta monocromática en tonos azules, evitando colores estridentes o distractores. La tipografía es clara y legible, y la interfaz mantiene una estética minimalista e intuitiva, centrada en facilitar la detección del nistagmo mediante la interpretación del resultado del movimiento ocular.

Figura 26. *Logotipo al iniciar aplicación NysCoss. Elaboración propia.*

Figura 27. *Prototipo de la aplicación NysCoss. Elaboración propia.*

Figura 28. *Prototipo de la aplicación NysCoss, posibles resultados. Elaboración propia.*

información.

6.1.3 Optimización

La propuesta actual tiene oportunidades de mejora, algunas podrían agregarse antes de implementarla y otras, podrían sumar cuando realicé el trabajo a futuro, incluyendo la mejora de la primera etapa y el desarrollo de la segunda.

La idea principal es desarrollar una aplicación móvil, aun así, el sistema podría ampliarse a versiones para escritorio y web, lo que aumentaría su accesibilidad y alcance.

Al lanzar la aplicación en una versión de prueba, se ampliaría la base de datos, lo que ayudaría a mejorar el análisis, o incluso implementar un método más sofisticado, dado que algunas bases actuales son de difícil acceso o requieren pago. Lo ideal es que, desde las versiones de prueba, ya se tenga colaboración médica, permitiendo la retroalimentación de profesionales para su implementación en el ámbito médico y su eventual posicionamiento en las tiendas de aplicaciones dentro de la categoría de salud.

Se plantea considerar la opción de ejecución en tiempo real mediante instrucciones y advertencias sonoras que guíen al usuario hasta obtener un resultado adecuado, incluyendo iluminación y enfoque del ojo.

La posibilidad de ingresar como médico con acceso a las métricas necesarias, con opción de agregar comentarios, o como paciente con un resultado y la confirmación de interpretación de un especialista.

Siempre garantizando la seguridad y privacidad de los datos de los usuarios, tanto médicos como pacientes.

6.2 Proyecto general

En esta sección se presentan las líneas de desarrollo futuro, contemplando la optimización de la primera etapa y el establecimiento de las bases de la segunda.

6.2.1 Primera etapa: detección de nistagmo de forma objetiva

Una de las principales áreas de oportunidad es colaborar directamente con el sector salud, no solo como consulta con un profesional, aportando así casos clínicos actuales y retroalimentación continua de especialistas.

Ampliar y balancear la base de datos con vídeos de nistagmo en condiciones controladas y registros de ojos sanos, permitiendo validar y ajustar los parámetros establecidos.

Definir especificaciones técnicas mínimas del vídeo, como resolución, frecuencia y velocidad de captura, para estandarizar las condiciones de análisis.

Optimizar el código tras la validación clínica de la primera etapa, reduciendo los tiempos de ejecución.

6.2.2 Segunda etapa: caracterización del movimiento

Para una caracterización completa del movimiento ocular, sería necesario incorporar parámetros como dirección y sentido, amplitud, frecuencia y velocidad de las fases. La integración de estas métricas permitiría una clasificación más detallada del nistagmo que incluso podría automatizarse de manera objetiva y contribuiría a la identificación de posibles causas.

Además, se podría implementar un sistema de registro y monitoreo por paciente, orientado a evaluar la evolución de la afección, su estabilidad o su reducción a lo largo del tiempo.

El fortalecimiento de la base de datos no solo incrementaría el volumen de información disponible, sino que permitiría realizar análisis estadísticos más robustos y modelos de interpretación más complejos.

REFERENCIAS

Referencias

1. Tomi Digital. *Sistema endocrino*. TOMI. Disponible en: https://tomi.digital/es/392594/sistema-endocrino
2. A. Singh y A. R. Varma. *Whole-Body Vibration Therapy as a Modality for Treatment of Senile and Postmenopausal Osteoporosis: A Review Article*. Cureus. vol. 15, n.º 1. p. e33690,
3. M. Ameer y A. Al Abbad. *R E V I E W Whole-Body Vibration to Enhance Skeletal Muscle Performance and Flexibility in Healthy Adults: A Narrative Review*. Muscle Ligaments Tendons J. vol. 13. pp. 504-515. sep. 2023,
4. *Whole-Body Vibration Training Increases Stem/Progenitor Cell Circulation Levels and May Attenuate Inflammation | Military Medicine | Oxford Academic*. Disponible en: https://academic.oup.com/milmed/article/185/Supplement_1/404/5740660 (Accedido: 18 de enero de 2026).
5. *Changes in Skin Microcirculation Resulting from Vibration Therapy in Women with Cellulite*. Disponible en: https://www.mdpi.com/1660-4601/19/6/3385 (Accedido: 18 de enero de 2026).
6. M. Huang, T. Miller, M. Ying, y M. Y. C. Pang. *Whole-body vibration modulates leg muscle reflex and blood perfusion among people with chronic stroke: a randomized controlled crossover trial*. Sci. Rep. vol. 10, n.º 1. p. 1473. ene. 2020,
7. F. Martinón-Torres, I. Ibarra de la Rosa, M. Fernández Sanmartín, E. García Menor, y J. M. Martinón Sánchez. *Ventilación de alta frecuencia*. An. Pediatría. vol. 59, n.º 2. pp. 172-180. ene. 2003,
8. D. R. Denisha y S. V. Nidhi. *Designing and Testing Compression Vibratory Combination Device for Lower Limb Lymphedema: An Exploratory Study*. Indian J. Phys. Ther. Res. vol. 6, n.º 1. p. 26. jun. 2024,
9. Á. D. R. Galindo, I. L. García, F. B. Carbajal, E. C. Littlewood, y J. L. H. Ávila. *ABSORBEDOR PASIVO DE VIBRACIONES PARA REDUCIR LOS FENÓMENOS FISIOLÓGICOS DE PRONACIÓN-SUPINACIÓN DE ANTEBRAZO DEBIDO A LA ENFERMEDAD DE PARKINSON (PASSIVE VIBRATION ABSORBER TO REDUCE THE PHYSIOLOGICAL PHENOMENA OF PRONATION-SUPINATION OF THE FOREARM DUE TO PARKINSON'S DISEASE)*. Pist. Educ. vol. 42, n.º 137. nov. 2020, Disponible en: https://pistaseducativas.celaya.tecnm.mx/index.php/pistas/article/view/2379 (Accedido: 18 de enero de 2026).

10. M. V. Perrotta, I. Kohler, y D. M. Eagleman. *Bimodal stimulation for the reduction of tinnitus using vibration on the skin*, 7 de diciembre de 2022, *bioRxiv*. Disponible en: https://www.biorxiv.org/content/10.1101/2022.11.28.518290v2 (Accedido: 18 de enero de 2026).
11. *A prospective comparison between skin cooling and skin vibration in reducing the pain of local anesthetic infiltration - Alshahwan - 2020 - Journal of Cosmetic Dermatology - Wiley Online Library*. Disponible en: https://onlinelibrary.wiley.com/doi/abs/10.1111/jocd.13160 (Accedido: 18 de enero de 2026).
12. T. Oroszi, M. J. G. van Heuvelen, C. Nyakas, y E. A. van der Zee. *Vibration detection: its function and recent advances in medical applications*. F1000Research. vol. 9. p. F1000 Faculty Rev-619. jun. 2020,
13. M. A. Pla. *NEUROPATÍA AUTONÓMICA CARDIOVASCULAR EN PACIENTES DIABÉTICOS SOMETIDOS A TRASPLANTE SIMULTÁNEO PÁNCREAS-RIÑÓN*.
14. M. K. Guess *et al. The Effects of a Genital Vibratory Stimulation Device on Sexual Function and Genital Sensation*. Urogynecology. vol. 23, n.º 4. p. 256. ago. 2017,
15. A. Bechara, A. Casabé, y W. D. Bonis. *Efectividad de las ondas de choque de baja intensidad con enfoque linear a los 6 meses del tratamiento de varones con disfunción eréctil y factores de riesgo vascular asociado*.
16. S. Marcos Alonso. *Nistagmo inducido por vibración mastoidea como predictor de ataques de vértigo en pacientes con enfermedad de ménière tratados con gentamicina intratimpánica*, http://purl.org/dc/dcmitype/Text, Universidad de Salamanca, Salamanca, España, 2024. Disponible en: https://dialnet.unirioja.es/servlet/tesis?codigo=329305 (Accedido: 19 de septiembre de 2025).
17. V. Díaz Rodríguez. *Manual de urgencias oftalmológicas para Enfermería*, Trabajo Fin de Máster (Tesis de maestría), Universidad de Valladolid, Facultad de Enfermería / Instituto Universitario de Oftalmobiología Aplicada (IOBA), Valladolid, España, 2017. Disponible en: https://uvadoc.uva.es/handle/10324/25313 (Accedido: 19 de septiembre de 2025).
18. *Qué es signo. Diccionario médico. Clínica U. Navarra*. https://www.cun.es. Disponible en: https://www.cun.es/diccionario-medico/terminos/signo (Accedido: 5 de octubre de 2025).

19. *Qué es Síntoma. Diccionario médico. Clínica U. Navarra*. https://www.cun.es. Disponible en: https://www.cun.es/diccionario-medico/terminos/sintoma (Accedido: 5 de octubre de 2025).
20. National Eye Institute (NEI). *National Eye Institute | National Eye Institute*. National Eye Institute. Disponible en: https://www.nei.nih.gov/ (Accedido: 19 de septiembre de 2025).
21. *Once millones de personas en México viven con ceguera o discapacidad visual*. Medscape. Disponible en: https://espanol.medscape.com/verarticulo/5911213 (Accedido: 19 de septiembre de 2025).
22. *4254 - Declaran el 15 de octubre Día Nacional de las Personas Ciegas y con otras Discapacidades Visuales / 15 / Octubre / 2020 / Boletines / Comunicación / Inicio - Camara de Diputados*. Disponible en: http://www5.diputados.gob.mx/index.php/esl/Comunicacion/Boletines/2020/Octubre/15/4254-Declaran-el-15-de-octubre-Dia-Nacional-de-las-Personas-Ciegas-y-con-otras-Discapacidades-Visuales (Accedido: 19 de septiembre de 2025).
23. D. C.V Desarrollo de Medios, S. A. de. *La Jornada: México, entre los 20 países donde más personas sufren discapacidad visual*. Disponible en: https://www.jornada.com.mx/2019/10/11/sociedad/036n2soc (Accedido: 19 de septiembre de 2025).
24. *44 POR CIENTO EN MÉXICO CON PROBLEMAS VISUALES: FES IZTACALA DA ATENCIÓN - UNAM Global*. Disponible en: https://unamglobal.unam.mx/global_tv/44-por-ciento-en-mexico-con-problemas-visuales-fes-iztacala-da-atencion/ (Accedido: 19 de septiembre de 2025).
25. J. M. S. Espinosa. *Nistagmo: fisiopatología y características clínicas*. Salud Areandina. vol. 2, n.º 1. pp. 58-69. 2013,
26. *What Is Nystagmus? - American Academy of Ophthalmology*. Disponible en: https://www.aao.org/eye-health/diseases/what-is-nystagmus (Accedido: 19 de septiembre de 2025).
27. *Nistagmo o Nistagmus | Qué es, Causas y Tratamientos*. Centro de oftalmología Barraquer. Disponible en: https://www.barraquer.com/patologia/nistagmo (Accedido: 19 de septiembre de 2025).
28. J. E. Ramírez-Salas *et al. Nistagmo no vestibular e intrusiones sacádicas no nistágmicas*. Rev. ORL. vol. 15, n.º 3. sep. 2024, Disponible en: https://scielo.isciii.es/scielo.php?script=sci_abstract&pid=S2444-79862024000300005&lng=es&nrm=iso&tlng=es (Accedido: 19 de septiembre de 2025).

29. Z. Fariñas Falcón, A. Hernández Camacho, y S. Álvarez Romero. *Nistagmo y baja visión*. Medicentro Electrónica. vol. 21, n.º 1. pp. 65-68. mar. 2017.
30. ICR. *El nistagmo - Neuroftalmología - Institut Català de retina*. ICR. Disponible en: https://icrcat.com/el-nistagmo/ (Accedido: 19 de septiembre de 2025).
31. *Eye Exam and Vision Testing Basics*. American Academy of Ophthalmology. Disponible en: https://www.aao.org/eye-health/tips-prevention/eye-exams-101 (Accedido: 19 de septiembre de 2025).
32. *Diagnóstico del nistagmo: cómo identificar este problema ocular*. Webconsultas Revista de salud y bienestar. Disponible en: https://www.webconsultas.com/salud-al-dia/nistagmo/diagnostico-del-nistagmo (Accedido: 19 de septiembre de 2025).
33. *Diagnosis – Nystagmus Network*. Nystagmus Network. Disponible en: https://nystagmusnetwork.org/information/diagnosis/ (Accedido: 19 de septiembre de 2025).
34. C. Oftalvist. *Nistagmo, ¿qué produce el movimiento involuntario de los ojos?* Disponible en: https://www.oftalvist.es/blog/nistagmo-causas-tipos (Accedido: 5 de octubre de 2025).
35. M. G. Cristell, G. D. Rosa, P. V. Miguel, y G. V. J. Ney. *Implementación de la telemática y monitoreo remoto para la atención adultos mayores en el ISSSTE*. Rev. Cuba. Cienc. Informáticas. vol. 18, n.º 4. 2024, Disponible en: https://rcci.uci.cu/index.php/RCCI/article/view/12982 (Accedido: 19 de septiembre de 2025).
36. Y.-W. Chen, S. Tanaka, R. J. Howlett, y L. C. Jain. *Innovation in Medicine and Healthcare: Proceedings of 12th KES-InMed 2024*. Springer Nature, 2025.
37. A. Caycedo, A. Serrano, y A. Ucros. *Telemedicina y oftalmología en tiempos de covid-19: un estudio descriptivo*. Univ. Medica. vol. 63, n.º 1. pp. 1-9. 2022.
38. D. Chueke. *What the Pandemic Left Us: Regulatory Advances for Telemedicine and Telehealth in Argentina*. Telehealth Med. Today. vol. 8, n.º 3. abr. 2023, Disponible en: https://telehealthandmedicinetoday.com/index.php/journal/article/view/413 (Accedido: 19 de septiembre de 2025).
39. B. López-Star, A. L. Ochoa-Ramírez, J. Vega-Lugo, R. Baralt-Zamudio, y P. Gonzalez-Daher. *Efficacy of teleophthalmology: Experience at the Mexican Institute of Ophthalmology*. Med. Res. Arch. vol. 13, n.º 2. feb. 2025, Disponible en: https://esmed.org/MRA/mra/article/view/6335 (Accedido: 19 de septiembre de 2025).
40. Á. C. Caparroso y L. M. R. López. *Telemedicina en Dermatología, Oftalmología y Urología: Una aproximación real a la clínica*. Sci. Educ. Med. J. vol. 5, n.º 2. pp. 24-32. may 2025.

41. S. Giuli Bello. *Diagnóstico optométrico mediante aplicaciones digitales*, Trabajo fin de grado, 2021. Disponible en: https://hdl.handle.net/11441/132688 (Accedido: 19 de septiembre de 2025).
42. W. Jerjes y D. Harding. *Telemedicine in the post-COVID era: balancing accessibility, equity, and sustainability in primary healthcare*. Front. Digit. Health. vol. 6. ago. 2024, Disponible en: https://www.frontiersin.org/journals/digital-health/articles/10.3389/fdgth.2024.1432871/full (Accedido: 19 de septiembre de 2025).
43. Z. P. M. Katherine y E. R. J. Guerrero. *Telemedicina en Tiempos de Crisis Sanitaria: Lecciones Aprendidas de la Pandemia Covid 19*. Rev. Veritas Difus. Científica. vol. 6, n.º 2. pp. 762-784. jul. 2025,
44. M. U. Friedrich *et al*. *Smartphone video nystagmography using convolutional neural networks: ConVNG*. J. Neurol. vol. 270, n.º 5. pp. 2518-2530. may 2023,
45. Oppenheim, Alan V. y Schafer, Ronald W. *Discrete-Time Signal Processing*, 2nd edition. NJ, USA: Prentice Hall, 1997.
46. R. M. Rangayyan. *Biomedical Signal Analysis*, 3rd edition. Hoboken, NJ, USA: Wiley-IEEE Press, 2022.
47. B. I. Gramatikov y D. L. Guyton. *Normalization of Retinal Birefringence Scanning Signals*. Sensors. vol. 25, n.º 1. p. 165. dic. 2024,
48. B. Balachandran y E. B. Magrab. *Vibrations*, 3rd edition. Cambridge, UK: Cambridge University Press, 2018. Disponible en: https://www.cambridge.org/highereducation/books/vibrations/D77B34F1320BDC1EAE74CCF5117E77C6#contents (Accedido: 19 de septiembre de 2025).
49. S. S. Rao y Y. F. Fah. *Mechanical vibrations*, 5. ed. in SI units. en Always learning. Singapore: Prentice Hall/Pearson, 2011.
50. V. Frick. *Lenguajes de programación 2025: los más demandados*. IT Patagonia. Disponible en: https://itpatagonia.com/lenguajes-de-programacion-2025/ (Accedido: 3 de octubre de 2025).
51. *TIOBE Index*. TIOBE. Disponible en: https://www.tiobe.com/tiobe-index/ (Accedido: 3 de octubre de 2025).
52. *Welcome to Python.org*. Python.org. Disponible en: https://www.python.org/ (Accedido: 3 de octubre de 2025).
53. *Visual Studio Code - Code Editing. Redefined*. Disponible en: https://code.visualstudio.com/ (Accedido: 3 de octubre de 2025).
54. R. J. Standing y P. S. Maulder. *The Biomechanics of Standing Start and Initial Acceleration: Reliability of the Key Determining Kinematics*. J. Sports Sci. Med. vol. 16, n.º 1. pp. 154-162. mar. 2017.

55. P. Terrier y Y. Schutz. *Variability of gait patterns during unconstrained walking assessed by satellite positioning (GPS)*. Eur. J. Appl. Physiol. vol. 90, n.º 5-6. pp. 554-561. nov. 2003,
56. S. Gebai y M. Hammoud. *Parkinson's Disease Treatment as Seen from a Mechanical Point of View*. Adv. Park. Dis. vol. 5, n.º 4. pp. 97-106. nov. 2016,
57. *Flutter - Build apps for any screen*. Disponible en: //flutter.dev/ (Accedido: 3 de octubre de 2025).
58. *¿Qué es Flutter? - Explicación de la aplicación Flutter - AWS*. Amazon Web Services, Inc. Disponible en: https://aws.amazon.com/es/what-is/flutter/ (Accedido: 3 de octubre de 2025).
59. *FastAPI*. Disponible en: https://fastapi.tiangolo.com/ (Accedido: 3 de octubre de 2025).

ANEXOS

ANEXO A

CÓDIGO DEL SISTEMA

A Código del sistema

Módulo `main.py`

```
import os
import sys

from preproceso import preproceso
from proceso import proceso

# - GUARDAR PRINTS EN ARCHIVO -
# Ruta absoluta de carpeta raíz del proyecto
ruta_base = os.path.dirname(
    os.path.dirname(os.path.abspath(__file__))
)  # ruta_base apunta a .../Proyecto
ruta_datos = os.path.join(
    ruta_base, "datos"
)  # Carpeta datos dentro de la raíz Proyecto
ruta_log = os.path.join(ruta_datos, "salida.log")


# Redirigir salida a consola y archivo
class DualWriter:
    def __init__(self, *writers):
        self.writers = writers

    def write(self, data):
        for w in self.writers:
            w.write(data)
            w.flush()

    def flush(self):
        for w in self.writers:
            w.flush()


# Conservar stdout original y agregar archivo
sys.stdout = DualWriter(
    sys.stdout, open(ruta_log, "w", encoding="utf-8")
)  # Sobrescribir cada vez "w" | Ir acumulando ejecuciones "a"
# sys.stderr = sys.stdout # Guardar errores
```

```
# - PROGRAMA PRINCIPAL -
def main():
    # Vídeo
    ruta_base = os.path.dirname(os.path.abspath(__file__))  # Carpeta src/
    ruta_video = os.path.join(ruta_base, "..", "videos", "nistagmo",
"N2.mp4")  # Vídeo
    ruta_video = os.path.normpath(ruta_video)  # Normalizar la ruta

    modo = preproceso(
        ruta_video
    )  # Procesar vídeo (formato legible > rotación > estabilización >
seguimiento)

    proceso(modo)  # Analizar ojo(s)


# - NO EJECUTAR SI ES IMPORTADO -
if __name__ == "__main__":
    main()
```

Módulo preproceso.py

```
from video.convertir_video import asegurar_h264
from video.corregir_movimiento import corregir_movimiento
from video.rotacion import rotacion
from video.seguimiento import seguimiento


# - PREPROCESO -
def preproceso(ruta_video_original):
    print("🟢 Prepoceso iniciado 🟢\n")

    ventana = False  # Pruebas rápidas False

    # Paso 0: formato H.264
    ruta_video_convertido = asegurar_h264(ruta_video_original)

    # Paso 1: corregir rotación
    ruta_video_rotado = rotacion(ruta_video_convertido)

    # Paso 2: corregir movimiento y guardar video corregido
    ruta_video_corregido, lado_fijo = corregir_movimiento(
        ruta_video_rotado, debug=ventana
    )

    # Paso 3: decidir modo de seguimiento
    modo = lado_fijo if lado_fijo in ("left", "right") else "both"

    # Paso 4: seguimiento con video corregido
    seguimiento(ruta_video_corregido, modo, debug=ventana)

    print("🟢 Prepoceso ejecutado 🟢\n")

    # Devolver modo
    return modo
```

Submódulo especializado convertir_vídeo.py

```
import os
import subprocess

import cv2


# - CONVERTIR VÍDEO -
def asegurar_h264(ruta_video):
    print("🎞️ Conversión de vídeo iniciada")

    # - RUTA ABSOLUTA -
    ruta_abs = os.path.abspath(ruta_video)

    # - VALIDAR EXISTENCIA -
    if not os.path.exists(ruta_abs):
        raise FileNotFoundError(f"El archivo no existe: {ruta_abs}")

    # - PROBAR LECTURA -
    cap = cv2.VideoCapture(ruta_abs)
    ok, _ = cap.read()
    cap.release()

    if ok:
        print("El video se puede leer, no es necesario convertir.")
        print("🎞️ Conversión de vídeo ejecutada\n")
        return ruta_abs  # No necesita conversión

    # - CONVERTIR A H.264 CON FFMPEG -
    base, _ = os.path.splitext(ruta_abs)
    nuevo_archivo = base + "_h264.mp4"

    print("El video no se puede leer, convirtiendo a H.264...")
    comando = [
        "ffmpeg",
        "-y",
        "-i",
        ruta_abs,
        "-c:v",
        "libx264",
        "-pix_fmt",
        "yuv420p",
        "-c:a",
        "aac",
```

```
        "-movflags",
        "+faststart",
        nuevo_archivo,
    ]

    # - EJECUTAR CONVERSIÓN -
    try:
        subprocess.run(comando, check=True)
    except FileNotFoundError:
        raise RuntimeError(
            "No se encontró ffmpeg. Instálalo y asegúrate de que esté en el
PATH."
        )
    except subprocess.CalledProcessError as e:
        raise RuntimeError(f"Error al convertir el video: {e}")

    # - VERIFICAR ARCHIVO CONVERTIDO -
    cap = cv2.VideoCapture(nuevo_archivo)
    ok, _ = cap.read()
    cap.release()

    if not ok:
        raise RuntimeError(f"No se pudo leer el video convertido:
{nuevo_archivo}")

    print("Conversión exitosa:", nuevo_archivo)

    print("🎞️ Conversión de vídeo ejecutada\n")
    return nuevo_archivo
```

Submódulo especializado rotación.py

```
import os
import subprocess  # Permite ejecutar comandos del sistema (como ffprobe)

import cv2
from pymediainfo import MediaInfo


# - ROTAR -
def rotacion(ruta_video):
    print("🔄 Rotación iniciada")

    # - CREAR RUTAS DE SALIDA -
    base, _ = os.path.splitext(ruta_video)  # Ignorar extensión original
    ruta_salida_temp = base + "_rotado_tmp.mp4"
    ruta_salida_final = base + "_rotado.mp4"

    # - SI YA ESTA PREPROCESADO -
    if os.path.exists(ruta_salida_final):
        print(f"Vídeo rotado -> {os.path.basename(ruta_salida_final)}")
        print("Video ya corregido encontrado. Usando vídeo existente.")
        print("🔄 Rotación ejecutada\n")
        return ruta_salida_final

    # - OBTENER ROTACIÓN -
    rotacion = obtener_rotacion(ruta_video)

    # - CONFIGURAR CAPTURA -
    cap = cv2.VideoCapture(ruta_video)  # Abrir video original y leerlo
frame por frame

    # - OBTENER PROPIEDADES -
    # Propiedades de vídeo
    width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    fps = cap.get(cv2.CAP_PROP_FPS)  # Cuadros por segundo

    # - VALIDAR ROTACIÓN -
    if rotacion == 0:
        print("Rotación ejecutada. No fue necesario corregir rotación.")
        print("🔄 Rotación ejecutada\n")
        return ruta_video  # Sin rotación, usar original

    # - EVITAR DOBLE ROTADO -
```

```
    if rotacion in [90, 270] and height > width:
        print("OpenCV ya aplicó la rotación; no se rota de nuevo.")
        cap.release()
        print("🔄 Rotación ejecutada\n")
        return ruta_video

    # - AJUSTAR DIMENSIONES Y ESCRITURA -
    out_w, out_h = (height, width) if rotacion in [90, 270] else (width,
height)

    fourcc = cv2.VideoWriter_fourcc(
        *"mp4v"
    )  # Definir códec (formato) para guardar video
    out = cv2.VideoWriter(
        ruta_salida_temp, fourcc, fps, (out_w, out_h)
    )  # Preparar objeto VideoWriter para guardar nuevo video corregido

    # - ROTAR Y GUARDAR FRAMES -
    while True:
        ret, frame = cap.read()  # Leer cada frame del video original
        if not ret:
            break
        frame_rotado = corregir_orientacion(frame, rotacion)
        out.write(frame_rotado)  # Escribir frame rotado en el nuevo video

    cap.release()  # Cerrar vídeo original (cap)
    out.release()  # Cerrar vídeo rotado (out)

    # - LIMPIAR METADATO DE ROTACIÓN -
    # Limpiar metadato para evitar rotación futura
    limpiar_metadato_rotate(ruta_salida_temp, ruta_salida_final)
    os.remove(ruta_salida_temp)

    print(f"Vídeo rotado -> {os.path.basename(ruta_salida_final)}")
    print("🔄 Rotación ejecutada\n")

    return ruta_salida_final  # dDvolver ruta del vídeo


# - OBTENER ROTACIÓN -
def obtener_rotacion(ruta_video):
    # pymediainfo para detectar rotación del video desde metadatos
    media_info = MediaInfo.parse(ruta_video)

    for track in (
```

```
        media_info.tracks
    ):  # Recorrer todas las pistas encontradas en el archivo de video
        if (
            track.track_type == "Video" and track.rotation is not None
        ):  # Verifica si la pista es tipo "Video" y contiene información de
rotación
            rotacion = int(float(track.rotation))  #  Convertir rotación a
entero
            print(f"Rotación detectada (MediaInfo): {rotacion}°")
            return rotacion  # Devolver rotación
    return 0  # Sin rotación detectada


# - CORREGIR ORIENTACIÓN -
def corregir_orientacion(frame, rotacion):
    # Aplicar rotación al frame si es necesario
    if rotacion == 90:
        return cv2.rotate(frame, cv2.ROTATE_90_CLOCKWISE)
    elif rotacion == 180:
        return cv2.rotate(frame, cv2.ROTATE_180)
    elif rotacion == 270:
        return cv2.rotate(frame, cv2.ROTATE_90_COUNTERCLOCKWISE)
    return frame  # Si no hay rotación, devuelve el frame tal cual


# - LIMPIAR METADATO DE ROTACIÓN -
def limpiar_metadato_rotate(ruta_in, ruta_out):
    # Eliminar metadato de rotación usando ffmpeg
    ffmpeg_path = r"C:\ffmpeg\ffmpeg-7.1.1-essentials_build\bin\ffmpeg.exe"
    subprocess.run(
        [
            ffmpeg_path,
            "-y",
            "-i",
            ruta_in,
            "-metadata:s:v:0",
            "rotate=0",
            "-c",
            "copy",
            ruta_out,
        ],
        check=True,
        stdout=subprocess.DEVNULL,  # Oculta salida estándar
        stderr=subprocess.DEVNULL,  # Oculta mensajes de error/info
    )
```

Submódulo especializado corregir_movimiento.py

```
import csv
import os  # Usar os para rutas
from collections import deque

import cv2
import numpy as np
from video.display import show_letterboxed
from video.roi import detectar_ojos_e_iris_multi, reset_roi_state  # Usar
ROI modular

# - PARÁMETROS DE FILTRO Y ESTABILIZACIÓN -
_MAX_DXY = 1.6  # Máx. píxeles por frame (clamp) → pasos mínimos
_REDETECT_EVERY = (
    30  # Re-detección cada N frames (menos frecuente, evita saltitos
periódicos)
)
_REDETECT_GAP = 25  # Mínimo de frames entre re-detecciones (cooldown)
_MIN_TM_SCORE = 0.78  # Score mínimo para aceptar match de plantilla
_MIN_TM_KEEP = 0.74  # Debajo de esto, no mover ni actualizar plantilla
_SMOOTH_WIN = 13  # Ventana mediana para suavizar dx,dy

# - ZONA MUERTA (HISTERESIS) -
_DEADBAND_LO = 1.6  # Dentro de ±1 px => no mover
_DEADBAND_HI = 3.2  # Solo si el error supera ±2.5 px permitimos mover
_DEADBAND = _DEADBAND_LO  # Usar misma zona muerta baja (±1 px)

# - RESTRICCIONES DE MOVIMIENTO -
_MAX_RELOC = 24  # Máx. reubicación frente a la referencia en re-detección
_RUN_MAX = 7  # Evitar rachas largas de movimientos en el mismo sentido

# - ESTADO GLOBAL DE MOVIMIENTO -
_prev_move_sign = {"x": 0, "y": 0}
_same_sign_run = {"x": 0, "y": 0}

# - HISTORIAL DE DESPLAZAMIENTO (SUAVIZADO) -
_hist_dx = deque(maxlen=_SMOOTH_WIN)  # Historial dx
_hist_dy = deque(maxlen=_SMOOTH_WIN)  # Historial dy
_ema_dx = 0.0
_ema_dy = 0.0


# - CONVERTIR A ESCALA DE GRISES Y APLICAR CLAHE -
def _to_gray_clahe(img):  # Aplicar CLAHE para mejorar contraste y robustez
```

```
    g = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)  # Convertir a gris
    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))  # Crear
objeto CLAHE
    return clahe.apply(g)  # Devolver imagen con CLAHE


# - CALCULAR BBOX PERIOCULAR DESDE PUNTOS O BBOX BASE -
def _bbox_from_pts_or_bbox(
    entry, W, H, margin
):  # Calcular bbox periocular desde puntos o bbox base
    if entry is None:
        return None
    # Usar bbox de ojo si está disponible
    if "eye_bbox" in entry and entry["eye_bbox"]:
        x, y, w, h = entry["eye_bbox"]
        x1, y1 = x, y
        x2, y2 = x + w, y + h
    else:
        # Usar puntos del contorno de ojo si existen
        pts = entry.get("eye_pts", None)
        if pts is None or len(pts) == 0:
            return None
        pts = np.array(pts, dtype=np.float32)  # Convertir a arreglo
        x1, y1 = pts.min(axis=0)
        x2, y2 = pts.max(axis=0)  # Medir extremos

    bw, bh = (x2 - x1), (y2 - y1)  # Calcular tamaño
    x1 -= bw * margin
    y1 -= bh * margin  # Expandir con margen
    x2 += bw * margin
    y2 += bh * margin
    x1 = int(max(0, round(x1)))
    y1 = int(max(0, round(y1)))  # Limitar a imagen
    x2 = int(min(W - 1, round(x2)))
    y2 = int(min(H - 1, round(y2)))
    return (x1, y1, max(0, x2 - x1), max(0, y2 - y1))  # Devolver bbox
válido


# - UNIR DOS BBOXES EN UNO -
def _union_bboxes(b1, b2):  # Unir dos bboxes en uno que englobe ambos
    if b1 is None and b2 is None:
        return None
    if b1 is None:
        return b2
```

```
    if b2 is None:
        return b1
    x1 = min(b1[0], b2[0])
    y1 = min(b1[1], b2[1])  # Calcular esquina superior izquierda
    x2 = max(b1[0] + b1[2], b2[0] + b2[2])
    y2 = max(b1[1] + b1[3], b2[1] + b2[3])  # Calcular esquina inferior
derecha
    return (x1, y1, x2 - x1, y2 - y1)  # Devolver bbox unido


# - RECORTAR IMAGEN SEGÚN RECTÁNGULO -
def _crop(img, rect):  # Recortar imagen usando rectángulo (x, y, w, h)
    x, y, w, h = rect
    if w <= 0 or h <= 0:
        return np.zeros((0, 0), dtype=np.uint8)
    return img[y : y + h, x : x + w].copy()  # Devolver recorte


# - COINCIDENCIA LOCAL DE PLANTILLA (MATCHING) -
def _match_near_gray_clahe(
    frame, tpl_gray, prev_bbox, search_pad=16
):  # Buscar plantilla cerca de la posición previa
    H, W = frame.shape[:2]  # Medir tamaño del frame
    x, y, w, h = prev_bbox
    sx = max(0, x - search_pad)
    sy = max(0, y - search_pad)  # Calcular ventana de búsqueda
    ex = min(W, x + w + search_pad)
    ey = min(H, y + h + search_pad)
    search = frame[sy:ey, sx:ex]  # Recortar ventana
    if search.size == 0 or tpl_gray.size == 0:
        return (x, y, -1.0)
    sG = _to_gray_clahe(search)  # Aplicar CLAHE a ventana
    if sG.shape[0] < tpl_gray.shape[0] or sG.shape[1] < tpl_gray.shape[1]:
        return (x, y, -1.0)
    res = cv2.matchTemplate(
        sG, tpl_gray, cv2.TM_CCOEFF_NORMED
    )  # Usar correlación normalizada
    _, max_val, _, max_loc = cv2.minMaxLoc(res)  # Calcular mejor
coincidencia
    best_x = sx + max_loc[0]
    best_y = sy + max_loc[1]  # Calcular posición global
    return (best_x, best_y, float(max_val))  # Devolver mejor posición y
score


# - APLICAR TRASLACIÓN AL FRAME -
```

```
def _translate(frame, dx, dy):  # Aplicar traslación (dx, dy) al frame
    H, W = frame.shape[:2]
    M = np.float32([[1, 0, dx], [0, 1, dy]])  # Calcular matriz de
traslación
    return cv2.warpAffine(
        frame, M, (W, H), flags=cv2.INTER_LINEAR,
borderMode=cv2.BORDER_REPLICATE
    )  # Devolver frame estabilizado


# - SUAVIZAR Y LIMITAR DESPLAZAMIENTO -
def _smooth_and_clamp(dx, dy):
    # Suavizado híbrido: mediana (contra outliers) + EMA (continuidad suave)
    global _ema_dx, _ema_dy

    _hist_dx.append(float(dx))
    _hist_dy.append(float(dy))
    mdx = np.median(_hist_dx)
    mdy = np.median(_hist_dy)

    # EMA: reduce vibración sin "escalonar"
    alpha = 0.25  # suavizado moderado (sube a 0.3 si aún vibra, baja si
sientes lag)
    _ema_dx = (1 - alpha) * _ema_dx + alpha * mdx
    _ema_dy = (1 - alpha) * _ema_dy + alpha * mdy

    # Clamp final por seguridad (no escalonado)
    sdx = max(-_MAX_DXY, min(_MAX_DXY, _ema_dx))
    sdy = max(-_MAX_DXY, min(_MAX_DXY, _ema_dy))
    return float(sdx), float(sdy)


# - CREAR PLANTILLA PERIOCULAR (MONOCULAR O BINOCULAR) -
def _init_periocular_template(first_frame, L, R):
    # Crear plantilla periocular estable (no usar iris) con margen según
mono/both
    H, W = first_frame.shape[:2]

    # Si sólo hay uno de L/R → monocular; si hay ambos → binocular
    mono = (L is None) ^ (R is None)  # XOR: True si hay exactamente uno
    margin = 0.18 if mono else 0.25  # usar mismos valores que definiste
(mono/both)

    bL = _bbox_from_pts_or_bbox(L, W, H, margin=margin) if L is not None
else None
```

```
    bR = _bbox_from_pts_or_bbox(R, W, H, margin=margin) if R is not None
else None

    if bL is None and bR is None:
        return None, None, None

    bbox_union = _union_bboxes(bL, bR)  # mono: toma el único; both: une
ambos
    tpl_rgb = _crop(first_frame, bbox_union)  # recorte de plantilla
    tpl_gray = _to_gray_clahe(tpl_rgb)  # GRAY+CLAHE para robustez
    ref_tl = (bbox_union[0], bbox_union[1])  # esquina sup. izq. como
referencia fija
    return tpl_gray, bbox_union, ref_tl


# - ESTABILIZAR -
def corregir_movimiento(
    ruta_video, debug, prefer_side="auto", ojos="auto", fullscreen=False
):  # Corregir movimiento por estabilización periocular
    print("☞ Estabilización iniciada")

    # - INICIALIZAR VARIABLES -

    # Control de estabilizar
    cum_dx, cum_dy = 0.0, 0.0
    stab_log = []

    # Control de estabilizar
    _hist_dx.clear()
    _hist_dy.clear()
    _prev_move_sign["x"] = _prev_move_sign["y"] = 0
    _same_sign_run["x"] = _same_sign_run["y"] = 0

    global _ema_dx, _ema_dy
    _ema_dx = 0.0
    _ema_dy = 0.0

    # - LEER VÍDEO -
    # Estabilizar vídeo usando plantilla periocular (párpados+comisuras) en
GRAY+CLAHE.
    # - Evita usar el iris como referencia (robusto a nistagmo).
    # - Usa plantilla que engloba ambos ojos si están disponibles; si no,
usa uno.
    cap = cv2.VideoCapture(ruta_video)  # Abrir vídeo de entrada
    if not cap.isOpened():
```

```
        print("No se pudo leer el vídeo.")  # Reportar error de lectura
        return None, None

    fps = cap.get(cv2.CAP_PROP_FPS) or 30.0  # Medir fps del vídeo
    W = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    H = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))  # Medir tamaño
    ok, first = cap.read()  # Leer primer frame
    if not ok:
        print("No se pudo leer el vídeo.")  # Reportar error de lectura
        cap.release()
        return None, None

    # - DETECTAR OJOS INICIALES -
    multi0 = detectar_ojos_e_iris_multi(
        first, lado_fijo=None
    )  # Usar detección multi-ojos
    L0 = multi0.get("left")
    R0 = multi0.get("right")  # Extraer entradas

    # - VALIDAR BBOXES INICIALES -
    # Márgenes para ROI
    margin_mono = 0.22  # Ojo único (recuadro más apretado)
    margin_both = 0.30  # Dos ojos (un poco más amplio)

    # Validar bboxes reales desde los resultados iniciales
    bL0 = (
        _bbox_from_pts_or_bbox(L0, W, H, margin=margin_both) if L0 is not
None else None
    )
    bR0 = (
        _bbox_from_pts_or_bbox(R0, W, H, margin=margin_both) if R0 is not
None else None
    )

    # Umbral mínimo para considerar un ROI utilizable
    MIN_W, MIN_H = 12, 8  # Tamaño mínimo de ancho y alto
    MIN_AREA = 96  # Área mínima
    validL = (
        bL0 is not None
        and bL0[2] >= MIN_W
        and bL0[3] >= MIN_H
        and (bL0[2] * bL0[3] >= MIN_AREA)
    )
    validR = (
        bR0 is not None
```

```
        and bR0[2] >= MIN_W
        and bR0[3] >= MIN_H
        and (bR0[2] * bR0[3] >= MIN_AREA)
    )

    # - FUNCIONES DE APOYO PARA VALIDACIÓN -
    # Heurísticas anti-duplicados/falsos positivos cuando ambos parecen
válidos
    def _area(b):
        return 0 if b is None else (b[2] * b[3])

    def _center(b):
        if b is None:
            return (None, None)
        return (b[0] + b[2] * 0.5, b[1] + b[3] * 0.5)

    def _iou(a, b):
        if a is None or b is None:
            return 0.0
        ax1, ay1, aw, ah = a
        ax2, ay2 = ax1 + aw, ay1 + ah
        bx1, by1, bw, bh = b
        bx2, by2 = bx1 + bw, by1 + bh
        ix1, iy1 = max(ax1, bx1), max(ay1, by1)
        ix2, iy2 = min(ax2, bx2), min(ay2, by2)
        iw, ih = max(0, ix2 - ix1), max(0, iy2 - iy1)
        inter = iw * ih
        if inter <= 0:
            return 0.0
        union = (aw * ah) + (bw * bh) - inter
        return inter / max(1.0, union)

    if validL and validR:
        areaL, areaR = _area(bL0), _area(bR0)
        cxL, cyL = _center(bL0)
        cxR, cyR = _center(bR0)
        iouLR = _iou(bL0, bR0)

        # Muy solapados o centros demasiado cerca → probablemente el mismo
ojo
        too_close_x = (cxL is not None and cxR is not None) and abs(
            cxL - cxR
        ) < 0.25 * max(bL0[2], bR0[2])
        big_overlap = iouLR > 0.20
```

```
        # Uno demasiado pequeño respecto al otro ⇒ descartar el pequeño
        tiny_vs_big = min(areaL, areaR) < 0.40 * max(areaL, areaR)

        # Orden incoherente (izquierdo a la derecha del derecho) → se queda
el más grande
        wrong_order = (cxL is not None and cxR is not None) and (cxL >= cxR)

        if big_overlap or too_close_x or tiny_vs_big or wrong_order:
            if areaL >= areaR:
                validR = False
            else:
                validL = False

    # - DETERMINAR MODO (MONOCULAR O BINOCULAR) -
    # Decidir modo usando SOLO bboxes válidos (evitar ojos fantasma)
    if ojos == "both" and validL and validR:
        modo = "both"
    elif ojos == "mono":
        if validL and validR:
            if prefer_side in ("left", "right"):
                modo = prefer_side
            else:
                areaL = bL0[2] * bL0[3]
                areaR = bR0[2] * bR0[3]
                modo = "left" if areaL >= areaR else "right"
        elif validL:
            modo = "left"
        elif validR:
            modo = "right"
        else:
            print("No se detectó ojo utilizable en el primer frame.")
            cap.release()
            return None, None
    else:
        # Auto: ambos si hay 2 válidos; si no, uno; si ninguno, error
        if validL and validR:
            modo = "both"
        elif validL:
            modo = "left"
        elif validR:
            modo = "right"
        else:
            print("No se detectó ojo utilizable en el primer frame.")
            cap.release()
            return None, None
```

```
    # - CONSTRUIR ROI PERIOCULAR -
    # Construir bbox(s) perioculares según modo (usar margen adecuado)
    if modo == "both":  # Ambos
        # Recalcular bboxes con margen para binocular
        bL0 = (
            _bbox_from_pts_or_bbox(L0, W, H, margin=margin_both)
            if L0 is not None
            else None
        )
        bR0 = (
            _bbox_from_pts_or_bbox(R0, W, H, margin=margin_both)
            if R0 is not None
            else None
        )
        bbox0 = _union_bboxes(bL0, bR0)
    elif modo == "left":  # Left
        # Recalcular con margen apretado para monocular
        bL0 = (
            _bbox_from_pts_or_bbox(L0, W, H, margin=margin_mono)
            if L0 is not None
            else None
        )
        bbox0 = bL0
    else:  # Right
        bR0 = (
            _bbox_from_pts_or_bbox(R0, W, H, margin=margin_mono)
            if R0 is not None
            else None
        )
        bbox0 = bR0

    if bbox0 is None or bbox0[2] == 0 or bbox0[3] == 0:
        print("No se pudo construir ROI periocular inicial.")  # Reportar
fallo de ROI
        cap.release()
        return None, None

    # - INICIALIZAR PLANTILLA PERIOCULAR -
    # Inicializar plantilla periocular con referencia fija
    tpl_gray, bbox_ref, ref_tl = _init_periocular_template(
        first, L0 if modo != "right" else None, R0 if modo != "left" else
None
    )  # Usar L/R según modo
    prev_bbox = bbox_ref  # ← SE FIJA UNA SOLA VEZ, NO SE VUELVE A CAMBIAR
```

```
    if tpl_gray is None:
        print(
            "No se pudo inicializar plantilla periocular."
        )  # Reportar fallo de plantilla
        cap.release()
        return None, None

    prev_bbox = bbox_ref  # Guardar bbox previo

    # - CONFIGURAR SALIDAS DE VÍDEO -
    base, ext = os.path.splitext(ruta_video)  # Separar ruta y extensión
    salida = f"{base}_estabilizado{ext}"  # Generar nombre de salida
    fourcc = cv2.VideoWriter_fourcc(*"mp4v")  # Seleccionar códec mp4
    writer = cv2.VideoWriter(salida, fourcc, fps, (W, H))  # Crear escritor
de vídeo

    # Vídeo con recuadro
    salida_dbg = f"{base}_estabilizado_debug{ext}"
    writer_dbg = cv2.VideoWriter(salida_dbg, fourcc, fps, (W, H))

    writer.write(first)  # Escribir primer frame (definido como referencia)

    # Vídeo con recuadro
    vis0 = first.copy()
    cv2.rectangle(
        vis0,
        (ref_tl[0], ref_tl[1]),
        (ref_tl[0] + bbox_ref[2], ref_tl[1] + bbox_ref[3]),
        (0, 255, 0),
        2,
    )
    writer_dbg.write(vis0)

    # - MOSTRAR PRIMER FRAME (DEBUG) -
    if debug:
        show_letterboxed(
            first, win="Estabilizar", screen_scale=0.92,
fullscreen=fullscreen
        )
        cv2.waitKey(1)

    # - PROCESAR FRAMES -
    frame_idx = 0  # Contador de frames
    last_score = 1.0  # score inicial "bueno" para que el primer pad sea 40
    last_redetect_idx = -(
```

```
        10**9
    )  # Muy negativo para que el primer uso no esté bloqueado
    while True:  # Procesar frames
        ok, frame = cap.read()  # Leer siguiente frame
        if not ok:
            break

        # Opción: re-detectar cada N frames para robustez
        need_redetect = (last_score < _MIN_TM_KEEP) and (
            (frame_idx - last_redetect_idx) >= _REDETECT_GAP
        )
        cur_tl = None  # Inicializar esquina actual

        # - RE-DETECCIÓN (SI ES NECESARIO) -
        if need_redetect:
            multi = detectar_ojos_e_iris_multi(frame, lado_fijo=None)
            L = multi.get("left")
            R = multi.get("right")
            if modo == "both" and L is not None and R is not None:
                bL = _bbox_from_pts_or_bbox(L, W, H, margin=margin_both)
                bR = _bbox_from_pts_or_bbox(R, W, H, margin=margin_both)
                b = _union_bboxes(bL, bR)
            elif modo == "left" and L is not None:
                b = _bbox_from_pts_or_bbox(L, W, H, margin=margin_mono)
            elif modo == "right" and R is not None:
                b = _bbox_from_pts_or_bbox(R, W, H, margin=margin_mono)
            else:
                b = None

            # --- Anti-salto de re-detección (comparado con referencia fija)
---
            if b is not None:
                if (
                    abs(b[0] - ref_tl[0]) > _MAX_RELOC
                    or abs(b[1] - ref_tl[1]) > _MAX_RELOC
                ):
                    # Demasiado lejos de la referencia → ignorar esta re-
detección
                    b = None

            if b is not None and b[2] > 0 and b[3] > 0:
                cur_tl = (b[0], b[1])  # Fijar esquina actual con detección
                last_redetect_idx = frame_idx
                last_score = 1.0
```

```
        # - MATCHING LOCAL DE PLANTILLA -
        if cur_tl is None:
            # Matching local alrededor del bbox previo
            pad = 28 if last_score >= _MIN_TM_SCORE else 56

            # Matching local con pad dinámico
            x, y, score = _match_near_gray_clahe(
                frame, tpl_gray, bbox_ref, search_pad=pad
            )

            if score >= _MIN_TM_SCORE:
                cur_tl = (x, y)
            else:
                # Score bajo → intentar detector PERO con márgenes correctos
y sin “saltos” grandes
                multi = detectar_ojos_e_iris_multi(frame, lado_fijo=None)
                L = multi.get("left")
                R = multi.get("right")

                if modo == "both" and L is not None and R is not None:
                    bL = _bbox_from_pts_or_bbox(L, W, H, margin=margin_both)
                    bR = _bbox_from_pts_or_bbox(R, W, H, margin=margin_both)
                    b = _union_bboxes(bL, bR)
                elif modo == "left" and L is not None:
                    b = _bbox_from_pts_or_bbox(L, W, H, margin=margin_mono)
                elif modo == "right" and R is not None:
                    b = _bbox_from_pts_or_bbox(R, W, H, margin=margin_mono)
                else:
                    b = None

                if b is not None and b[2] > 0 and b[3] > 0:
                    cand_tl = (b[0], b[1])

                    # Evaluar salto candidato: no aceptar desplazamientos
enormes (antisalto)
                    dx_c = ref_tl[0] - cand_tl[0]
                    dy_c = ref_tl[1] - cand_tl[1]
                    if abs(dx_c) <= 2 * _MAX_DXY and abs(dy_c) <= 2 *
_MAX_DXY:
                        cur_tl = cand_tl

            # Guardar score para decidir el pad del próximo frame
            last_score = score

        # - FRAME SIN INFORMACIÓN FIABLE -
```

```
        if cur_tl is None:
            # Sin información fiable este frame → escribir tal cual
            writer.write(frame)  # Mantener continuidad temporal
            if debug:
                show_letterboxed(
                    frame, win="Estabilizar", screen_scale=0.92,
fullscreen=fullscreen
                )
                if (cv2.waitKey(1) & 0xFF) == 27:
                    break
            frame_idx += 1  # Avanzar contador
            continue

        # - CALCULAR DESPLAZAMIENTO -
        # Calcular delta para llevar cur_tl → ref_tl (traslación pura)
        dx = ref_tl[0] - cur_tl[0]  # Delta crudo en x
        dy = ref_tl[1] - cur_tl[1]  # Delta crudo en y

        # - SUAVIZAR Y LIMITAR MOVIMIENTO -
        dx, dy = _smooth_and_clamp(
            dx, dy
        )  # Suavizar + limitar salto por frame (antisaltos)

        # Histéresis tipo Schmitt: no mover si error es pequeño

        # - APLICAR HISTÉRESIS Y FILTROS ANTI-RACHA -
        # (Usar dx,dy ya suavizados como corrección propuesta)
        def _apply_hysteresis(val, axis):
            # 1) Zona muerta estricta
            if abs(val) <= _DEADBAND_LO:
                return 0.0

            # 2) Banda intermedia: solo mover si mantiene signo (evita
serrucho)
            if abs(val) < _DEADBAND_HI:
                sign = 1 if val > 0 else -1
                if _prev_move_sign[axis] != 0 and sign !=
_prev_move_sign[axis]:
                    return 0.0  # No cambiar de sentido en banda intermedia

            return val

        dx = _apply_hysteresis(dx, "x")
        dy = _apply_hysteresis(dy, "y")
```

```
        # Anti-racha: cortar carreras largas en el mismo sentido
        def _update_run(val, axis):
            if val == 0.0:
                _same_sign_run[axis] = 0
                return val
            sign = 1 if val > 0 else -1
            if sign == _prev_move_sign[axis]:
                _same_sign_run[axis] += 1
                if _same_sign_run[axis] >= _RUN_MAX:
                    _same_sign_run[axis] = 0
                    return 0.0  # Forzar pausa (corta el "piloto
automático")
            else:
                _same_sign_run[axis] = 1
                _prev_move_sign[axis] = sign
            return val

        dx = _update_run(dx, "x")
        dy = _update_run(dy, "y")

        # Clamp final de seguridad (por si algo pasó el filtro)
        dx = max(-_MAX_DXY, min(_MAX_DXY, dx))
        dy = max(-_MAX_DXY, min(_MAX_DXY, dy))

        # Deadband para eliminar microtemblores
        if abs(dx) < _DEADBAND:
            dx = 0.0
        if abs(dy) < _DEADBAND:
            dy = 0.0

        # Control de estabilizar
        cum_dx += dx
        cum_dy += dy
        stab_log.append((frame_idx, dx, dy, cum_dx, cum_dy))

        # - ACTUALIZAR Y GUARDAR FRAME ESTABILIZADO -
        stab = _translate(frame, dx, dy)  # Aplicar traslación estable
        writer.write(stab)  # Guardar frame estabilizado

        # Vídeo con recuadro
        vis = stab.copy()
        cv2.rectangle(
            vis,
            (ref_tl[0], ref_tl[1]),
            (ref_tl[0] + bbox_ref[2], ref_tl[1] + bbox_ref[3]),
```

```
            (0, 255, 0),
            2,
        )
        writer_dbg.write(vis)

        if debug:
            vis = stab.copy()
            cv2.rectangle(
                vis,
                (ref_tl[0], ref_tl[1]),
                (ref_tl[0] + bbox_ref[2], ref_tl[1] + bbox_ref[3]),
                (0, 255, 0),
                2,
            )
            show_letterboxed(
                vis, win="Estabilizar", screen_scale=0.92,
fullscreen=fullscreen
            )
            if (cv2.waitKey(1) & 0xFF) == 27:
                break

        frame_idx += 1  # Incrementar contador

    # - CERRAR Y GUARDAR RESULTADOS -
    cap.release()
    writer.release()
    writer_dbg.release()  # Liberar recursos
    if debug:
        cv2.destroyAllWindows()  # Cerrar ventanas

    # - GUARDAR REGISTRO DE DESPLAZAMIENTOS -
    # Guardar log de estabilización junto al vídeo de salida
    # Control de estabilizar
    log_csv = f"{base}_estabilizado_deltas.csv"
    with open(log_csv, "w", newline="") as f:
        w = csv.writer(f)
        w.writerow(["frame", "dx", "dy", "cum_dx", "cum_dy"])
        w.writerows(stab_log)

    print(f"Log de estabilización -> {os.path.basename(log_csv)}")
    print(f"Debug de estabilización -> {os.path.basename(salida_dbg)}")
    print(f"Vídeo estabilizado -> {os.path.basename(salida)}")

    print("☞ Estabilización ejecutada. Datos guardados\n")
    return salida, (
```

```
        modo if modo in ("left", "right") else None
    )  # Devolver "left"/"right" o None (para both)


# - REINICIAR ESTADO GLOBAL -
# Limpieza de estado
def reset_corregir_movimiento_state():  # Reiniciar estado global del módulo
ROI
    reset_roi_state()  # Usar reset del módulo roi
```

Submódulo especializado seguimiento.py

```
import math
import os

import cv2  # Importar OpenCV
import numpy as np
from video.display import show_letterboxed
from video.roi import detectar_ojos_e_iris_multi, reset_roi_state

# - PARÁMETROS (BASE ROBUSTA) -
DEBUG_OVERLAY = True  # Mostrar puntitos/bandas/métricas
SAVE_DEBUG_VIDEO = True  # Guardar MP4 con overlay en carpeta datos
POINTS_PER_BAND = 13  # Puntitos por banda (arriba/abajo/laterales)
WIN_LK = (31, 31)  # Ventana para LK Optical Flow
MAXLEVEL_LK = 3  # Niveles de pirámide para LK
RANSAC_THRESH = 3.0  # Umbral de reproyección px para RANSAC
MIN_INLIERS = 16  # Mínimo de inliers para confiar en la transformación
USE_SKIN_MASK = True  # Usar máscara de piel para evitar cabello en anclajes

# - SUAVIZADO (EMA) -
USE_EMA = True  # Usar suavizado de s,theta,tx,ty
EMA_ALPHA = 0.4  # Peso del frame actual (0.0-1.0)
LIMIT_ROT_DEG = 8.0  # Límite por cuadro para |rotación|
LIMIT_SCALE_LOW = 0.90  # Límite inferior de escala por cuadro
LIMIT_SCALE_HIGH = 1.10  # Límite superior de escala por cuadro


# - SANITIZAR BOUNDING BOXES CONTRA BORDES -
def _sanitize_bbox(frame_shape, bbox):
    if bbox is None:
        return None
    x, y, w, h = bbox
    vals = [x, y, w, h]
    if any(
        (v is None) or (isinstance(v, float) and (math.isnan(v) or
math.isinf(v)))
        for v in vals
    ):
        return None
    x, y, w, h = float(x), float(y), float(w), float(h)
    if w <= 0 or h <= 0:
        return None
    H, W = frame_shape[:2]
    x = max(0.0, min(x, W - 1.0))
```

```
    y = max(0.0, min(y, H - 1.0))
    w = max(1.0, min(w, W - x))
    h = max(1.0, min(h, H - y))
    return (x, y, w, h)


# - CREAR TRACKER CSRT (COMPATIBLE CON OPENCV LEGACY) -
def _crear_tracker_csrt():
    try:
        return cv2.TrackerCSRT_create()
    except AttributeError:
        return cv2.legacy.TrackerCSRT_create()


# - RECORTAR RECTÁNGULO A IMAGEN -
def _clip_rect(x1, y1, x2, y2, W, H):
    x1 = int(max(0, min(x1, W - 1)))
    y1 = int(max(0, min(y1, H - 1)))
    x2 = int(max(0, min(x2, W - 1)))
    y2 = int(max(0, min(y2, H - 1)))
    if x2 <= x1 or y2 <= y1:
        return None
    return (x1, y1, x2, y2)


# - APLICAR RECTÁNGULO A MÁSCARA BINARIA -
def _rect_to_mask(mask, rect):
    if rect is None:
        return
    x1, y1, x2, y2 = rect
    mask[y1:y2, x1:x2] = 255


# - MÁSCARA DE PIEL (YCrCb) -
def _skin_mask(bgr):
    ycb = cv2.cvtColor(bgr, cv2.COLOR_BGR2YCrCb)
    Y, Cr, Cb = cv2.split(ycb)
    # Rangos típicos de piel en YCrCb (suaves)
    skin = cv2.inRange(ycb, (0, 133, 77), (255, 173, 127))
    return skin

    # - ANCLAJES, BANDAS Y ESTABILIZACIÓN -


# - MONOCULAR -
# DEFINIR BANDAS ALREDEDOR DEL OJO EVITANDO PÁRPADOS Y CABELLO
```

```
def _bands_from_eye_bbox(frame_shape, eye_bbox):
    H, W = frame_shape[:2]
    x, y, w, h = eye_bbox
    x2 = x + w
    y2 = y + h
    # Bandas: top/bottom/laterales con márgenes prudentes
    top = _clip_rect(
        int(x + 0.10 * w), int(y - 0.45 * h), int(x + 0.90 * w), int(y -
0.15 * h), W, H
    )
    bot = _clip_rect(
        int(x + 0.10 * w),
        int(y2 + 0.15 * h),
        int(x + 0.90 * w),
        int(y2 + 0.45 * h),
        W,
        H,
    )
    # Laterales más estrechos (evitar cabello) y centrados en la hendidura
palpebral
    lef = _clip_rect(
        int(x - 0.25 * w), int(y + 0.25 * h), int(x - 0.10 * w), int(y +
0.75 * h), W, H
    )
    rig = _clip_rect(
        int(x2 + 0.10 * w),
        int(y + 0.25 * h),
        int(x2 + 0.25 * w),
        int(y + 0.75 * h),
        W,
        H,
    )
    return {"top": top, "bottom": bot, "left": lef, "right": rig}


# - DETECTAR ESQUINAS FUERTES EN UNA BANDA (GFTT + MÁSCARA DE PIEL) -
def _gftt_in_band(gray, band_rect, k, frame_bgr, apply_skin):
    if band_rect is None:
        return np.empty((0, 1, 2), dtype=np.float32)
    band_mask = np.zeros_like(gray, dtype=np.uint8)
    _rect_to_mask(band_mask, band_rect)
    if apply_skin:
        skin = _skin_mask(frame_bgr)
        mask = cv2.bitwise_and(band_mask, skin)
    else:
```

```
        mask = band_mask
    pts = cv2.goodFeaturesToTrack(
        gray,
        maxCorners=k,
        qualityLevel=0.008,
        minDistance=6,
        blockSize=7,
        useHarrisDetector=False,
        mask=mask,
    )
    if pts is None:
        return np.empty((0, 1, 2), dtype=np.float32)
    return np.array(pts, dtype=np.float32)


# - MONOCULAR -
# INICIALIZAR ANCLAJES PERIOCULARES POR OJO
def _init_anchors(frame, eye_bbox):
    if eye_bbox is None:
        return None
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    bands = _bands_from_eye_bbox(frame.shape, eye_bbox)

    # Usar skin-mask en laterales y top para evitar cabello/ceja
    pts_top = _gftt_in_band(gray, bands["top"], POINTS_PER_BAND, frame,
USE_SKIN_MASK)
    pts_bot = _gftt_in_band(
        gray, bands["bottom"], POINTS_PER_BAND, frame, USE_SKIN_MASK
    )
    pts_lef = _gftt_in_band(gray, bands["left"], POINTS_PER_BAND, frame,
USE_SKIN_MASK)
    pts_rig = _gftt_in_band(gray, bands["right"], POINTS_PER_BAND, frame,
USE_SKIN_MASK)

    parts, weights, labels = [], [], []
    if len(pts_lef):
        parts.append(pts_lef)
        weights += [1.5] * len(pts_lef)
        labels += ["L"] * len(pts_lef)
    if len(pts_rig):
        parts.append(pts_rig)
        weights += [1.5] * len(pts_rig)
        labels += ["R"] * len(pts_rig)
    if len(pts_bot):
        parts.append(pts_bot)
```

```
        weights += [1.2] * len(pts_bot)
        labels += ["B"] * len(pts_bot)
    if len(pts_top):
        parts.append(pts_top)
        weights += [0.8] * len(pts_top)
        labels += ["T"] * len(pts_top)
    if not parts:
        return None

    p0 = np.vstack(parts).astype(np.float32).reshape(-1, 1, 2)
    wv = np.array(weights, dtype=np.float32).reshape(-1, 1)

    return {
        "p0": p0.copy(),  # Puntos base (frame 0)
        "p_prev": p0.copy(),  # Puntos en frame previo
        "bands": bands,  # Bandas para overlay
        "weights": wv,  # Pesos informativos
        "M": None,  # Transformación actual (2x3)
        "ema": None,  # Estado EMA (s,theta,tx,ty)
        "last_draw": {  # Datos para overlay
            "p_new": None,
            "inlier_mask": None,
        },
    }


# - OPTICAL FLOW + AJUSTE DE SIMILITUD (RANSAC) -
def _update_transform(prev_gray, gray, anchors):
    if anchors is None or anchors["p_prev"] is None or
len(anchors["p_prev"]) == 0:
        return None, 0, float("inf")
    p_prev = anchors["p_prev"]
    p_new, st, err = cv2.calcOpticalFlowPyrLK(
        prev_gray,
        gray,
        p_prev,
        None,
        winSize=WIN_LK,
        maxLevel=MAXLEVEL_LK,
        criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 30,
0.01),
    )
    if p_new is None or st is None:
        return None, 0, float("inf")
```

```
    good_new = p_new[st.flatten() == 1].reshape(-1, 1, 2)
    good_base = anchors["p0"][st.flatten() == 1].reshape(-1, 1, 2)
    if len(good_new) < 4:
        anchors["p_prev"] = p_new if p_new is not None else p_prev
        anchors["last_draw"]["p_new"] = good_new
        anchors["last_draw"]["inlier_mask"] = None
        return None, len(good_new), float("inf")

    M, inliers = cv2.estimateAffinePartial2D(
        good_base,
        good_new,
        method=cv2.RANSAC,
        ransacReprojThreshold=RANSAC_THRESH,
        confidence=0.99,
        refineIters=10,
    )
    ninl = int(inliers.sum()) if inliers is not None else 0

    rmse = float("inf")
    if M is not None and ninl > 0:
        A = good_base[inliers.flatten() == 1].reshape(-1, 2)
        B = good_new[inliers.flatten() == 1].reshape(-1, 2)
        A1 = np.hstack([A, np.ones((A.shape[0], 1), dtype=np.float32)])
        Bp = A1 @ M.T
        err = np.linalg.norm(Bp - B, axis=1)
        rmse = float(np.sqrt((err**2).mean()))

    anchors["p_prev"] = p_new
    anchors["M"] = M
    anchors["last_draw"]["p_new"] = good_new
    anchors["last_draw"]["inlier_mask"] = inliers
    return M, ninl, rmse


# - EXTRAER s, ϑ, tx, ty DESDE MATRIZ DE TRANSFORMACIÓN M (2x3) -
def _sim_params(M):
    if M is None or not isinstance(M, np.ndarray) or M.shape != (2, 3):
        return None, None, None, None
    a, b, tx = float(M[0, 0]), float(M[0, 1]), float(M[0, 2])
    c, d, ty = float(M[1, 0]), float(M[1, 1]), float(M[1, 2])
    theta = math.atan2(c, a)
    s = (a * a + c * c) ** 0.5
    return s, theta, tx, ty
```

```python
# - RECONSTRUIR MATRIZ DE SIMILITUD M DESDE s, ϑ, tx, ty -
def _sim_matrix(s, theta, tx, ty):
    cs = math.cos(theta)
    sn = math.sin(theta)
    return np.array([[s * cs, -s * sn, tx], [s * sn, s * cs, ty]],
dtype=np.float32)


# - NORMALIZAR DIFERENCIA ANGULAR A [-π, π] -
def _angle_diff(a, b):
    d = a - b
    while d > math.pi:
        d -= 2 * math.pi
    while d < -math.pi:
        d += 2 * math.pi
    return d


# - APLICAR INVERSA DE MATRIZ M AFÍN A UN PUNTO (x, y) -
def _apply_inv_affine(M, x, y):
    if M is None:
        return x, y
    Minv = cv2.invertAffineTransform(M)
    v = np.array([x, y, 1.0], dtype=np.float32)
    out = Minv @ v
    return float(out[0]), float(out[1])


# - DIBUJAR BANDAS Y PUNTOS DE ANCLAJE (DEPURACIÓN VISUAL) -
def _draw_anchors(frame, anchors):
    if anchors is None:
        return
    if DEBUG_OVERLAY and anchors.get("bands"):
        for r in anchors["bands"].values():
            if r is None:
                continue
            x1, y1, x2, y2 = r
            cv2.rectangle(frame, (x1, y1), (x2, y2), (100, 100, 100), 1)
    pts = anchors["last_draw"].get("p_new")
    inl = anchors["last_draw"].get("inlier_mask")
    if pts is None:
        return
    pts = pts.reshape(-1, 2)
    if inl is None:
        for x, y in pts:
            cv2.circle(frame, (int(x), int(y)), 2, (0, 0, 255), -1)
```

```
        return
    inl = inl.flatten().astype(bool)
    for (x, y), ok in zip(pts, inl):
        cv2.circle(frame, (int(x), int(y)), 2, (0, 255, 0) if ok else (0, 0,
255), -1)


# - SEGUIMIENTO -
# Orquestar detección, tracking, estabilización y guardado CSV
def seguimiento(ruta_video, modo, debug, fullscreen=False):
    print("👁 Seguimiento iniciado")

    # - INICIALIZAR VÍDEO -
    video = cv2.VideoCapture(ruta_video)
    fps_video = video.get(cv2.CAP_PROP_FPS) or 30.0
    try:
        fps_video = float(fps_video)
    except Exception:
        fps_video = 30.0
    if fps_video <= 0 or math.isnan(fps_video):
        fps_video = 30.0
    delay_ms = max(1, int(1000 / fps_video))

    # - RUTA PARA GUARDAR DATOS -
    ruta_base = os.path.abspath(os.path.join(os.path.dirname(__file__),
"..", ".."))
    ruta_datos = os.path.join(ruta_base, "datos")
    os.makedirs(ruta_datos, exist_ok=True)

    trayectoria_L, trayectoria_R = [], []

    ok, frame0 = video.read()
    if not ok:
        print("No se pudo leer el video.")
        _guardar_csv(trayectoria_L, trayectoria_R, ruta_datos)
        return

    print(f"Modo: {modo}")

    # - LEER FRAME INICIAL Y DETECTAR OJOS/IRIS (FRAME 0) -
    datos_multi = detectar_ojos_e_iris_multi(frame0)
    intentos = 0
    while (not datos_multi["left"] and not datos_multi["right"]) and
intentos < 10:
        ok, frame0 = video.read()
```

```
        if not ok:
            print("No hay más frames para intentar detección.")
            _guardar_csv(trayectoria_L, trayectoria_R, ruta_datos)
            return
        datos_multi = detectar_ojos_e_iris_multi(frame0)
        intentos += 1
    if not datos_multi["left"] and not datos_multi["right"]:
        print("No se pudo detectar ningún ojo/iris.")
        _guardar_csv(trayectoria_L, trayectoria_R, ruta_datos)
        return

    # - CREAR TRACKERS CSRT PARA CADA IRIS -
    tracker_L = tracker_R = None
    eye_bbox_L = eye_bbox_R = None

    if modo in ("both", "left") and datos_multi["left"]:
        ib = _sanitize_bbox(frame0.shape, datos_multi["left"]["iris_bbox"])
        eb = _sanitize_bbox(frame0.shape, datos_multi["left"]["eye_bbox"])
        if ib and ib[2] >= 12 and ib[3] >= 8:
            tracker_L = _crear_tracker_csrt()
            tracker_L.init(frame0, tuple(int(v) for v in ib))
            eye_bbox_L = eb
        else:
            print("ROI izquierdo inválido, se omite.")

    if modo in ("both", "right") and datos_multi["right"]:
        ib = _sanitize_bbox(frame0.shape, datos_multi["right"]["iris_bbox"])
        eb = _sanitize_bbox(frame0.shape, datos_multi["right"]["eye_bbox"])
        if ib and ib[2] >= 12 and ib[3] >= 8:
            tracker_R = _crear_tracker_csrt()
            tracker_R.init(frame0, tuple(int(v) for v in ib))
            eye_bbox_R = eb
        else:
            print("ROI derecho inválido, se omite.")

    # - INICIALIZAR ANCLAJES PERIOCULARES (FRAME 0) -
    anchors_L = _init_anchors(frame0, eye_bbox_L) if tracker_L and
eye_bbox_L else None
    anchors_R = _init_anchors(frame0, eye_bbox_R) if tracker_R and
eye_bbox_R else None
    prev_gray = cv2.cvtColor(frame0, cv2.COLOR_BGR2GRAY)

    # - CONFIGURAR VIDEO DE SALIDA (DEBUG VISUAL) -
    if SAVE_DEBUG_VIDEO:
        fourcc = cv2.VideoWriter_fourcc(*"mp4v")
```

```
        ruta_video_out = os.path.join(ruta_datos, "video_seguimiento.mp4")
        video_salida = cv2.VideoWriter(
            ruta_video_out, fourcc, fps_video, (frame0.shape[1],
frame0.shape[0])
        )
    else:
        video_salida = None

    # - INICIALIZAR ESTADOS DE SEGUIMIENTO -
    estado_L = estado_R = False
    avg_L = avg_R = 0.0
    cx_prev_L = cy_prev_L = None
    cx_prev_R = cy_prev_R = None
    origin_L = origin_R = None

    # - BUCLE PRINCIPAL DE SEGUIMIENTO -
    frame_idx = 0
    while True:
        ok, frame = video.read()
        if not ok:
            break
        frame_idx += 1
        t_frame = frame_idx / fps_video
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

        # - ACTUALIZAR TRACKERS (L/R) -
        cx_L = cy_L = None
        cx_R = cy_R = None

        if tracker_L:
            ok_L, bbL = tracker_L.update(frame)
            if ok_L:
                xf, yf, wf, hf = bbL
                cx_L, cy_L = xf + wf / 2.0, yf + hf / 2.0
                if DEBUG_OVERLAY:
                    cv2.rectangle(
                        frame,
                        (int(xf), int(yf)),
                        (int(xf + wf), int(yf + hf)),
                        (0, 255, 0),
                        2,
                    )
                    cv2.circle(frame, (int(cx_L), int(cy_L)), 4, (255, 0,
0), -1)
```

```
        if tracker_R:
            ok_R, bbR = tracker_R.update(frame)
            if ok_R:
                xf, yf, wf, hf = bbR
                cx_R, cy_R = xf + wf / 2.0, yf + hf / 2.0
                if DEBUG_OVERLAY:
                    cv2.rectangle(
                        frame,
                        (int(xf), int(yf)),
                        (int(xf + wf), int(yf + hf)),
                        (0, 255, 255),
                        2,
                    )
                    cv2.circle(frame, (int(cx_R), int(cy_R)), 4, (0, 0,
255), -1)

        # - ACTUALIZAR ANCLAJES Y ESTIMAR MATRIZ DE TRANSFORMACIÓN M -
        M_L, ninl_L, rmse_L = None, 0, float("inf")
        M_R, ninl_R, rmse_R = None, 0, float("inf")
        if anchors_L:
            M_L, ninl_L, rmse_L = _update_transform(prev_gray, gray,
anchors_L)
        if anchors_R:
            M_R, ninl_R, rmse_R = _update_transform(prev_gray, gray,
anchors_R)

        # - SUAVIZAR Y LIMITAR TRANSFORMACIÓN (EMA) -
        def _smooth_and_limit(anchors, M):
            if M is None:
                return None
            s, th, tx, ty = _sim_params(M)
            if s is None:
                return None
            if USE_EMA:
                if anchors["ema"] is None:
                    anchors["ema"] = (s, th, tx, ty)
                else:
                    ps, pth, ptx, pty = anchors["ema"]
                    dth = _angle_diff(th, pth)
                    th = pth + EMA_ALPHA * dth
                    s = ps + EMA_ALPHA * (s - ps)
                    tx = ptx + EMA_ALPHA * (tx - ptx)
                    ty = pty + EMA_ALPHA * (ty - pty)
                    anchors["ema"] = (s, th, tx, ty)
            # Limitar por cuadro
```

```
            th = max(-math.radians(LIMIT_ROT_DEG),
min(math.radians(LIMIT_ROT_DEG), th))
            s = max(LIMIT_SCALE_LOW, min(LIMIT_SCALE_HIGH, s))
            return _sim_matrix(s, th, tx, ty)

        M_L_used = _smooth_and_limit(anchors_L, M_L) if anchors_L else None
        M_R_used = _smooth_and_limit(anchors_R, M_R) if anchors_R else None

        # - ESTABILIZAR IRIS Y GUARDAR DESPLAZAMIENTOS Δ -
        if tracker_L and cx_L is not None:
            if anchors_L and M_L_used is not None and ninl_L >= MIN_INLIERS:
                x_est, y_est = _apply_inv_affine(M_L_used, cx_L, cy_L)
            else:
                x_est, y_est = cx_L, cy_L
            if origin_L is None:
                origin_L = (x_est, y_est)
            trayectoria_L.append((t_frame, x_est - origin_L[0], y_est -
origin_L[1]))

        if tracker_R and cx_R is not None:
            if anchors_R and M_R_used is not None and ninl_R >= MIN_INLIERS:
                x_est, y_est = _apply_inv_affine(M_R_used, cx_R, cy_R)
            else:
                x_est, y_est = cx_R, cy_R
            if origin_R is None:
                origin_R = (x_est, y_est)
            trayectoria_R.append((t_frame, x_est - origin_R[0], y_est -
origin_R[1]))

        # - MOSTRAR OVERLAY DE ANCLAJES Y MÉTRICAS -
        if DEBUG_OVERLAY:
            if anchors_L:
                _draw_anchors(frame, anchors_L)
            if anchors_R:
                _draw_anchors(frame, anchors_R)
            # Métricas
            h = frame.shape[0]
            ox, oy = 10, h - 30

            def _lbl(img, text, x, y, color=(255, 255, 255)):
                (tw, th), _ = cv2.getTextSize(text,
cv2.FONT_HERSHEY_SIMPLEX, 0.6, 2)
                cv2.rectangle(img, (x, y - th - 6), (x + tw + 10, y + 6),
(0, 0, 0), -1)
                cv2.putText(
```

```
                    img,
                    text,
                    (x + 5, y),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    0.6,
                    color,
                    2,
                    cv2.LINE_AA,
                )
                return y + th + 14

            oy = _lbl(frame, f"L  inl={ninl_L:02d}  rmse={rmse_L:.2f}", ox,
oy)
            oy = _lbl(frame, f"R  inl={ninl_R:02d}  rmse={rmse_R:.2f}", ox,
oy)

        # - MOSTRAR VIDEO -
        if debug:
            show_letterboxed(
                frame,
                win="Seguimiento de iris",
                screen_scale=0.92,
                fullscreen=fullscreen,
            )
            cv2.waitKey(1)

            # - SALIR CON ESC -
            if cv2.waitKey(delay_ms) & 0xFF == 27:
                print("ESC presionado, guardando datos y saliendo...")
                break

        # - GUARDAR VIDEO -
        if SAVE_DEBUG_VIDEO and video_salida is not None:
            video_salida.write(frame)

        # - PREPARAR SIGUIENTE ITERACIÓN -
        prev_gray = gray
        if tracker_L and cx_L is not None and cy_L is not None:
            cx_prev_L, cy_prev_L = cx_L, cy_L
        if tracker_R and cx_R is not None and cy_R is not None:
            cx_prev_R, cy_prev_R = cx_R, cy_R

    # - CIERRE -
    video.release()
    if SAVE_DEBUG_VIDEO and video_salida is not None:
```

```
        video_salida.release()
    cv2.destroyAllWindows()
    reset_roi_state()

    # - GUARDAR CSV (PARA proceso.py) -
    _guardar_csv(trayectoria_L, trayectoria_R, ruta_datos)

    print(f"Vídeo de seguimiento -> {os.path.basename(ruta_video_out)}")

    ruta_csv_L = os.path.join(ruta_datos, "datos_seguimiento_L.csv")
    ruta_csv_R = os.path.join(ruta_datos, "datos_seguimiento_R.csv")
    if os.path.exists(ruta_csv_L):
        print(f"Log de seguimiento (L) -> {os.path.basename(ruta_csv_L)}")
    if os.path.exists(ruta_csv_R):
        print(f"Log de seguimiento (R) -> {os.path.basename(ruta_csv_R)}")

    print("👁 Seguimiento ejecutado\n")


# - GUARDAR CSV DE SEGUIMIENTO -
# Guardar desplazamientos Δx,Δy en datos_seguimiento_L/R.csv
def _guardar_csv(tray_L, tray_R, ruta_datos):
    ruta_csv_L = os.path.join(ruta_datos, "datos_seguimiento_L.csv")
    ruta_csv_R = os.path.join(ruta_datos, "datos_seguimiento_R.csv")

    if tray_L and len(tray_L) >= 3:
        with open(ruta_csv_L, "w") as f:
            f.write("tiempo,x,y\n")
            for t, dx, dy in tray_L:
                f.write(f"{t:.3f},{dx:.1f},{dy:.1f}\n")
    else:
        if os.path.exists(ruta_csv_L):
            os.remove(ruta_csv_L)

    if tray_R and len(tray_R) >= 3:
        with open(ruta_csv_R, "w") as f:
            f.write("tiempo,x,y\n")
            for t, dx, dy in tray_R:
                f.write(f"{t:.3f},{dx:.1f},{dy:.1f}\n")
    else:
        if os.path.exists(ruta_csv_R):
            os.remove(ruta_csv_R)
```

Submódulo auxiliar roi.py

```
import cv2
import mediapipe as mp
import numpy as np

# - CONSTANTES DE DETECCIÓN -
mp_face = mp.solutions.face_mesh  # Crea acceso rápido al módulo FaceMesh de
MediaPipe
# Índices de puntos clave para ojo e iris en malla facial
LEFT_EYE_IDX = [33, 133, 159, 145, 153, 154, 155, 246]  # Contorno ojo izq
LEFT_IRIS_IDX = [468, 469, 470, 471]  # iris izq
RIGHT_EYE_IDX = [362, 263, 386, 374, 380, 381, 382, 466]  # Contorno ojo der
RIGHT_IRIS_IDX = [473, 474, 475, 476]  # iris der

# - VARIABLES GLOBALES (ESTADO DE DETECCIÓN) -
# Guardar lado fijo y última caja detectada
_lado_fijo = None  # Guardar elegido
_ultimo_bbox_ojo = None
_ultimo_bbox_iris = None
_ultimo_iris_data = None

# Cache por ojo (para modo binocular y/o cuando falte un lado)
_ultimo_bbox_ojo_L = None  # Última caja ojo izquierdo
_ultimo_bbox_iris_L = None  # Última caja iris izquierdo
_ultimo_iris_data_L = None  # Último (cx,cy,r) izquierdo

_ultimo_bbox_ojo_R = None  # Última caja ojo derecho
_ultimo_bbox_iris_R = None  # Última caja iris derecho
_ultimo_iris_data_R = None  # Último (cx,cy,r) derecho

UMBRAL_AREA = 3000  # Ojo chico => close-up (ajusta según tu resolución)
MODO_ANALISIS = "auto"  # "auto" | "binocular" | "monocular"


# - CREAR RECUADRO ALREDEDOR DEL OJO -
def _bbox_from_points(points, img_w, img_h, margin=0.3):
    # Calcular recuadro alrededor del ojo a partir de puntos,
    # ampliar un margen y ajustar para que no se salga de la imagen

    pts = np.array(points, dtype=np.float32)  # Lista → array NumPy
    x1, y1 = pts.min(axis=0)  # Punto mínimo (esquina sup izq)
    x2, y2 = pts.max(axis=0)  # Punto máximo (esquina inf der)
    w, h = x2 - x1, y2 - y1  # Ancho y alto
    # Ampliar caja con margen
```

```
    x1 -= w * margin
    y1 -= h * margin  # Restar sup izq
    x2 += w * margin
    y2 += h * margin  # Restar inf der
    # Limitar caja a bordes de imagen
    x1 = int(max(0, x1))
    y1 = int(max(0, y1))
    x2 = int(min(img_w - 1, x2))
    y2 = int(min(img_h - 1, y2))
    return (x1, y1, x2 - x1, y2 - y1)  # (x, y, ancho, alto)


# - CALCULAR CENTRO Y RADIO -
def _center_radius(points):
    pts = np.array(points, dtype=np.float32)  # Lista → array
    cx, cy = pts.mean(axis=0)  # Centro promedio
    r = np.mean(np.linalg.norm(pts - [cx, cy], axis=1))  # Radio medio
    return float(cx), float(cy), float(r)  # Centro y radio


# - MONOCULAR -
# - GENERAR BBOX A PARTIR DEL CENTRO DEL IRIS -
# Helper para crear un bbox a partir del centro del iris y su radio
def _bbox_from_center(cx, cy, r, m=2.5, w=0, h=0):
    # Generar rectángulo (x,y,w,h) alrededor de iris con margen m*r,
recortado a la imagen
    x1 = int(max(0, cx - m * r))
    y1 = int(max(0, cy - m * r))
    x2 = int(min(w - 1, cx + m * r))
    y2 = int(min(h - 1, cy + m * r))
    return (x1, y1, x2 - x1, y2 - y1)


# - MONOCULAR -
# - DETECCIÓN DE PUPILA POR HOUGH (FALLBACK) -
# Fallback con HoughCircles para detectar pupila cuando MediaPipe no da
landmarks (close-up de un solo ojo)
def _fallback_pupil_hough(frame_bgr, prev=None):
    # Normalizar contraste y suavizar para resaltar pupila
    g = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2GRAY)
    g = cv2.equalizeHist(g)
    g = cv2.medianBlur(g, 5)
    # Buscar círculos oscuros (pupila). Ajustar min/maxRadius video cambia
de escala
    circles = cv2.HoughCircles(
        g,
```

```
        cv2.HOUGH_GRADIENT,
        dp=1.2,
        minDist=30,
        param1=80,
        param2=18,
        minRadius=5,
        maxRadius=120,
    )
    if circles is None:
        return None
    circles = np.uint16(np.around(circles[0]))
    if prev is not None:
        px, py = prev
        # El círculo más cercano al anterior para dar estabilidad temporal
        c = min(circles, key=lambda c: (c[0] - px) ** 2 + (c[1] - py) ** 2)
    else:
        # Al iniciar, el más grande suele ser más estable
        c = max(circles, key=lambda c: c[2])
    return int(c[0]), int(c[1]), int(c[2])


# - ESTRUCTURA DE DATOS POR OJO -
# Construir estructura de datos por ojo de forma consistente
def _make_eye_entry(bbox_ojo, bbox_iris, cx, cy, r, monocular_flag):
    # Devolver dict estándar para un ojo
    return {
        "eye_bbox": bbox_ojo,  # (x,y,w,h) ojo
        "iris_bbox": bbox_iris,  # (x,y,w,h) iris
        "center": (float(cx), float(cy)),  # Centro (cx,cy) iris
        "radius": float(r),  # Radio iris
        "monocular": bool(monocular_flag),  # Flag de modo monocular
    }


# - BINOCULAR -
# - DETECTAR OJOs E IRIS -
def detectar_ojos_e_iris_multi(frame_bgr, lado_fijo=None):
    # Detecta ambos ojos cuando existan. Si solo hay uno, devuelve solo ese.
    # Mantiene caché por lado para tolerar pérdidas breves.
    global _lado_fijo
    global _ultimo_bbox_ojo_L, _ultimo_bbox_iris_L, _ultimo_iris_data_L
    global _ultimo_bbox_ojo_R, _ultimo_bbox_iris_R, _ultimo_iris_data_R

    h, w = frame_bgr.shape[:2]  # Dimensiones del frame
```

```
    # Respetar lado forzado si se pasa
    if lado_fijo is not None:
        _lado_fijo = lado_fijo  # Guardar elección externa

    # Atajo: modo monocular forzado → usar Hough si es posible
    if MODO_ANALISIS == "monocular":
        # Elegir "lado lógico" para etiquetar el monocular (persistente)
        prefer = _lado_fijo if _lado_fijo in ("left", "right") else "right"
        prev = (
            _ultimo_iris_data_L[:2]
            if prefer == "left" and _ultimo_iris_data_L
            else (
                _ultimo_iris_data_R[:2]
                if prefer == "right" and _ultimo_iris_data_R
                else None
            )
        )
        circ = _fallback_pupil_hough(frame_bgr, prev=prev)  # Buscar pupila
        L = R = None  # Inicializar respuestas
        if circ is not None:
            cx, cy, r = circ
            bbox_iris = _bbox_from_center(cx, cy, r, m=2.0, w=w, h=h)
            bbox_ojo = _bbox_from_center(cx, cy, r, m=2.5, w=w, h=h)
            if prefer == "left":
                _ultimo_bbox_ojo_L, _ultimo_bbox_iris_L, _ultimo_iris_data_L
= (
                    bbox_ojo,
                    bbox_iris,
                    (cx, cy, r),
                )
                L = _make_eye_entry(bbox_ojo, bbox_iris, cx, cy, r,
monocular_flag=True)
            else:
                _ultimo_bbox_ojo_R, _ultimo_bbox_iris_R, _ultimo_iris_data_R
= (
                    bbox_ojo,
                    bbox_iris,
                    (cx, cy, r),
                )
                R = _make_eye_entry(bbox_ojo, bbox_iris, cx, cy, r,
monocular_flag=True)
            return {
                "left": L,
                "right": R,  # Devuelve solo ojo detectado
```

```
                "mode": {"requested": "monocular", "auto_decision":
"monocular"},
                "has_face": False,  # No dependió de cara
            }
        # Si Hough falla, continuar con FaceMesh abajo (por si hay cara
entera)

    # FaceMesh para detectar landmarks
    with mp_face.FaceMesh(
        static_image_mode=False,
        refine_landmarks=True,  # Necesario para iris
        max_num_faces=1,  # Solo 1 cara
        min_detection_confidence=0.3,
        min_tracking_confidence=0.3,
    ) as mesh:
        rgb = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2RGB)  # BGR → RGB
        res = mesh.process(rgb)  # Procesar cara
        if not res.multi_face_landmarks:  # Si no detecta
            # Reconstruir desde caché correctamente por lado
            L = (
                _make_eye_entry(
                    _ultimo_bbox_ojo_L,
                    _ultimo_bbox_iris_L,
                    _ultimo_iris_data_L[0],
                    _ultimo_iris_data_L[1],
                    _ultimo_iris_data_L[2],
                    monocular_flag=True,
                )
                if _ultimo_bbox_ojo_L and _ultimo_bbox_iris_L and
_ultimo_iris_data_L
                else None
            )
            R = (
                _make_eye_entry(
                    _ultimo_bbox_ojo_R,
                    _ultimo_bbox_iris_R,
                    _ultimo_iris_data_R[0],
                    _ultimo_iris_data_R[1],
                    _ultimo_iris_data_R[2],
                    monocular_flag=True,
                )
                if _ultimo_bbox_ojo_R and _ultimo_bbox_iris_R and
_ultimo_iris_data_R
                else None
            )
```

```
            # Si no hay caché, último intento con Hough (un ojo)
            if L is None and R is None:
                prev = None
                circ = _fallback_pupil_hough(frame_bgr, prev=prev)
                if circ is not None:
                    cx, cy, r = circ
                    bbox_iris = _bbox_from_center(cx, cy, r, m=2.0, w=w,
h=h)
                    bbox_ojo = _bbox_from_center(cx, cy, r, m=2.5, w=w, h=h)
                    if (_lado_fijo or "right") == "left":
                        _ultimo_bbox_ojo_L, _ultimo_bbox_iris_L,
_ultimo_iris_data_L = (
                            bbox_ojo,
                            bbox_iris,
                            (cx, cy, r),
                        )
                        L = _make_eye_entry(
                            bbox_ojo, bbox_iris, cx, cy, r,
monocular_flag=True
                        )
                    else:
                        _ultimo_bbox_ojo_R, _ultimo_bbox_iris_R,
_ultimo_iris_data_R = (
                            bbox_ojo,
                            bbox_iris,
                            (cx, cy, r),
                        )
                        R = _make_eye_entry(
                            bbox_ojo, bbox_iris, cx, cy, r,
monocular_flag=True
                        )

            return {
                "left": L,
                "right": R,
                "mode": {"requested": MODO_ANALISIS, "auto_decision":
"monocular"},
                "has_face": False,
            }

        # Sí hay landmarks → extraer coordenadas
        lm = res.multi_face_landmarks[0].landmark  # Landmarks cara

        def xy(i):
```

```
            return (lm[i].x * w, lm[i].y * h)  # Escalar coord a pixeles

        # Obtener puntos ojo coordenadas (píxeles)
        left_eye = [xy(i) for i in LEFT_EYE_IDX]
        right_eye = [xy(i) for i in RIGHT_EYE_IDX]
        left_iris = [xy(i) for i in LEFT_IRIS_IDX]
        right_iris = [xy(i) for i in RIGHT_IRIS_IDX]

        # Calcular área
        # Convertir lista de puntos en array de NumPy de enteros (int32).
OpenCV trabajará con datos en formato estándar.
        area_L = cv2.contourArea(np.array(left_eye, dtype=np.int32))
        area_R = cv2.contourArea(np.array(right_eye, dtype=np.int32))

        # Decisión automática de modo según tamaño aparente
        modo_auto = "binocular"  # Por defecto binocular
        if MODO_ANALISIS == "auto":
            if area_L < UMBRAL_AREA and area_R < UMBRAL_AREA:
                modo_auto = (
                    "monocular"  # Close-up (ambos chicos) → tratar como
monocular
                )

        # Construcción por ojo
        L = None
        R = None  # Inicializar

        # - OJO IZQUIERDO -
        if len(left_iris) > 0 and area_L > 0:
            cxL, cyL, rL = _center_radius(left_iris)  # Centro/radio iris L
            if MODO_ANALISIS == "monocular" or modo_auto == "monocular":
                bbox_ojo_L = _bbox_from_center(
                    cxL, cyL, rL, m=2.5, w=w, h=h
                )  # Close-up: ROI por iris
                bbox_iris_L = _bbox_from_center(
                    cxL, cyL, rL, m=2.2, w=w, h=h
                )  # Iris box
                monoL = True
            else:
                bbox_ojo_L = _bbox_from_points(
                    left_eye, w, h, margin=0.35
                )  # Cara completa: contorno ojo
                bbox_iris_L = _bbox_from_center(cxL, cyL, rL, m=2.2, w=w,
h=h)
                monoL = False
```

```
            _ultimo_bbox_ojo_L, _ultimo_bbox_iris_L, _ultimo_iris_data_L = (
                bbox_ojo_L,
                bbox_iris_L,
                (cxL, cyL, rL),
            )  # Cache
            L = _make_eye_entry(
                bbox_ojo_L, bbox_iris_L, cxL, cyL, rL, monocular_flag=monoL
            )
        else:
            # Usar caché si existe
            if _ultimo_bbox_ojo_L and _ultimo_bbox_iris_L and
_ultimo_iris_data_L:
                cxL, cyL, rL = _ultimo_iris_data_L
                L = _make_eye_entry(
                    _ultimo_bbox_ojo_L,
                    _ultimo_bbox_iris_L,
                    cxL,
                    cyL,
                    rL,
                    monocular_flag=True,
                )

        # - OJO DERECHO -
        if len(right_iris) > 0 and area_R > 0:
            cxR, cyR, rR = _center_radius(right_iris)  # Centro/radio iris R
            if MODO_ANALISIS == "monocular" or modo_auto == "monocular":
                bbox_ojo_R = _bbox_from_center(
                    cxR, cyR, rR, m=2.5, w=w, h=h
                )  # Close-up: ROI por iris
                bbox_iris_R = _bbox_from_center(
                    cxR, cyR, rR, m=2.2, w=w, h=h
                )  # Iris box
                monoR = True
            else:
                bbox_ojo_R = _bbox_from_points(
                    right_eye, w, h, margin=0.35
                )  # Cara completa: contorno ojo
                bbox_iris_R = _bbox_from_center(cxR, cyR, rR, m=2.2, w=w,
h=h)
                monoR = False
            _ultimo_bbox_ojo_R, _ultimo_bbox_iris_R, _ultimo_iris_data_R = (
                bbox_ojo_R,
                bbox_iris_R,
                (cxR, cyR, rR),
            )  # Cache
```

```
            R = _make_eye_entry(
                bbox_ojo_R, bbox_iris_R, cxR, cyR, rR, monocular_flag=monoR
            )
        else:
            # Usar caché si existe
            if _ultimo_bbox_ojo_R and _ultimo_bbox_iris_R and
_ultimo_iris_data_R:
                cxR, cyR, rR = _ultimo_iris_data_R
                R = _make_eye_entry(
                    _ultimo_bbox_ojo_R,
                    _ultimo_bbox_iris_R,
                    cxR,
                    cyR,
                    rR,
                    monocular_flag=True,
                )

        # Si sigue sin haber nada (caso raro), último intento con Hough
global
        if L is None and R is None:
            circ = _fallback_pupil_hough(frame_bgr, prev=None)
            if circ is not None:
                cx, cy, r = circ
                bbox_iris = _bbox_from_center(cx, cy, r, m=2.0, w=w, h=h)
                bbox_ojo = _bbox_from_center(cx, cy, r, m=2.5, w=w, h=h)
                # Etiquetar al lado "fijo" si existe, sino elegir por X
(derecha)
                side = _lado_fijo if _lado_fijo in ("left", "right") else
"right"
                if side == "left":
                    _ultimo_bbox_ojo_L, _ultimo_bbox_iris_L,
_ultimo_iris_data_L = (
                        bbox_ojo,
                        bbox_iris,
                        (cx, cy, r),
                    )
                    L = _make_eye_entry(
                        bbox_ojo, bbox_iris, cx, cy, r, monocular_flag=True
                    )
                else:
                    _ultimo_bbox_ojo_R, _ultimo_bbox_iris_R,
_ultimo_iris_data_R = (
                        bbox_ojo,
                        bbox_iris,
                        (cx, cy, r),
```

```
                    )
                    R = _make_eye_entry(
                        bbox_ojo, bbox_iris, cx, cy, r, monocular_flag=True
                    )

        # Decidir _lado_fijo si no está y hay ambos
        if _lado_fijo is None:
            if L and R:
                # Elegir por área mayor si ambas presentes
                _lado_fijo = "left" if area_L >= area_R else "right"
            elif L:
                _lado_fijo = "left"
            elif R:
                _lado_fijo = "right"

        return {
            "left": L,
            "right": R,
            "mode": {"requested": MODO_ANALISIS, "auto_decision":
modo_auto},
            "has_face": True,
        }

    # - DETECTAR OJO E IRIS -


def detectar_ojos_e_iris(frame_bgr, lado_fijo=None):
    global _lado_fijo, _ultimo_bbox_ojo, _ultimo_bbox_iris,
_ultimo_iris_data

    # Detectar cara con MediaPipe FaceMesh, obtiener puntos clave
(landmarks) y convertirlos a coordenadas (píxeles)
    h, w = frame_bgr.shape[
        :2
    ]  # Obtener dimensiones de imagen frame_bgr. | .shape devuelve (alto,
ancho, canales), [:2] toma alto y ancho

    # Usar la versión multi y adaptar al contrato antiguo para no romper al
llamador actual.
    multi = detectar_ojos_e_iris_multi(frame_bgr, lado_fijo=lado_fijo)  #
Obtener ambos

    # Elegir qué lado devolver manteniendo API anterior
    if lado_fijo is not None:
        _lado_fijo = lado_fijo  # Respetar orden externo
```

```
    # Decidir lado a reportar
    L = multi.get("left")
    R = multi.get("right")
    side = _lado_fijo
    if side is None:
        # Si no hay fijo, elegir el que exista o el más grande (si ambos)
        if L and R:
            side = (
                "left"
                if L["eye_bbox"][2] * L["eye_bbox"][3]
                >= R["eye_bbox"][2] * R["eye_bbox"][3]
                else "right"
            )
        elif L:
            side = "left"
        elif R:
            side = "right"

    # Preparar retorno compatible
    entry = L if side == "left" else (R if side == "right" else None)

    if entry is None:
        # Mantener tus caídas de respaldo previas (historial simple)
        if _ultimo_bbox_ojo is not None:
            return {
                "lado": _lado_fijo,
                "bbox_ojo": _ultimo_bbox_ojo,
                "bbox_iris": _ultimo_bbox_iris,
                "iris": _ultimo_iris_data,
            }
        # Sin historial → None
        return None

    # Actualizar historial simple (compat)
    _ultimo_bbox_ojo = entry["eye_bbox"]
    _ultimo_bbox_iris = entry["iris_bbox"]
    _ultimo_iris_data = (entry["center"][0], entry["center"][1],
entry["radius"])

    # Devolver datos
    return {
        "lado": side if side else _lado_fijo,  # Etiqueta lado reportado
        "bbox_ojo": entry["eye_bbox"],
        "bbox_iris": entry["iris_bbox"],
```

```
        "iris": (entry["center"][0], entry["center"][1], entry["radius"]),
        "monocular": entry["monocular"],
    }


# - REINICIAR ESTADO ROI -
def reset_roi_state():
    global _lado_fijo, _ultimo_bbox_ojo, _ultimo_bbox_iris,
_ultimo_iris_data
    _lado_fijo = None
    _ultimo_bbox_ojo = None
    _ultimo_bbox_iris = None
    _ultimo_iris_data = None

    # limpiar caches por ojo
    global _ultimo_bbox_ojo_L, _ultimo_bbox_iris_L, _ultimo_iris_data_L
    global _ultimo_bbox_ojo_R, _ultimo_bbox_iris_R, _ultimo_iris_data_R
    _ultimo_bbox_ojo_L = None
    _ultimo_bbox_iris_L = None
    _ultimo_iris_data_L = None
    _ultimo_bbox_ojo_R = None
    _ultimo_bbox_iris_R = None
    _ultimo_iris_data_R = None
```

Submódulo auxiliar `display.py`

```
import ctypes

import cv2
import numpy as np


# - OBTENER TAMAÑO DE PANTALLA -
def _get_screen_size(screen_scale):
    # Usar caché para evitar recalcular en llamadas repetidas
    if not hasattr(_get_screen_size, "_cache"):
        _get_screen_size._cache = {}

    # Retornar valor almacenado si existe
    if screen_scale in _get_screen_size._cache:
        return _get_screen_size._cache[screen_scale]

    # Obtener resolución de pantalla (ancho, alto)
    sw = ctypes.windll.user32.GetSystemMetrics(0)
    sh = ctypes.windll.user32.GetSystemMetrics(1)

    # Escalar dimensiones según factor dado
    max_w, max_h = int(sw * screen_scale), int(sh * screen_scale)
    _get_screen_size._cache[screen_scale] = (max_w, max_h)
    return max_w, max_h


# - MOSTRAR VISTA CON BARRAS NEGRAS (LETTERBOXED) -
def show_letterboxed(frame, win="Vista", screen_scale=1, fullscreen=False):
    # Crear la ventana una sola vez
    if not hasattr(show_letterboxed, "_inited"):
        cv2.namedWindow(win, cv2.WINDOW_NORMAL)
        if fullscreen:
            cv2.setWindowProperty(win, cv2.WND_PROP_FULLSCREEN,
cv2.WINDOW_FULLSCREEN)
        show_letterboxed._inited = True

    # Obtener resolución de pantalla
    sw = ctypes.windll.user32.GetSystemMetrics(0)
    sh = ctypes.windll.user32.GetSystemMetrics(1)
    max_w, max_h = int(sw * screen_scale), int(sh * screen_scale)

    # Escalar frame sin deformar
    h, w = frame.shape[:2]
    s = min(max_w / w, max_h / h, 1.0)
```

```
    new_w, new_h = int(w * s), int(h * s)
    interp = cv2.INTER_AREA if s < 1.0 else cv2.INTER_LINEAR
    resized = cv2.resize(frame, (new_w, new_h), interpolation=interp)

    # Lienzo negro centrado
    canvas = np.zeros((max_h, max_w, 3), dtype=np.uint8)
    y = (max_h - new_h) // 2
    x = (max_w - new_w) // 2
    canvas[y : y + new_h, x : x + new_w] = resized

    # Mostrar imagen en ventana
    cv2.imshow(win, canvas)
```

Módulo proceso.py

```
import os

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from scipy.fft import fft, fftfreq
from scipy.signal import find_peaks


# - PROCESO -
def proceso(modo):
    print("\n🟢 Proceso iniciado 🟢")

    # - RUTA PARA GUARDAR DATOS -
    # Ruta absoluta de carpeta raíz del proyecto
    ruta_base = os.path.dirname(
        os.path.dirname(os.path.abspath(__file__))
    )  # ruta_base apunta a .../Proyecto
    ruta_datos = os.path.join(
        ruta_base, "datos"
    )  # Carpeta datos dentro de la raíz Proyecto
    ruta_csv_legacy = os.path.join(
        ruta_datos, "datos_seguimiento.csv"
    )  # Determinar CSV legado
    ruta_csv_L = os.path.join(
        ruta_datos, "datos_seguimiento_L.csv"
    )  # Determinar CSV ojo izquierdo
    ruta_csv_R = os.path.join(
        ruta_datos, "datos_seguimiento_R.csv"
    )  # Determinar CSV ojo derecho

    # - OBTENER DATOS -
    datasets = {}  # Usar contenedor de series disponibles
    if os.path.exists(ruta_csv_L):
        dfL = pd.read_csv(ruta_csv_L)
        dfL = (
            dfL.dropna()
            .drop_duplicates(subset="tiempo")
            .sort_values("tiempo")
            .reset_index(drop=True)
        )
        datasets["L"] = dfL
```

```
    if os.path.exists(ruta_csv_R):
        dfR = pd.read_csv(ruta_csv_R)
        dfR = (
            dfR.dropna()
            .drop_duplicates(subset="tiempo")
            .sort_values("tiempo")
            .reset_index(drop=True)
        )
        datasets["R"] = dfR

    if not datasets and os.path.exists(ruta_csv_legacy):
        datasets["LEGACY"] = pd.read_csv(ruta_csv_legacy)

    if not datasets:  # Validar que exista al menos un dataset
        print("No se encontraron archivos de datos de seguimiento.")
        return

    # - FILTRAR SEGÚN MODO -
    # Validar modo de análisis y seleccionar datasets apropiados
    if modo in ("left", "right"):  # Si es left o right, toma solo ese ojo
        etiqueta = "L" if modo == "left" else "R"
        if etiqueta in datasets:
            datasets = {etiqueta: datasets[etiqueta]}
        elif "LEGACY" in datasets:
            datasets = {"LEGACY": datasets["LEGACY"]}
        else:
            print(f"No se encontraron datos válidos para el modo {modo}.")
            return

    elif modo == "both":
        pass  # En modo binocular se mantienen ambos si existen

    else:
        print(f"Modo desconocido: {modo}")
        return

    # - ANALIZAR SERIES DISPONIBLES -
    for etiqueta, df in datasets.items():  # Iterar por cada serie
disponible
        # Validar datos suficientes
        if df.shape[0] < 2:
            print(f"Dataset {etiqueta} vacío o con muy pocos puntos, se
omite.")
            continue
```

```
        # Extraer columnas (asegurar tipo float)
        t = df["tiempo"].to_numpy(dtype=float)
        x = df["x"].to_numpy(dtype=float)
        y = df["y"].to_numpy(dtype=float)

        # Asegurar tiempos estrictamente crecientes (si hay empates o
regresiones)
        mt = np.diff(t) > 0
        if not np.all(mt):
            keep = np.hstack(
                [[True], mt]
            )  # Conservar el primero y luego sólo pasos positivos
            t, x, y = t[keep], x[keep], y[keep]
            if t.size < 2:
                print(
                    f"Dataset {etiqueta} sin suficientes puntos tras
limpieza, se omite."
                )
                continue

        # - DISTANCIA VS TIEMPO -
        x0, y0 = x[0], y[0]  # Punto inicial
        distancia = (
            (x - x0) ** 2 + (y - y0) ** 2
        ) ** 0.5  # Calcular distancia al primer punto

        # - GRÁFICAS -
        plt.figure(figsize=(9, 7))  # Determinar tamaño de figura
        plt.subplot(3, 1, 1)
        plt.plot(t, x, label="x(t)", color="blue")
        plt.xlabel("Tiempo [s]")
        plt.ylabel("x")
        plt.title(f"Distancia x vs tiempo ({etiqueta})")
        plt.grid(True)

        plt.subplot(3, 1, 2)
        plt.plot(t, y, label="y(t)", color="red")
        plt.xlabel("Tiempo [s]")
        plt.ylabel("y")
        plt.title(f"Distancia y vs tiempo ({etiqueta})")
        plt.grid(True)

        plt.subplot(3, 1, 3)
        plt.plot(t, distancia, label="distancia(t)", color="green")
        plt.xlabel("Tiempo [s]")
```

```
        plt.ylabel("d")
        plt.title(f"Distancia total al punto inicial vs tiempo
({etiqueta})")
        plt.grid(True)

        # - AJUSTE -
        plt.tight_layout()  # Ajustar automáticamente espacios entre
gráficas

        # - GUARDAR IMAGEN -
        ruta_graficas = os.path.join(
            ruta_datos, f"xyd_{etiqueta}.png"
        )  # Guardar por ojo
        plt.savefig(ruta_graficas)
        plt.close()  # Liberar figura para memoria

        # - SELECCIONAR SEÑAL -
        senal_principal, nombre_senal = seleccionar_senal_principal(
            x, y, distancia
        )  # Determinar señal principal

        # - NORMALIZAR SEÑAL SELECCIONADA -
        senal_principal = normalizar_senal(
            senal_principal, t
        )  # Usar señal normalizada para análisis y gráficas

        # - GRÁFICA SEÑAL SELECCIONADA -
        plt.figure()
        plt.plot(t, senal_principal)
        plt.xlabel("Tiempo [s]")
        plt.ylabel(nombre_senal)
        plt.title(
            f"Señal principal seleccionada: {nombre_senal} vs tiempo
({etiqueta})"
        )
        plt.grid(True)
        plt.tight_layout()

        # - GUARDAR IMAGEN -
        ruta_senalPrincipal = os.path.join(
            ruta_datos, f"senalPrincipal_{etiqueta}.png"
        )  # Guardar por ojo
        plt.savefig(ruta_senalPrincipal)
        plt.close()
```

```
        # - ¿MOVIMIENTO RÍTMICO? -
        print(f"\n--- Análisis básico con picos ({etiqueta}) ---")
        ritmico_basico = analizar_ritmicidad_basica(
            senal_principal, t, ruta_datos, etiqueta, nombre_senal
        )  # Delegar análisis
        print(f"\n--- Análisis Transformada de Fourier FFT ({etiqueta}) ---
")
        ritmico_ftt = analizar_ritmicidad_fft(
            senal_principal, t, ruta_datos, etiqueta, nombre_senal
        )  # Delegar análisis

    print("\n🟢 Proceso ejecutado 🟢")


# - SELECCIONAR SEÑAL -
def seleccionar_senal_principal(x, y, distancia):
    # Rango de movimiento (variación) en cada eje
    rango_x = np.max(x) - np.min(x)
    rango_y = np.max(y) - np.min(y)

    # Si un eje domina, seleccionarlo
    if rango_x > 1.7 * rango_y:
        print("\n📈 Se seleccionó el eje X como señal principal.")
        return x, "x"
    elif rango_y > 1.7 * rango_x:
        print("\n📉 Se seleccionó el eje Y como señal principal.")
        return y, "y"
    else:
        print("\n📊 Movimiento en ambos ejes → se usará la distancia al
punto inicial.")
        return distancia, "distancia"


# - NORMALIZAR SEÑAL -
def normalizar_senal(senal, t):
    # Delegar normalización general de señal
    fs = 1.0 / np.median(np.diff(t))  # Determinar frecuencia de muestreo
    win = max(5, int(round(1.0 * fs)))  # Definir ventana de ~1 s
    trend = np.convolve(
        senal, np.ones(win) / win, mode="same"
    )  # Suavizar señal para estimar tendencia
    x = senal - trend  # Quitar deriva lenta de la señal

    # Escalar con min-max a rango 0-1
    x_min, x_max = np.min(x), np.max(x)
```

```
    if x_max - x_min < 1e-9:
        return np.zeros_like(x)  # Usar vector de ceros si no hay variación
    return (x - x_min) / (x_max - x_min)  # Normalizar señal a [0, 1]

    # - ¿RÍTMICO? -


# - ANÁLISIS BÁSICO (PICOS) -
def analizar_ritmicidad_basica(senal, t, ruta_datos, etiqueta,
nombre_senal):
    # Detectar picos con prominencia mínima
    peaks, _ = find_peaks(senal, prominence=0.1)  # Picos con prominencia
mínima de 0.1
    tiempos_picos = t[peaks]  # Tiempos de los picos detectados

    # Si no hay suficientes picos, no es rítmico
    if len(tiempos_picos) < 3:
        print("⚠ No hay suficientes picos para analizar el rítmo.")
        return False

    # Calcular intervalos entre picos
    intervalos = np.diff(tiempos_picos)  # Intervalos de tiempo
    prom = np.mean(intervalos)  # Promedio o media
    std = np.std(intervalos)  # Desviación estándar

    plt.subplot()  # Normalizada
    plt.plot(t, senal, label="Señal normalizada")
    plt.plot(t[peaks], senal[peaks], "rx", label="Picos")
    plt.xlabel("Tiempo [s]")
    plt.ylabel("Señal")
    plt.title(f"Picos detectados de {nombre_senal} ({etiqueta})")
    plt.legend()
    ruta_picos = os.path.join(ruta_datos, f"picos_{etiqueta}.png")
    plt.savefig(ruta_picos)
    plt.close()

    # Criterio: si la variabilidad de intervalos es baja → rítmico
    cv = (
        std / prom if prom > 0 else 1e9
    )  # Coeficiente de variación | Usar valor alto si el promedio es cero
    print(f"Media.Int = {prom:.3f}s, Desv.std = {std:.3f}s, Coef.Var =
{cv:.2f}")

    if cv < 0.25:
        print("✅ Movimiento RÍTMICO (nistagmo detectado)")
```

```
        return True
    else:
        print("❌ Movimiento ARÍTMICO (sin nistagmo)")
        return False


# - ANÁLISIS FFT (Transformada de Fourier) -
def analizar_ritmicidad_fft(senal, t, ruta_datos, etiqueta, nombre_senal):
    # Frecuencia de muestreo
    fs = 1.0 / np.median(
        np.diff(t)
    )  # Frecuencia de muestreo a partir del intervalo medio de tiempo
    N = len(senal)  # Número total de muestras de la señal

    # FFT (quitando media)
    senal_centrada = senal - np.mean(
        senal
    )  # Centrar señal eliminando su componente DC (promedio)
    yf = fft(senal_centrada)  # Transformada rápida de Fourier (FFT)
    xf = fftfreq(N, 1 / fs)[
        : N // 2
    ]  # Frecuencias correspondientes (mitad positiva del espectro)
    amplitud = 2.0 / N * np.abs(yf[: N // 2])  # Amplitud normalizada del
espectro

    # Normalizar amplitud, rango 0-1
    A_min, A_max = (
        np.min(amplitud),
        np.max(amplitud),
    )  # Valores mínimo y máximo de la amplitud
    amplitud_norm = (amplitud - A_min) / (
        A_max - A_min + 1e-12
    )  # Normalizar amplitud al rango 0-1 evitano división entre cero

    # Gráfica de espectro de frecuencias
    plt.figure(figsize=(9, 6))  # Determinar tamaño de figura
    plt.subplot(2, 1, 1)  # Real
    plt.plot(xf, amplitud)
    plt.xlabel("Frecuencia [Hz]")
    plt.ylabel("Amplitud")
    plt.title(f"FFT de {nombre_senal} ({etiqueta})")
    plt.grid(True)

    plt.subplot(2, 1, 2)  # Normalizada
    plt.plot(xf, amplitud_norm)
```

```
    plt.xlabel("Frecuencia [Hz]")
    plt.ylabel("Amplitud Normalizada")
    plt.title(f"FFT de {nombre_senal} ({etiqueta})")
    plt.grid(True)

    # Ajuste
    plt.tight_layout()  # Ajustar automáticamente espacios entre gráficas

    # Guardar gráfica de espectro de frecuencias
    ruta_fft = os.path.join(ruta_datos, f"fft_{etiqueta}.png")
    plt.savefig(ruta_fft)
    plt.close()

    # Analizar frecuencia dominante
    idx_max = (
        np.argmax(amplitud[1:]) + 1
    )  # Índice de la frecuencia con mayor amplitud, ignorando DC (0 Hz)
    freq_dom = xf[idx_max]  # Frecuencia dominante
    amp_dom = amplitud[idx_max]  # Amplitud dominante

    print(f"Frecuencia dominante: {freq_dom:.2f} Hz con amplitud
{amp_dom:.3f}")

    # Criterio: si la amplitud dominante es clara respecto al promedio →
rítmico
    amp_rel = (
        amp_dom / np.mean(amplitud[1:]) if np.mean(amplitud[1:]) > 0 else 0
    )  # Relación entre amplitud dominante y media del espectro
    print(f"Relación amplitud dominante / media espectro = {amp_rel:.2f}")

    if amp_rel > 5:
        print("✅ Movimiento RÍTMICO (nistagmo detectado)")
        return True
    else:
        print("❌ Movimiento ARÍTMICO (sin nistagmo)")
        return False
```

ANEXO B

TABLAS DE VALIDACIÓN

B Tablas de validación

Pruebas

La Tabla 9 muestra las características detalladas de cada vídeo, incluyendo observaciones específicas que amplían la información de las pruebas presentadas en el Capítulo 4.

Tabla 9. *Características de los vídeos.*

Vídeo	Ojo(s)	Duración [s]	¿Reflejo en ojo(s)?	Movimiento		¿Zona de ojos visible?	Observaciones
				Cámara	Usuario		
0	Derecho	9	Considerable	No	Considerable	Sí	-
1	Ambos	6	No	Considerable	Considerable	Sí	-
2	Ambos	8	Casi imperceptible en iris	Casi imperceptible	Casi imperceptible	Sí	-
3	Ambos	3	Considerable	Considerable	Considerable	Sí	-
4	Izquierdo	1	Casi imperceptible en iris	No	Sí	Sí	Se ve un poco del ojo derecho abierto y el usuario se acerca.
5	Izquierdo	12	Sí	No	Considerable	Insuficiente	-
6	Izquierdo	4	Considerable	No	Considerable	Sí	Hay zoom y el usuario cierra un poco el ojo.
7	Derecho	3	Casi imperceptible en iris	No	Casi imperceptible	Sí	
8	Ambos	8	Casi imperceptible	Sí	Sí	Sí	Se ve la cara completa y el usuario está un poco lejos.
9	Ambos	1	Sí en pupila	Considerable	Casi imperceptible	Sí	-
10	Izquierdo	4	Sí	No	Considerable	Insuficiente	Usuario cierra el ojo al final
11	Derecho	4	Considerable	No	Considerable	Sí	-
12	Ambos	6	Considerable	Sí	Casi imperceptible	Insuficiente	Cara muy cerca, se ven ambos ojos, pero casi no se ve nariz ni otros puntos clase del rostro. Dos
13	Ambos	3	Considerable	Sí	Casi imperceptible	Insuficiente	Cara muy cerca, se ven ambos ojos, pero casi no se ve nariz ni otros puntos clave de del rostro. Un
14	Ambos	3	Considerable	Sí	Casi imperceptible	Insuficiente	Cara muy cerca, se ven ambos ojos, pero casi no se ve nariz ni otros puntos clave de del rostro.
15	Ambos	1	No	Sí	Casi imperceptible	Sí	Se ve la cara completa y el usuario está lejos.

Resultados

La Tabla 10 presenta los resultados detallados, incluyendo consideraciones específicas que amplían la información de las pruebas descritas en el Capítulo 4.

Tabla 10. *Resultados de detección del movimiento mediante los métodos básico y FFT.*

Vídeo	¿Rítmico?	Básico		¿Acerto?	FFT		¿Acerto?	Ojo(s)	¿Acerto ojo(s)?	Consideraciones
		Izquierdo	Derecho		Izquierdo	Derecho				
0	No	ND	✓	✓	✓	ND	✓	Derecho	Sí	-
1	Sí	✗	✗	✗	✓	✓	✓	Ambos	Sí	-
2	Sí	✓	✓	✓	✓	✓	✓	Ambos	Sí	-
3	Sí	⚠	✗	✗	✓	✓	✓	Ambos	Sí	-
4	Sí	✗	⚠	✗	✗	✗	✗	Izquierdo	Sí	Cuando se ve un poco ojo derecho, se detecta y se descarta su análisis.
5	Sí	ND	✗	✗	ND	✓	✓	Izquierdo	El contrario	Detección inestable, cambia entre iris y pupila, se pierde al parpadear.
6	Sí	✓	ND	✓	✗	ND	✗	Izquierdo	Sí	-
7	Sí	ND	✓	✓	ND	✗	✗	Derecho	Sí	Detecta la orilla de pupila.
8	Sí	✗	✗	✗	✓	✓	✓	Ambos	Sí	-
9	Sí	✗	✓	✗	✗	✗	✗	Ambos	Sí	-
10	Sí	ND	✓	✓	ND	✓	✓	Izquierdo	El contrario	Detecta la orilla de pupila.
11	Sí	ND	✗	✗	ND	✗	✗	Derecho	Sí	Detecta a media pupila.
12	Sí	✗	✗	✗	✓	✗	✓	Ambos	No	Detecta izquierdo en la orrila de la pupila derecho. Detecta derecho en parpado derecho; se descarta su análisis. Se pierden al parpadear.
13	Sí	-	-	-	-	-	-	Ambos	No	Detecta izquierdo en parpado izquierdo; se descarta totalmente el análsis de ambos.
14	Sí	✗	✗	✗	✓	✓	✓	Ambos	No	Detecta izquierdo en parpado derecho; se descarta su análisis. Detecta derecho en la orilla del iris derecho. Se pierden al parpadear.
15	Sí	✓	✗	✗	✗	✗	✗	Ambos	Sí	-

La figura ✓ indica un resultado positivo y ✗ uno negativo, mientras que ⚠ señala que no hay suficientes picos para el análisis.

El color también es relevante, cuando cualquiera de estas figuras (✓, ✗ o ⚠) aparece en negro, significa que la prueba fue descartada.

ANEXO C

RESULTADOS

C Resultados

Caso 0: un ojo sin nistagmo

```
🟠 Prepoceso iniciado 🟠

🎥 Conversión de vídeo iniciada
El video se puede leer, no es necesario convertir.
🎥 Conversión de vídeo ejecutada

🔄 Rotación iniciada
Rotación detectada (MediaInfo): 90°
Vídeo rotado -> 7_rotado.mp4
🔄 Rotación ejecutada

📼 Estabilización iniciada
Log de estabilización -> 7_rotado_estabilizado_deltas.csv
Debug de estabilización -> 7_rotado_estabilizado_debug.mp4
Vídeo estabilizado -> 7_rotado_estabilizado.mp4
📼 Estabilización ejecutada

👁 Seguimiento iniciado
Modo: right
Vídeo de seguimiento -> video_seguimiento.mp4
Log de seguimiento (R) -> datos_seguimiento_R.csv
👁 Seguimiento ejecutado

🟠 Prepoceso ejecutado 🟠


🔵 Proceso iniciado 🔵

📊 Movimiento en ambos ejes → se usará la distancia al punto inicial.

--- Análisis básico con picos (R) ---
Promedio = 0.184s, Desv.std = 0.100s, Coef.Var = 0.55
❌ Movimiento ARÍTMICO (sin nistagmo)

--- Análisis Transformada de Fourier FFT (R) ---
Frecuencia dominante: 1.57 Hz con amplitud 0.068
Relación amplitud dominante / media espectro = 4.55
❌ Movimiento ARÍTMICO (sin nistagmo)

🔵 Proceso ejecutado 🔵
```

Caso 2: ambos ojos con nistagmo

```
🟠 Prepoceso iniciado 🟠

🎥 Conversión de vídeo iniciada
El video se puede leer, no es necesario convertir.
🎥 Conversión de vídeo ejecutada

🔄 Rotación iniciada
Rotación detectada (MediaInfo): 0°
Rotación ejecutada. No fue necesario corregir rotación.
🔄 Rotación ejecutada

📼 Estabilización iniciada
Log de estabilización -> N2_estabilizado_deltas.csv
Debug de estabilización -> N2_estabilizado_debug.mp4
Vídeo estabilizado -> N2_estabilizado.mp4
📼 Estabilización ejecutada

👁 Seguimiento iniciado
Modo: both
Vídeo de seguimiento -> video_seguimiento.mp4
Log de seguimiento (L) -> datos_seguimiento_L.csv
Log de seguimiento (R) -> datos_seguimiento_R.csv
👁 Seguimiento ejecutado

🟠 Prepoceso ejecutado 🟠


🔵 Proceso iniciado 🔵

📊 Movimiento en ambos ejes → se usará la distancia al punto inicial.

--- Análisis básico con picos (L) ---
Promedio = 0.224s, Desv.std = 0.038s, Coef.Var = 0.17
✅ Movimiento RÍTMICO (nistagmo detectado)

--- Análisis Transformada de Fourier FFT (L) ---
Frecuencia dominante: 4.31 Hz con amplitud 0.147
Relación amplitud dominante / media espectro = 9.68
✅ Movimiento RÍTMICO (nistagmo detectado)
```

```
📊 Movimiento en ambos ejes → se usará la distancia al punto inicial.

--- Análisis básico con picos (R) ---
Promedio = 0.213s, Desv.std = 0.049s, Coef.Var = 0.23
✅ Movimiento RÍTMICO (nistagmo detectado)

--- Análisis Transformada de Fourier FFT (R) ---
Frecuencia dominante: 4.43 Hz con amplitud 0.109
Relación amplitud dominante / media espectro = 6.84
✅ Movimiento RÍTMICO (nistagmo detectado)

🔵 Proceso ejecutado 🔵
```