UTN

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL CÓRDOBA
SECRETARIA DE POSGRADO

TRABAJO FINAL INTEGRADOR
PARA CARRERA DE POSGRADO DE
ESPECIALIZACIÓN EN AUTOMÓVILES DE COMPETICIÓN

Desarrollo de algoritmo para tratamiento y fusión de señales aplicado a un sistema de adquisición de datos de bajo costo para automovilismo de competición

Autor: Carfagna, Gastón

Tutor: Pérez Paina, Gonzalo
Co-tutor: Ciabattari, Javier

Córdoba
2026

Tabla de contenido

Resumen

En el automovilismo de competición nacional, y especialmente en el ámbito zonal, existe una brecha tecnológica significativa causada por el alto costo de los sistemas de adquisición de datos (DAQ) profesionales. Esta barrera económica limita la capacidad de optimización basada en evidencia para una gran parte de los competidores.

Para abordar esta problemática se propone desarrollar y validar un sistema DAQ de precio competitivo que mediante un algoritmo de post-procesamiento compense las limitaciones metrológicas inherentes a los sistemas de adquisición de bajo costo. Se parte de la premisa de que el hardware económico, por sí solo, carece de la estabilidad y relación señal-ruido requeridas para el análisis dinámico profesional.

En consecuencia, la contribución central de este trabajo no reside en la electrónica física, sino en la implementación de una metodología de mejora algorítmica que, mediante técnicas de filtrado digital y fusión sensorial, corrija las deficiencias del sensor. Este enfoque permite transformar un dispositivo de arquitectura abierta en una opción tecnológicamente competitiva, capaz de entregar métricas cualitativas y cuantitativas con una fidelidad equivalente a los estándares de referencia de la industria.

El algoritmo desarrollado implementa una secuencia de diagnóstico y corrección totalmente automatizada:

- Detección y Corrección de Latencia Interna: Identifica y corrige el desfase temporal (latencia) entre los registros de la IMU y el GPS.
- Diagnóstico de Orientación y Montaje: A través de la correlación de la aceleración medida por el IMU con la derivada de la velocidad del GPS. Esto le permite identificar correctamente los ejes (cuando montaje hace que no coincida la orientación del acelerómetro con la del vehículo) y sus signos según convención SAE, además corregir el sesgo característico del sensor empleado.
- Fusión de Sensores: Implementa un Filtro de Kalman para reconstruir la señal de velocidad durante las pérdidas momentáneas del GPS.
- Filtrado de Señal: Aplica un filtro Savitzky-Golay para atenuar el ruido de alta frecuencia (vibraciones) sin introducir retardo de fase, preservando la morfología de las maniobras.

Los resultados de la validación experimental en pista demostraron una excelente correlación cuantitativa. El algoritmo de post-procesado redujo significativamente las métricas de error en relación al obtenido de las señales en bruto. La capacidad del software para diagnosticar y corregir errores de montaje físico, validó su robustez y funcionalidad. Las mediciones finales de velocidad y aceleración, ya procesadas, replican fielmente la dinámica registrada por el sistema comercial de referencia.

Como implicación principal, este proyecto valida la viabilidad de construir una herramienta DAQ de bajo costo que, potenciada por un software robusto, ofrece resultados de alta fidelidad. Representando una solución accesible que permite optimizar el rendimiento de los vehículos de competición de una forma precisa.

Palabras Clave: Adquisición de Datos, DAQ, Motorsports, Bajo Costo, Dinámica Vehicular, Python, Procesamiento de Señales, Telemetría, Filtro Kalman.

Capítulo 1. Introducción

1.1 Contexto y antecedentes del problema

El automovilismo de competición ha transitado una profunda transformación en las últimas décadas. La intuición del piloto y la experiencia del mecánico, si bien siguen siendo pilares fundamentales, han sido complementadas por el análisis riguroso de datos. Hoy en día, un auto de carreras es una compleja plataforma de sensores que genera un volumen masivo de información en cada vuelta. La correcta adquisición, procesamiento e interpretación de estos datos se ha convertido en el factor diferenciador clave entre el éxito y el estancamiento.

En las categorías de élite a nivel mundial, la telemetría en tiempo real y el análisis de datos post-ensayo son herramientas indispensables. Permiten a los ingenieros comprender de manera cuantitativa y objetiva la compleja interacción entre el piloto, el vehículo y el circuito. Variables como la aceleración lateral en el ápex de una curva, la progresividad en la aplicación del freno o la velocidad mínima en un viraje lento dejan de ser conceptos subjetivos y se convierten en métricas cuantificables y optimizables.

Sin embargo, esta era de la información no ha alcanzado de manera homogénea a todas las esferas del deporte motor. En el contexto del automovilismo nacional y zonal en Argentina, se observa una realidad heterogénea. Mientras las categorías de mayor envergadura disponen de estas tecnologías, una vasta mayoría de las divisiones de ascenso, campeonatos zonales y categorías de formación operan con recursos limitados. Para estos equipos, el acceso a sistemas de adquisición de datos (DAQ) profesionales, dominados por marcas de prestigio como MoTeC, AIM o Racelogic, representa una barrera económica. El costo de estos sistemas puede equivaler a una porción significativa del presupuesto total de una temporada.

Esta disparidad tecnológica genera un círculo vicioso: los equipos con mayor presupuesto acceden a mejores herramientas de datos, lo que les permite optimizar su rendimiento de manera más eficiente y perpetuar su ventaja competitiva. En contrapartida, los equipos de menores recursos se ven forzados a depender de métodos más tradicionales y subjetivos, como la intuición del piloto o la observación visual, limitando su capacidad de aprendizaje técnico y su potencial de mejora continua.

1.2 Problema de la investigación

La problemática central de esta investigación radica en la barrera tecnológica y económica que restringe el acceso a sistemas de adquisición de datos (DAQ) de grado metrológico en el automovilismo zonal y nacional.

Es importante destacar que el presente trabajo no tiene como objetivo el desarrollo de un sistema de navegación inercial completo ni la investigación teórica de nuevos algoritmos de estimación, sino la aplicación práctica e integrada de técnicas conocidas de procesamiento digital de señales y fusión sensorial. El foco del proyecto se centra en evaluar su efectividad dentro del contexto del automovilismo de competición, priorizando la utilidad operativa y la coherencia dinámica de los datos obtenidos por sobre el desarrollo teórico formal

La inviabilidad presupuestaria para implementar equipos profesionales genera una asimetría técnica estructural que obstaculiza la profesionalización del deporte. Esta carencia no representa simplemente una desventaja competitiva, sino una limitación sistémica en el ciclo de desarrollo de la ingeniería de pista. Por un lado, impide la transición de una puesta a punto basada en datos subjetivos a una metodología de optimización basada en evidencia. Por otro lado, restringe la curva de aprendizaje de pilotos e ingenieros, quienes se ven privados de interactuar con herramientas de análisis que constituyen el estándar técnico en la industria global.

La pregunta de investigación que surge de esta problemática es multifacética:

¿Es posible mitigar las limitaciones de precisión y ruido inherentes al hardware de arquitectura abierta mediante la implementación de algoritmos de procesamiento digital de señales? Específicamente, ¿puede un sistema de bajo costo, asistido por un algoritmo de corrección y fusión sensorial en software, entregar métricas de dinámica vehicular con una fidelidad funcionalmente equivalente a la de un sistema comercial de referencia certificado?

1.3 Hipótesis

La aplicación de un algoritmo avanzado de post-procesamiento de señales, basado en técnicas de sincronización, filtrado y fusión sensorial, permite mejorar significativamente la calidad de datos provenientes de un sistema de adquisición vehicular de bajo costo, corrigiendo deficiencias inherentes como ruido, sesgos, errores de montaje y latencias, generando un conjunto de señales capaz de describir con fidelidad la dinámica vehicular. Los resultados obtenidos son funcionalmente comparables a los provistos por un sistema comercial de referencia, validando la utilidad del enfoque algorítmico como herramienta profesional para el análisis y la optimización del rendimiento en automovilismo de competición.

1.4 Objetivos

1.4.1 Objetivo general

Desarrollar y validar un algoritmo de post-procesamiento de señales aplicado a datos vehiculares, orientado a mejorar la calidad, coherencia y utilidad práctica de la información adquirida mediante un sistema de adquisición de bajo costo, con el fin de permitir un análisis confiable de la dinámica vehicular en el contexto del automovilismo de competición.

1.4.2 Objetivos específicos

- Implementar un algoritmo de diagnóstico automático de señales inerciales que permita identificar y corregir errores de montaje y orientación de sensores sin requerir calibración manual previa.
- Desarrollar un método de sincronización temporal entre señales provenientes de sensores heterogéneos (IMU y GPS), considerando latencias internas y diferencias en las tasas de muestreo.
- Aplicar técnicas de filtrado digital orientadas a reducir el ruido y los efectos de vibraciones propias del entorno automovilístico, preservando la información relevante para el análisis dinámico del vehículo.

- Integrar un esquema de fusión sensorial que combine información inercial y de posicionamiento para mejorar la estimación de variables cinemáticas clave, tales como velocidad y aceleraciones longitudinales y laterales.
- Evaluar el desempeño del algoritmo desarrollado mediante ensayos en pista, comparando los resultados obtenidos con los provistos por un sistema comercial de referencia utilizado en automovilismo de competición.
- Analizar las limitaciones del algoritmo frente a condiciones desfavorables, como pérdidas temporales de señal GPS, y discutir su impacto en la utilización práctica de los datos obtenidos.

Capítulo 2. Revisión de la literatura

Para fundamentar el desarrollo de este sistema DAQ de bajo costo, se ha realizado una revisión técnica focalizada en cuatro pilares: la física del vehículo, las técnicas de análisis, los algoritmos de tratamiento de señales y el estado del arte tecnológico.

2.1 Dinámica Vehicular

El marco teórico general para este proyecto se basa en la obra de referencia Milliken (1995). Si bien la dinámica vehicular moderna abarca múltiples variables, este proyecto delimita su alcance al estudio de la cinemática del chasis. Siguiendo los principios descritos en esta obra, se ha priorizado el uso de una configuración de sensores no intrusiva compuesta por Acelerómetro, Giroscopio y GPS. De este conjunto se obtienen las variables fundamentales: aceleraciones inerciales (G-G) y, crucialmente, la velocidad de avance proporcionado por el módulo satelital, así como la trayectoria seguida por el vehículo en pista.

2.2 Técnicas de Análisis de Adquisición de Datos

Complementando la teoría dinámica, se adoptaron los criterios de análisis presentados por Segers (2008). Este autor establece los estándares de calidad de señal necesarios para la interpretación efectiva de la telemetría, definiendo métricas críticas como la aceleración longitudinal, lateral y velocidad instantánea del vehículo.

La relevancia de Segers para este trabajo radica en que define los "requisitos de usuario": para que un sistema DAQ sea útil en competición, no basta con acumular datos; las señales deben tener una relación correcta entre la señal y ruido eléctrico, que permita apreciar los fenómenos dinámicos que afectan al vehículo.

2.3 Métodos de Tratamiento de Señales

El ecosistema científico de Python constituye la base metodológica de este trabajo. Para la gestión eficiente de los datos del microcontrolador, se empleó la biblioteca NumPy, aprovechando las técnicas descritas en el trabajo Harris, C. R. (2020).

Dado que el sistema maneja distintas frecuencias de muestreo (GPS e IMU), se utilizó la librería pandas para la sincronización de series temporales, tal como se detalla en McKinney, W. (2010). Para la generación de las curvas comparativas y la visualización científica de la telemetría, se utilizaron las herramientas presentadas en Hunter, J. D. (2007).

Finalmente, para mitigar el ruido de los sensores de bajo costo, se aplicaron algoritmos avanzados de la librería SciPy, fundamentados en Virtanen, P. G. (2020):

- Filtro Savitzky-Golay: Se implementó el método de suavizado presentado en Savitzky, A. & Golay, M. (1964). Su aplicación permitió eliminar el ruido de vibración del motor sin distorsionar los picos de frenado.
- Filtro de Kalman: Para la fusión sensorial y la reconstrucción de la trayectoria ante pérdidas de señal GPS, se utilizó la teoría expuesta en Welch, G. & Bishop, G. (2007), implementada siguiendo las prácticas de Labbe, R. (2014) que permitió reconstruir datos perdidos por falta de señal del sensor de bajo costo que se empleó.

2.4 Estado del Arte en Tecnología DAQ

Para establecer un estándar de validación, se analizaron las especificaciones detalladas en Motec (2025). A diferencia de estos sistemas comerciales de alta gama, el prototipo desarrollado se define bajo una arquitectura autónoma limitada únicamente a sensores inerciales y GPS. El objetivo fue demostrar una fidelidad funcionalmente equivalente en la dinámica del vehículo sin la complejidad de conexión a la ECU ni sensores externos al sistema.

Finalmente, el diseño se verificó contra la normativa vigente, consultando el ACTC (2025) y el APAT (2025), asegurando que el uso de sistemas DAQ autónomos está permitido, siempre y cuando se cumpla con las regulaciones de no tener telemetría y solo adquirir datos post competencia.

Capítulo 3. Metodología

El desarrollo del proyecto se estructuró en un enfoque modular e iterativo, abarcando desde la selección de componentes hasta la validación final en pista. La metodología se puede dividir en dos grandes fases: el diseño e integración del hardware y por el otro, el desarrollo del software de post procesado, que se complementan para obtener las mediciones de alta fidelidad. A través de la incorporación de estos algoritmos y correcciones, se busca obtener resultados que representen de forma correcta los fenómenos dinámicos descriptos por el vehículo, siendo esto el núcleo innovador de este trabajo y lo que sienta las bases para a futuro mejorar el sistema.

En este sentido, el desarrollo del hardware se aborda únicamente como una plataforma necesaria para la adquisición de las señales primarias, mientras que el aporte principal del trabajo reside en el diseño e implementación del algoritmo de post-procesado en software, encargado de mejorar la calidad, coherencia y utilidad práctica de los datos adquiridos.

3.1 Diseño e Integración del Hardware

Siguiendo el objetivo de bajo costo y accesibilidad, se seleccionaron componentes electrónicos de código abierto y amplia disponibilidad (ver sección 8.1 ” Arquitectura del Sistema”:

3.2 Desarrollo de Software de Post-Procesado

El aporte innovador central de este trabajo reside en el algoritmo de post-procesado desarrollado en Python (ver sección 8.2 "Codigo: algoritmo-post-procesadoTFI.py"). Este proceso automatizado transforma la señal en bruto de los sensores del sistema de bajo costo en un conjunto de datos limpio, alineado, sincronizado y de alta fidelidad, comparable a los sistemas comerciales de alta gama.

El código ejecuta una secuencia de operaciones, diseñadas de forma específica para diagnosticar y corregir los errores inherentes al hardware y al montaje del equipo.

Para facilitar la comprensión del algoritmo desarrollado, la Tabla 1 resume el origen de cada señal y su función dentro del post-procesado:

Señal	Origen	Función dentro algoritmo
Velocidad	GPS	Referencia para corrección sesgo (ZUPT), y como variable de medición en Kalman
Aceleración	IMU	Diagnóstico de ejes, predicción Kalman, corrección de latencia y filtrado Savitzky-Golay
Posición	GPS	Detección de signo lateral
Señales procesadas	Software	Análisis dinámico final

Tabla 1: Resumen señales

3.2.1 Diagnóstico Automatizado de Sensores y Corrección de Montaje

A diferencia de los sistemas que requieren una configuración inicial manual precisa, el algoritmo, inicia con un diagnóstico para auto calibrar el sistema. Esta fase, realiza tres tareas críticas:

- Detección de Ejes (Longitudinal/Lateral): El sistema determina la orientación física del IMU. Para ello, calcula la aceleración longitudinal de referencia derivando la señal de velocidad del GPS y la contrasta con la aceleración en eje x e y obtenida en bruto con el IMU. Este procedimiento permite determinar cuál de los ejes presenta la mayor correlación cruzada, identificando automáticamente el eje longitudinal del vehículo. Esto elimina por completo los errores de montaje.
- Detección de Signo de Ejes: El signo del eje longitudinal se infiere de la correlación anterior. Para el eje lateral, el script implementa un algoritmo que analiza la trayectoria del GPS (cambios en Latitud y Longitud). Al detectar la dirección de un giro, la compara con la lectura del acelerómetro lateral, asignando el signo correcto, conforme a la norma SAE (positivo cuando el giro es hacia la derecha).
- Corrección de sesgo: Simultáneamente, el script implementa una Actualización en Velocidad Cero (ZUPT). Detecta períodos en los que el vehículo está detenido (velocidad < 1.0 km/h) y calcula el promedio de la lectura de los sensores en esos tramos. Este promedio, que representa el sesgo del sensor, se sustrae de toda la señal para centrarla perfectamente en cero.

3.2.2 Detección y Corrección de Latencia Interna (IMU-GPS)

Un desafío fundamental en la fusión de sensores de bajo costo es tener un proceso robusto que pueda corregir errores generados por la latencia característica del hardware. El algoritmo de post-procesado identifica y corrige el desfase temporal que existe entre los datos del IMU y los del GPS.

A través de la correlación cruzada entre la aceleración del GPS y la señal del eje longitudinal (ya identificado) del IMU. El pico de esta correlación revela el desfase exacto en muestras, que se convierte a segundos.

Posteriormente, el script aplica la corrección encontrada, generando un vector de tiempo desfasado y utiliza interpolación lineal para muestrear nuevamente las lecturas del acelerómetro de forma corregida tanto en eje X como Y. Este paso crucial asegura que los datos de la IMU y del GPS estén perfectamente alineados temporalmente antes de cualquier fusión o filtrado.

3.2.3 Fusión de Señal y Corrección de Anomalías GPS

Para garantizar la integridad de la señal de velocidad ante interrupciones o degradaciones severas del enlace satelital, se implementa una arquitectura de estimación híbrida basada en un Filtro de Kalman Lineal.

Definición del modelo de estado: El núcleo del algoritmo se configura bajo un esquema de predicción-corrección. Se define la velocidad longitudinal como la variable de estado. La etapa de predicción es gobernada por la integración numérica de la aceleración longitudinal del IMU, previamente rectificada por los algoritmos de alineación y corrección de sesgo (bias).

Posteriormente, la etapa de actualización incorpora la velocidad medida por el módulo GPS como observación externa. Esta configuración permite fusionar la alta frecuencia de muestreo y respuesta transitoria del sensor inercial con la estabilidad a largo plazo y referencia absoluta del sistema satelital.

Lógica de validación y rechazo de mediciones: El sistema incorpora un módulo de que supervisa la calidad de la señal. Mediante un umbral de validación dinámico, el algoritmo detecta ventanas de baja confiabilidad en la telemetría GPS (típicamente asociadas a velocidades < 5.0 km/h en condiciones de pista).

Al detectar una anomalía, el sistema conmuta su lógica de operación: rechaza la medición satelital corrupta y opera en modo de navegación inercial asistida, confiando temporalmente en la estimación del modelo interno del Filtro de Kalman. Para asegurar la consistencia cinemática en las fronteras de estos eventos, se aplican técnicas de suavizado en los puntos de conmutación, garantizando una señal de velocidad continua y derivable, apta para el análisis dinámico posterior.

3.2.4 Filtrado de Señal (Savitzky-Golay)

Como etapa final del algoritmo, se aplica un filtro de suavizado Savitzky-Golay para eliminar el ruido de alta frecuencia generado por las vibraciones mecánicas del chasis y el motor. Este proceso se aplica de manera independiente sobre las tres señales cinemáticas resultantes:

- Velocidad Final: Sobre la señal híbrida resultante de la fusión (GPS + Kalman).

- Aceleración Longitudinal: Sobre los datos crudos del IMU ya reorientados.
- Aceleración Lateral: Sobre los datos crudos del IMU ya reorientados.

Al ajustar un polinomio local a cada ventana de datos, este método preserva la amplitud real de los picos transitorios, sin introducir el retardo de fase característico de los filtros de media móvil.

3.2.5 Sincronización de Ensayos por Correlación Cruzada

Para la comparación directa entre el prototipo y el sistema de referencia, el algoritmo programado, implementa un método de sincronización temporal robusto. En lugar de alinear por un único punto (como la velocidad máxima), el script utiliza una correlación cruzada sobre la morfología completa de ambas señales de velocidad procesadas. El algoritmo calcula el desfase temporal que produce la máxima superposición entre las dos curvas completas. Este método garantiza una alineación precisa y objetiva de los ensayos.

Cabe aclarar que este procedimiento de sincronización se emplea exclusivamente con fines de validación experimental y comparación contra el sistema de referencia (MoTeC). En una aplicación operativa normal, el sistema DAQ experimental funciona de manera autónoma y el análisis se realiza sobre su propia base temporal relativa, sin necesidad de sincronización externa.

3.3 Métricas de Evaluación de Desempeño

Para cuantificar la fidelidad de la señal reconstruida por el algoritmo propuesto, se definen formalmente las métricas de error, utilizando como Patrón las señales provenientes del sistema MoTeC calibrado.

3.3.1 Error Absoluto Medio (MAE)

Se utiliza para evaluar la exactitud promedio del sistema, proporcionando una magnitud lineal del error esperado en cada muestra:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|yref,i - \hat{y}\,i|$$

Donde $\hat{y}\,i$ representa la señal procesada por el prototipo en el instante i, y n es el número total de muestras

3.3.2 Raíz del Error Cuadrático Medio (RMSE)

Se emplea para evaluar la estabilidad del filtro ante transitorios y valores atípicos. Al elevar las diferencias al cuadrado antes de promediar, esta métrica penaliza severamente las desviaciones grandes, siendo un indicador crítico de la robustez del Filtro de Kalman ante pérdidas de señal:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(yref,i - \hat{y}\,i)^2}$$

Capítulo 4. Resultados

Este capítulo presenta el análisis cuantitativo y cualitativo del sistema de adquisición de datos (DAQ) desarrollado. El objetivo central es validar la hipótesis de que, mediante un software de procesamiento avanzado, es posible obtener datos de dinámica vehicular de alta fidelidad utilizando hardware de bajo costo y arquitectura autónoma (GPS + IMU)

La prueba donde se realizó la adquisición de datos se realizó el 06/09/25 en el Autódromo Juan y Oscar Gálvez en CABA. Para esta prueba, el sistema fue montado en un vehículo de Turismo Carretera 2000, más específicamente en un Honda Civic (ver Figura 1) del equipo FDC. Este vehículo contaba con un sistema comercial MoTeC instalado, el cual se utilizó como patrón de referencia para validar los datos obtenidos con el sistema experimental.

Figura 1: Honda Civic de TURCAR2000 donde se efectuó la prueba.

4.1 Protocolo de Validación y Selección de Escenarios

Para evaluar el rendimiento del sistema de manera integral, se diseñó un protocolo de pruebas en pista (Autódromo Juan y Oscar Gálvez) que somete al prototipo a niveles de exigencia progresivos. Se seleccionaron tres escenarios de análisis específicos, descartando aquellos que no ofrecían condiciones de borde relevantes. La selección obedece a los siguientes criterios de ingeniería:

- Escenario de Alta Velocidad (Rendimiento Nominal): Representa una vuelta rápida en condiciones de carrera con cobertura GPS óptima. Este escenario actúa como el punto de comparación principal para demostrar la equivalencia funcional con el sistema de referencia (MoTeC) bajo altas fuerzas G laterales y longitudinales.
- Escenario de Baja Velocidad (Sensibilidad): Corresponde a una vuelta de instalación/calentamiento. Su inclusión es crítica para evaluar la sensibilidad de los sensores inerciales ante aceleraciones de baja magnitud y validar que el filtrado digital no elimine información útil en el rango dinámico inferior.
- Escenario de Estrés (Robustez del Filtro de Kalman): Se aisló una serie de datos con interrupciones severas y "zonas de sombra" en la señal satelital del GPS. El objetivo es validar la capacidad del algoritmo de fusión de sensores para reconstruir la trayectoria y la velocidad cuando la fuente primaria falla, diferenciando esta propuesta de un sistema convencional.

4.2 Escenario de Alta Velocidad (Rendimiento Nominal)

Se observa que, al superponer las trayectorias registradas por ambos sistemas, se observa una alta coincidencia geométrica, presentando discrepancias marginales (Ver figura 2)

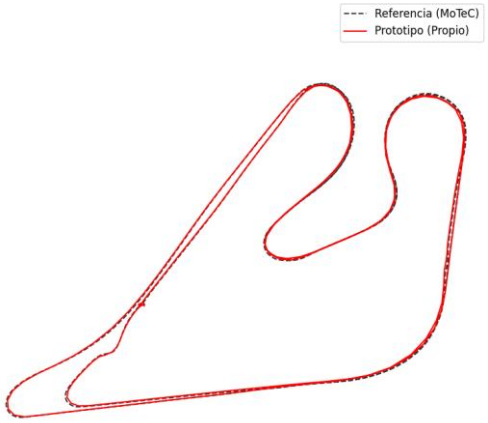


Figura 2: Comparativa posición GPS sistema MoTec vs sistema experimental

En cuanto a la velocidad (ver Figura 3), en esta experiencia se observa una gran relación entre las mediciones de los 2 sistemas. Cualitativamente, la señal procesada (línea roja) se encuentra alineada con la señal de referencia (línea punteada verde).

El análisis cuantitativo lo confirma:

	MAE	RMSE
Señal en Bruto	2,2176 km/h	3,5744 km/h
Señal procesada	1,5944 km/h	2,4805 km/h
Porcentaje de mejora	28,1%	30,6%

Tabla 2: Comparativa MAE y RMSE en bruto vs post-procesado para velocidad en escenario alta velocidad

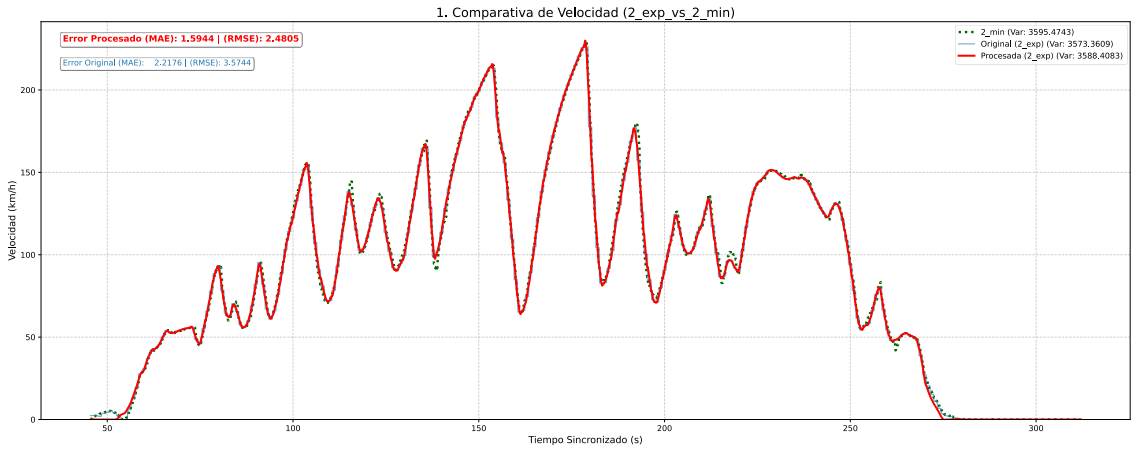


Figura 3: Comparativa gráfico velocidad escenario rendimiento nominal.

Analizando la varianza de las señales, observamos que inicialmente con la señal en bruto teníamos un valor de 3573,36 $(km/h)^2$. Luego del post-procesado el

resultado se modificó quedando en 3588,40 $(km/h)^2$, la cual es una medida semejante a la de señal de referencia, que tiene un valor de 3595,47 $(km/h)^2$.

Para las aceleraciones, el objetivo principal del algoritmo de post-procesado es eliminar el ruido de alta frecuencia. Los gráficos de Aceleración Longitudinal (ver Figura 4) y Lateral (ver Figura 5) muestran una señal base (azul) con picos muy marcados.

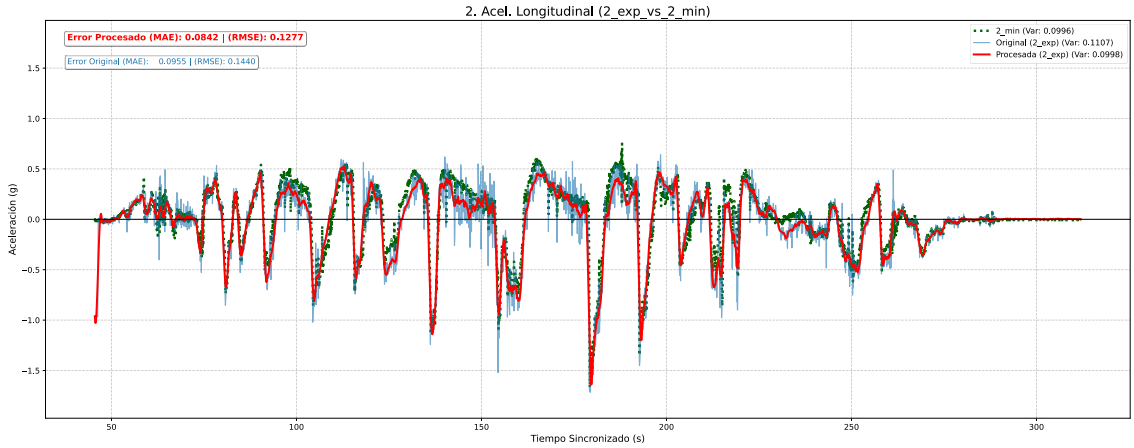


Figura 4: Comparativa gráfico aceleración longitudinal rendimiento nominal.

	MAE	RMSE
Señal en Bruto	0,0955 g	0,1416 g
Señal procesada	0,0842 g	0,1277 g
Porcentaje de mejora	13,7%	10%

Tabla 3: Comparativa MAE y RMSE en bruto vs post-procesado para aceleración long en escenario alta velocidad

En este caso el análisis de la Varianza, nos dio que la señal en bruto tiene un valor de 0,1117 g^2, mientras que luego del post-procesado se obtiene un valor de 0.0990 g^2, que es casi idéntica a la de la referencia (0.0996 g^2).

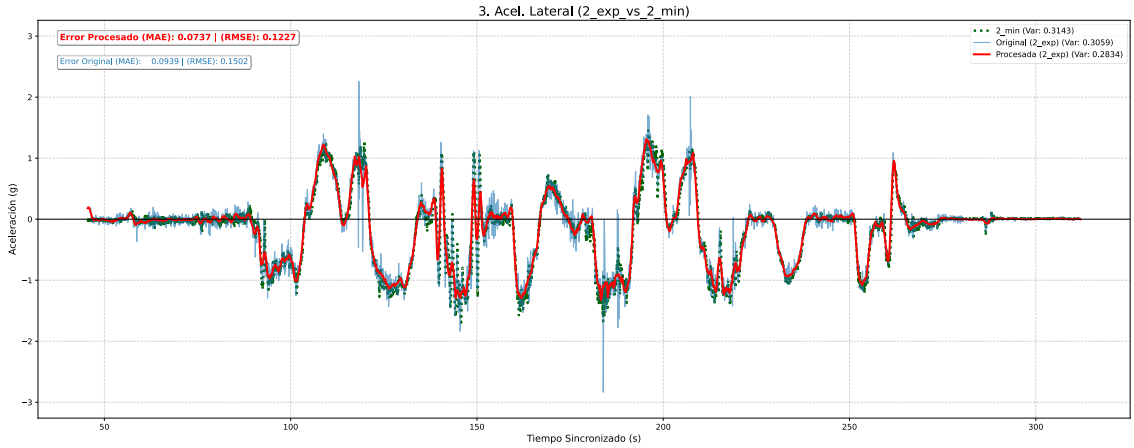


Figura 5: Comparativa gráfico aceleración lateral rendimiento nominal.

	MAE	RMSE
Señal en Bruto	0,0839 g	0,1502 g
Señal procesada	0,0737 g	0,1227 g
Porcentaje de mejora	13,7%	10%

Tabla 4: Comparativa MAE y RMSE en bruto vs post-procesado para aceleración lat en escenario alta velocidad

Al analizar la varianza, se observa que el valor de la señal bruta (0,3059 g^2) parece estar numéricamente más cerca de la referencia (0,3143 g^2) que la señal procesada (0,2834 g^2). Sin embargo, esto no representa un empeoramiento en la fidelidad de la señal. La varianza de la señal original se encontraba afectada por la presencia de picos de ruido de alta frecuencia (claramente visibles en la gráfica), los cuales no representan aceleraciones laterales físicamente realistas y que modifican la varianza, por eso en este caso este parámetro no define y la mejora se ve claramente en el MAE y RMSE

4.3 Escenario de Baja Velocidad (Sensibilidad)

La comparativa de las trayectorias también es muy similar, dado que no se registraron pérdidas de señal de ningún tipo (ver Figura 6)

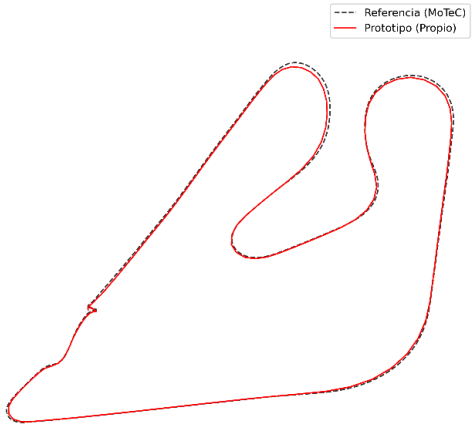


Figura 6: Comparativa posición GPS sistema MoTec vs sistema experimental (baja velocidad)

Pese a en este caso tener velocidades bajas al ser una vuelta de calentamiento, observamos que, en esta situación, también el sistema demuestra una alta fidelidad (ver Figura 7).

	MAE	RMSE
Señal en Bruto	1,5178 km/h	2,1402 km/h
Señal procesada	1,0320 km/h	1,5069 km/h
Porcentaje de mejora	32%	29,6%

Tabla 5: Comparativa MAE y RMSE en bruto vs post-procesado para velocidad escenario baja velocidad

En cuanto a la varianza, inicialmente con la señal en bruto era de 2049,06 $(km/h)^2$, luego del post-procesado alcanzo un valor de 2075,60 $(km/h)^2$, el cual es un valor semejante al de señal de referencia, el cual es de 2061,78 $(km/h)^2$.

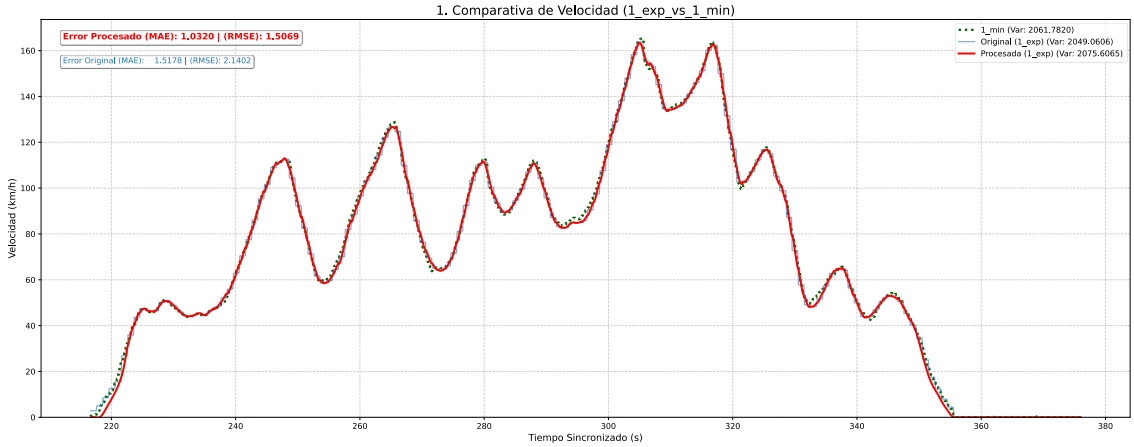


Figura 7: Comparativa gráfico velocidad (baja velocidad).

Analizando el gráfico de la aceleración longitudinal, podemos ver que luego del post-procesado, obtenemos una mejora notable en la comparativa con respecto a la referencia (ver Figura 8)

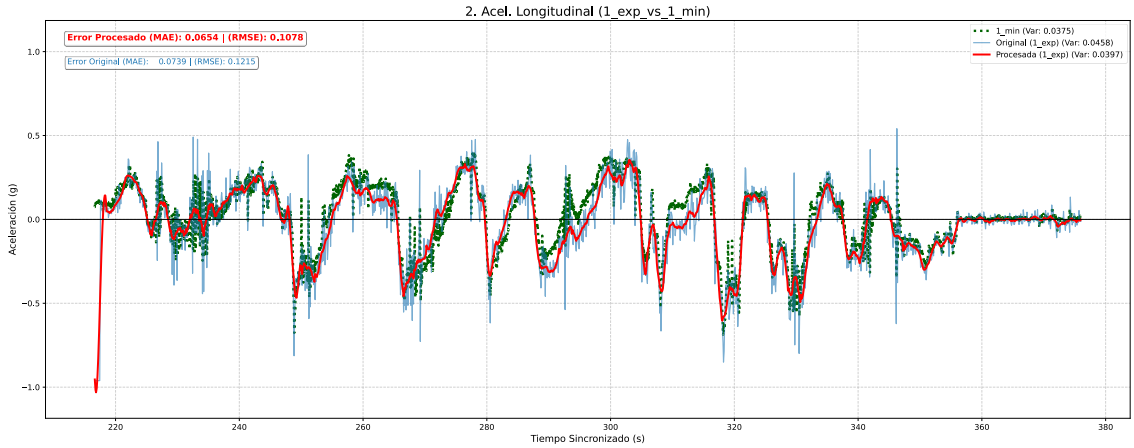


Figura 8: Comparativa gráfico aceleración longitudinal escenario de baja velocidad.

	MAE	RMSE
Señal en Bruto	0,0654 g	0,1215 g
Señal procesada	0,0739 g	0,1078 g
Porcentaje de mejora	25%	28%

Tabla 6: Comparativa MAE y RMSE en bruto vs post-procesado para aceleración lat en escenario baja velocidad

En cuanto a la varianza, inicialmente con la señal en bruto era de 0,0458 g^2, al hacer el post-procesado quedo en 0,0397 g^2, el cual es un valor semejante a la de señal de referencia, que es de 0,0375 g^2

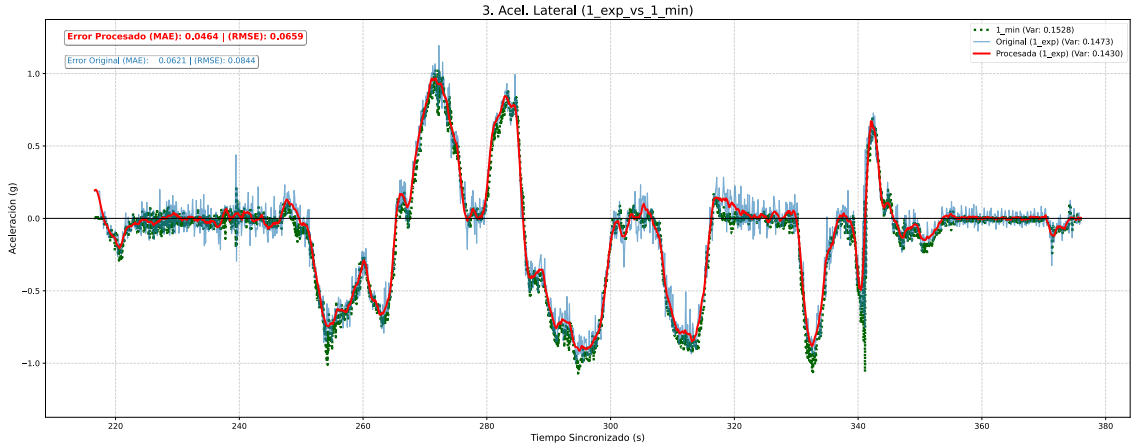


Figura 9: Comparativa gráfico aceleración lateral escenario de baja velocidad.

	MAE	RMSE
Señal en Bruto	0,0621 g	0,0914 g
Señal procesada	0,0464 g	0,0659 g
Porcentaje de mejora	25%	28%

Tabla 7: Comparativa MAE y RMSE en bruto vs post-procesado para aceleración long en escenario baja velocidad

Al analizar la varianza, se observa nuevamente que el valor de la señal bruta (0,1473 g^2) parece estar numéricamente más cerca de la referencia (0,1430 g^2) que la señal procesada (0,1528 g^2). Sin embargo, nuevamente como en el escenario anterior, la varianza no representa un empeoramiento en la fidelidad de la señal y la mejora se ve claramente en el MAE y RMSE.

4.4 Escenario de Estrés (Robustez del Filtro de Kalman)

Este escenario es la prueba más crítica para el algoritmo de post-procesado debido a que durante esta prueba, el sistema se vio afectado por severas y sucesivas perdidas de señal por parte del GPS, lo que produjo zonas donde la velocidad se perdió momentáneamente, presentando caídas espurias a 0 km/h que no se corresponden con la dinámica real del vehículo y exponen las debilidades del hardware de bajo costo.

En cuanto a la trayectoria observamos que la medición tuvo deriva, produciendo errores gráficos en la construcción de la trayectoria (ver Figura 10). Cabe destacar que ningún dato de otro sensor puede ayudar a corregir esta problemática, con lo cual se debe buscar que el GPS no sufra estos inconvenientes (para alcanzar este objetivo se debe: colocar una antena de mayor potencia o colocar un módulo de mayor calidad que tenga conexión con mayor cantidad de satélites).

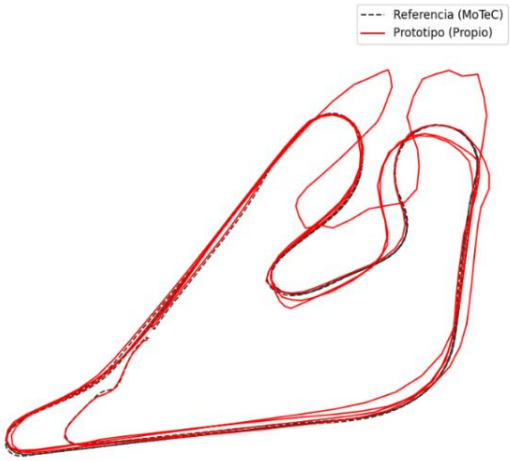


Figura 10: Comparativa posición GPS sistema MoTec vs sistema experimental (perdida señal)

La señal "Original" (azul) colapsa a cero en repetidas ocasiones, resultando en errores inutilizables para el análisis:

- Error Original (MAE): 15.3144 km/h | (RMSE): 38.9203 km/h.

Aquí es donde el algoritmo a través de fusión de sensores demuestra su valor. La "Señal Procesada" (roja) activa la lógica de corrección: ignora los datos GPS anómalos y reconstruye la velocidad utilizando la señal "donante" del Filtro de Kalman, la cual se basa en los datos del acelerómetro. El resultado es una reducción drástica del error:

- Error Procesado (MAE): 5.0287 km/h | (RMSE): 8.6849 km/h.

El algoritmo de post-procesado reduce el Error Absoluto Medio en un 67% y el Error Cuadrático Medio en un 78%. Aunque la estimación del Kalman inevitablemente acumula un pequeño error por deriva (como se ve en la discrepancia entre 275s y 375s), transforma una señal completamente corrupta en un conjunto de datos coherente y analizable (ver Figura 11).

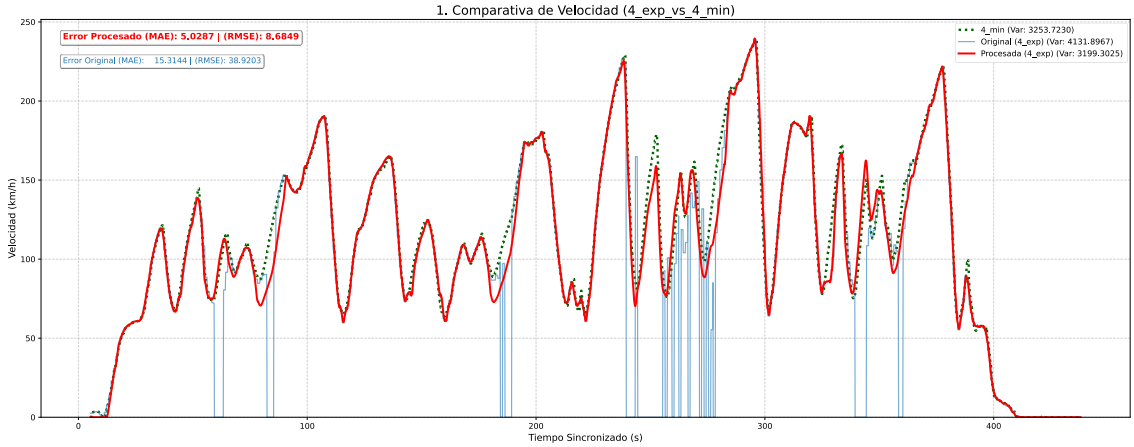


Figura 11: Comparativa gráfico velocidad escenario estrés con pérdidas de señal GPS.

El análisis sobre la varianza nos muestra que la señala en bruto tenia un valor de 4131,89 $(km/h)^2$, que luego del post-procesado se modificó quedando en 3199,30 $(km/h)^2$ el cual esta muy aproximado al valor de referencia para esta señal (3253,72 $(km/h)^2$)

Comparativa de Aceleraciones (Escenario de estrés) Dado que el sensor IMU no se vio afectado por la pérdida de GPS, por lo cual el algoritmo simplemente aplicó su filtrado Savitzky-Golay estándar, obteniendo los siguientes resultados:

Aceleración longitudinal (ver Figura 12):

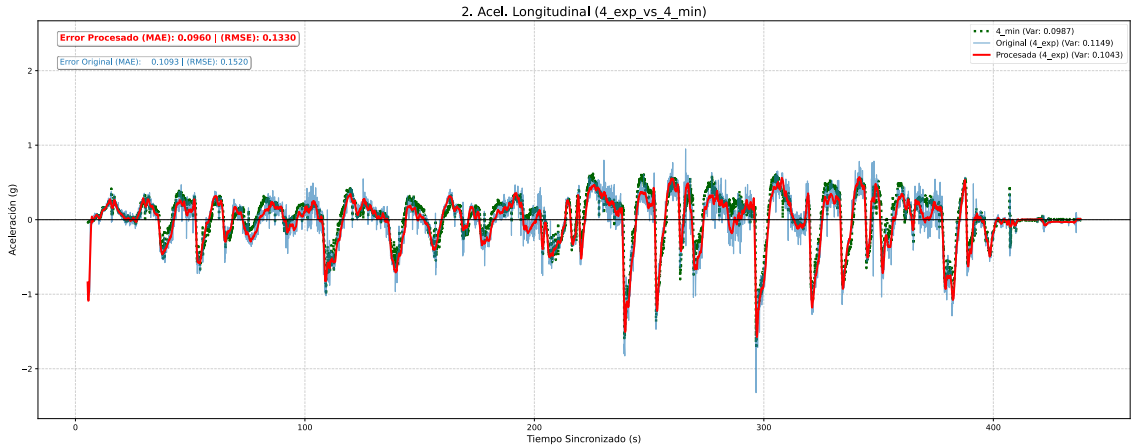


Figura 12: Comparativa gráfico aceleración long. escenario estrés con pérdidas de señal GPS.

	MAE	RMSE
Señal en Bruto	0,1093 g	0,1520 g
Señal procesada	0,0960 g	0,1330 g
Porcentaje de mejora	12%	12,5%

Tabla 8: Comparativa MAE y RMSE en bruto vs post-procesado para aceleración long en escenario de estrés.

En cuanto a la varianza, inicialmente con la señal en bruto era de 0,1149 g^2 luego del post-procesado quedo en 0,1043 g^2 que es semejante al valor de la señal de referencia, con una magnitud de 0,0987 g^2

Aceleración lateral (ver Figura 13):

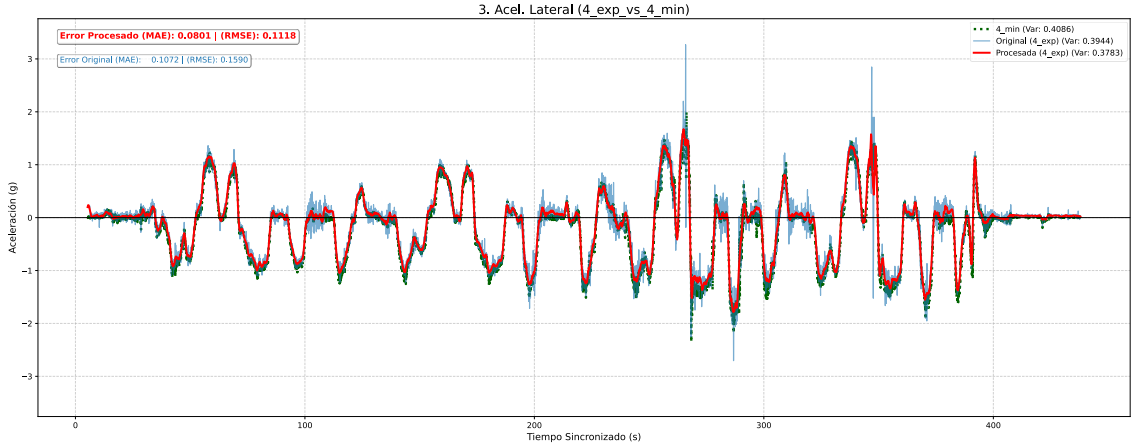


Figura 13: Comparativa gráfico aceleración lat. escenario stress con pérdidas de señal GPS.

	MAE	RMSE
Señal en Bruto	0,1072 g	0,1590 g
Señal procesada	0,0801 g	0,1118 g
Porcentaje de mejora	47%	29,6%

Tabla 9: Comparativa MAE y RMSE en bruto vs post-procesado para aceleración lat escenario estrés

Al analizar la varianza, se observa (como en todos los análisis sobre aceleración lateral) que el valor de la señal bruta (0,3944 g^2) parece estar numéricamente más cerca de la referencia (0,3783 g^2) que la señal procesada (0,4086 g^2).

En conclusión, los resultados cuantitativos validan la hipótesis planteada: el algoritmo de post-procesado no solo filtra eficazmente el ruido del hardware de bajo costo, sino que detecta y reconstruye activamente las mediciones erróneas generadas durante fallos críticos del GPS (caídas a cero). El resultado es un conjunto de datos de alta fidelidad, cuyo rendimiento es comparable al de sistemas comerciales de alta gama.

Capítulo 5. Discusión

El análisis de los datos obtenidos en la fase de validación experimental trasciende la mera cuantificación de métricas de error; permite confirmar la hipótesis central de que el post-procesamiento algorítmico permite elevar las prestaciones de un sistema de adquisición de bajo costo. Los resultados evidencian que la fidelidad del dato final depende en gran medida del software.

5.1 Validación de la Integridad de la Captura de Datos

El primer requisito para la viabilidad del algoritmo propuesto era contar con un flujo de datos crudos continuo y confiable. En este aspecto, la integración de los sensores inerciales y el módulo GNSS demostró la estabilidad necesaria para alimentar el algoritmo de post-procesado. La obtención de series temporales coherentes valida al hardware no como un fin en sí mismo, sino como una plataforma de entrada confiable, capaz de soportar las exigencias vibratorias de la pista y proveer la materia prima necesaria para que el script de Python ejecute las rutinas de mejora.

5.2 Análisis de la Validación Experimental

La comparación crítica contra el sistema de referencia (MoTeC) permitió someter al software a una prueba no planificada de robustez. Durante el análisis preliminar, se detectó una discrepancia severa en la asignación de ejes de aceleración, la cual fue diagnosticada mediante el script como un error de montaje físico del IMU (rotación de 90° respecto al eje longitudinal del chasis).

Este evento, lejos de ser un fallo negativo, se convirtió en una demostración clave de la potencia del algoritmo desarrollado:

- Capacidad de Autodiagnóstico y Corrección: El hallazgo valida el módulo de detección de ejes del software. El algoritmo no se limitó a filtrar una señal errónea, sino que, al correlacionar la derivada del GPS con los datos inerciales, fue capaz de identificar la inconsistencia vectorial y reasignar matemáticamente los ejes. Esto demuestra que el algoritmo puede compensar

errores de instalación humana, una característica vital para equipos amateurs que carecen de protocolos de montaje profesionales.

- Maximización del Hardware Económico: Se confirmó que los sensores de bajo costo poseen la sensibilidad suficiente para capturar la dinámica vehicular, pero que su utilidad depende estricta y exclusivamente de una interpretación correcta por software. Sin la intervención del algoritmo de reorientación, los datos del sensor habrían sido inservibles y con el procesamiento aplicado, se aumentó también, la fidelidad de la medición, igualando la funcionalidad de un sistema comercial.

5.2.1 Análisis de Estabilidad y Deriva GPS en situación de señal estable

Una vez que el algoritmo corrige la rotación de ejes, y los errores inducidos por el hardware y su orientación, podemos observar una alta correlación visual de los gráficos (confirmado por las mejoras en con respecto al error MAE y RMSE).

Esto nos confirma que el sistema DAQ experimental en conjunto con el algoritmo de post-procesado creado en Python, son funcionalmente comparables a un sistema comercial utilizado para el análisis de la dinámica en automovilismo de competición.

5.2.2 Análisis de Estabilidad y Deriva GPS en situación de señal deficiente

En el "Escenario de Estrés" (Sección 4.4), se observa que, si bien el Filtro de Kalman logra reconstruir la velocidad durante cortes intermitentes, existe una divergencia acumulativa en lapsos prolongados de pérdida de señal satelital (por ejemplo, en la zona de 275s a 375s).

El comportamiento observado durante pérdidas prolongadas de señal GNSS no se interpreta como un fallo del algoritmo, sino como una limitación inherente a la integración inercial sin corrección externa. Dentro del contexto de uso del sistema (orientado al análisis de dinámica vehicular) esta limitación resulta aceptable, dado que el algoritmo prioriza la coherencia temporal y dinámica de la señal por sobre la exactitud absoluta durante eventos transitorios de pérdida de señal.

Desde el punto de vista de la teoría de estimación, esto responde al comportamiento de bucle abierto del filtro. Al perderse la etapa de Corrección, el filtro depende exclusivamente de la etapa de Predicción basada en la integración de la aceleración. Dado que los acelerómetros MEMS presentan un error de sesgo (inestabilidad bias) estocástico, la integración de este error genera ruido en la señal evidenciando un fenómeno que es conocido como caminata aleatoria o Random Walk en la velocidad estimada.

Aunque el algoritmo ZUPT mitiga el bias estático, el bias dinámico remanente provoca la deriva observada.

Capítulo 6. Conclusión

El desarrollo del presente trabajo permitió demostrar que la aplicación de un algoritmo de post-procesamiento de señales constituye una herramienta eficaz para mejorar la calidad y coherencia de los datos vehiculares adquiridos mediante hardware de bajo costo. A través de la implementación integrada de técnicas de diagnóstico, sincronización, filtrado y fusión sensorial, fue posible corregir deficiencias inherentes a

este tipo de sistemas, tales como ruido, sesgos, errores de montaje y desalineaciones temporales entre sensores.

Los resultados obtenidos en ensayos reales en pista evidencian que las señales procesadas describen de manera consistente la dinámica vehicular, mostrando un alto grado de concordancia con las provistas por un sistema comercial de referencia ampliamente utilizado en el automovilismo de competición. En particular, las variables cinemáticas clave presentan errores reducidos y un comportamiento temporal coherente, lo que valida la utilidad del enfoque algorítmico propuesto para el análisis de rendimiento y la toma de decisiones en ingeniería de pista.

Asimismo, el análisis de escenarios de estrés permitió identificar las principales limitaciones del sistema, especialmente aquellas asociadas a pérdidas prolongadas de señal GPS. Estos comportamientos no se interpretan como fallos del algoritmo, sino como restricciones inherentes a la navegación inercial sin corrección externa, consideradas aceptables dentro del contexto de uso del sistema, que prioriza la coherencia dinámica por sobre la exactitud absoluta durante eventos transitorios.

En conjunto, el trabajo confirma que un enfoque basado en post-procesamiento de señales puede transformar datos adquiridos con hardware accesible en información útil y confiable para aplicaciones profesionales en automovilismo de competición, constituyendo una alternativa viable para equipos que buscan soluciones de análisis de desempeño con menores costos sin comprometer la calidad funcional de los datos.

Desde una perspectiva aplicada, el sistema desarrollado cumple con los requerimientos fundamentales de una herramienta de análisis para ingeniería de pista, alineándose con los objetivos del automovilismo de competición y demostrando que el valor diferencial del sistema reside principalmente en el tratamiento inteligente de la información más que en la complejidad del hardware empleado.

6.1 Trabajo a Futuro

A partir de los resultados y la experiencia obtenidos, se proponen las siguientes líneas de trabajo para futuras iteraciones del proyecto:

- Mejorar la calidad de modulo GPS, acelerómetro y giróscopo para obtener datos sin pérdida de información y con menor ruido.
- Expansión de Canales: Integrar más sensores al sistema, crear una conexión CAN Bus entre el sistema experimental y la ECU motor capaz de almacenar los datos que maneja la misma (rpm, combustible inyectado, temperaturas, presiones, etc). También sería interesante poder conectar potenciómetros lineales para medir el recorrido de la suspensión, sensores de posición del volante o poder sincronizar una cámara de video, para enriquecer aún más el análisis.
- Para mitigar la deriva inherente a la integración inercial en zonas de señal GNSS deficiente, se propone integrar también la velocidad de rueda mediante protocolo CAN Bus o sensores inductivos. Esta señal actuaría como una fuente secundaria de velocidad en la matriz de medición del Filtro de Kalman, bajando el error de la estimación y acotando las discrepancias generadas por la integración. De esta forma se puede reducir al mínimo la divergencia en la medición, como la observada en el escenario de estrés

- Creación de una Interfaz Gráfica de Usuario (GUI) Completa: Empaquetar el algoritmo creado en Python en una aplicación de escritorio intuitiva y que pueda ser utilizada fácilmente por cualquier usuario sin necesidad de que cuente con conocimientos en programación.

Capítulo 7. Bibliografía

ACTC. (2025). *Reglamento Deportivo y Técnico de Turismo Carretera*. Obtenido de Asociación Corredores de Turismo Carretera: https://www.actc.org.ar

APAT. (2025). *Reglamento Deportivo y Técnico de Turismo Nacional*. Obtenido de Asociación Pilotos Automóviles de Turismo: https://www.apat.org.ar

Harris, C. R. (2020). Array programming with NumPy. *Science*, 585(7825), 357–362.

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering (IEEE)*, 9(3), 90–95.

Labbe, R. (2014). *Kalman and Bayesian Filters in Python*. Obtenido de https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python

McKinney, W. (2010). Data structures for statistical computing in Python. *9th Python in Science Conference (SciPy 2010)*, (págs. (pp. 51–56)).

Milliken, W., & Milliken, D. (1995). *Race car vehicle dynamics* (2 ed.). New York: SAE International.

Motec. (2025). *MoTeC Data Acquisition Systems and Software*. Obtenido de https://www.motec.com.au

Savitzky, A. &. (1964). Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry (American Chemical Society)*, 36(8), 1627–1639.

Segers, C. (2008). *Analysis Techniques for Racecar Data Acquisition. Second edition.* Bentley Publishers.

Virtanen, P. G. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3), 261–272.

Welch, G. &. (2007). *An Introduction to the Kalman Filter.* Departamento de Ciencias de la Computación de la Universidad de Carolina del Norte (UNC-Chapel Hill).

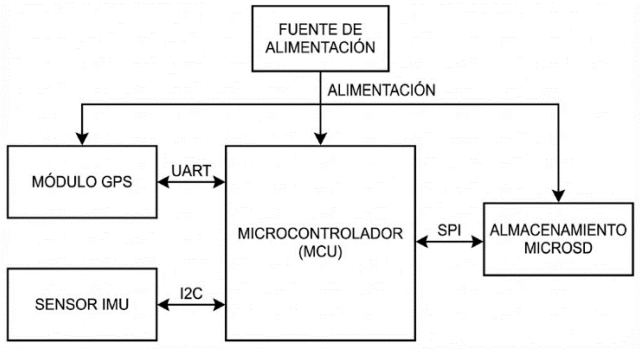
Capítulo 8. Anexos

8.1 Hardware

8.1.1 Arquitectura del Sistema

Para la adquisición de los datos cinemáticos utilizados en la validación del algoritmo, se diseñó e integró un sistema embebido basado en componentes COTS (Commercial Off-The-Shelf). La arquitectura sigue un esquema modular centralizado, donde una unidad de procesamiento gestiona la sincronización de los buses de comunicación digital para garantizar la coherencia temporal de las muestras.

A continuación, se presenta el diagrama de bloques funcional del prototipo, detallando las interconexiones lógicas entre los sensores y el núcleo de procesamiento:



8.1.2 Especificaciones de Componentes

El hardware se dimensionó para cumplir con los requisitos de frecuencia de muestreo y ancho de banda necesarios para la dinámica vehicular

- Unidad de Procesamiento Central (MCU): Se empleó una placa de desarrollo basada en un microcontrolador de 32 bits de alto rendimiento.
  Función: guardar los datos medidos por el sistema, lee los registros FIFO del IMU, decodifica las trazas NMEA del GPS y escribe los buffers en la memoria SD.
  Interfaz de Datos: SPI a 25 MHz para escritura en tarjeta SD.
- Unidad de Medición Inercial (IMU): Sensor MEMS de 6 grados de libertad (6-DoF).
  Comunicación: Bus I2C de alta velocidad (400 kHz).
  Configuración: Rango dinámico de acelerómetros ajustado a 4g (suficiente para las fuerzas laterales máximas de los vehículos de competición locales) y giroscopios a ± 500 °/s.
  Frecuencia de Muestreo: Configurada a 100 Hz para capturar transitorios de suspensión y vibraciones.
- Módulo de Posicionamiento Global (GNSS): Receptor satelital de actualización rápida.
  Tasa de Refresco: 10 Hz (10 muestras por segundo), estándar mínimo para aplicaciones en motorsport
  Protocolo: NMEA 0183 vía UART a 9600 baudios.
  Precisión de Velocidad: 0.1m/s en condiciones de cielo abierto.
- Sistema de Almacenamiento: Módulo de tarjeta MicroSD.
  Formato: Sistema de archivos FAT32.
  Salida: Archivos .csv de texto plano para post-procesamiento universal.

8.1.3 Integración y Montaje Mecánico

La electrónica fue integrada en un PCB de propósito general para asegurar la rigidez de las conexiones y minimizar el ruido eléctrico inducido por vibraciones mecánicas. Todo el conjunto se aloja en una carcasa de polímero (impresión 3D) diseñada para proteger los componentes del polvo y permitir una fijación rígida al chasis del vehículo.

Estos componentes fueron integrados en una única placa de circuito impreso (ver Figura 15) y alojados en una carcasa robusta para protegerlos de las vibraciones y condiciones del entorno de competición (ver Figura 14).

Figura 14: Prototipo final montado para evaluación en pista

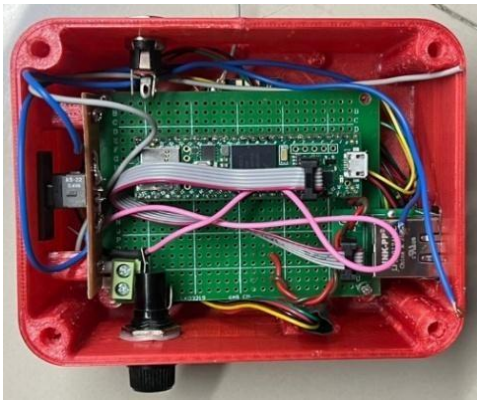


Figura 15: PCB montado en carcasa del prototipo

8.2 Código: algoritmo-post-procesadoTFI.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
from filterpy.kalman import KalmanFilter
from scipy.signal import savgol_filter
from scipy import signal
import os
import traceback
import tkinter as tk
from tkinter import filedialog

# --- 1. CONFIGURACIÓN GLOBAL ---
SAVGOL_WINDOW = 21
SAVGOL_ORDER = 3
G = 9.80665
STOP_THRESHOLD_S = 15

# 2. FUNCIONES DE DETECCIÓN DE EJES Y LAG INTERNO

def find_axes_by_correlation(df):
    print(" -> Iniciando detección de ejes por correlación con
d(Velocidad)/dt...")

    # 1. Centrar los acelerómetros usando ZUPT (Quitar bias/gravedad)
    stationary_exp = (df['spd_kmh'] < 1.0) &
(df['spd_kmh'].rolling(window=50, center=True, min_periods=1).mean() <
1.0)
    if stationary_exp.any():
        ax_bias = df.loc[stationary_exp, 'ax_ms2'].mean()
        ay_bias = df.loc[stationary_exp, 'ay_ms2'].mean()
```

```
        print(f"    - ZUPT para detección: ax_bias={ax_bias:.4f},
ay_bias={ay_bias:.4f}")
    else:
        ax_bias, ay_bias = 0.0, 0.0
        print("    - ZUPT para detección: No se encontraron puntos de
reposo, asumiendo bias=0.")

    ax_centered = df['ax_ms2'] - ax_bias
    ay_centered = df['ay_ms2'] - ay_bias

    # 2. Obtener la "verdadera" aceleración (desde GPS)
    spd_smooth_ms = savgol_filter(df['spd_kmh'] / 3.6, 51, 3)
    accel_from_speed = np.gradient(spd_smooth_ms, df['time_s'])

    # 3. Calcular correlación (para seleccionar el eje)
    accel_from_speed_clean = np.nan_to_num(accel_from_speed)
    ax_centered_clean = np.nan_to_num(ax_centered)
    ay_centered_clean = np.nan_to_num(ay_centered)

    corr_ax = np.corrcoef(accel_from_speed_clean, ax_centered_clean)[0,
1]
    corr_ay = np.corrcoef(accel_from_speed_clean, ay_centered_clean)[0,
1]

    print(f" -> Correlación con d(Velocidad)/dt: [ax]: {corr_ax:.4f},
[ay]: {corr_ay:.4f}")

    # 4. Asignar ejes
    if abs(corr_ax) > abs(corr_ay):
        long_axis_name = 'ax_ms2'
        lat_axis_name = 'ay_ms2'
        long_sign = np.sign(corr_ax)
        long_bias = ax_bias
        lat_bias = ay_bias
        imu_long_signal = ax_centered_clean
    else:
        long_axis_name = 'ay_ms2'
        lat_axis_name = 'ax_ms2'
        long_sign = np.sign(corr_ay)
        long_bias = ay_bias
        lat_bias = ax_bias
        imu_long_signal = ay_centered_clean

    print(f"    - Eje Longitudinal detectado: '{long_axis_name}' (Signo:
{'Invertido' if long_sign < 0 else 'Normal'})")

    # 5. LÓGICA DE DETECCIÓN DE LAG CORREGIDA

    dt = df['time_s'].diff().mean()
```

```
    if pd.isna(dt) or dt <= 0: dt = 0.01

    # Crear la señal IMU con el signo ya corregido
    imu_long_signal_signed = imu_long_signal * long_sign

    # Correlacionar A (Verdad-GPS) con B (IMU-Corregido)
    # Ahora el pico (argmax) SIEMPRE será la correlación correcta
    correlation = signal.correlate(accel_from_speed_clean,
imu_long_signal_signed, mode='full', method='fft')
    lags = signal.correlation_lags(len(accel_from_speed_clean),
len(imu_long_signal_signed), mode='full')

    lag_index = np.argmax(correlation)
    best_lag_samples = lags[lag_index]

    # El lag resultante es el offset de IMU relativo a GPS
    # Si lag > 0, IMU está RETRASADO (ocurre DESPUÉS) que GPS.
    internal_lag_s = best_lag_samples * dt

    print(f"    - Latencia interna IMU vs GPS detectada:
{internal_lag_s:.3f} s")

    # 6. Detectar signo lateral
    lat_sign = detect_lateral_axis_sign(df, lat_axis_name, 'lat', 'lon')

    # Retornar el nuevo valor de lag
    return long_axis_name, long_sign, long_bias, lat_axis_name, lat_sign,
lat_bias, internal_lag_s

def detect_lateral_axis_sign(df, lat_axis_name, lat_col, lon_col):
    """Detecta el signo del eje lateral usando los giros del GPS."""
    print(" -> Iniciando detección de signo de eje lateral por GPS...")
    if not all(col in df.columns for col in [lat_col, lon_col]):
        print(f"    - Faltan columnas '{lat_col}'/'{lon_col}'. Asumiendo
signo lateral Normal (1.0).")
        return 1.0

    # Evitar cálculos en dataframes vacíos o con NaNs
    if df[lat_col].isnull().all() or df[lon_col].isnull().all():
        print(f"    - Columnas '{lat_col}'/'{lon_col}' están vacías
(NaN). Asumiendo signo lateral Normal (1.0).")
        return 1.0

    df['rad_lat'] = np.radians(df[lat_col])
    df['rad_lon'] = np.radians(df[lon_col])
    df['dLat'] = df['rad_lat'].diff()
    df['dLon'] = df['rad_lon'].diff()

    window = 100
```

```
    for i in range(len(df) - window):
        if df['spd_kmh'].iloc[i] > 20:
            v1_lat, v1_lon = df['dLat'].iloc[i], df['dLon'].iloc[i]
            v2_lat, v2_lon = df['dLat'].iloc[i+10], df['dLon'].iloc[i+10]
            if any(pd.isna([v1_lat, v1_lon, v2_lat, v2_lon])): continue

            cross_product = v1_lon * v2_lat - v1_lat * v2_lon

            if abs(cross_product) > 1e-12:
                avg_lat_accel = df[lat_axis_name].iloc[i:i+window].mean()

                if (cross_product > 0 and avg_lat_accel < 0) or \
                   (cross_product < 0 and avg_lat_accel > 0):
                    print("    - Signo del eje lateral: Invertido
(Corregido según giro GPS)")
                    return -1.0

                print("    - Signo del eje lateral: Normal (Confirmado
por giro GPS)")
                return 1.0

    print("    - No se encontró un giro claro para validar el eje
lateral. Asumiendo signo Normal (1.0).")
    return 1.0

# 3. FUNCIÓN DE KALMAN
def apply_kalman_filter(df, time_col='time_s', accel_col='a_long_g',
speed_col='spd_kmh'):
    print(f" -> Generando señal Kalman de fondo para usar como
'donante'...")
    R_noise = 2.0; Q_noise = 0.05
    dt = df[time_col].diff().mean()
    if pd.isna(dt) or dt == 0: dt = 0.01

    kf = KalmanFilter(dim_x=2, dim_z=1)

    # Inicializar velocidad de Kalman con la primera velocidad GPS válida
    initial_speed_ms = df[speed_col].fillna(0).iloc[0] / 3.6

    kf.x = np.array([[0.], [initial_speed_ms]])
    kf.F = np.array([[1., dt], [0., 1.]]); kf.B = np.array([[(0.5 *
dt**2)], [dt]]); kf.H = np.array([[0., 1.]]); kf.R =
np.array([[R_noise]]);
    kf.Q = np.array([[(0.25*dt**4), (0.5*dt**3)], [(0.5*dt**3),
(dt**2)]]) * Q_noise

    accel_measurements = df[accel_col].values * G
    speed_measurements_ms = df[speed_col].values / 3.6
    filtered_states = []
```

```
    for accel, speed_gps in zip(accel_measurements,
speed_measurements_ms):
        # Manejar NaNs en las mediciones
        if pd.isna(accel):
            accel = 0.0 # Asumir 0 aceleración si falta el dato

        kf.predict(u=accel)

        # No actualizar si la velocidad GPS es inválida o si estamos
parados
        if pd.isna(speed_gps) or speed_gps < 1.0 / 3.6:
            kf.R = np.array([[R_noise * 10000]]) # Confianza muy baja en
la medición
        elif speed_gps < 5.0 / 3.6:
            kf.R = np.array([[R_noise * 100]]) # Confianza baja
        else:
            kf.R = np.array([[R_noise]]) # Confianza normal

        kf.update(z=speed_gps)
        filtered_states.append(kf.x)

    full_states_array = np.array(filtered_states)
    filtered_velocities_ms = full_states_array[:, 1, 0]
    return np.maximum(0, filtered_velocities_ms * 3.6)

# 4. FUNCIÓN DE PLOTEO
def create_comparison_plot(data, col1, col2, col1_filtered, title,
ylabel, filename,
                           label1, label2, is_acceleration=False,
                           var_original=None, var_processed=None,
var_reference=None,
                           mae_raw=None, mae_proc=None, rmse_raw=None,
rmse_proc=None):
    """
    Crea un gráfico comparativo, incluyendo Varianza en leyenda y
métricas de Error en el gráfico.
    """
    plt.figure(figsize=(20, 8))

    # Crear etiquetas con Varianza
    label_ref = f'{label2} (Var: {var_reference:.4f})' if var_reference
is not None else label2
    label_orig = f'Original ({label1}) (Var: {var_original:.4f})' if
var_original is not None else f'Original ({label1})'
    label_proc = f'Procesada ({label1}) (Var: {var_processed:.4f})' if
var_processed is not None else f'Procesada ({label1})'

    # Plotear Señales
```

```
    plt.plot(data['time_s'], data[col2], linestyle=':',
color='DarkGreen', label=label_ref, linewidth=3.0)
    plt.plot(data['time_s'], data[col1], linestyle='-', color='C0',
label=label_orig, linewidth=1.5, alpha=0.6)
    plt.plot(data['time_s'], data[col1_filtered], linestyle='-',
color='red', label=label_proc, linewidth=2.5)

    # Configuración de Ejes
    if is_acceleration:
        plt.axhline(0, color='black', linestyle='-', linewidth=0.5)
        valid_data = pd.concat([data[col1], data[col1_filtered],
data[col2]]).dropna()
        if not valid_data.empty:
            max_abs = valid_data.abs().max()
            y_limit = max(0.2, max_abs * 1.15)
            plt.ylim(-y_limit, y_limit)
    else:
        plt.ylim(bottom=0)

    plt.title(title, fontsize=16)
    plt.xlabel('Tiempo Sincronizado (s)', fontsize=12)
    plt.ylabel(ylabel, fontsize=12)
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.legend(loc='upper right', fontsize=10)

    # Añadir texto de Error (MAE/RMSE)
    if mae_raw is not None and mae_proc is not None and rmse_raw is not
None and rmse_proc is not None:
        ax = plt.gca()
        ymin, ymax = ax.get_ylim()
        xmin, xmax = ax.get_xlim()

        # Posición en la esquina superior izquierda
        x_pos = xmin + (xmax - xmin) * 0.02 # 2% desde la izquierda
        y_pos_base = ymax - (ymax - ymin) * 0.05 # 5% desde arriba

        # Texto para Procesado
        text_proc = f"Error Procesado (MAE): {mae_proc:.4f} | (RMSE):
{rmse_proc:.4f}"
        plt.text(x_pos, y_pos_base, text_proc, fontsize=11, color='red',
weight='bold',
                 bbox=dict(facecolor='white', alpha=0.8,
boxstyle='round,pad=0.3'))

        # Texto para Raw
        text_raw  = f"Error Original (MAE):    {mae_raw:.4f} | (RMSE):
{rmse_raw:.4f}"
        plt.text(x_pos, y_pos_base - (ymax - ymin) * 0.06, text_raw,
fontsize=10, color='C0',
```

```
                bbox=dict(facecolor='white', alpha=0.8,
boxstyle='round,pad=0.3'))

    plt.tight_layout()

    # --- MODIFICACIÓN OPCIÓN 1: GUARDAR COMO SVG ---
    plt.savefig(filename, format='svg', bbox_inches='tight')
    # -----------------------------------------------

    plt.close()
    print(f"Gráfico guardado en: {filename}")

# 5. FUNCIONES DE LÓGICA MODULARIZADAS

def load_and_process_exp_file(filepath):
    """
    Carga un archivo _exp y aplica corrección de latencia interna,
    luego ejes, ZUPT y Kalman.
    """
    print(f"\n--- Procesando Archivo Experimental:
{os.path.basename(filepath)} ---")
    df_exp = pd.read_csv(filepath)
    df_exp.columns = df_exp.columns.str.strip()

    if 't_ms' not in df_exp.columns:
        raise KeyError("El archivo _exp no contiene la columna 't_ms'.
Verifique el archivo.")
    if 'ax_ms2' not in df_exp.columns or 'ay_ms2' not in df_exp.columns:
        raise KeyError("El archivo _exp no contiene 'ax_ms2' o 'ay_ms2'.
Verifique el archivo.")

    df_exp['time_s'] = (df_exp['t_ms'] - df_exp['t_ms'].iloc[0]) / 1000.0

    # 1. Detección automática
    long_axis, long_sign, long_bias, lat_axis, lat_sign, lat_bias,
internal_lag_s = find_axes_by_correlation(df_exp)

    # 2. CORRECCIÓN DE LATENCIA INTERNA
    if abs(internal_lag_s) > 0.001: # Solo corregir si el lag es
significativo
        print(f" -> Aplicando corrección de latencia interna de
{internal_lag_s:.3f} s a los acelerómetros.")

        # Crear funciones de interpolación basadas en el tiempo ORIGINAL
        interp_ax = interp1d(df_exp['time_s'], df_exp['ax_ms2'],
kind='linear', bounds_error=False, fill_value=0.0)
        interp_ay = interp1d(df_exp['time_s'], df_exp['ay_ms2'],
kind='linear', bounds_error=False, fill_value=0.0)
```

```
        # Calcular el vector de tiempo corregido
        corrected_time_vector = df_exp['time_s'] - internal_lag_s

        # Sobrescribir las columnas 'ax_ms2' y 'ay_ms2' con los valores
re-muestreados
        df_exp['ax_ms2'] = interp_ax(corrected_time_vector)
        df_exp['ay_ms2'] = interp_ay(corrected_time_vector)
        print(" -> Corrección de latencia aplicada.")

    # 3. Aplicar corrección de ZUPT y signos
    df_exp['a_long_g_centered'] = long_sign * (df_exp[long_axis] -
long_bias) / G
    df_exp['a_lat_g_centered']  = lat_sign * (df_exp[lat_axis] -
lat_bias) / G

    # 4. Procesamiento Híbrido de Velocidad
    kalman_full_signal = apply_kalman_filter(df_exp,
accel_col='a_long_g_centered', speed_col='spd_kmh')

    is_bad_gps_region = (df_exp['spd_kmh'] < 5.0).rolling(window=100,
center=True, min_periods=1).max().astype(bool)
    is_stopped = df_exp['spd_kmh'] < 1.0
    stop_blocks = (is_stopped.diff().ne(0)).cumsum()
    block_durations = stop_blocks.map(stop_blocks.value_counts())

    dt_calc = df_exp['time_s'].diff().mean()
    if pd.isna(dt_calc) or dt_calc == 0: dt_calc = 0.01

    is_long_stop = (block_durations * dt_calc > STOP_THRESHOLD_S) &
is_stopped
    region_to_correct = is_bad_gps_region & ~is_long_stop
    print(f" -> Se corregirán {region_to_correct.sum()} muestras de GPS
con Kalman.")

    hybrid_signal = np.where(region_to_correct, kalman_full_signal,
df_exp['spd_kmh'])
    df_exp['spd_kmh_processed'] = savgol_filter(hybrid_signal,
SAVGOL_WINDOW, SAVGOL_ORDER)

    # 5. Filtrado de Aceleraciones
    df_exp['a_long_g_filtered'] =
savgol_filter(df_exp['a_long_g_centered'], SAVGOL_WINDOW, SAVGOL_ORDER)
    df_exp['a_lat_g_filtered'] =
savgol_filter(df_exp['a_lat_g_centered'], SAVGOL_WINDOW, SAVGOL_ORDER)

    # 6. Renombrar columnas
    df_exp.rename(columns={
        'spd_kmh': 'spd_raw',
        'spd_kmh_processed': 'spd_processed',
```

```
        'a_long_g_centered': 'a_long_raw',
        'a_long_g_filtered': 'a_long_processed',
        'a_lat_g_centered': 'a_lat_raw',
        'a_lat_g_filtered': 'a_lat_processed',
    }, inplace=True)

    print(f"--- Fin de procesamiento: {os.path.basename(filepath)} ---")
    return df_exp

def load_and_process_min_file(filepath):
    """
    Carga un archivo _min, aplica corrección de latencia interna,
    y luego aplica corrección de ejes.
    """
    print(f"\n--- Procesando Archivo de Referencia (con Detección):
{os.path.basename(filepath)} ---")
    df_min = pd.read_csv(filepath)
    df_min.columns = df_min.columns.str.strip()

    if 'time' not in df_min.columns:
        raise KeyError("El archivo _min no contiene la columna 'time'.
Verifique el archivo.")

    df_min.rename(columns={'time': 'time_s'}, inplace=True)

    # 1. Preparar columnas para la detección
    if 'a_longitudinal' not in df_min.columns or 'a_lateral' not in
df_min.columns:
        if 'ax_ms2' in df_min.columns and 'ay_ms2' in df_min.columns:
            print(" -> Usando 'ax_ms2' y 'ay_ms2' como entrada.")
        else:
            raise KeyError("Columnas de aceleración de referencia no
encontradas.")
    else:
        print(" -> Columnas 'a_longitudinal'/'a_lateral' encontradas.
Convirtiendo G's a m/s^2 para detección.")
        df_min['ax_ms2'] = df_min['a_longitudinal'] * G
        df_min['ay_ms2'] = df_min['a_lateral'] * G

    # 2. Detección automática
    long_axis, long_sign, long_bias, lat_axis, lat_sign, lat_bias,
internal_lag_s = find_axes_by_correlation(df_min)

    # 3. CORRECCIÓN DE LATENCIA INTERNA
    if abs(internal_lag_s) > 0.001:
        print(f" -> Aplicando corrección de latencia interna de
{internal_lag_s:.3f} s a los acelerómetros.")
```

```
        interp_ax = interp1d(df_min['time_s'], df_min['ax_ms2'],
kind='linear', bounds_error=False, fill_value=0.0)
        interp_ay = interp1d(df_min['time_s'], df_min['ay_ms2'],
kind='linear', bounds_error=False, fill_value=0.0)

        corrected_time_vector = df_min['time_s'] - internal_lag_s

        df_min['ax_ms2'] = interp_ax(corrected_time_vector)
        df_min['ay_ms2'] = interp_ay(corrected_time_vector)
        print(" -> Corrección de latencia aplicada.")

    # 4. Aplicar corrección de ZUPT y signos (a las columnas ya
corregidas en lag)
    df_min['a_long_g_centered'] = long_sign * (df_min[long_axis] -
long_bias) / G
    df_min['a_lat_g_centered']  = lat_sign * (df_min[lat_axis] -
lat_bias) / G

    # 5. Renombrar columnas
    df_min.rename(columns={
        'spd_kmh': 'spd_processed',
        'a_long_g_centered': 'a_long_processed',
        'a_lat_g_centered': 'a_lat_processed',
    }, inplace=True)

    # 6. Añadir columnas "raw"
    df_min['spd_raw'] = df_min['spd_processed']
    df_min['a_long_raw'] = df_min['a_long_processed']
    df_min['a_lat_raw'] = df_min['a_lat_processed']

    print(f"--- Fin de procesamiento: {os.path.basename(filepath)} ---")
    return df_min

def sync_and_plot_data(df1, df2, label1, label2, output_dir_base):
    """
    Sincroniza, calcula errores/varianzas y genera los 3 gráficos
comparativos.
    """
    print(f"\n--- Iniciando Sincronización y Ploteo: {label1} vs {label2}
---")

    # E. SINCRONIZACIÓN (CORRELACIÓN CRUZADA)
    print(" -> Calculando time shift robusto usando Correlación
Cruzada...")

    sig1 = np.nan_to_num(df1['spd_processed'].values)
    sig2 = np.nan_to_num(df2['spd_processed'].values)

    dt_common = df1['time_s'].diff().mean()
```

```
    if pd.isna(dt_common) or dt_common <= 0:
        dt_common_df2 = df2['time_s'].diff().mean()
        dt_common = 0.02 if (pd.isna(dt_common_df2) or dt_common_df2 <=
0) else dt_common_df2

    interp1 = interp1d(df1['time_s'], sig1, kind='linear',
bounds_error=False, fill_value=0.0)
    interp2 = interp1d(df2['time_s'], sig2, kind='linear',
bounds_error=False, fill_value=0.0)

    t_min = min(df1['time_s'].min(), df2['time_s'].min())
    t_max = max(df1['time_s'].max(), df2['time_s'].max())

    if (t_max - t_min) / dt_common > 500000:
        print(f"ADVERTENCIA: El rango de tiempo es demasiado grande.
Reduciendo la resolución de sincronización.")
        dt_common = (t_max - t_min) / 500000.0

    common_time_grid = np.arange(t_min, t_max, dt_common)

    resampled1 = interp1(common_time_grid)
    resampled2 = interp2(common_time_grid)

    correlation = signal.correlate(np.nan_to_num(resampled2),
np.nan_to_num(resampled1), mode='full', method='fft')
    lags = signal.correlation_lags(len(resampled2), len(resampled1),
mode='full')

    lag_index = np.argmax(correlation)
    best_lag_samples = lags[lag_index]

    time_shift_s = best_lag_samples * dt_common
    print(f"Time Shift (Correlación Cruzada) calculado:
{time_shift_s:.2f} s")

    # F. INTERPOLACIÓN Y UNIFICACIÓN

    df1['synced_time_s'] = df1['time_s'] + time_shift_s
    min_common_time = max(df1['synced_time_s'].min(),
df2['time_s'].min())
    max_common_time = min(df1['synced_time_s'].max(),
df2['time_s'].max())

    if min_common_time >= max_common_time:
        print(f"ERROR: No hay solapamiento de tiempo (Min:
{min_common_time:.2f}, Max: {max_common_time:.2f}). Verifique los
archivos.")
        return
```

```
    time_mask = (df2['time_s'] >= min_common_time) & (df2['time_s'] <=
max_common_time)
    df_ref_trimmed = df2.loc[time_mask].copy()
    plotting_time = df_ref_trimmed['time_s']

    if plotting_time.empty:
        print("ERROR: No se encontraron datos en el rango de tiempo
común.")
        return

    df_synced = pd.DataFrame({'time_s': plotting_time.values})

    interp_cols = {
        'df1_spd_raw': df1['spd_raw'].values, 'df1_spd_processed':
df1['spd_processed'].values,
        'df1_a_long_raw': df1['a_long_raw'].values,
'df1_a_long_processed': df1['a_long_processed'].values,
        'df1_a_lat_raw': df1['a_lat_raw'].values, 'df1_a_lat_processed':
df1['a_lat_processed'].values,
    }
    for col, data in interp_cols.items():
        interp_func = interp1d(df1['synced_time_s'], data, kind='linear',
bounds_error=False, fill_value=np.nan)
        df_synced[col] = interp_func(plotting_time.values)

    df_synced['df2_spd_processed'] =
df_ref_trimmed['spd_processed'].values
    df_synced['df2_a_long_processed'] =
df_ref_trimmed['a_long_processed'].values
    df_synced['df2_a_lat_processed'] =
df_ref_trimmed['a_lat_processed'].values

    df_synced.bfill(inplace=True)
    df_synced.ffill(inplace=True)
    if df_synced.empty:
        print("ERROR: El dataframe final está vacío.")
        return

    # G. CÁLCULO DE ERRORES (MAE y RMSE)

    # Velocidad
    mae_spd_raw = np.mean(np.abs(df_synced['df1_spd_raw'] -
df_synced['df2_spd_processed']))
    mae_spd_proc = np.mean(np.abs(df_synced['df1_spd_processed'] -
df_synced['df2_spd_processed']))
    rmse_spd_raw = np.sqrt(np.mean((df_synced['df1_spd_raw'] -
df_synced['df2_spd_processed'])**2))
    rmse_spd_proc = np.sqrt(np.mean((df_synced['df1_spd_processed'] -
df_synced['df2_spd_processed'])**2))
```

```
    # Acel. Longitudinal
    mae_long_raw = np.mean(np.abs(df_synced['df1_a_long_raw'] -
df_synced['df2_a_long_processed']))
    mae_long_proc = np.mean(np.abs(df_synced['df1_a_long_processed'] -
df_synced['df2_a_long_processed']))
    rmse_long_raw = np.sqrt(np.mean((df_synced['df1_a_long_raw'] -
df_synced['df2_a_long_processed'])**2))
    rmse_long_proc = np.sqrt(np.mean((df_synced['df1_a_long_processed'] -
df_synced['df2_a_long_processed'])**2))

    # Acel. Lateral
    mae_lat_raw = np.mean(np.abs(df_synced['df1_a_lat_raw'] -
df_synced['df2_a_lat_processed']))
    mae_lat_proc = np.mean(np.abs(df_synced['df1_a_lat_processed'] -
df_synced['df2_a_lat_processed']))
    rmse_lat_raw = np.sqrt(np.mean((df_synced['df1_a_lat_raw'] -
df_synced['df2_a_lat_processed'])**2))
    rmse_lat_proc = np.sqrt(np.mean((df_synced['df1_a_lat_processed'] -
df_synced['df2_a_lat_processed'])**2))

    print("\n--- Métricas de Error (vs Referencia) ---")
    print(f"Canal Velocidad ({label1}):")
    print(f"  -> Error Promedio (MAE) [Raw]:      {mae_spd_raw:.4f}
km/h")
    print(f"  -> Error Promedio (MAE) [Procesado]: {mae_spd_proc:.4f}
km/h  (Mejora: {mae_spd_raw - mae_spd_proc:.4f})")
    print(f"  -> Error (RMSE) [Raw]:             {rmse_spd_raw:.4f}
km/h")
    print(f"  -> Error (RMSE) [Procesado]:        {rmse_spd_proc:.4f}
km/h  (Mejora: {rmse_spd_raw - rmse_spd_proc:.4f})")

    print(f"\nCanal Acel. Long. ({label1}):")
    print(f"  -> Error Promedio (MAE) [Raw]:      {mae_long_raw:.4f} g")
    print(f"  -> Error Promedio (MAE) [Procesado]: {mae_long_proc:.4f}
g  (Mejora: {mae_long_raw - mae_long_proc:.4f})")
    print(f"  -> Error (RMSE) [Raw]:             {rmse_long_raw:.4f} g")
    print(f"  -> Error (RMSE) [Procesado]:        {rmse_long_proc:.4f}
g  (Mejora: {rmse_long_raw - rmse_long_proc:.4f})")

    print(f"\nCanal Acel. Lat. ({label1}):")
    print(f"  -> Error Promedio (MAE) [Raw]:      {mae_lat_raw:.4f} g")
    print(f"  -> Error Promedio (MAE) [Procesado]: {mae_lat_proc:.4f}
g  (Mejora: {mae_lat_raw - mae_lat_proc:.4f})")
    print(f"  -> Error (RMSE) [Raw]:             {rmse_lat_raw:.4f} g")
    print(f"  -> Error (RMSE) [Procesado]:        {rmse_lat_proc:.4f}
g  (Mejora: {rmse_lat_raw - rmse_lat_proc:.4f})")
    print("----------------------------------------")
```

```
    # H. CÁLCULO DE VARIANZAS
    var_spd_orig = df_synced['df1_spd_raw'].var()
    var_spd_proc = df_synced['df1_spd_processed'].var()
    var_spd_ref  = df_synced['df2_spd_processed'].var()
    var_long_orig = df_synced['df1_a_long_raw'].var()
    var_long_proc = df_synced['df1_a_long_processed'].var()
    var_long_ref  = df_synced['df2_a_long_processed'].var()
    var_lat_orig = df_synced['df1_a_lat_raw'].var()
    var_lat_proc = df_synced['df1_a_lat_processed'].var()
    var_lat_ref  = df_synced['df2_a_lat_processed'].var()

    # I. GENERACIÓN DE GRÁFICOS (con Varianza y Errores)
    os.makedirs(output_dir_base, exist_ok=True)
    plot_title_prefix = f"{label1}_vs_{label2}"

    # --- MODIFICACIÓN OPCIÓN 1: NOMBRES DE ARCHIVO SVG ---

    create_comparison_plot(df_synced,
                           'df1_spd_raw', 'df2_spd_processed',
'df1_spd_processed',
                           f'1. Comparativa de Velocidad
({plot_title_prefix})',
                           'Velocidad (km/h)',
                           os.path.join(output_dir_base,
f'{plot_title_prefix}_velocidad.svg'), # .svg
                           label1, label2, False,
                           var_original=var_spd_orig,
var_processed=var_spd_proc, var_reference=var_spd_ref,
                           mae_raw=mae_spd_raw, mae_proc=mae_spd_proc,
rmse_raw=rmse_spd_raw, rmse_proc=rmse_spd_proc)

    create_comparison_plot(df_synced,
                           'df1_a_long_raw', 'df2_a_long_processed',
'df1_a_long_processed',
                           f'2. Acel. Longitudinal
({plot_title_prefix})',
                           'Aceleración (g)',
                           os.path.join(output_dir_base,
f'{plot_title_prefix}_acel_long.svg'), # .svg
                           label1, label2, True,
                           var_original=var_long_orig,
var_processed=var_long_proc, var_reference=var_long_ref,
                           mae_raw=mae_long_raw, mae_proc=mae_long_proc,
rmse_raw=rmse_long_raw, rmse_proc=rmse_long_proc)

    create_comparison_plot(df_synced,
                           'df1_a_lat_raw', 'df2_a_lat_processed',
'df1_a_lat_processed',
                           f'3. Acel. Lateral ({plot_title_prefix})',
```

```
                            'Aceleración (g)',
                            os.path.join(output_dir_base,
f'{plot_title_prefix}_acel_lat.svg'), # .svg
                            label1, label2, True,
                            var_original=var_lat_orig,
var_processed=var_lat_proc, var_reference=var_lat_ref,
                            mae_raw=mae_lat_raw, mae_proc=mae_lat_proc,
rmse_raw=rmse_lat_raw, rmse_proc=rmse_lat_proc)

    print(f"--- Gráficos guardados en: {output_dir_base} ---")

# 6. PUNTO DE ENTRADA INTERACTIVO

def main():
    """
    Función principal que muestra el menú y guía al usuario.
    """
    print("=====================================================")
    print(" Herramienta de Comparación y Corrección de Datos VBO ")
    print("=====================================================")

    while True:
        print("\n¿Qué tipo de comparación deseas realizar?")
        print("  1. Comparar Experimental (_exp) vs Referencia
(_min)  (Modo estándar)")
        print("  2. Comparar Experimental (_exp) vs Experimental (_exp)
(Ambos se procesarán)")
        print("  3. Comparar Referencia (_min) vs Referencia
(_min)    (Ambos se centrarán)")
        print("  0. Salir")

        choice = input("Ingresa tu opción (1, 2, 3 o 0): ")

        if choice == '1':
            print("\n--- Modo 1: Exp vs Min ---")
            try:
                file1_path = filedialog.askopenfilename(title="1.
Selecciona el archivo Experimental (_exp)")
                if not file1_path: raise InterruptedError("Selección
cancelada")

                file2_path = filedialog.askopenfilename(title="2.
Selecciona el archivo de Referencia (_min)")
                if not file2_path: raise InterruptedError("Selección
cancelada")

                df1 = load_and_process_exp_file(file1_path)
                df2 = load_and_process_min_file(file2_path)
```

```
                label1 = os.path.basename(file1_path).replace('.csv', '')
                label2 = os.path.basename(file2_path).replace('.csv', '')
                output_dir = os.path.join(os.path.dirname(file1_path),
f"Comparacion_{label1}_vs_{label2}")

                sync_and_plot_data(df1, df2, label1, label2, output_dir)

            except InterruptedError as e:
                print(f"Operación cancelada: {e}")
            except Exception as e:
                print(f"ERROR FATAL durante el Modo 1: {e}")
                traceback.print_exc()

        elif choice == '2':
            print("\n--- Modo 2: Exp vs Exp ---")
            try:
                file1_path = filedialog.askopenfilename(title="1.
Selecciona el PRIMER archivo Experimental (_exp)")
                if not file1_path: raise InterruptedError("Selección
cancelada")

                file2_path = filedialog.askopenfilename(title="2.
Selecciona el SEGUNDO archivo Experimental (_exp) (Este será la
referencia verde)")
                if not file2_path: raise InterruptedError("Selección
cancelada")

                df1 = load_and_process_exp_file(file1_path)
                df2 = load_and_process_exp_file(file2_path)

                label1 = os.path.basename(file1_path).replace('.csv', '')
                label2 = os.path.basename(file2_path).replace('.csv', '')
                output_dir = os.path.join(os.path.dirname(file1_path),
f"Comparacion_{label1}_vs_{label2}")

                sync_and_plot_data(df1, df2, label1, label2, output_dir)

            except InterruptedError as e:
                print(f"Operación cancelada: {e}")
            except Exception as e:
                print(f"ERROR FATAL durante el Modo 2: {e}")
                traceback.print_exc()

        elif choice == '3':
            print("\n--- Modo 3: Min vs Min ---")
            try:
                file1_path = filedialog.askopenfilename(title="1.
Selecciona el PRIMER archivo de Referencia (_min)")
```

```
                if not file1_path: raise InterruptedError("Selección
cancelada")

                file2_path = filedialog.askopenfilename(title="2.
Selecciona el SEGUNDO archivo de Referencia (_min) (Este será la
referencia verde)")
                if not file2_path: raise InterruptedError("Selección
cancelada")

                df1 = load_and_process_min_file(file1_path)
                df2 = load_and_process_min_file(file2_path)

                label1 = os.path.basename(file1_path).replace('.csv', '')
                label2 = os.path.basename(file2_path).replace('.csv', '')
                output_dir = os.path.join(os.path.dirname(file1_path),
f"Comparacion_{label1}_vs_{label2}")

                sync_and_plot_data(df1, df2, label1, label2, output_dir)

            except InterruptedError as e:
                print(f"Operación cancelada: {e}")
            except Exception as e:
                print(f"ERROR FATAL durante el Modo 3: {e}")
                traceback.print_exc()

        elif choice == '0':
            print("Saliendo del programa.")
            break

        else:
            print("Opción no válida. Por favor, ingresa 1, 2, 3 o 0.")

if __name__ == '__main__':
    try:
        root = tk.Tk()
        root.withdraw()
        main()
    except Exception as e:
        print(f"Se produjo un error al iniciar el script: {e}")
    finally:
        print("\nAnálisis finalizado.")
```