UNIVERSIDAD NACIONAL DE CHIMBORAZO
FACULTAD DE INGENIERÍA
CARRERA TELECOMUNICACIONES

Implementación de un sistema para optimizar el posicionamiento de un vehículo mediante Machine Learning y filtrado de posiciones geolocalizadas

Trabajo de Titulación para optar al título de
Ingeniero en Telecomunicaciones

Autor:
Francis Winder Remache Yucailla

Tutor:
Mgs. José Luis Jinez Tapia

Riobamba, Ecuador. 2025

DECLARATORIA DE AUTORÍA

Yo, **Francis Winder Remache Yucailla**, con cédula de ciudadanía **0605638444**, autor (a) (s) del trabajo de investigación titulado: **Implementación de un sistema para optimizar el posicionamiento de un vehículo mediante Machine Learning y filtrado de posiciones geolocalizadas**, certifico que la producción, ideas, opiniones, criterios, contenidos y conclusiones expuestas son de mí exclusiva responsabilidad.

Asimismo, cedo a la Universidad Nacional de Chimborazo, en forma no exclusiva, los derechos para su uso, comunicación pública, distribución, divulgación y/o reproducción total o parcial, por medio físico o digital; en esta cesión se entiende que el cesionario no podrá obtener beneficios económicos. La posible reclamación de terceros respecto de los derechos de autor (a) de la obra referida, será de mi entera responsabilidad; librando a la Universidad Nacional de Chimborazo de posibles obligaciones.

En Riobamba, 24 de noviembre de 2025.

Francis Winder Remache Yucailla
C.I: 0605638444

DICTAMEN FAVORABLE DEL PROFESOR TUTOR

Quien suscribe, **José Luis Jinez Tapia** catedrático adscrito a la Facultad de Ingeniería, por medio del presente documento certifico haber asesorado y revisado el desarrollo del trabajo de investigación titulado: **Implementación de un sistema para optimizar el posicionamiento de un vehículo mediante Machine Learning y filtrado de posiciones geolocalizadas**, bajo la autoría de **Francis Winder Remache Yucailla**; por lo que se autoriza ejecutar los trámites legales para su sustentación.

Es todo cuanto informar en honor a la verdad; en Riobamba, a los 24 días del mes de noviembre de 2025.

Firmado electrónicamente por:
JOSE LUIS JINEZ TAPIA
Validar únicamente con FirmaEC

Mgs. José Luis Jinez Tapia
C.I: 0602899007

CERTIFICADO DE LOS MIEMBROS DEL TRIBUNAL

Quienes suscribimos, catedráticos designados Miembros del Tribunal de Grado para la evalissaión del tralıajo de im estigación "**Implementación de un sistema para optimizar al posicinnamiento de un vehiculo mediante Machine Learning y filtrado de posiciones geolncalizadas**", presentado por **Francis Winder Remache Yncuilla**, con céitula de idemidail número 0605635444, bajo la tuiorio de Mgs. **José Lnis Jinez Tapia**; certificamos que recomendamos la **APROBACIÓN** de este con fines de titulación. Previamanie se ha evaluado el trabsjo de investigación y escuchnda la susientación por parte de su autor, so teniendo mis mda que observar.

De conformidad a la normativa aplicable firmamos, en Riobamba el 6 de enero de 2026.

Haro Mendoza Daniel Eduarrlo, PhD.
PRESIDENTE DEL TRIBUNAL DE GRADO

Torres Redciguez Klever Herrsin, Dr.
MIEMBRO DEL TRIBUNAL DE GRADO

Peiiafiel Ojeda Cedos Ramiro, PhD.
MIEMBRO DEL TRIBUNAL DE GRADO

CERTIFICACIÓN

Que, **Remache Yucailla Francis Winder** con CC: **0605638444**, estudiante de la Carrera **Telecomunicaciones**, Facultad de **Ingeniería**; ha trabajado bajo mi tutoría el trabajo de investigación titulado **"Implementación de un sistema para optimizar el posicionamiento de un vehículo mediante Machine Learning y filtrado de posiciones geolocalizadas"**, cumple con el 7**%** de similitud y 3% de Inteligencia Artificial, de acuerdo al reporte del sistema Anti plagio **COMPILATIO**, porcentaje aceptado de acuerdo a la reglamentación institucional, por consiguiente autorizo continuar con el proceso.

Riobamba, 11 de diciembre de 2025.

Mgs. José Luis Jinez Tapia
TUTOR

DEDICATORIA

A mi familia, por su apoyo constante, por su amor incondicional y por enseñarme el valor del esfuerzo y perseverancia, gracias por la fe y la esperanza que me transmitieron. A mis hermanos, gracias por creer en mí, gracias por sus consejos, gracias por haber sido el pilar fundamental durante el proceso y a mis amigos gracias por su paciencia, por brindar su apoyo y ser parte de este logro.

AGRADECIMIENTO

Agradezco primeramente a Dios, por esta oportunidad que me a dado en la vida, por haberme dado la fuerza, salud y sabiduría para cumplir una meta más. A mi familia por haberme guiado por el camino correcto, por su paciencia y apoyo constante. Agradezco a todos mis docentes de carrera por compartir sus conocimientos y guiarme durante mi formación académica, a mi tutor de tesis por su orientación y compromiso, el cual a sido clave durante el desarrollo de este trabajo de investigación.

ÍNDICE GENERAL

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El siguiente trabajo de investigación se enfoca en implementar un sistema de posicionamiento que integre IMU/GPS (Unidad de medida Inercial/Sistema de Posicionamiento Global) con ayuda de fusión de sensores y ML (Maching Learning). Este sistema recopila datos del GPS e IMU, los cuales son tecnologías de bajo costo.

La fusión de sensores es implementada usando el algoritmo de EKF (Filtro extendido de Kalman), este algoritmo es evaluado usando datos reales recopilados por la unidad de medida Inercial (MPU6050) y el receptor GPS NEO-6M, estos datos previamente estimados por el Filtro pasan a ser procesados por los modelos Random Forest y LSTM para suavizar y optimizar mejor las estimaciones. Los resultados obtenidos de este sistema muestran tener una mejor estimación de posición de un vehículo.

Palabras claves: GPS, ML, EFK, optimizar, posicionamiento.

ABSTRACT

The following research work focuses on implementing a positioning system that integrates IMU/GPS (Inertial Measurement Unit/Global Positioning System) using sensor fusion and ML (Machine Learning). This system collects data from GPS and IMU, which are low-cost technologies.

Sensor fusion is implemented using the EKF (Extended Kalman Filter). This algorithm is evaluated using real data collected by the Inertial Measurement Unit (MPU6050) and the NEO-6M GPS receiver. These data, previously estimated by the Filter, are processed by the Random Forest and LSTM models to smooth and better optimize the estimates. The results from this system show better vehicle position estimation.

Keywords: GPS, ML, EFK, optimize, positioning.

Reviewed by:

Mgs. Sofía Freire Carrillo

ENGLISH PROFESSOR

C.C. 0604257881

CAPÍTULO I

1.1 Introducción

En los últimos años, el Sistema de posicionamiento global (GPS) se ha considera como uno de los más importantes avances tecnológicos empleados en la navegación y localización. Durante varios años, el territorio ecuatoriano ha empleado tecnología GPS en diversos sectores como: vehicular, industrial, policial y militar lo cual ha contribuido a optimizar operativos de patrullaje, logística, entre otras aplicaciones [1].

Un receptor GPS necesita un mínimo de 4 satélites para determinar su posición actual con un error inferior a 20 m. En términos generales el receptor GPS en condiciones ideales proporciona una precisión aproximada de 2 a 5 m [2], pero está precisión disminuye por la influencia de varios factores como: malas condiciones climáticas, áreas urbanas o donde haya mala cobertura satelital, esto debido a que las señales GPS pueden llegar a rebotar en árboles, montañas, grandes edificios, etc. [3], esto provoca que estas señales lleguen distorsionadas haciendo que el receptor no pueda determinar correctamente la posición.

En la actualidad la precisión en la geolocalización se ha convertido en un componente esencial para una variedad de aplicaciones críticas que van desde la navegación autónoma, donde la exactitud milimétrica juega un papel fundamental. Asimismo, en la seguridad vial, agricultura de precisión, aplicaciones comerciales, robótica, seguimiento y monitoreo de vehículos, activos, animales, entre otros.

La navegación integrada como tecnología, permite combinar datos de GPS con otros sistemas como son los inerciales para corregir errores y mejorar la exactitud de la posición, incorporando técnicas de procesamiento de señales avanzadas y algoritmos. Varios autores han contribuido al desarrollo de algoritmos de navegación integrada, con un conjunto mínimo de sensores para mantener las condiciones de bajo costo [4].

Con el presente proyecto de investigación se pretende dar una opción económica y tecnológica en cuanto a los sistemas de posicionamiento, lo cual se consigue mediante la integración de técnicas avanzadas de Machine Learning y filtrado de posiciones geolocalizadas. La combinación de estas metodologías no solo promete superar las limitaciones tradicionales de los sistemas GPS como: la precisión limitada bajo malas condiciones climáticas, en entornos urbanos densos o áreas con obstrucciones, sino que también busca establecer fiabilidad en la determinación de la ubicación.

1.2 Planteamiento del Problema

El posicionamiento de vehículos es un elemento clave para numerosos sistemas de asistencia al conductor, sistemas de transporte inteligente, sistemas de seguridad, entre otros. En función de estas diversas aplicaciones, se requieren mayores o menores niveles de fiabilidad y precisión en la estimación del posicionamiento [5].

El GPS en condiciones ideales no es capaz de proporcionar información completamente aceptable y continua sobre la posición de un cuerpo en movimiento y presentan varios desafíos con respecto a la localización que dependen del entorno local. En presencia de interferencia electromagnética, ruido, problemas de sincronización, malas condiciones climáticas, zonas urbanas densas, entornos indoor, las señales GPS pueden llegar a distorsionarse agresivamente generando errores en la estimación de la posición.

A pesar de que los sistemas GPS convencionales proporcionan ubicaciones con un margen de error aceptable, existen limitaciones en términos de precisión y exactitud, ya que una ubicación imprecisa o una estimación errónea puede llegar a afectar negativamente la eficiencia operativa y la seguridad de un vehículo. Para abordar esta problemática, se propone implementar un sistema que combine técnicas de Machine Learning con métodos de filtrado de posiciones para optimizar la estimación de posición y fiabilidad de la localización de un vehículo.

1.3 Justificación

Las señales GPS aún bajo condiciones ideales son vulnerables a diversos factores físicos y ambientales que afectan su fiabilidad, en consecuencia, el receptor entrega coordenadas con errores significativos, por lo cual la implementación de un sistema que optimice la estimación de posición de un vehículo se vuelve fundamental para los sistemas de navegación.

La aplicación de algoritmos de Machine Learning pueden corregir errores sistemáticos y optimizar las estimaciones de posición, aprovechando patrones y correlaciones en los datos históricos de posicionamiento. Además, el filtro de Kalman o el uso de técnicas más avanzadas de fusión de sensores permiten integrar datos de múltiples fuentes, logrando filtrar el ruido que provocan estos sensores como: GPS e IMU. El trabajo conjunto de estos sistemas nos puede ayudar a obtener estimaciones más fiables y robustas.

Mediante el presente proyecto de investigación se busca desarrollar una alternativa económica en cuanto a sistemas de navegación se refiere, esto abre nuevas oportunidades para aplicaciones vehiculares comerciales rentables, por tal motivo la implementación de este sistema proporcionará un dispositivo que sea fiable y que integre tecnología de bajo costo.

1.4 Objetivos

1.4.1 Objetivo General

Implementar un sistema para optimizar el posicionamiento de un vehículo mediante Machine Learning y filtrado de posiciones geolocalizadas.

1.4.2 Objetivos Específicos

- Estudiar los entornos de desarrollo y sistemas implicados en el vehículo para la estimación del posicionamiento.
- Implementar técnicas de fusión de algoritmos de estimación y el sistema de localización sobre un medio de transporte utilizando algoritmos de inteligencia artificial.
- Diseñar una interfaz gráfica para monitorear la ubicación del vehículo en tiempo real.
- Evaluar el funcionamiento del sistema mediante pruebas de campo y análisis estadístico.

CAPÍTULO II.

MARCO TEÓRICO

2.1 Estado del Arte.

El posicionamiento por GPS en zonas urbanas densamente pobladas puede ser un reto, principalmente debido al bloqueo de señales por edificios o túneles. En el caso de los vehículos autónomos se necesitan de contar con un posicionamiento extremadamente preciso para trazar rutas y moverse por carreteras, especialmente en vías complejas como intersecciones y autopistas, por lo que se propone una estructura de localización para zonas urbanas densamente pobladas que incluye tanto un algoritmo robusto de detección de errores, capaz de evaluar el rango de confianza de cada estimación, como una precisa técnica de localización alternativa basada en un algoritmo de map matching de bajo coste computacional. Finalmente se demuestra que esta propuesta no solo se centra en detectar correctamente los errores a lo largo de la trayectoria, sino que además acota el efecto de dichos errores mejorando la precisión del sistema [6].

Por otro lado, los INS son diseñados para ayudar a la navegación, ya que proporciona una posición estimada del vehículo, entonces se implementó un sistema de navegación de bajo costo hecho con filtros de Kalman, esta solución se montó sobre un automóvil de juguete y el navegador se implementó por medio de Arduino, logrando tener una mejora significativa en la precisión de los sensores utilizados para hallar la posición del automóvil, la posición se determinó por medio de las variables de distancia y los ángulos Roll, Pitch, Yaw o RPY. Para validar los resultados obtenidos por el algoritmo de Kalman, se realizaron pruebas específicas en el procesamiento con registros y gráficas [7].

Ante las carencias de cobertura de posicionamiento de GPS y de otros sistemas de radiofrecuencia en entornos hostiles, como túneles, aeropuertos o almacenes, se propone estudiar el rango de frecuencias de la luz visible para el posicionamiento (VLP). Se plantea así emplear VLP en estos entornos hostiles, donde tanto el comportamiento como guía de onda, la presencia de metales o las restricciones del uso de la radiofrecuencia hacen que utilizar VLP sea una buena solución. Finalmente, mediante pruebas de campo se ha demostrado que es posible implementar un sistema de geolocalización o de posicionamiento en lugares hostiles para la radiofrecuencia, pudiendo posicionar un vehículo de manera más confiable en dichos entornos [8].

A nivel internacional en la investigación realizada se evaluó un sistema de navegación inercial GNSS/IMU en automoción para sistemas de automatización a la conducción, en donde para integrar las soluciones de los sistemas satelital de navegación global (GNSS) e INS se combinó en un FK para obtener la solución del sistema integrado, el FK se utilizó para estimar los errores INS. En entornos urbanos y montañosos las pérdidas de precisión fueron notables, alcanzando hasta los 40 metros y 7 metros respectivamente, mientras que en un entorno de cielo abierto se alcanzan los 0.2 metros. Además, se observó las pérdidas de precisión a pesar de contar con un número óptimo de satélites, siendo destacable que las

precisiones no superan los 0.05 m de error. Los resultados obtenidos destacan la influencia crucial del GNSS en la precisión de la solución de posicionamiento [9].

La Inteligencia Artificial y el Machine Learning son fundamentales para el desarrollo de vehículos autónomos, entonces se apuesta por utilizar redes neuronales para procesar datos de múltiples sensores lo que permite a los vehículos autónomos tomar decisiones informadas en tiempo real. Como punto de partida es indispensable la optimización del posicionamiento de un vehículo, lo cual contribuye a la efectividad de los sistemas de conducción autónoma, mejorando la navegación y la respuesta a situaciones del entorno, el resultado de esta implementación se refleja en la reducción de accidentes en pruebas de vehículos autónomos y la mejora en la eficiencia del tráfico en entornos urbanos [10].

Los vehículos autónomos prometen numerosos beneficios para el tráfico vehicular, incluida una mayor capacidad vial y flujo de tráfico, menos accidentes como resultado de los sistemas de prevención de colisiones, en este contexto se propone implementar un sistema de visión artificial basado en redes convolucionales para el correcto posicionamiento de un vehículo en un carril compuesto por dos módulos. La primera, de procesamiento de imágenes que son capturadas por un sensor óptico por medio de inteligencia artificial aplicando OpenCV, Tensorflow y Keras. El segundo modulo se encarga del control de los motores y la interpretación de los datos obtenidos por el primer módulo de procesamiento, los resultados obtenidos apuntan a el sistema de detección de carril está dentro de los rangos precisos de interpretación para evitar salirse de los limites trazados en un carril a escala, la precisión de entendimiento de la red neuronal llega a un 98.54%. Se concluye que el prototipo de sistema permite una conducción estable de un vehículo y de interpretación de imágenes en óptimas condiciones de iluminación [11].

En otro ámbito se planteó un modelo de predicción de tiempos de traslado mediante el modelo de machine Learning de Random Forest programado en el lenguaje de programación Python, esto utilizando GPS de vehículos registrados en la plataforma de SimpliRoute y de Transantiago para complementar zonas faltantes en el mapa. En primer lugar, se calculó la velocidad promedio de movimiento de los vehículos, para luego obtener un algoritmo de cálculo de tiempos históricos de traslado. Con los valores históricos obtenidos se realizó un modelo de entrenamiento de Random Forest que realiza una predicción de los tiempos en base a datos históricos. Dicha predicción se realiza con un 96.88 % de precisión calculado, utilizando la medida de error porcentual MAPE. Para asegurar que la predicción obtenida sea certera se realizó una comparación con los valores obtenidos de una llamada a la API de Google Maps, obteniendo como resultado que la predicción calculada por el modelo de Random Forest tiene una diferencia de ±5 minutos con los obtenidos por Google Maps [12].

Uno de los principales puntos que se debe resolver es el mejorar la propia localización del vehículo dentro de su entorno, para ello se propone una técnica basada en la fusión del posicionamiento absoluto en un mapa del vehículo a partir de sensores de visión (Visual-SLAM) o LiDAR (LiDAR-SLAM) en conjunto con el Aprendizaje profundo o Deep Learning para mejorar el posicionamiento de vehículos autónomos, el resultado obtenido

consigue una consistencia mayor en la estimación de la posición y una mejora notable en la calidad de la misma [13].

Los sistemas de posicionamiento son utilizados en diferentes aplicaciones, uno de ellos es en la robótica, en donde se propone utilizar dicho sistema para la localización de un robot autónomo en superficies exteriores, como resultado el autor tuvo diferentes problemas y desafíos con respecto a la capacidad de procesamiento que se requiere para trabajar con inteligencia artificial y con el entorno exterior que alberga multitud de interferencias y malas condiciones ambientales [14].

A nivel internacional también se propone mejorar la precisión del posicionamiento en entornos afectados por multipath, mediante la aplicación de técnicas avanzadas de inteligencia artificial y machine Learning, en donde se realizó un estudio exhaustivo sobre las medidas GNSS, explorando diversas características como la elevación, los residuos, el SNR (Signal-to-Noise Ratio), el CRC (Cyclic Redundancy Check) y el CMC (Code Multipath Correction). Estas características se agrupan y se comparan utilizando dos enfoques: el modelo K-Means y el uso de mapas autoorganizados, como resultado se ha observado una reducción en los picos de error durante el cálculo de posicionamiento [15].

La aplicación de filtros de Kalman para mejorar el posicionamiento de un objeto se ha vuelto una de las alternativas utilizadas en la conducción autónoma, como se menciona en el estudio se determinó la orientación de un vehículo no tripulado mediante el sistema GNSS, con lo cual se concluyó que se necesita aplicar rutinas de filtrado de Kalman para mejorar el error estático y así conseguir mayor precisión en la orientación y trayectoria [16]. Otro trabajo se basó en este mismo principio anterior, pero aplicando el filtrado de Kalman a un robot autónomo, con lo cual se consiguió describir su movimiento y planificar su trayectoria con mejores resultados [17].

Entre las más recientes investigaciones hechas para mejorar la ubicación precisa y confiable en interiores, se propone analizar y mejorar la tecnología de posicionamiento por Ultra Wideband (UWB), la cual se destaca por su alta precisión, bajo consumo de energía y resistencia a interferencias. En este trabajo, la tecnología UWB emplea el algoritmo TDoA (diferencia de tiempo de llegada) y ToA (Tiempo de llegada) para medir el tiempo que tardan las ondas electromagnéticas en viajar desde el transmisor hasta el receptor, gracias a esta fusión de algoritmos se consiguió una precisión de posicionamiento a nivel de centímetros, satisfaciendo ampliamente los requisitos de posicionamiento de alta precisión en diversos escenarios [18].

2.2 Fundamento Teórico.

2.2.1 Sistema de Navegación Inercial (INS).

La navegación inercial se basa en los principios de la cinemática, que partiendo de un punto de partida puede calcular posiciones futuras en cualquier momento cuando se conocen la velocidad relativa, la dirección y la aceleración, sin embargo, la precisión de este tipo de navegación se degrada con el tiempo, debido a los errores acumulativos causados por offsets, bias, factores de escala y no linealidades presentes en los sensores inerciales [19].

Unidad de Medida Inercial (IMU).

La IMU es un dispositivo electrónico, está conformado por un conjunto de sensores como: acelerómetro, giroscopio y opcionalmente un magnetómetro, se utilizan para rastrear la posición y orientación de un objeto en relación con un punto de partida [9].

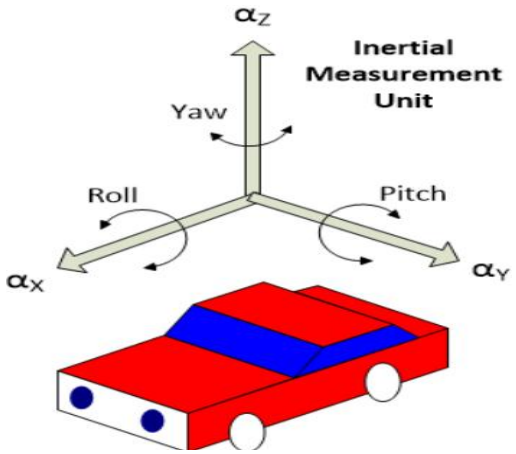


Figura. 2.1. Unidad de medida inercial [20].

Componentes principales.

- **Acelerómetro:** Mide la aceleración lineal en los ejes x, y, z en m/s^2. Los acelerómetros son sensibles a entornos vibrantes y a la diferencia entre la aceleración lineal del sensor y el campo gravitacional local.
- **Giroscopio:** Mide la velocidad angular o cambios de orientación en los ejes x, y, z en rad/s o $°/s$. Son sensibles a errores que pueden acumularse a lo largo de tiempo.
- **Magnetómetro:** Mide la dirección del campo magnético, usado principalmente para determinar la orientación.

Características de Ruido en Giroscopio y Acelerómetro.

Entre las principales características de ruido encontramos las siguientes:

Offset o desviación de cero. También conocido como bias es un valor de desviación del valor real medido dado en unidades rad/s o m/s^2 respectivamente. Para el giroscopio cuando el error offset se integra, causa que el error de posición angular crezca con el tiempo linealmente [19]. La aceleración se integra dos veces para obtener la posición, por tanto, el error de posición crece con el cuadrado del tiempo [19].

Ruido Blanco Termo-Mecánico. La salida de los giroscopios y acelerómetros pueden ser perturbados por la presencia de un ruido blanco termo mecánico el cual oscila a una velocidad mucho mayor que el tiempo de muestreos del sensor [19].

Filtración de ruido.

Los acelerómetros inerciales y giroscopios son propensos a ser afectados por ruido, para filtrar este ruido, se puede utilizar una **media móvil** (ecuación. 1), o un **filtro exponencial (**ecuación. 2), con los cuales se podrá atenuar en gran medida este problema.

$$y(k) = \left(\frac{1}{N}\right) \sum_{i=0}^{N-1} x(k-i) \tag{1}$$

y(k): representa el promedio filtrado en el instante k.

***N*:** es el número de muestras a promediar.

x(k-i): se atribuye al valor tomado en el instante anterior.

$$y(k) = \alpha * x(k) + (1-\alpha) * y(k-1) \tag{2}$$

y(k): representa el valor filtrado en el instante k.

x(k): es el valor de entrada actual.

y(k-1): es el valor en el instante anterior.

∝: es el coeficiente de suavizado.

Estabilidad de bias. La forma de hallar este error es promediando una serie de datos ecuación [3], cuando no está bajo ninguna rotación o movimiento y una vez que se conoce este error simplemente se resta de la salida de la IMU para compensar.

$$y = \left(\frac{1}{N}\right) \sum_{i=0}^{N-1} x(i) \tag{3}$$

y: representa el promedio de las mediciones cuando la IMU está en reposo.

N: es el número de muestras.

x(i): valor de la medición en el instante i.

2.2.2 Sistema de Posicionamiento Global (GPS).

Este sistema proporciona al usuario información específica sobre posicionamiento, navegación y cronometraje en un punto concreto. La ubicación y la velocidad del vehículo están determinadas por GPS. Esta tecnología de navegación permite obtener mediciones aproximadamente cada 1 segundo y el error se limita a 3-5 metros [21].

Las órbitas de los satélites se distribuyen de manera que al menos 4 satélites sean siempre visibles desde cualquier punto de la Tierra en cualquier instante dado. Cada satélite lleva consigo un reloj atómico que funciona con una precisión de 1 nano segundo [22].

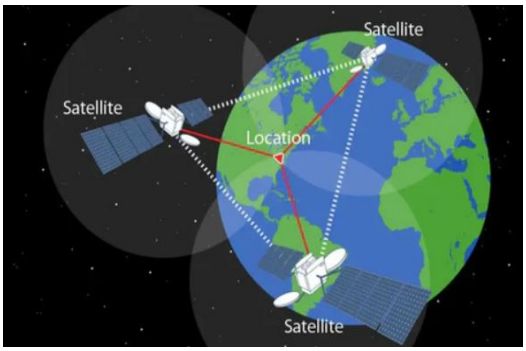


Figura. 2.2. Sistema de posicionamiento Global [23].

Protocolo NMEA. NMEA 0183 es un estándar para la comunicación de datos desarrollada por el U.S. Nacional Marine Electronics Association (NMEA), para evitar incompatibilidades entre datos y formatos de mensajes entre dispositivos electrónicos marinos. NMEA es fundamentalmente usado para la transmisión de datos entre un receptor GPS/GNSS y otros dispositivos. Es un formato ASCII fácilmente legible, pero menos compacto que un formato binario [24].

Trama NMEA. Es una línea de código en formato hexadecimal que contiene toda la información evaluada por el dispositivo GPS. Incluye datos como: tiempo, fecha, coordenadas (latitud, longitud), orientación, numero de satélites visibles, velocidad, orientación, entre otros. Las tramas de GPS por estandarización utilizan comas, letras o símbolos para separar los campos.

Trama	Descripción	Datos Clave
GGA	Información de posición	Latitud, Longitud, Hora, Calidad de la señal
GLL	Posición geográfica	Latitud, Longitud, Hora, Indicador de posición válida
GSA	Satélites activos	Satélites utilizados, DOP
GSV	Satélites en vista	Número total de satélites, ID, Elevación, Azimut
RMC	Información mínima de navegación	Latitud, Longitud, Hora, Velocidad, Rumbo

Figura. 2.3. Tipos de tramas NMEA [25].

2.2.3 Filtro de Kalman (FK).

El filtro de Kalman realiza proceso de estimación de estados aplicado a sistemas dinámicos que involucran cambios aleatorios. El filtro de Kalman ofrece un algoritmo recursivo lineal, sin sesgo y con un mínimo error de varianza, que de manera óptima estima los estados futuros de un sistema dinámico, con ruido discreto en tiempo real [26].

Filtro extendido de Kalman (EKF).

Es una evolución del FK clásico, utilizado para sistemas no lineales, aplicado en campos como navegación, robótica, procesamiento de señales y control de vehículos autónomos, este filtro se compone de dos fases: fase de predicción y fase de corrección, se basa en la linealización de las funciones que describen el sistema dinámico, considerando únicamente los términos de primer orden del desarrollo en serie de Taylor [19] alrededor del punto estimado $\hat{x}_{\bar{k}\,-1}$ y el Jacobiano representa la matriz de derivadas parciales de esas funciones.

- Variables de estado *x*:

$$x \;=\; [x, y, v, \theta]$$

- Variables de control *u*:

$$u \;=\; [a, \omega]$$

- Variables de medición *z*:

$$z \;=\; [x, y, v, \theta]$$

- Modelo de transición del estado, donde *wk* representa el ruido de proceso:

$$\hat{x}_{\bar{k}} = f(x_{k-1}, uk) + wk \tag{4}$$

- Matriz jacobiana de la función de transición *f* respecto al estado:

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{x_{k-1},\, u_k} \tag{5}$$

- Modelo de observación, donde *vk* representa en ruido de medición:

$$zk = h(xk) + vk \tag{6}$$

- Matriz jacobiana de la función de medición *h* antes de aplicar la corrección del estado con la medición:

$$H_k = \left.\frac{\partial f}{\partial x}\right|_{x_{\bar{k}}} \tag{7}$$

Fase de predicción. Se calcula la predicción de los estados $\hat{x}_{\bar{k}}$ en el instante *k* y su matriz de covarianza P_k^-, donde *Q* es la matriz de ruido del proceso.

$$P_k^- = F_k P_{k-1} F_k^\top + Q \tag{8}$$

Para ello se hacen uso de las variables de estado *x*, utilizando como variables de control *u* como entrada para EKF, para luego desarrollar las funciones que describen el movimiento simple de un vehículo.

$$x_k = x_{k-1} + v_{k-1} \cdot \cos(\theta_{k-1}) \cdot \Delta t \quad (9)$$

$$y_k = y_{k-1} + v_{k-1} \cdot \sin(\theta_{k-1}) \cdot \Delta t \quad (10)$$

$$v_k = v_{k-1} + a \cdot \Delta t \quad (11)$$

$$\theta_k = \theta_{k-1} + \omega \cdot \Delta t \quad (12)$$

Estas ecuaciones realizan la predicción de los estados: posición (*x, y*), velocidad (*v*) y orientación (θ).

Fase de corrección. Se evalúa las variables de medición *z* en el instante *k*, junto a la matriz de medición H_k y se calcula la diferencia entre la medida real y la estimada respectivamente. Se calcula la ganancia de Kalman K_k donde *R* es la matriz de ruido de medición, además se corrige las estimaciones de los estados x_k y la covarianza de estado P_k^- .

$$K_k = P_k^- H_k^\top (H_k P_k^- H_k^\top + R)^{-1} \quad (12)$$

$$x_k = x_k^- + K_k\left(z_k - h(x_k^-)\right) \quad (13)$$

$$P_k = (I - K_k H_k) P_k^- \quad (14)$$

2.2.4 Aprendizaje autónomo (ML).

Los algoritmos de machine Learning, se basan en el aprendizaje a partir de una muestra de datos de patrones y relaciones funcionales entre distintas variables. Las redes neuronales recurrentes (RNN) son una clase de aprendizaje profundo, son conocidas por su capacidad para procesar y obtener información de datos secuenciales [27].

Random Forest (RF).

Es una técnica de aprendizaje supervisado que genera múltiples árboles de decisión sobre un conjunto de datos de entrenamiento. Cada árbol contiene un grupo de observaciones aleatorias (elegidas mediante bootstrap, que es una técnica estadística para obtener muestras de una población donde una observación se puede considerar en más de una muestra). Las observaciones no estimadas en los árboles (también conocidas como “out of the bag”) se utilizan para validar el modelo. Las salidas de todos los árboles se combinan en una salida final (conocida como ensamblado) que se obtiene mediante alguna regla (generalmente el promedio, cuando las salidas de los árboles del ensamblado son numéricas [28].

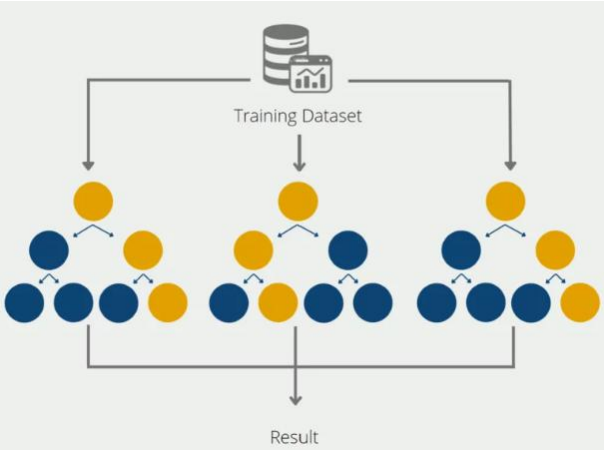


Figura. 2.4. Estructura Random Forest [29].

Ventajas:

- Es un modelo simple de entrenar y muy eficiente con base de datos grandes.
- Mantiene un grado de precisión aceptable cuando tenemos porciones de datos perdidos.

Desventajas:

- Puede sobre ajustar los datos cuando hay presencia de ruido.
- Las predicciones no son de naturaleza continua y no puede predecir más allá del rango de valores del conjunto de datos usado para entrenar el modelo [28].

Long Short-Term Memory (LSTM).

Las LSTM poseen la capacidad de procesar datos secuenciales y retener información de pasos anteriores en la secuencia, lo que les permite predecir los pasos futuros de manera efectiva. Esta característica las hace altamente adecuadas para tareas que involucran dependencias a largo plazo [30].

El flujo de información dentro de las redes LSTM está gobernado por tres puertas internas durante el proceso de aprendizaje: la puerta de olvido, la puerta de entrada y la puerta de salida. La puerta de olvido (ft) determina qué información del estado de la celda debe ser descartada. La puerta de entrada (it) decide qué nueva información se añadirá al estado de la celda. Finalmente, la puerta de salida (ot) controla la salida del estado oculto. Este mecanismo de puertas permite a las unidades LSTM gestionar eficazmente las dependencias a largo plazo al actualizar y retener información selectivamente a lo largo del tiempo [30].

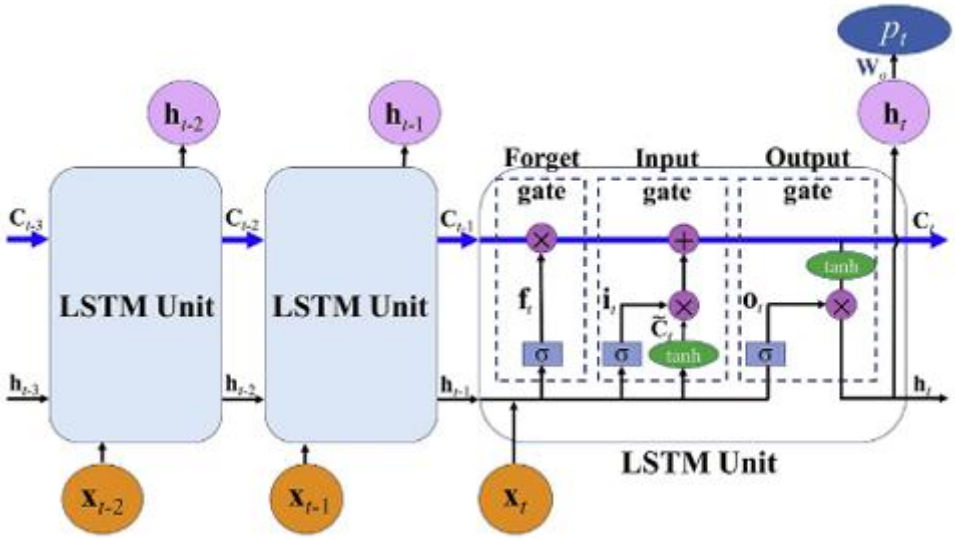


Figura. 2.5. Arquitectura LSTM [30].

La complejidad temporal para un paso de entrenamiento implica $mn(n+(m-1)) = n^2m+nm^2-nm$ operaciones. Esto se realiza para cada una de las tres puertas, el estado de la celda, y durante k pasos de tiempo, resultando en $4k(n^2m+nm^2-nm)$. En consecuencia, esto puede resultar en tiempos de entrenamiento más largos, especialmente cuando se trabaja con grandes conjuntos de datos o cuando se utiliza un alto número de unidades LSTM, esto supone recursos computacionales significativos [30].

CAPÍTULO III

METODOLOGÍA.

3.1 Tipo de investigación.

3.1.1 Investigación cuantitativa.

Se basó en la recolección y análisis de datos numéricos para evaluar la efectividad del sistema propuesto en la optimización del posicionamiento de un vehículo. Esto permitió obtener resultados medibles con el fin de comparar el desempeño del sistema implementado frente al sistema GPS tradicional.

3.2 Métodos de Investigación.

3.2.1 Método Experimental.

Se utilizó la recolección de datos a través de pruebas de campo con el fin de analizar cuál de los sistemas (sistema propuesto vs tradicional) proporciona el menor margen de error. Finalmente se interpretó los resultados de forma objetiva.

3.3 Diseño de Investigación.

Este trabajo de investigación se desarrolló en tres fases que se describen a continuación:

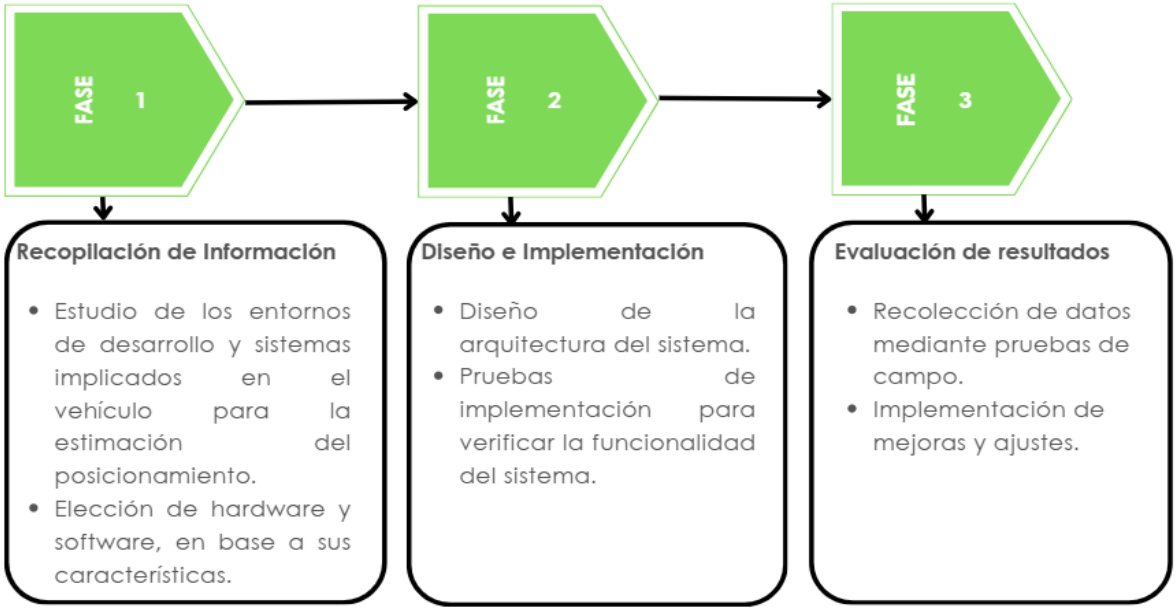


Figura. 3.1. Fases de Investigación

3.3.1 Fase 1

1. **Recopilación de Información.**

En esta primera fase se realizó el estudio de los entornos de desarrollo y sistemas implicados en el vehículo para la estimación del posicionamiento, para lo cual se investigó en artículos

y publicaciones similares al tema propuesto para estudiar las diversas tecnologías, algoritmos y sistemas disponibles para la geolocalización. Además, se optó por software y hardware específico, tomando en cuenta características, ventajas, requerimientos y objetivos, con el fin de desarrollar e implementar un sistema económico.

Módulo MPU-6050.

Es un sensor de medición inercial, que combina un giroscopio de 3 ejes y un acelerómetro de 3 ejes en el mismo chip de silicio junto con un Procesador de Movimiento Digital a bordo capaz de procesar complejos algoritmos de fusión de sensores de 9 ejes. Los algoritmos de fusión de movimiento de 9 ejes integrados en el MPU-6000 y el MPU-6050 acceden a magnetómetros externos u otros sensores a través de un bus I2C [31].

Entre sus principales ventajas encontramos que: Es un sensor de muy bajo costo y fácil de adquirir, además tiene un gran soporte comunitario con codificación, módulos y librerías gratis, soporta el protocolo I2C, lo cual es compatible con diferentes microcontroladores, para el caso una ESP32.

- El protocolo I2C, se utiliza para la comunicación síncrona entre circuitos integrados a corta distancia, utilizando el sistema maestro-esclavo.

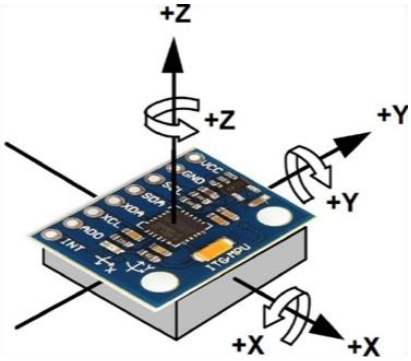


Figura. 3.2. Sensor MPU-6050 [32].

Los pines utilizados son:

- VCC, para la alimentación 3.3V y GND, pin a tierra.
- SCL, línea de reloj conectado al pin GPIO22 de la ESP32.
- SDA, línea de datos conectado al pin GPIO21 de la ESP32.

Tabla 3.1 Características técnicas de la MPU-6050.

Voltaje de operación	2.375V-3.46V
Interfaz serial	I2C
Rango del giroscopio	±250, ±500, ±1000, ±2000 °/seg
Rango del acelerómetro	±2g, ±4g, ±8g, ±16g

Corriente de operación Giroscopio	3.6 mA
Corriente de operación acelerómetro	500 uA
Frecuencia de muestreo Giroscopio	8 KHz programable
Frecuencia de muestreo Acelerómetro	1 KHz programable

Módulo GPS NEO-6M.

Es un receptor GPS ampliamente utilizado, diseñado para un consumo de energía bajo [33], entre sus ventajas tenemos: es económico y compacto, utiliza interfaz serial UART, compatible con diferentes microcontroladores pal caso ESP32, además cuenta con librerías gratis que facilitan su implementación.

- El protocolo UART, se utiliza para la comunicación asíncrona entre dispositivos TX y RX. La sincronización se logra mediante la coincidencia de las tasas de baudios y un formato de trama de bits.

Figura. 3.3. Módulo y antena GPS NEO-6M [34].

- Aquí los pines RX - TX, se conectan a los pines TX - RT de la ESP32.
- VCC, conectado a 3.3 V y GND a tierra.

Tabla 3.2 Características técnicas de GPS NEO-6M.

Voltaje de operación	2.7V-3.6V
Interfaz serial	UART
Sensibilidad de rastreo	Hasta -161 dBm
Sensibilidad de adquisición	Hasta -148 dBm
Baud rate por defecto	9600 bps
Protocolo	NMEA
Frecuencia de muestreo	1 a 5 Hz programable
Precisión de posición horizontal	2.5 m CEP en condiciones ideales [35].

Módulo ESP32.

ESP32 es un microcontrolador combinado de Wi-Fi y Bluetooth de 2.4 GHz. Está diseñado para lograr el mejor rendimiento de potencia y RF, mostrando robustez, versatilidad y fiabilidad en una amplia variedad de aplicaciones como IOT y sistemas embebidos [36].

Figura. 3.4. Microcontrolador ESP32 [37].

Entre sus principales características podemos encontrar que: posee un procesador de doble núcleo de 32 bits operando a 240 MHz como máximo, además posee un coprocesador de ultra bajo consumo para tareas simples, una de sus ventajas es que la corriente de reposo es inferior a 5 µA, por lo que es adecuado para aplicaciones de electrónica portátiles con batería, además es compatible con ARDUINO IDE, Lua y Micro Python, lo cual hace que tenga una amplia comunidad, por tanto tienes acceso a librerías codificación gratis.

Tabla 3.3 Características Técnicas ESP32.

Voltaje de operación	3.3 V
Interfaz serial	UART, I2C, SPI, Ethernet, I2S
Numero de pines	30
SRAM	520 KB
Memoria Flash SPI	4 MB
Interfaces UART	3
Interfaces SPI	4
Interfaces I2C	2

- La mayoría de los pines GPIOx son utilizados con entrada o salida digital, el voltaje lógico es de 3.3 V, pin Vin para alimentar con 5V, pin 3V3 para salida de voltaje, pines ADC, DAC y para el caso se utilizan los pines TX, RX, SDA, SCL.

Python.

Es un lenguaje de programación versátil, utilizada en una enorme gama de aplicaciones como: desarrollo web, análisis de datos, ML, automatización, IA, entre otros. Entre sus ventajas esta que: tiene una gran colección de librerías, módulos, bibliotecas, útil para desarrollar diversos proyectos, además de que es un lenguaje eficiente y fácil de aprender.

Micropython.

Es un lenguaje de programación, el cual viene a ser una versión optimizada de Python para microcontroladores y sistemas embebidos de bajos recursos, su ventaja es que también posee

una gran cantidad de módulos para hardware, lo cual te permite interactuar directamente con los pines del microcontrolador, además de que es un lenguaje fácil de aprender.

Thonny IDE.

Es un editor de código, que soporta Python y Micropython, posee una interfaz de usuario sencilla y fácil de usar, además de que es una plataforma que no consume muchos recursos.

VS Code IDE.

Es un editor de código multi idioma que soporta diferentes lenguajes de programación entre ellos Python, es versátil y potente, además también es una plataforma ligera.

2. Análisis de la Información.

En base al fundamento teórico, en este apartado se analizó los principios que se tomaron en cuenta para diseñar e implementar el sistema.

- Para obtener los datos crudos de los sensores MPU-6050 y GPS NEO-6M, se utiliza los módulos y librerías disponibles en repositorios y páginas web, los cuales ayudaron a interpretar y transformar estos datos a un formato global como es: acelerómetro m/s^2, Giroscopio *rad/s,* latitud-longitud *DD,* velocidad *m/s* y orientación en *rad.*

- Para filtrar el ruido de los sensores de la MPU-6050 se utiliza un filtro pasa bajos exponencial, el cual me permite reducir el ruido, además este introduce poco retardo en la señal filtrada y supone un coste computacional bajo.

- En el caso de las bias, se implementa una media móvil que va recoger N muestras, cuando el dispositivo este en absoluto reposo, es resultado se resta a las salidas de los datos de los sensores, mientras que cuando haya movimiento las bias se estimaron de manera suave, estos datos son enviados por cable hacia la PC.

- En la Pc para fusionar los sensores, se utiliza el filtro extendido de Kalman (EKF), el cual tendrá con estados a la posición, velocidad y orientación, como entradas de control a la aceleración y orientación, como estados de medición a la posición, velocidad y orientación medidas directamente del GPS, el cual ayuda a corregir las estimaciones que hace el EKF.

- El EKF trabaja conjuntamente con un modelo de ML Random Forest, el cual se utiliza porque no consume muchos recursos y permite encontrar patrones no lineales en los datos, sin necesidad de mucha parametrización, además es útil para corregir errores o sesgos que puede tener las estimaciones del EKF.

- Para la etapa final se implementa una red neuronal simple LSTM, porque es capaz de aprender dependencias a largo plazo y patrones dinámicos en la trayectoria del vehículo, lo cual permite suavizar mejor la trayectoria aprovechando el historial de

los estados estimados, esta trayectoria será finalmente mostrado en una interfaz gráfica.

3.3.2 Fase 2.

1. **Diseño e implementación:**

En esta fase se procedió a diseñar y construir cada etapa del sistema, el cual incluye programación para la IMU y GPS dentro de la ESP32, programación para el EKF, Random Forest y LSTM dentro de la PC y finalmente el diseño de una página web.

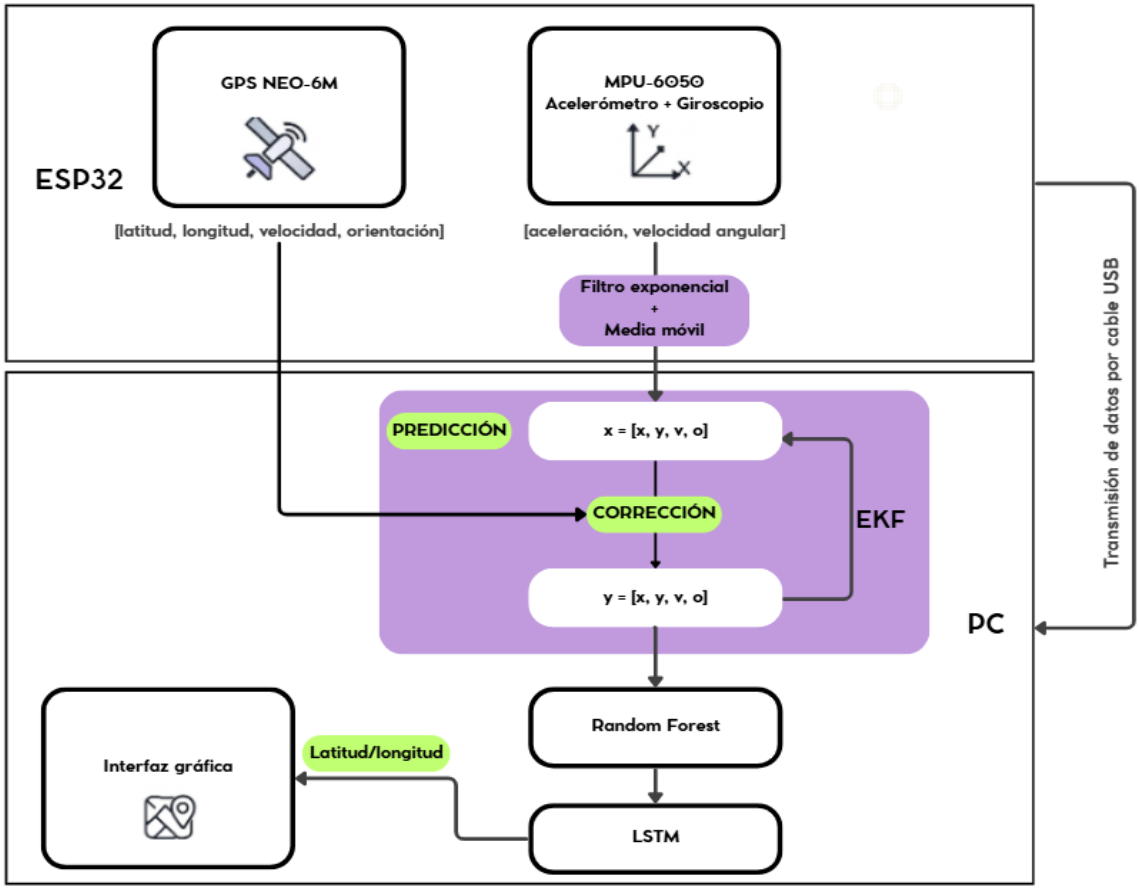


Figura. 3.5. Arquitectura general del sistema.

Diseño del dispositivo

Para diseñar el dispositivo que se encarga de recolectar datos se utilizó Wokwi, un simulador de circuitos online que trabaja con micropython. En la (figura. 3.6), se aprecia la conexión de cada sensor con la ESP32, para el caso del GPS el pin TX al pin RX GPIO16 de la ESP32 el pin RX del GPS al TX GPIO17 de la esp32, para la MPU el pin SCL al pin SCL GPIO22 de la ESP32, el pin SDA al SDA GPIO21 de la ESP32, y finalmente VCC del GPS y MPU a 3.3 V de la ESP32, lo mismo sucede con GND, está configuración es útil a la hora de programar cada sensor, ya que se debe mencionar al puerto o pin que se va ocupar.

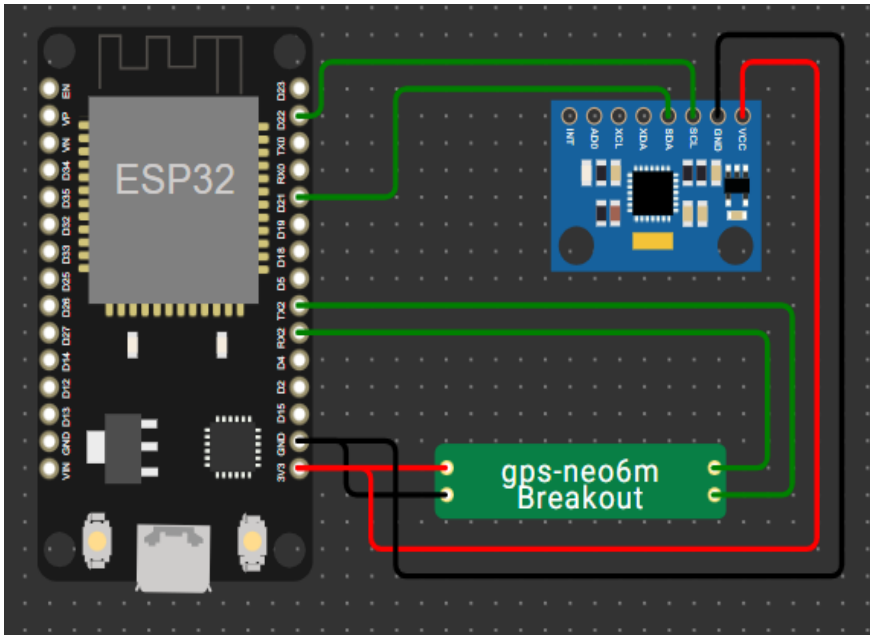


Figura. 3.6. Esquemático del dispositivo.

Programación en la ESP32

Para la parte de programación, la ESP32 es la encargada de controlar cada uno de los sensores, para eso se utilizó Thonny.

GPS NEO-6M

Se procedió a programar este módulo, en donde primeramente se debe saber que los datos que genera el receptor vienen en tramas NMEA, cada una contiene un tipo de información, para lo cual se consideró las tramas de tipo, GPRMC, GPGLL, GPVTG, GPGGA, GPGSA, GPGSV. La biblioteca que me permitió obtener e interpretar la información de cada una de estas tramas se obtuvo desde la página web [38], está biblioteca cargada contiene una clase llamada “MicropyGPS”, en donde se modificó ciertos parámetros en la función “latitude” y “longitude”, para que devuelvan la posición en formato grado-decimal (DD), además se modificó la función “compass_direction” para que devuelta la dirección en radianes, esta información la obtiene la ESP32 por la Interfaz serial UART.

MPU-6050

Para este módulo se carga la biblioteca que se encuentra en el repositorio de GitHub [39], la cual contiene la clase “MPU6050” necesaria para interpretar los datos del acelerómetro y giroscopio, dentro de esta clase también se realizaron ciertos cambios. Se modificó los rangos de medida y se induce por forzar un rango de +- 4G para el acelerómetro y +- 500 grados para el giroscopio, esto con el fin de que la señal proveniente de los sensores no se sature cuando haya, maniobras fuertes o aceleraciones bruscas del vehículo, además se modificó las funciones “read_gyro_data” y “read_accel_data” para que devuelvan datos en rad/s y m/s^2, esta información la obtiene la ESP32 por la interfaz serial I2C.

MAIN

Es el script principal de donde se gestiona cada proceso y a cada sensor, aquí se procedió a instanciar a las dos clases mencionadas anteriormente, además aquí se encuentra alojado la lógica del filtro exponencial y la media móvil para estimar las bias de la IMU, estos se utilizaron para suavizar y mejor los datos. Los datos del receptor GPS, se muestrearon a 1 Hz y se obtuvieron a través de la interfaz serial UART2 a 9600 baudios lo cual viene por defecto. Los datos de la IMU se muestrearon a 50 Hz y se obtuvieron a través de la Interfaz serial I2C a 100Khz por defecto. Finalmente, estos datos filtrados y suavizados se enviaron a través de la Interfaz UART/USB hacia la PC.

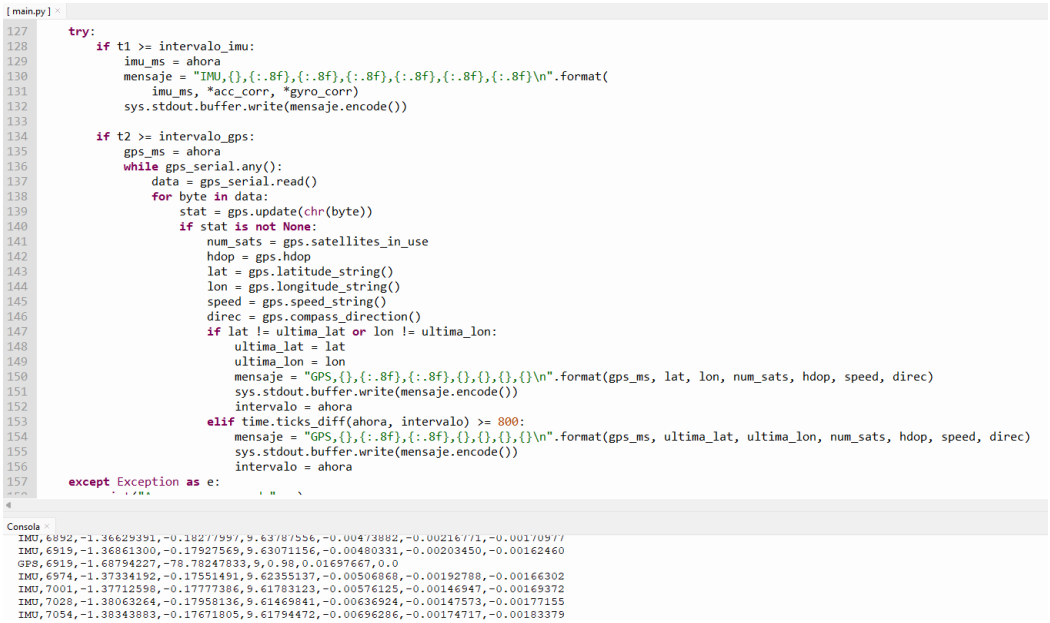


Figura. 3.7. Fragmento de codificación para main.

Programación en la PC.

En la PC se codifican 5 scripts: el primero para el EKF, el segundo para entrenar el modelo Random Forest, el tercero para entrenar LSTM, el cuarto para mostrar la posición del vehículo en una interfaz gráfica y el último el cual es el script principal que se encarga de gestionar todos los scripts anteriores.

EKF

Este script la cual es una clase se programó en base al fundamento teórico, en donde se utilizó las ecuaciones y fases que el filtro ejecuta para estimar la posición. Como variables de estado tenemos $x = [x, y, v, o]$, entonces comenzamos inicializando el tiempo que se ejecutara cada ciclo del filtro que viene dado por la frecuencia de muestreo de la IMU, también se inicializó las bias estimadas por el ESP32, de la misma manera se inicializó los factores de ajuste dinámico para corrección de bias.
Pasamos a la fase de predicción, aquí es donde se predice o estima las variables de estado como: posición, velocidad, orientación, además se calculó la covarianza o incertidumbre del

modelo, para el caso la velocidad y orientación no son tan importantes como la posición del vehículo. Las entradas de control *u = [a, w]* se utilizaron para modelar el movimiento, para ellos se usaron las ecuaciones de movimiento de la cinemática. La predicción se realiza aproximadamente cada 20 ms frecuencia a la que la IMU muestrea, entonces como el filtro no puede trabajar con sistemas no lineales, lo que hiso es aproximar el modelo a una versión lineal a través de las series de Tylor de primer orden, además para el caso de las bias que se generan en el filtro, se realizó un proceso suave de corrección.

Para la fase de corrección, igual que en la predicción se linealiza el modelo no lineal, entonces se tomó los valores de la variable de medición *z = [x, y, v, o]* para corregir los estados estimados en el proceso anterior, además se calculó la ganancia de Kalman, lo cual representa la incertidumbre del modelo y finalmente se actualiza la covarianza, esta fase se realiza aproximadamente cada 1000 ms frecuencia a la que muestrea el GPS.

Estas fases de corrección y predicción se ejecutan continuamente a medida que los datos de la IMU lleguen, entonces la posición corregida vuelve a ser tomada como estado inicial del filtro, además cada fase trabaja con las matrices Q y R que representan la incertidumbre de predicción y de medición respectivamente. Estos datos estimados por Kalman pasan a ser procesados por el modelo Random Forest.

```python
class ExtendedKalmanFilter:

    def predict(self, a_x, g_z, Q):
        x, y, v, theta, bias_ax, bias_gz = self.x.flatten()

        # Compensar bias
        a_x_corr = a_x - bias_ax
        g_z_corr = g_z - bias_gz

        # Modelo de movimiento
        v_new = v + a_x_corr * self.dt
        theta_new = theta + g_z_corr * self.dt
        x_new = x + v_new * np.cos(theta_new) * self.dt
        y_new = y + v_new * np.sin(theta_new) * self.dt

        # Actualizar estado
        self.x[:4, 0] = [x_new, y_new, v_new, theta_new]

        # Ajuste de bias dinámico
        moving = abs(v_new) >= 0.05 or abs(a_x) >= 0.05 or abs(g_z) >= 0.01
        ajuste = self.lento if moving else self.rapido
        self.x[4, 0] *= (1 - ajuste)  # bias_ax
        self.x[5, 0] *= (1 - ajuste)  # bias_gz

        # Jacobiano F
        F = np.eye(6)
        F[0, 2] = np.cos(theta_new) * self.dt
        F[0, 3] = -v_new * np.sin(theta_new) * self.dt
        F[1, 2] = np.sin(theta_new) * self.dt
        F[1, 3] = v_new * np.cos(theta_new) * self.dt
        F[3, 5] = -self.dt

        # Actualizar covarianza
        self.P = F @ self.P @ F.T + Q
```

Figura. 3.8. Fragmento de codificación para EKF.

Random Forest y LSTM.

Para estos modelos de ML se programó en base a [40], [41], [42], [43]. Ramdon Forest y LSTM fueron utilizados como un complemento para mejorar las estimaciones que el EKF predecía, pero cada modelo cumple un papel diferente, en el caso de RF se utilizó con el objetivo de corregir errores o patrones no lineales que el EKF puede generar en las estimaciones hechas, esto se nota más cuando hay demasiado ruido entre las mediciones crudas de los sensores, entonces como RF no necesita de supuestos estadísticos puede aprender a predecir mejor la posición. Para RF se colocó como variables de entrada: *x = [x_est, y_est, v_est, ort_est, accel_imu, gyro_imu, v_gps, ort_gps, num_sat, hdop]*, estas variables el modelo los utilizó para aprender patrones y combinaciones que los relaciona con la posición real *y = [x_r, y_r]* y de esa manera pude ajustar los umbrales en cada árbol de decisión para minimizar el error entre la predicción y la posición real durante el entrenamiento.

El modelo de LSTM se lo utilizó con el objetivo de procesar datos temporales, entonces como entrada se colocó a la posición corregida por RF *x = [x_rf, y_rf, v_est, ort_est, accel_imu, gyro_imu, v_gps, ort_gps, num_sat, hdop]*, con estos datos el modelo internamente ajusta sus pesos minimizando el error para que la secuencia de entrada produzca la posición final más cercana a la real *y = [x_r, y_r]*. En general LSTM usa la historia de posiciones para inferir la tendencia de movimiento suavizando la posición final en base a los patrones temporales aprendidos.

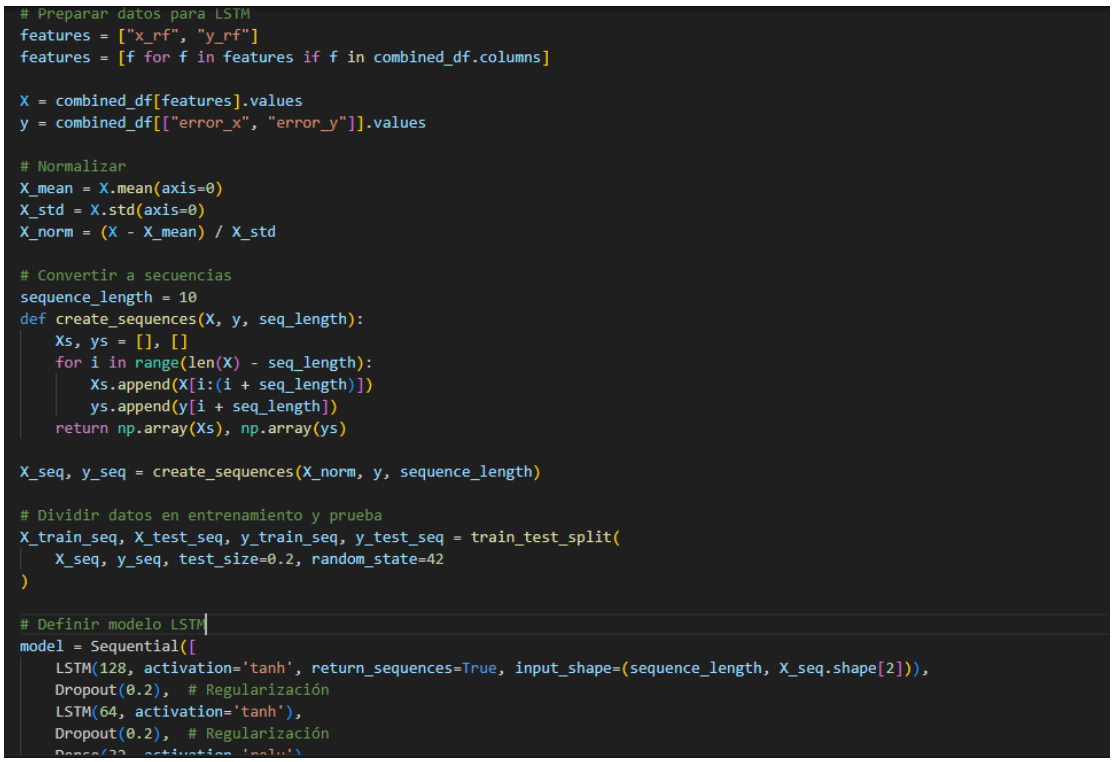
```
csv_files = [
    "R1combinado.csv",
    "TR2combinado.csv",
    "R2combinado.csv",
    "TR3combinado.csv",
    "TR11combinado.csv",
    "R3combinado.csv",
    "TR22combinado.csv"
]
dfs = []
for file in csv_files:
    df = pd.read_csv(file)
    if {"x_r", "y_r", "x_est", "y_est"}.issubset(df.columns):
        df["error_x"] = df["x_r"] - df["x_est"]
        df["error_y"] = df["y_r"] - df["y_est"]
        dfs.append(df)
combined_df = pd.concat(dfs, ignore_index=True)


features = ["x_est", "y_est", "v_est", "ort_est",
            "accel_imu", "gyro_imu", "v_gps", "ort_gps", "num_sat", "hdop"]
X = combined_df[features]
y = combined_df[["error_x", "error_y"]]

# Dividir datos
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Ajustar RandomForest
rf = RandomForestRegressor(
    n_estimators=300,
    max_depth=20,
    min_samples_split=2,
    random_state=42,
)
```

Figura. 3.9. Fragmento de codificación para Random Forest.



```
# Preparar datos para LSTM
features = ["x_rf", "y_rf"]
features = [f for f in features if f in combined_df.columns]

X = combined_df[features].values
y = combined_df[["error_x", "error_y"]].values

# Normalizar
X_mean = X.mean(axis=0)
X_std = X.std(axis=0)
X_norm = (X - X_mean) / X_std

# Convertir a secuencias
sequence_length = 10
def create_sequences(X, y, seq_length):
    Xs, ys = [], []
    for i in range(len(X) - seq_length):
        Xs.append(X[i:(i + seq_length)])
        ys.append(y[i + seq_length])
    return np.array(Xs), np.array(ys)

X_seq, y_seq = create_sequences(X_norm, y, sequence_length)

# Dividir datos en entrenamiento y prueba
X_train_seq, X_test_seq, y_train_seq, y_test_seq = train_test_split(
    X_seq, y_seq, test_size=0.2, random_state=42
)

# Definir modelo LSTM
model = Sequential([
    LSTM(128, activation='tanh', return_sequences=True, input_shape=(sequence_length, X_seq.shape[2])),
    Dropout(0.2),  # Regularización
    LSTM(64, activation='tanh'),
    Dropout(0.2),  # Regularización
```

Figura. 3.10. Fragmento de codificación para LSTM

Interfaz gráfica.

La interfaz gráfica se diseñó utilizando HTML y se optó por utilizar Openstrepmap, el cual me permite utilizar sus mapas digitales de forma gratuita, en general la interfaz es una página web que se aloja en un servidor local (PC), al cual se puede acceder a través de una dirección IP o localhost, esta interfaz está diseñada para observar la trayectoria del vehículo en tiempo real a través de un mapa geográfico y también en un plano x-y, para ello se hace uso de la librería leaflet que me ayuda a mostrar el mapa que me proporciona Openstrepmap, luego se ajusta diferentes parámetros como estilo, color, tipo de texto, tamaño entre otros de acuerdo al resultado que se espera obtener, posteriormente se llama a la función *actualizarDatos()* que me sirve para capturar la posición del vehículo para su posterior visualización, finalmente se integró un botón que sirve para ocultar o visualizar la trayectoria en un plano x-y según se requiera.

```
index.html > html > body > script
<html lang="es">
<body>
<script>
var map = L.map('map').setView([0.0, 0.0], 15);
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        maxZoom: 20,
        attribution: '© OpenStreetMap contributors'
    }).addTo(map);

    // Polylines para cada sistema
    var polylines = {
        gps: L.polyline([], {color: 'black'}).addTo(map),
        ekf: L.polyline([], {color: 'red'}).addTo(map),
        rf: L.polyline([], {color: 'blue'}).addTo(map),
        lstm: L.polyline([], {color: 'green'}).addTo(map)
    };

    // Plotly
    var traceData = [
        {x: [], y: [], mode: 'lines', line: {color: 'black'}, name: 'GPS'},
        {x: [], y: [], mode: 'lines', line: {color: 'red'}, name: 'EKF'},
        {x: [], y: [], mode: 'lines', line: {color: 'blue'}, name: 'EKF+RF'},
        {x: [], y: [], mode: 'lines', line: {color: 'green'}, name: 'EKF+RF+LSTM'}
    ];
    var layout = {margin: {l: 50, r: 50, t: 50, b: 50}, title:"Trayectoria", xaxis:{title:'X (m)'}, yaxis:{title:'Y (m)'}};
    Plotly.newPlot('plot', traceData, layout);
    var centrado = true;
    // Actualizar datos cada 500ms
    async function actualizarDatos() {
        try {
            const res = await fetch('/data');
            const datos = await res.json();

            // Leaflet
            polylines.gps.setLatLngs(datos.gps);
            polylines.ekf.setLatLngs(datos.ekf);
            polylines.rf.setLatLngs(datos.rf);
            polylines.lstm.setLatLngs(datos.lstm);
```

Figura. 3.11. Fragmento de codificación para la interfaz gráfica.

Main.

Este script se encarga de controlar todos los procesos como son: filtrado, predicción, corrección, envió de datos hacia la web y visualización. Primero instanciamos la clase EKF, importamos las bibliotecas necesarias para RF y LSTM y también cargamos los modelos entrenados anteriormente, luego recibimos los datos por serial/USB desde la ESP32 a 115200 baudios por defecto, estos datos son procesados con ayuda de funciones y sentencias, para luego ser redirigidos hacia cada clase según corresponda, como se mencionó los datos de entrada para el EKF son provistas por la ESP32 esos datos son procesados por este filtro los cuales se convierten en las entradas para el modelo de RF y luego esos datos corregidos por dicho modelo se convierten en las entradas para el modelo LSTM. Finalmente se creó un servidor local utilizando Flask el cual me ayuda en el proceso de enviar las posiciones finales corregidas a una página web para su posterior visualización de posición y trayectoria del vehículo en tiempo real.

```python
while True:

    trama = ser.readline().decode('utf-8', errors='ignore').strip()
    #print(line)
    if not trama.strip():
        continue

    archivo_tramas.write(trama + "\n")
    archivo_tramas.flush()

    if not trama.startswith("GPS") and not trama.startswith("IMU"):
        continue

    posicion = trama.split(',')

    try:
        tipo = posicion[0]
    except (IndexError, ValueError):
        continue

    if tipo == 'IMU':
        if len(posicion) < 8:
            continue
        try:
            timestamp = float(posicion[1]) / 1000.0
            accel_imu = float(posicion[2])
            ay = float(posicion[3])
            az = float(posicion[4])
            gx = float(posicion[5])
            gy = float(posicion[6])
            gyro_imu = float(posicion[7])
        except ValueError:
            continue

        if lt_imu is None:
            lt_imu = timestamp
            continue
```

Figura. 3.12. Fragmento de codificación para main.

2. Pruebas de implementación:

Se realizó el ensamblado del dispositivo que se encarga de recoger mediciones en un Protoboard para posteriormente ensamblarlo en un PCB, entonces se comenzó a verificar el funcionamiento correcto de cada etapa, primero se verificó el funcionamiento de la ESP32 y sus sensores, los cuales respondían acorde a lo programado, recolectando posiciones, filtrando y suavizando esos datos para posteriormente ser enviados a la PC a través de cable USB, obteniendo el resultado esperado.

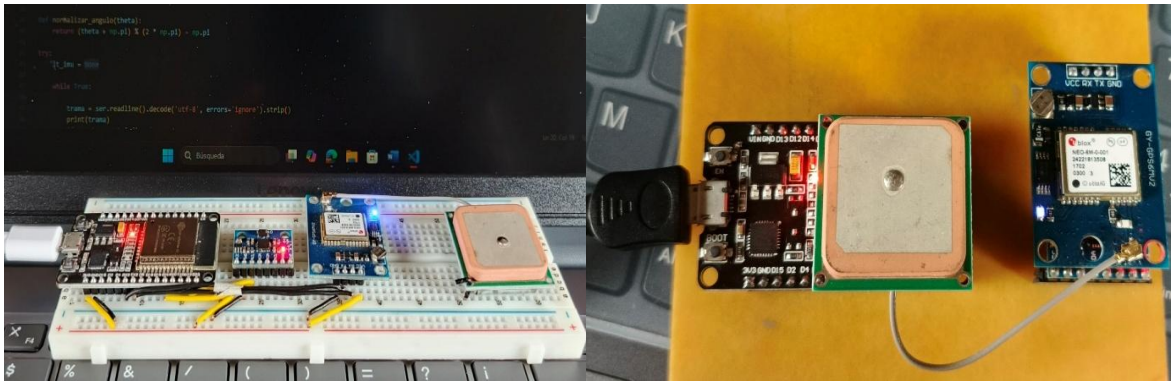


Figura. 3.13. Ensamblado del dispositivo en el Protoboard y PCB.

En la PC se realizaron las pruebas preliminares de cada proceso por separado, entonces se verificó que el EKF estime las posiciones del vehículo las cuales tenían un cierto grado de error especialmente en curvas, además se verifico que la página web y el servidor local funcionen correctamente. Luego se procedió a recopilar bases de datos de diferentes trayectorias reales recorridas para entrenar cada modelo de ML implementado, cada modelo predecía con un cierto grado de error que se medió a través de la variable RMSE (raíz cuadrática media), mientras más pequeño sea este valor el modelo predecía de mejor manera, una vez entrenado los modelos de ML se procedió a verificar el funcionamiento en conjunto de toda la arquitectura del sistema obteniendo los resultados que se esperaba.

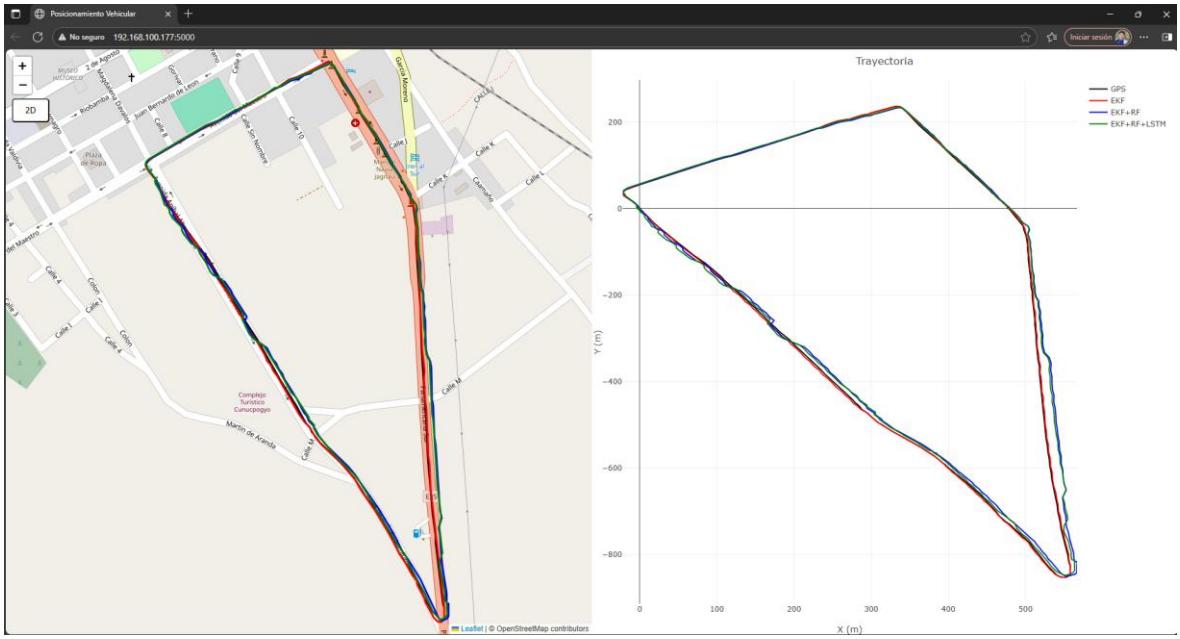


Figura. 3.14. Trayectoria recorrida por el vehículo en tiempo real al realizar las pruebas de funcionamiento.

3.3.3 Fase 3.

1. **Evaluación de resultados:**

Se llevó a cabo la recolección y análisis de datos obtenidos durante las pruebas de campo en entornos controlados, en donde se consideró implementar algunas mejoras a la etapa de estimación por EKF y los modelos de ML. Para entrenar los modelos de ML finalmente se escogió 11 trayectorias distintas conformando una base de 3205 datos totales.

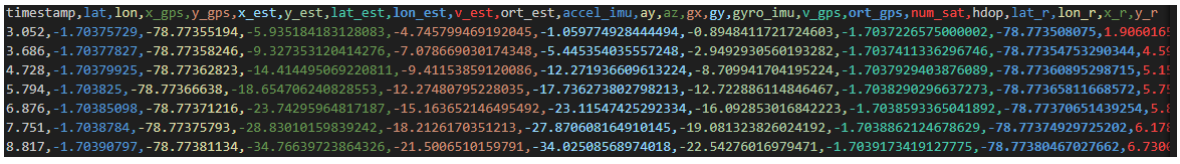


Figura. 3.15. Estructura general de la base de datos para entrenar los modelos de ML.

2. **Implementación de mejoras y ajustes.**

Con base en los resultados se implementó algunas mejoras y ajustes para que el sistema funcione de mejor manera. Uno de los ajustes hechos fue para la etapa del EKF, al cual se le agrego una fase de ajuste de la matriz R de manera dinámica que me ayudó a detectar deriva de las mediciones de GPS y según esos datos la matriz R sube o baja su confianza en la medición, también se implementó un umbral que me sirve para detectar saltos grandes en las mediciones GPS y descartar automáticamente esa medición. De la misma manera se optó por optimar los modelos de ML, para el caso de RF se aumentó el número de estimaciones lo cual hace al modelo más estable, pero a la misma ves le cuesta más procesar datos, lo mismo sucede con LSTM al cual se le incremento más la dependencia temporal y se optó por usar una pausa de entrenamiento para que el modelo no sobreajuste los parámetros.

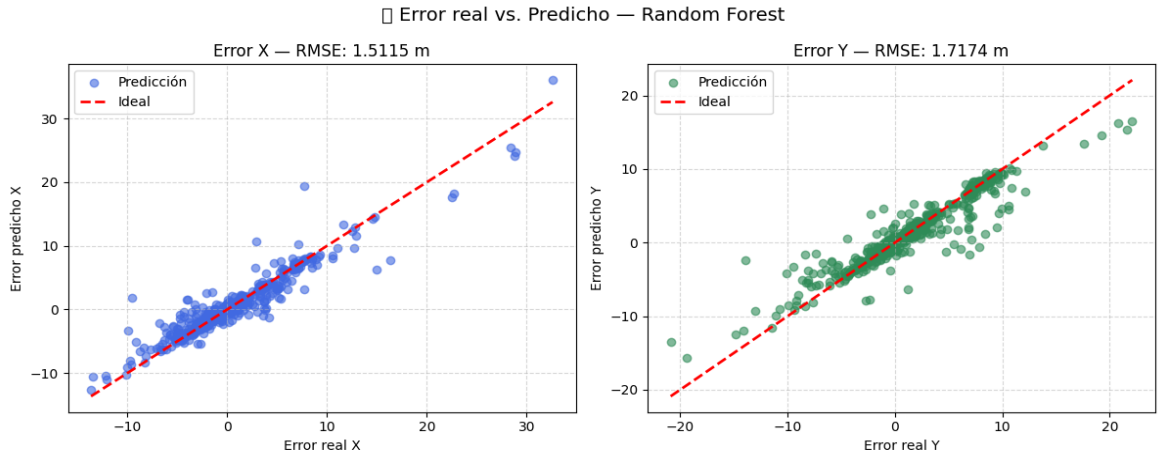


Figura. 3.16. Diagrama de error real vs estimado en el modelo RF.

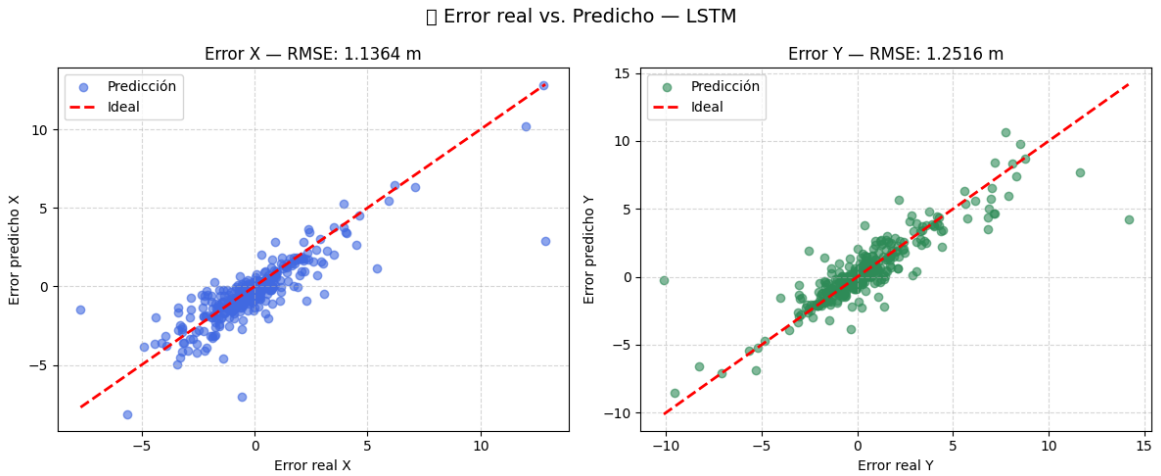


Figura. 3.17. Error real vs estimado en el modelo LSTM

Los dos anteriores gráficos fueron generadas luego de implementar los ajustes descritos en el apartado anterior, en las cuales se observa que los modelos predicen el error de posición de manera más óptima, ya que la mayoría de puntos se agrupan y se alinean más a la línea ideal, pero también se observan algunos puntos más alejados lo que indican que ubo eventos atípicos causados por una pérdida agresiva de señal, ruido en los datos o saltos temporales,

pero si se analiza la raíz cuadrática media (RMSE) de los modelos, se sabe que LSTM tiene un mejor comportamiento al momento de predecir los errores de posición y eso también se evidencia en su gráfico de comportamiento, ya que LSTM predijo errores con valores más pequeños, mientras que RF predijo errores con valores más grandes. A la final estos valores atípicos no influyeron fuertemente ya que mientras RMSE sea más pequeño el modelo en promedio predice mejor los errores con respecto a la posición real.

3.4 Población y Muestra

3.4.1 Población

La población está determinada por los datos generados en la variable dependiente Margen de error mínimo, estos datos estarán conformados por la posición en la que se encuentra el vehículo. Además, la población contemplará tres grupos de datos que serán medidos por el sistema GPS tradicional y por el nuevo sistema propuesto.

3.4.2 Muestra

La muestra fue tomada aleatoriamente a partir de los datos obtenidos de la población.

3.5 Operacionalización de las variables:

Tabla 3.4. Variables dependientes e Independientes

Variables	**Tipo**	**Descripción**	**Indicador**	**Instrumento de medición**
Margen de error mínimo	Cuantitativa Dependiente	Error entre la ubicación real y la medida.	Valor en metros	Observación
Tipo de sistema	Cualitativa Independiente	El tipo de sistema utilizado para determinar la posición.	Tipo	Observación

3.6 Hipótesis

El sistema implementado utilizando filtrado de posiciones geolocalizadas y modelos de Machine Learning (RF y LSTM) logra optimizar el posicionamiento del vehículo, comparado con el sistema GPS tradicional.

- Hipótesis nula (H_0).

La implementación del sistema utilizando modelos de ML y filtrado de posiciones geolocalizadas no mejora el rendimiento de posicionamiento del vehículo, comparado con el sistema GPS tradicional.

- Hipótesis alternativa (H_1).

La implementación del sistema utilizando modelos de ML y filtrado de posiciones geolocalizadas mejora el rendimiento de posicionamiento del vehículo, comparado con el sistema GPS tradicional.

CAPÍTULO IV

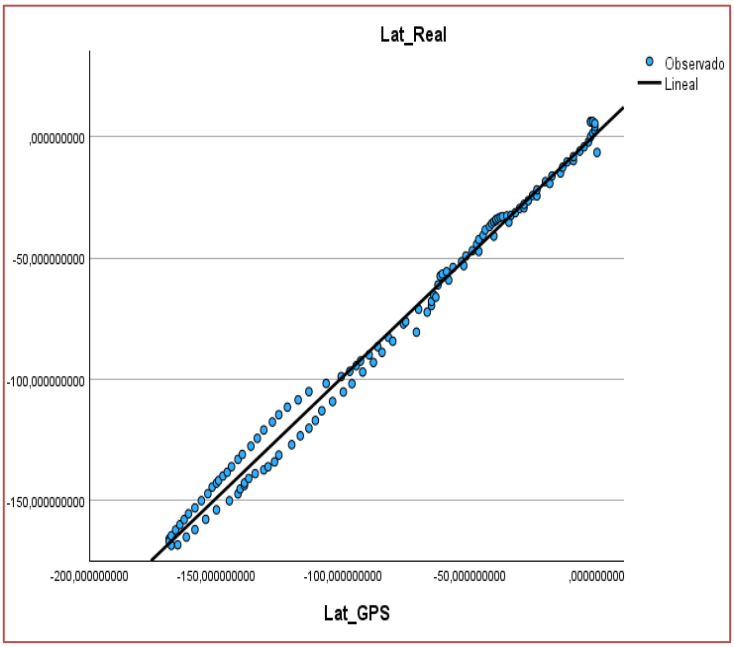
RESULTADOS Y DISCUSIÓN

Para validar el funcionamiento del sistema se realizaron pruebas de campo en entornos reales pero controlados, es decir se sigo una trayectoria predefinida a una velocidad constante comparando el error que existe entre la posición real y la estimada por el sistema GPS tradicional y el sistema implementado. Debido a la usencia de receptores por corrección de cinemática en tiempo real (RTK), se usó un GPS con soporte de múltiples costelaciones de doble banda integrado en un móvil, como referencia de posición real confiable para evaluar la precisión de estimación de posicionamiento del vehículo utilizando el sistema GPS tradicional y el sistema implementado utilizando filtrado de posiciones geolocalizadas y los modelos de Machine Learning RF y LSTM.

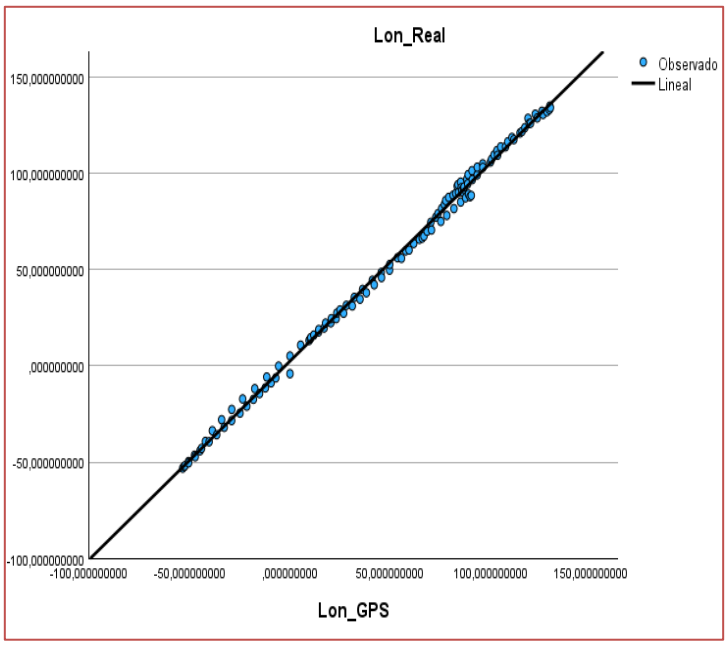
4.1 Modelo de regresión lineal.

Tabla. 4.5. Resumen de modelo de estimación lineal para el GPS.

Variable dependiente: (Latitud, Longitud) Real Variable independiente: (Latitud, Longitud) GPS	
R cuadrado	**Sig. (p)**
0,992	0,001
0,997	0,001



a) Latitud



b) Longitud

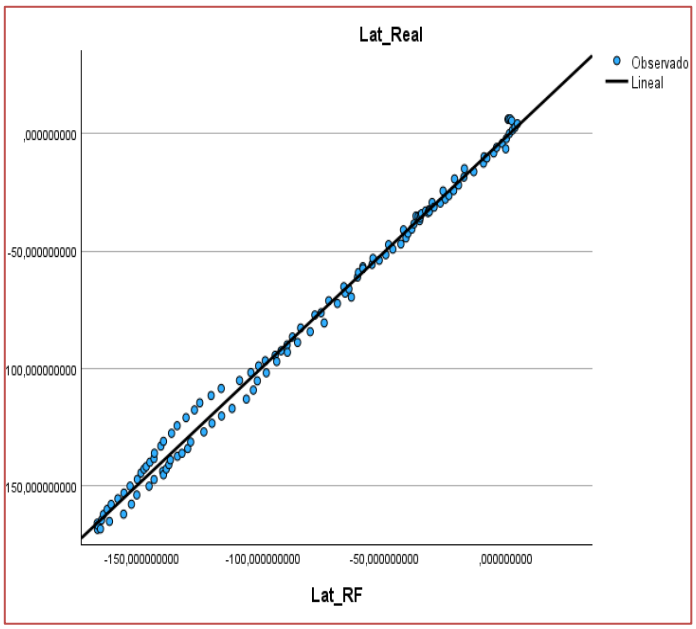
Figura. 4.18. Posición real vs estimada por el GPS tradicional: a) Latitud, b) Longitud.

Para este primer análisis, las estimaciones son hechas por el GPS tradicional, se evidencia que según el coeficiente de determinación R^2 indica que el GPS es capaz de estimar la posición con más del 92% de precisión con respecto a la real.

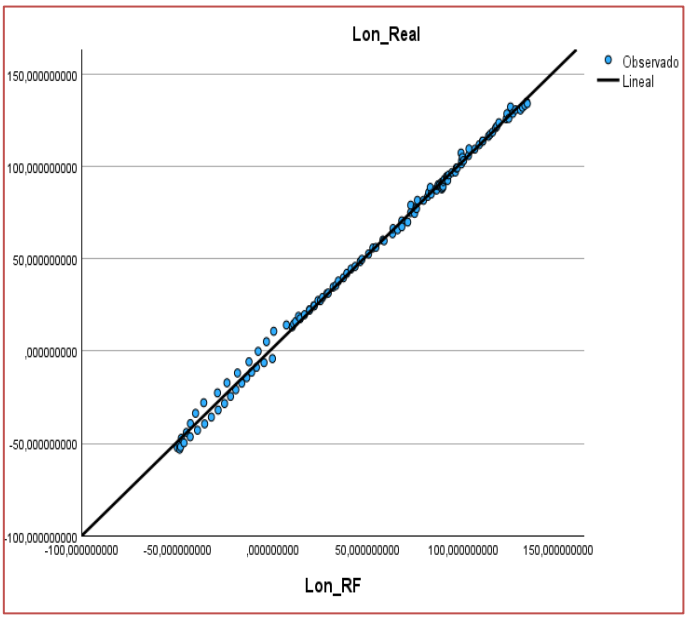
Gracias al gráfico de posición se evidencia que las posiciones GPS para esta trayectoria están casi alineadas con respecto a la línea diagonal con algunas posiciones dispersas, esto a la final indica que el GPS estimo la posición del vehículo con una precisión moderada frente a la real.

Tabla. 4.6. Resumen de modelo de estimación lineal para RF.

Variable dependiente: (Latitud, Longitud) Real Variable independiente: (Latitud, Longitud) RF	
R cuadrado	**Sig. (p)**
0,995	0,001
0,998	0,001



a) Latitud



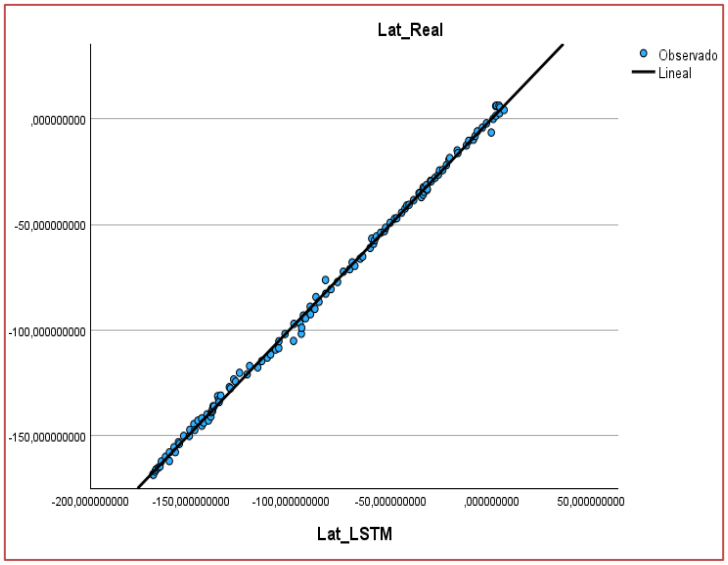
b) Longitud

Figura. 4.19. Posición real vs estimada por el modelo RF: a) Latitud, b) Longitud.

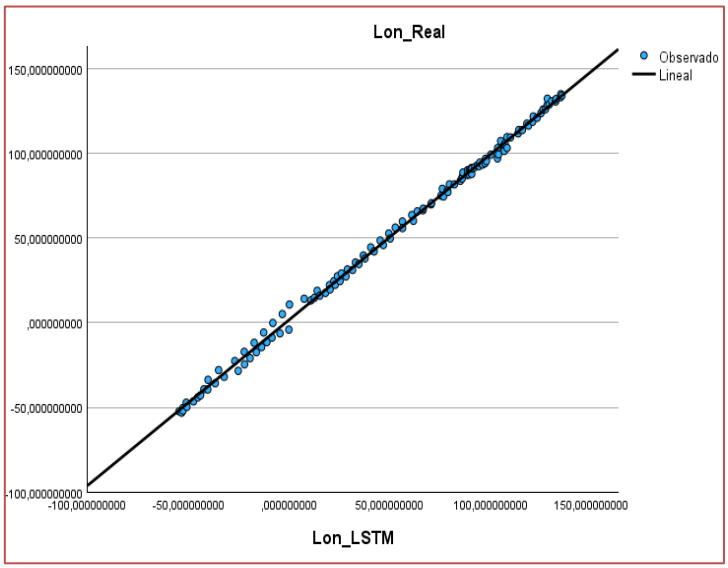
Para este segundo análisis, las estimaciones son hechas por el sistema propuesto utilizando el modelo RF, donde el coeficiente de determinación R^2 indica que RF es capaz de estimar la posición con más del 95% de precisión con respecto a la real, además por el gráfico de posición se evidencia que las estimaciones para esa trayectoria están un poco más alineadas con respecto a la línea diagonal, esto a la final indica que RF tiene mejor comportamiento a la hora de estimar la posición en comparación con el GPS.

Tabla. 4.7. Resumen de modelo de estimación lineal para LSTM.

Variable dependiente: (Latitud, Longitud) Real Variable independiente: (Latitud, Longitud) LSTM	
R cuadrado	**Sig. (p)**
0,999	0,001
0,998	0,001



a) Latitud



b) Longitud

Figura. 4.20. Posición real vs estimada por el modelo LSTM: a) Latitud, b) Longitud.

Para el tercer análisis, las estimaciones son hechas por el sistema propuesto utilizando el modelo LSTM, donde el coeficiente de determinación R^2 indica que LSTM es capaz de estimar la posición con más del 98% de precisión con respecto a la real, además por el gráfico de posición se evidencia que las estimaciones para esa trayectoria están mucho más alineadas con respecto a la línea diagonal, esto a la final indica que LSTM tiene un mejor comportamiento al estimar la posición del vehículo en comparación con el GPS y RF.

En comparación durante el recorrido de la trayectoria, aunque el GPS tiene un R^2 alto, no llega a estar al mismo nivel que el sistema utilizando los dos modelos de ML, ya que el sistema implementado demuestra tener un mejor rendimiento al estimar la posición.

Tabla. 4.8. Resumen estadístico para GPS, RF y LSTM.

		Estadístico	**Error estándar**
E_GPS	Media	5,99860	0,334898
	Mediana	6,64714	
	Varianza	14,244	
	Desv. estándar	3,774114	
E_RF	Media	4,64594	0,262658
	Mediana	3,93227	
	Varianza	8,762	
	Desv. estándar	2,960000	
E_LSTM	Media	2,81619	0,200097
	Mediana	2,12237	
	Varianza	5,085	
	Desv. estándar	2,254976	

En este caso se va analizar el error euclidiano cometido por cada sistema, este error representa cuanta diferencia en metros hay entre la posición real del vehículo y la estimada, esto nos indica que tan precisas son las estimaciones realizadas por el sistema con respecto a los valores reales, por lo cual los datos de posición fueron transformados a coordenadas (x,y) en metros para un mejor análisis.

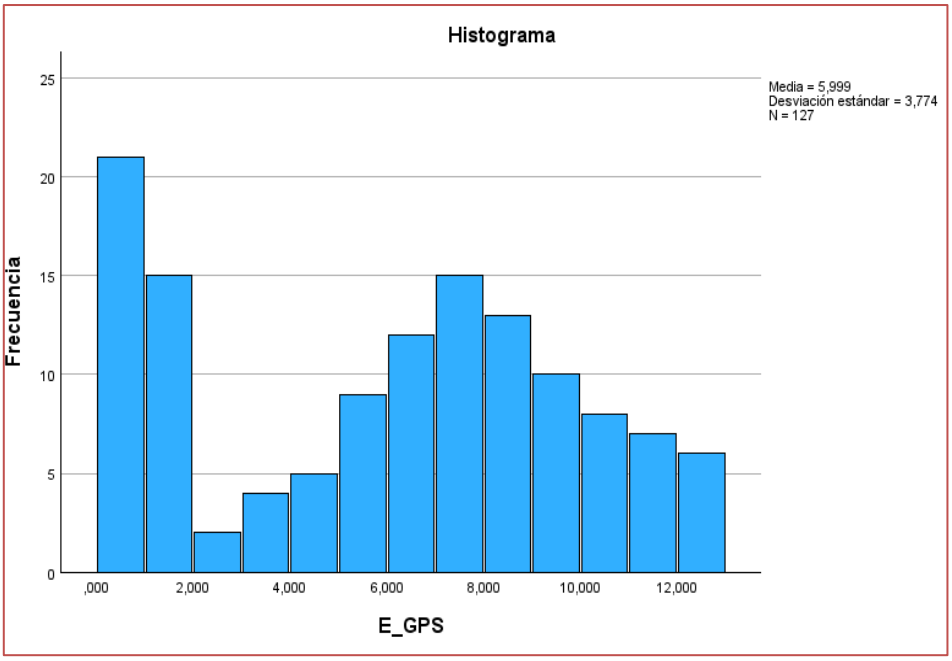
Las estimaciones hechas por el GPS convencional tienen una mediana 6,64714 m, este valor es alto e indica que las estimaciones no son muy precisas, esto se evidencia también en la desviación estándar de 3,774114 lo cual indica que las estimaciones están muy dispersas, lo que indica que hay casos en que el GPS estima bien la posición mientras en otros casos no, por lo cual las estimaciones son menos consistentes, también otra variable a considerar es el error estándar 0,334898 indica que tan precisa es la media calculada con respecto al valor real, pal caso tenemos un valor pequeño.

Las estimaciones hechas por el sistema propuesto utilizando RF tienen una mediana de error de 3,93227 m, este valor es menor al del GPS, pero sigue siendo un error considerable, pal caso la desviación estándar es 2,960000 con un error estándar de 0,262658, esto indica que

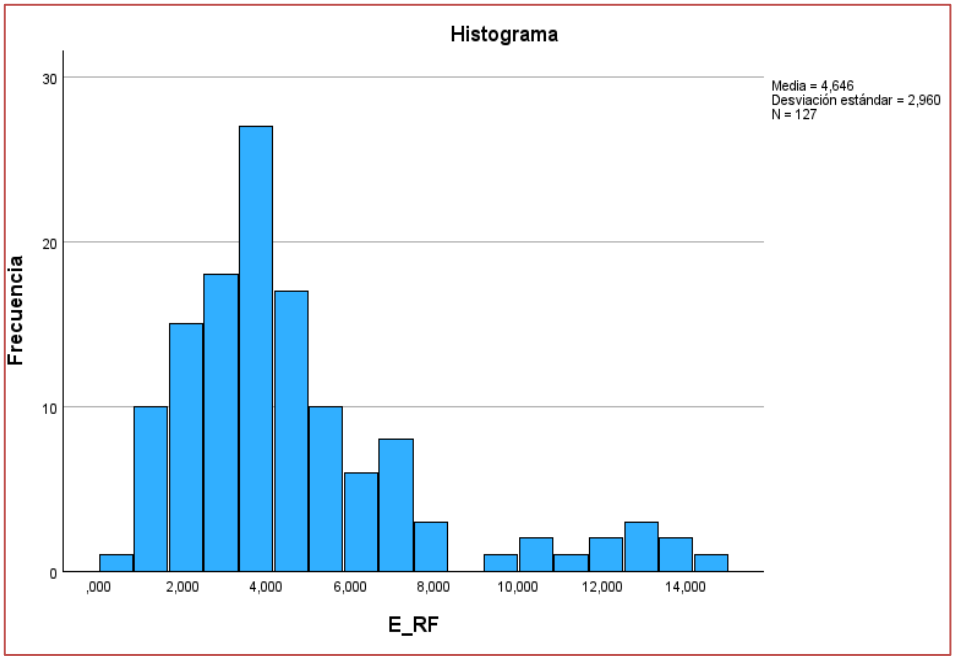
RF tiene menos errores al estimar la posición que el GPS y también sus estimaciones están menos dispersas y son consistentes.

Las estimaciones hechas por el sistema utilizando LSTM tienen una mediana de error de 2,12237 m, un valor menor al GPS y RF, tiene una desviación estándar de 2,254976 y un error estándar de 0,200097, estos datos indican que LSTM tiene menos errores al estimar la posición y que sus estimaciones son más consistentes que los anteriores modelos.

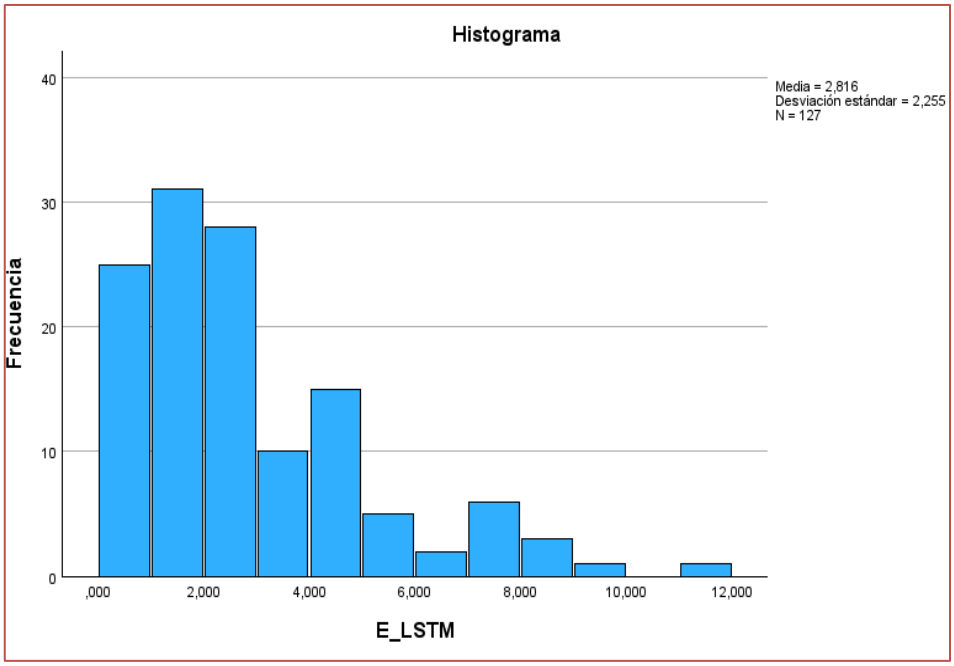
En general durante el recorrido de la trayectoria se optó por analizar la mediana de error ya que los datos tienden a no seguir una distribución normal, lo cual se analizara y comprobará más adelante, a la final la mediana representa un punto central, en donde el 50% de errores serán mayores o iguales que la mediana y el otro 50% serán iguales o menores de la misma, por tanto el sistema utilizando LSTM con una mediana menor que el resto estima las posiciones del vehículo con menos errores, esto también se refleja en sus valores de media, desviación estándar y error estándar, los cuales son más pequeños que el resto, por tanto, el sistema implementado demuestra ser más confiable, lo que sugiere que la media de error cometido por LSTM es más confiable por tanto el rendimiento general del sistema es mejor que el sistema GPS y RF.



a) GPS



b) RF



c) LSTM

Figura. 4.21. Distribución de errores: a) GPS, b) RF y c) LSTM.

En estas figuras se evidencian los errores cometidos por cada sistema al estimar la posición y con qué frecuencia se produjeron dichos errores, entonces como se evidencia el GPS convencional comete errores grandes y pequeños con mayor frecuencia, mientras que RF también comete errores grandes pero con menos frecuencia y errores pequeños con mayor frecuencia, pal caso de LSTM comete errores pequeños con más frecuencia y aunque también comete errores grande lo hace con mucha menos frecuencia que el resto

evidenciando que el sistema propuesto usando el modelo LSTM generalmente es más confiable, ya que sus estimaciones de posición del vehículo están más cercanas a los valores reales.

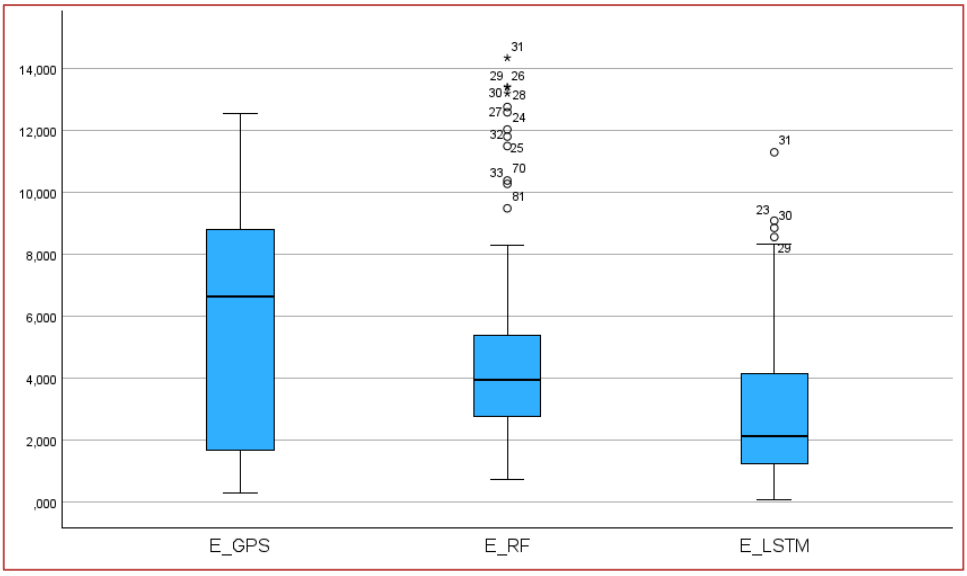


Figura. 4.22. Comparación de errores para los grupos (GPS, RF y LSTM).

Este diagrama de cajas representa la distribución de errores cometidos al estimar la posición del vehículo hecha por cada modelo, entonces se evidencia que GPS tiene una dispersión de errores bastante grande esto se observa por el ancho de su caja con una mediana bastante alta de 6,64714 m, mientras que RF con una mediana de 3,93227 m cuenta con una dispersión de errores mucho más reducido que GPS, pero con la presencia de valores atípicos que indica que hay casos en donde este modelo realiza predicciones con un error mucho mayor al medio, para el caso de LSTM con una mediana de 2,12237 m tenemos una distribución de errores mucho menor que el GPS y RF, pero también con presencia de valores atípicos que son mucho menos frecuentes con respecto al modelo de RF.

En general si comparamos los tres modelos, GPS presenta errores muy grandes de manera habitual, lo que indica que tiene una baja precisión y gran variabilidad de errores, mientras que el sistema propuesto usando los modelos de ML evidencian ser más estables ya que logran reducir el error medio en gran medida aunque cuentan con valores atípicos, estos valores fueron generalmente ocasionados por perdida agresiva de la señal GPS y también posiblemente porque los modelos de ML no generalizan correctamente esa trayectoria, es decir les falta más datos de entrenamiento que les ayude a describir y predecir mejor la trayectoria, estos errores atípicos presentes se mantienen dentro del rango normal de errores del GPS lo que significa que incluso dichos errores no afectan directamente al comportamiento general del sistema, ya que su promedio de error sigue estando muy por debajo del GPS, entonces a la final el sistema usando el modelo LSTM claramente tiene mejor precisión al momento de estimar la posición del vehículo, ya que cuenta con errores más pequeños y consistentes que el resto de modelos, además de que este modelo logra reducir en gran medida los errores atípicos, lo que demuestra ser más robusto.

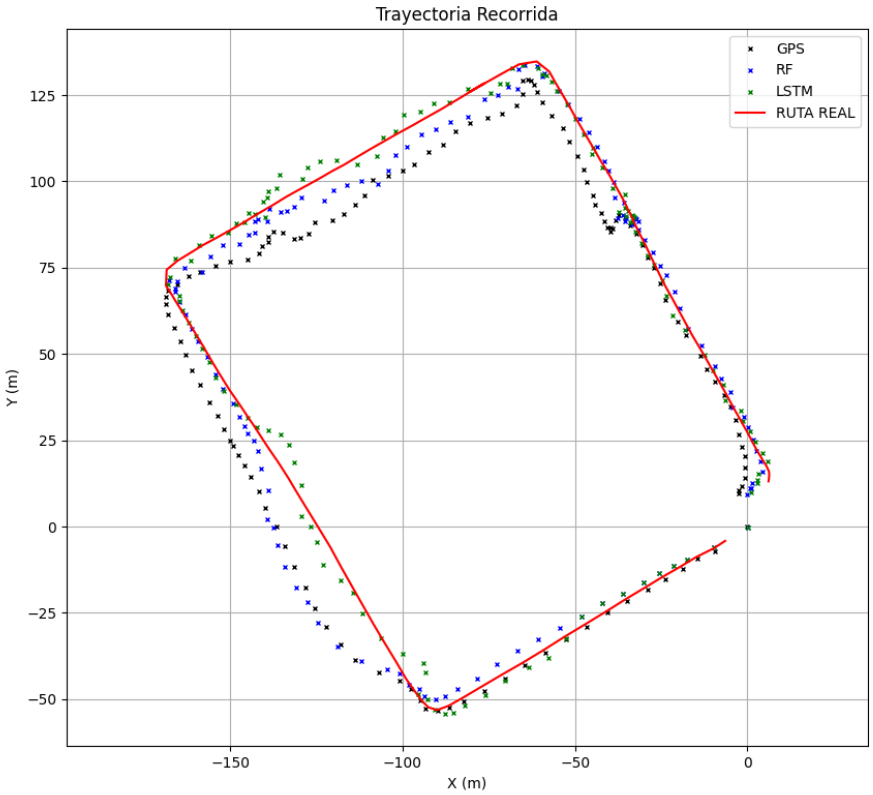


Figura. 4.23. Trayectoria recorrida para evaluar los resultados.

4.2 Test de Normalidad.

Con el fin de determinar que prueba estadística se ajusta para analizar los datos obtenidos se evaluó la normalidad de los datos, entonces se optó por trabajar con (n = 127) dado el tamaño de los datos ($n \geq 50$) se aplicó la prueba de Kolmogorov–Smirnov como método de referencia para saber si estos conjuntos de datos siguen una distribución normal, este enfoque ayuda a definir si se puede usar pruebas paramétricas o no paramétricas.

Hipótesis:

- Hipótesis nula (H_0).

Los datos de los grupos analizados siguen una distribución normal.

- Hipótesis alternativa (H_1).

Los datos de los grupos analizados no siguen una distribución normal.

Tabla. 4.9. Prueba de normalidad.

Grupo	Sig (p).	P-valor
E_GPS	0,001	0,001 es menor a 0,05
E_RF	0,001	0,001 es menor a 0,05
E_LSTM	0,001	0,001 es menor a 0,05

Evaluado el Test de Normalidad se rechaza la hipótesis nula y se acepta la hipótesis alternativa con un nivel de significancia = 0,05, lo que sugiere que los grupos de datos analizados no siguen una distribución normal con un nivel de confianza del 95%.

4.3 Test de Friedman.

Se realizó un test de Friedman puesto que los grupos son relacionados y no siguen una distribución normal, entonces se tomó la mediana de error de posicionamiento de cada grupo como referencia para evaluar cuál de ellos tuvo un mejor rendimiento al estimar la posición del vehículo.

- Hipótesis nula (H_0).

Las medianas de error de posicionamiento del sistema GPS tradicional y el sistema implementado utilizando modelos de ML y filtrado de posiciones geolocalizadas son significativamente iguales.

- Hipótesis alternativa (H_1).

Las medianas de error de posicionamiento del sistema GPS tradicional y el sistema implementado utilizando modelos de ML y filtrado de posiciones geolocalizadas son significativamente diferentes.

Tabla. 4.10. Test de Friedman.

N	127
Chi-cuadrado	76,621
gl	2
Sig. Asintótica.	0,001

Evaluado el Test de Friedman se rechaza la hipótesis nula y se acepta la hipótesis alternativa con un nivel de significancia = 0,05, lo que sugiere que las medianas de los tres grupos son significativamente diferentes con un 95% de confianza.

4.4 Comparaciones Post-Hoc

Tabla. 4.11. Comparaciones por pares.

Grupos	Sig (p)	Sig. ajustada
E_LSTM-E_RF	0,001	0,000
E_LSTM-E_GPS	0,001	0,000
E_RF-E_GPS	0,013	0,040

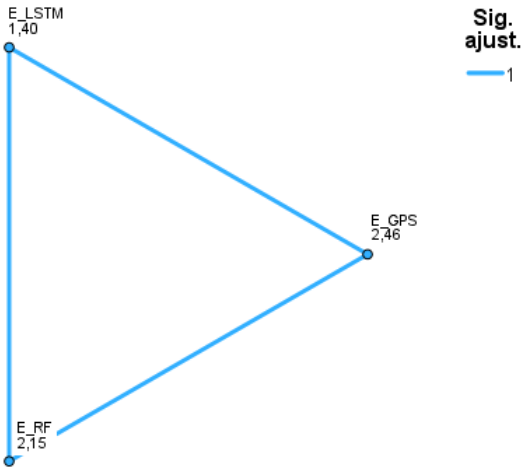


Figura. 4.24. Comparaciones por parejas (GPS, RF y LSTM).

Evaluado las comparaciones Post-Hoc, en donde los grupos de errores se comparan uno a uno, nos demuestra que sin excepción cada grupo tiene mediana de errores distintos, pero se observa que entre los pares GPS y RF esa diferencia es más pequeña con p = 0,013 mientras que en el resto de pares la diferencia es más grande con p = 0,001, lo mismo se evidencia en el gráfico de comparaciones por parejas, en donde LSTM tiene un rango de error medio más bajo con 1,40, seguido de RF con 2,15 y por ultimo GPS con 2,46, por tanto el sistema implementado utilizando LSTM tiene un rendimiento significativamente mejor que RF y GPS.

4.4.1 Discusión Final.

El estudio estadístico realizado, demuestra que el sistema implementado para optimizar el posicionamiento del vehículo mediante filtrado de posiciones geolocalizadas y Machine Learning logra tener un mejor rendimiento al estimar la posición frente al GPS tradicional con una confianza del 95%, esto se evidencia en sus variables estadísticas como: media, mediana y desviación estándar que demuestran que el sistema utilizando los modelos de ML reducen progresivamente el error de posicionamiento y su variabilidad.

El análisis comparativo demuestra que LSTM tiene un rango de errores menor que el resto, logrando ser más confiable y estable, lo que deja al modelo RF en segundo lugar, por tanto LSTM tiene un mejor desempeño proporcionando estimaciones más estables gracias a su aprendizaje secuencial demostrando ser robusto frente a datos ruidosos, estos resultados validan la hipótesis de que el sistema implementado usando filtrado y ML optimizan de manera significativa la precisión y estabilidad de posicionamiento vehicular frente al GPS tradicional.

CAPÍTULO V

CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

Este proyecto de investigación implementó un sistema de posicionamiento usando ML y filtrado de posiciones geolocalizadas para optimizar el posicionamiento de un vehículo, por lo cual se diseñó una página web que permitió mostrar los resultados de posición y trayectoria que siguió el vehículo en tiempo real.

La implementación del sistema usando el modelo LSTM redujo de manera considerable el error de posicionamiento vehicular con respecto al GPS tradicional, donde LSTM cuenta con un error central de 2,12237m, frente a 3,93227m de RF y 6,64714m del GPS confirmando que el uso del EKF y ML permite optimizar el posicionamiento vehicular y mejorar el rendimiento del sistema asta en un 68% con respecto al GPS tradicional.

El modelo LSTM al ser más robusto frente a condiciones externas como ruido o perdida de señal satelital muestra tener una menor dispersión de errores, por tanto, sus estimaciones son más estables y confiables.

La combinación de EKF y los modelos de ML demostraron ser una combinación efectiva y robusta al optimizar el posicionamiento vehicular, ya que especialmente LSTM tiene la capacidad de modelar patrones temporales y capturar errores que el filtro de Kalman no puede.

Es importante destacar que la referencia de posición real fue tomada de un sistema GNSS de doble banda, cuyo margen de error aproximado es de 1.5m [2], esto implica que los errores de posición reportados tanto por el GPS tradicional y el sistema implementado incluyan un pequeño valor de incertidumbre, ya que el error reportado no podrá ser menor al del sistema de referencia real, por tanto el error final reportado por LSTM variaría ligeramente, esto no afecta al resultado final, ya que la diferencia entre estos errores es grande, por tanto la mejora sigue siendo estadísticamente significativa.

5.2 Recomendaciones

Se recomienda la fusión de sensores con técnicas de filtrado como el EKF y ML para aprovechar los beneficios que cada tecnología ofrece, ya que ayudan a optimizar la precisión de estimación vehicular en tiempo real, incluyo se podría utilizar técnicas de filtrado y ML más robustas con lo cual posiblemente se conseguiría mejores resultados, pero sacrificando la parte de costo computacional el cual sería más elevado.

Es recomendable estudiar cómo afectan las condiciones del entorno a la calidad de las mediciones, esto permite implementar ajustes y calibraciones tanto a la etapa de recolección de datos, filtrado y para la etapa de estimación con ML, logrando una mejor respuesta frente a estas condiciones, por ende, los resultados serán más confiables.

Se recomienda tener una dase de entrenamiento variada para los modelos de ML como ejemplo: contar con rutas y condiciones ambientales diferentes, para que estos modelos aprendan a generalizar y estimar mejor una trayectoria, con esto se logra que las estimaciones sean más precisas y consistentes.

Para futuros estudios se recomienda, tomar como referencia de posición real de alta precisión al sistema RTK, ya que este ofrece estimaciones con errores a nivel de cm, por ende, se podría contar con una posición de referencia sin incertidumbre, lo cual aria que el error reportado sea más fiable, por otro lado se podría adquirir sensores más robustos, los cuales ayudarían a que las estimaciones finales sean más precisas, pero todas estas consideraciones incrementarían considerablemente es costo total del sistema.

BIBLIOGRAFÍA

[1] «Sistema GPS,» Ministerio de Gobierno, 2021. [En línea]. Available: https://www.ministeriodegobierno.gob.ec/sistema-gps-y-modernos-vehiculos-optimizan-patrullaje-policial/.

[2] «Advanced Navigation,» 2023. [En línea]. Available: https://n9.cl/t701d.

[3] «Paratopografía. Sistema satelital de navegación global,» 2024. [En línea]. Available: https://paratopografia.com/geodesia/sistema-satelital-de-navegacion-global/.

[4] L. H. S. D. G. G. Darvis Dorvigny Dorvigny, «Algoritmo de navegación integrada para vehículos autónomos con tecnología de bajo costo.,» *Revista Cubana de Ciencias Informáticas*, vol. 3, nº 12, pp. 121-139, 22 Abril 2018.

[5] S. M. J. N. Felipe Jimenez, «Mejora del posicionamiento de vehículos de carretera en áreas de mala cobertura,» Madrid, 2016.

[6] M. D. A. Z. Kerman Viana, «FUSION DE SENSORES PARA LOCALIZACIÓN ROBUSTA DE VEHÍCULOS AUTONOMOS EN ÁREAS URBANAS,» Bilbao, 2021.

[7] B. D. G. Niño, «Posicionamiento de un automóvil en espacios reducidos utilizando un sistema de navegación inercial con filtros de kalman.,» Bucaramanga, 2020.

[8] A. Q. González, «Implementación de un Sistema de Geolocalización y Posicionamiento mediante Comunicaciones por Luz visible,» Madrid, 2021.

[9] P. d. l. V. d. l. Iglesia, «Evaluación de un sistema de navegación inercial GNSS/IMU en automoción para sistemas de automatización a la conducción.,» Salamanca, 2023.

[10] A. S. Allende, «Implementación de redes neuronales para conducción autónoma,» Cantabria, 2021.

[11] Y. A. J. MONTERO, «DESARROLLO DE UN SISTEMA DE VISIÓN ARTIFICIAL BASADO EN REDES CONVOLUCIONALES PARA EL CORRECTO POSICIONAMIENTO DE UN VEHÍCULO AUTÓNOMO EN UN CARRIL,» Riobamba, 2021.

[12] J. F. M. BENÍTEZ, «MODELO DE PREDICCIÓN Y ESTIMACIÓN DE TIEMPOS DE TRASLADO ENTRE DOS PUNTOS UTILIZANDO DATOS DE GPS.,» Santigo de Chile, 2020.

[13] S. M. Hamdoun, «Fusión de Lidar-SLAM y Deep learning para mejorar el posicionamiento en vehículos autónomos,» Madrid, 2022.

[14] C. A. Falagán, «SISTEMA DE POSICIONAMIENTO Y LOCALIZACIÓN DE UN ROBOT AUTÓNOMO EN SUPERFICIES EXTERIORES,» Cantabria, 2021.

[15] Á. T. Tamayo, «Detección de multipath en mediciones GNSS urbanas y semiurbanas mediante inteligencia artificial,» Madrid, 2023.

[16] L. S. O. D. L. D. T. J. P. C. Ignacio Zaradnik, «Determinación de la Orientación en Vehículos No Tripulados,» Buenos Aires, 2023.

[17] R. E. C.-F. I. E. E.-V. Leonela Del Rocio De La A Salinas, «Aplicación de los filtros de Kalman en los sistemas de navegación autónoma,» *Ibero-American Journal of Engineering & Technology Studies*, vol. 1, nº 3, pp. 198-204, 2023.

[18] G. Yujun, «Análisis y mejoras en algoritmos para posicionamiento y localización basado en Ultra-Wide Band,» Valencia, 2024.

[19] A. M. C. Pescador, «Fusión Sensórica INS/ GPS para Navegación en Plataformas,» Bogotá, 2013.

[20] A. Elmquist y D. Negrut, «Virtual Sensing for Autonomous Vehicle Simulation in Chrono,» 2017.

[21] S. Z. E.-S. N. Aggarwal P, «MEMS-based integrated navigation.,» Canadá, Artech House, 2010, pp. 63-80.

[22] F. Villegas, «Relatividad y el Sistema de Posicionamiento Global (GPS).,» *Revista de investigación de Física,* vol. 1, nº 23, pp. 44-47, 2020.

[23] «Seinxon,» 25 Abril 2024. [En línea]. Available: https://www.seinxon.com/blogs/blog-posts/how-does-a-gps-tracker-work. [Último acceso: 18 Julio 2025].

[24] A. H. Herrada y S. A. Miranda, «PROTOCOLOS Y FORMATOS PARA LA DISEMINACIÓN DE DATOS GNSS,» vol. 1, nº 39, pp. 174-180, 2014.

[25] «Security GPS,» 1 Octubre 2022. [En línea]. Available: https://securitygps.com.ar/protocolo-nmea-gps/. [Último acceso: 18 Julio 2025].

[26] M. Mejia, «Integración del filtro de Kalman a un Sistema de Posicionamiento Global (GPS) para aplicación en vehículos autónomos.,» Monterrey, 2023.

[27] C. Arana, «Redes neuronales recurrentes: Análisis de los modelos especializados en datos secuenciales.,» Buenos Aires, 2021.

[28] E. Z. J. Jesús, «Aplicación de algoritmos Random Forest y XGBoost en una base de solicitudes de tarjetas de crédito,» vol. XXI, nº 3, 2020.

[29] «DATA BASE CAMP,» What is Randon Fores?, 22 06 2022. [En línea]. Available: https://databasecamp.de/en/ml/random-forests. [Último acceso: 24 07 2025].

[30] M. F. H. S. J. A. A. M. E. H. S. A. A. M. G. R. Safwan Mahmood Al-Selwi, « RNN-LSTM:Fromapplicationstomodelingtechniquesand beyond—Systematicreview,» Malaysia, 2024.

[31] «ALLDATASHEET,» MPU-6050, [En línea]. Available: https://www.alldatasheet.com/datasheet-pdf/pdf/1132807/TDK/MPU-6050.html. [Último acceso: 24 07 2025].

[32] «Tutorial MPU6050, Acelerómetro y Giroscopio,» NAYLAMP MECHATRONICS, [En línea]. Available: https://naylampmechatronics.com/blog/45_tutorial-mpu6050-acelerometro-y-giroscopio.html. [Último acceso: 24 07 2025].

[33] «ALLDATASHEET,» GPS NEO 6M, [En línea]. Available: https://www.alldatasheet.com/datasheet-pdf/view/1283987/U-BLOX/NEO-6M.html. [Último acceso: 24 07 2025].

[34] «AV Electronics,» Módulo GPS NEO-6M, [En línea]. Available: https://avelectronics.cc/producto/modulo-gps-neo-6m/?srsltid=AfmBOoqp5RL8_6FU59zSlETJJAx04OiYt32g_dJxf3Et_RseQPcVAm2E. [Último acceso: 24 07 2025].

[35] [En línea]. Available: https://share.google/4nt0ftwbHCfbgEZZ3. [Último acceso: 11 07 2025].

[36] «ALLDATASHEET,» ESP32, [En línea]. Available: https://www.alldatasheet.com/datasheet-pdf/pdf/1148023/ESPRESSIF/ESP32.html. [Último acceso: 25 07 2025].

[37] «TECmikro,» Modulo ESP32 Wifi Bluetooth, [En línea]. Available: https://tecmikro.com/modulos-shields/630-modulo-esp32-wifi-bluetooth.html. [Último acceso: 25 07 2025].

[38] MicroPython: ESP32 con módulo GPS NEO-6M, RANDOM NERD TUTORIALS. [En línea]. Available: https://randomnerdtutorials.com/micropython-esp32-neo-6m-gps/. [Último acceso: 25 07 2025].

[39] «GITHUB,» MPU6050-MicroPython, [En línea]. Available: https://github.com/Lezgend/MPU6050-MicroPython/blob/main/MPU6050.py.

[40] A. Shafi, «Datacamp,» 29 02 2024. [En línea]. Available: https://www.datacamp.com/es/tutorial/random-forests-classifier-python. [Último acceso: 28 07 2025].

[41] «Geeksforgeeks,» Random Forest Regression in Python, 12 07 2025. [En línea]. Available: https://www.geeksforgeeks.org/machine-learning/random-forest-regression-in-python/. [Último acceso: 28 07 2025].

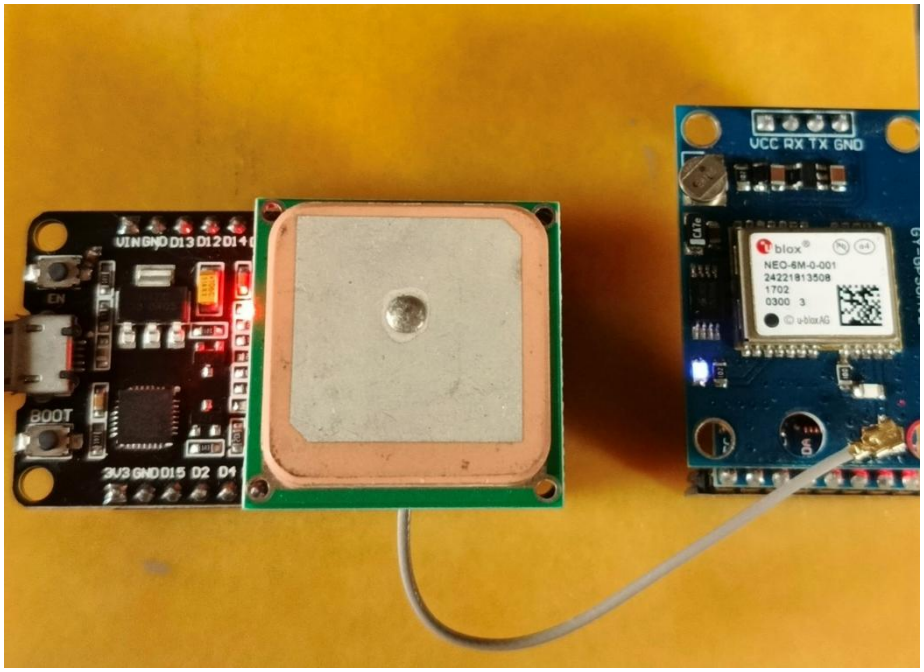
[42] J. Brownlee, «Machine Learning Mastery,» 07 08 2022. [En línea]. Available: https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/. [Último acceso: 28 07 2025].

[43] Julles, «Medium,» 12 10 2024. [En línea]. Available: https://medium.com/@techwithjulles/recurrent-neural-networks-rnns-and-long-short-term-memory-lstm-creating-an-lstm-model-in-13c88b7736e2. [Último acceso: 28 07 2025].

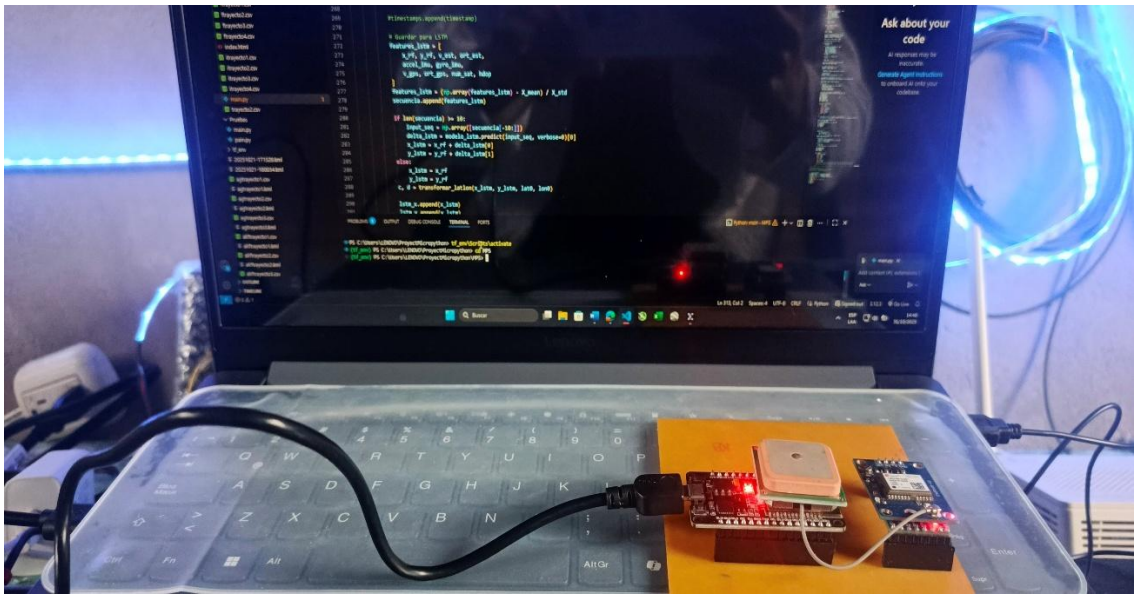
[44] J. C. D. T. F. C. M. K. Julián Tomaštík, «Avances en el posicionamiento de smartphones en bosques: receptores de doble frecuencia y datos GNSS en bruto,» p. 292–310, 07 09 2020.

ANEXOS

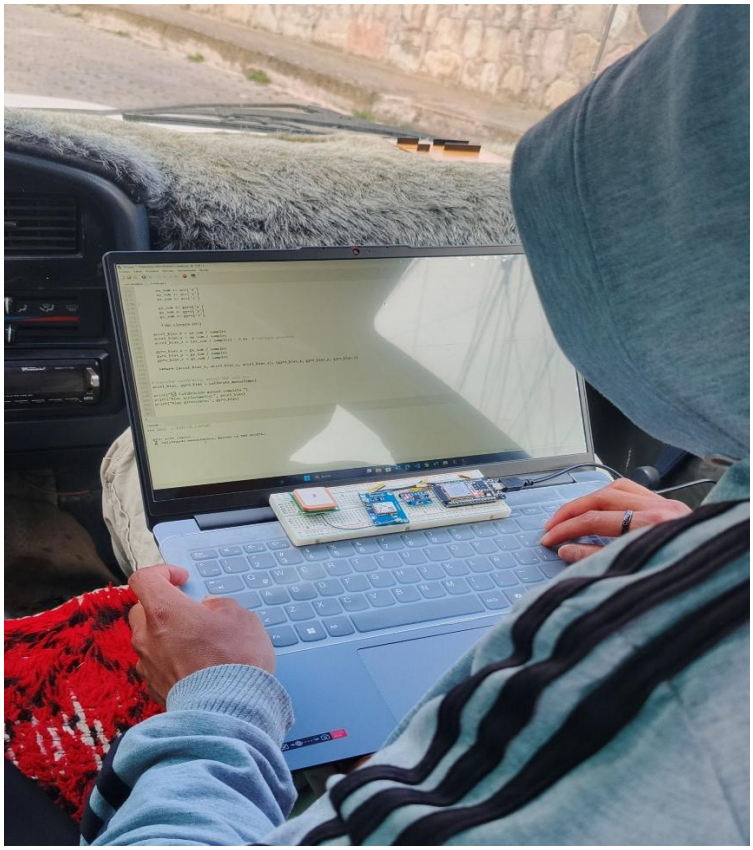
Anexo 1: Dispositivo ensamblado.



Anexo 2: Pruebas iniciales del funcionamiento del sistema.

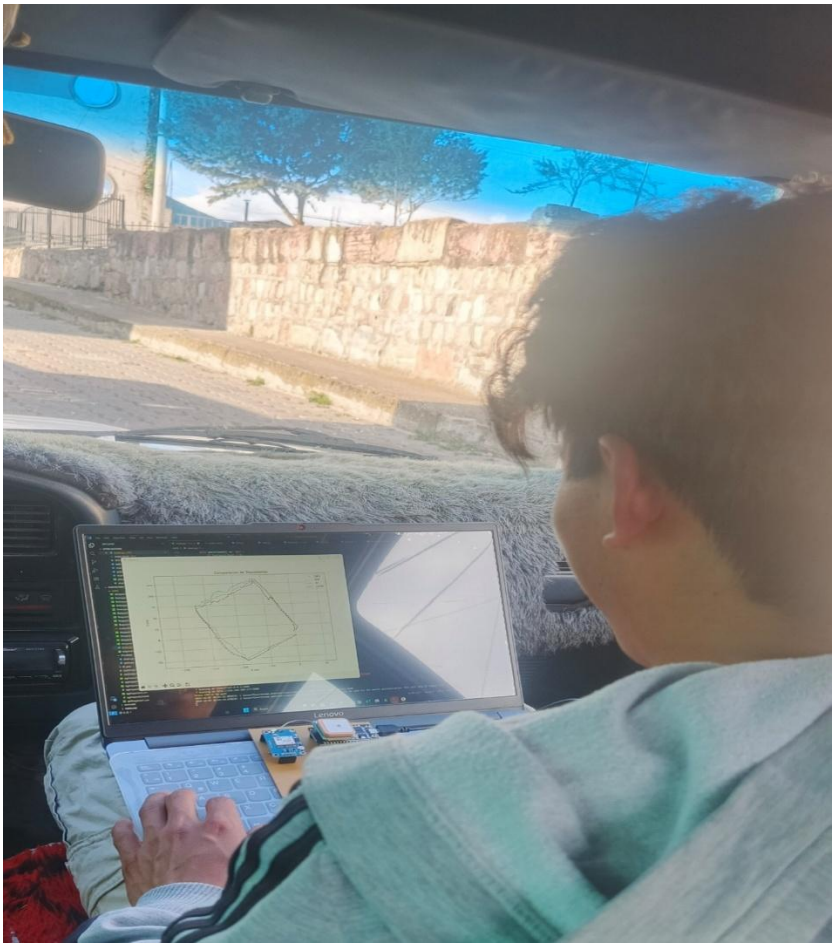


Anexo 3: Calibración de los sensores.

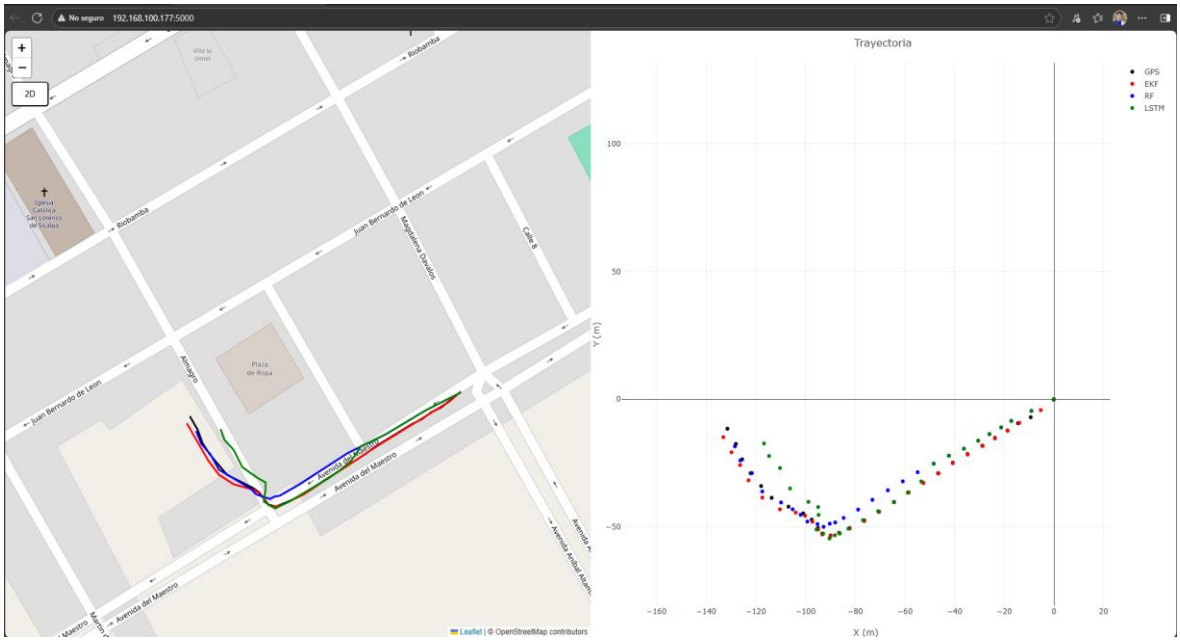


Anexo 4: Recopilación de información para armar la base de datos.

Anexo 5: Pruebas de campo.



Anexo 6: Seguimiento de vehículo en tiempo real.



Anexo 7: Código para GPS NEO-6M.

```
def latitude_string(self):
    """
    Create a readable string of the current latitude data
    :return: string
    """
    if self.coord_format == 'dd':
        lat_string = self.latitude
    elif self.coord_format == 'dms':
        formatted_latitude = self.latitude
        lat_string = str(formatted_latitude[0]) + '° ' + str(formatted_latitude[1]) + "' " + str(formatted_latitude[2]) + '" ' + str(formatted_latitude[3])
    else:
        lat_string = str(self._latitude[0]) + '° ' + str(self._latitude[1]) + "' " + str(self._latitude[2])
    return lat_string

def longitude_string(self):
    """
    Create a readable string of the current longitude data
    :return: string
    """
    if self.coord_format == 'dd':
        lon_string = self.longitude
    elif self.coord_format == 'dms':
        formatted_longitude = self.longitude
        lon_string = str(formatted_longitude[0]) + '° ' + str(formatted_longitude[1]) + "' " + str(formatted_longitude[2]) + '" ' + str(formatted_longitude[3])
    else:
        lon_string = str(self._longitude[0]) + '° ' + str(self._longitude[1]) + "' " + str(self._longitude[2])
    return lon_string

def speed_string(self, unit='km/h'):
    """
    Creates a readable string of the current speed data in one of three units
    :param unit: string of 'kph','mph, or 'knot'
    :return:
    """
    if unit == 'mph':
        speed_string = str(self.speed[1]) + ' mph'

    elif unit == 'knot':
        if self.speed[0] == 1:
            unit_str = ' knot'
        else:
            unit_str = ' knots'
        speed_string = str(self.speed[0]) + unit_str
```

Anexo 8: Código para MPU 6050.

```
def get_gyro_range(self, raw = False):
    # Get the raw value
    raw_data = self.i2c.readfrom_mem(self.addr, _GYRO_CONFIG, 2)

    if raw is True:
        return raw_data[0]
    elif raw is False:
        if raw_data[0] == _GYR_RNG_250DEG:
            return 250
        elif raw_data[0] == _GYR_RNG_500DEG:
            return 500
        elif raw_data[0] == _GYR_RNG_1000DEG:
            return 1000
        elif raw_data[0] == _GYR_RNG_2000DEG:
            return 2000
        else:
            return -1

# Gets and returns the GyX, GyY and GyZ values from the gyroscope.
# Returns the read values in a dictionary.
def read_gyro_data(self):
    gyro_data = self._readData(_GYRO_XOUT0)
    gyro_range = self._gyro_range
    scaler = None
    if gyro_range == _GYR_RNG_250DEG:
        scaler = _GYR_SCLR_250DEG
    elif gyro_range == _GYR_RNG_500DEG:
        scaler = _GYR_SCLR_500DEG
    elif gyro_range == _GYR_RNG_1000DEG:
        scaler = _GYR_SCLR_1000DEG
    elif gyro_range == _GYR_RNG_2000DEG:
        scaler = _GYR_SCLR_2000DEG
    else:
        print("Unkown range - scaler set to _GYR_SCLR_250DEG")
        scaler = _GYR_SCLR_250DEG

    x = gyro_data["x"] / scaler
    y = gyro_data["y"] / scaler
    z = gyro_data["z"] / scaler

    xr = x * pi / 180
    yr = y * pi / 180
    zr = z * pi / 180
    return {"x": xr, "y": yr, "z": zr}
```

Anexo 9: Código para calibración de sensores.

```
from MPU6050 import MPU6050
import time

mpu = MPU6050()

def calibrate_manual(mpu, samples=500):
    print("⏳ Calibrando manualmente… Mantén la IMU QUIETA…")
    ax_sum = ay_sum = az_sum = 0
    gx_sum = gy_sum = gz_sum = 0

    for _ in range(samples):
        acc = mpu.read_accel_data()  # devuelve dict {'x':.., 'y':.., 'z':..}
        gyro = mpu.read_gyro_data()  # devuelve dict {'x':.., 'y':.., 'z':..}

        ax_sum += acc['x']
        ay_sum += acc['y']
        az_sum += acc['z']

        gx_sum += gyro['x']
        gy_sum += gyro['y']
        gz_sum += gyro['z']

        time.sleep(0.005)

    accel_bias_x = ax_sum / samples
    accel_bias_y = ay_sum / samples
    accel_bias_z = (az_sum / samples) - 9.81  # corregir gravedad

    gyro_bias_x = gx_sum / samples
    gyro_bias_y = gy_sum / samples
    gyro_bias_z = gz_sum / samples

    return (accel_bias_x, accel_bias_y, accel_bias_z), (gyro_bias_x, gyro_bias_y, gyro_bias_z)


# Ejecutar calibración manual UNA sola vez
accel_bias, gyro_bias = calibrate_manual(mpu)

print("Calibración manual completa.")
print("Bias acelerómetro:", accel_bias)
print("Bias giroscopio:", gyro_bias)
```

Anexo 10: Código para la ESP32

```
time.sleep(2)
while True:
    ahora = time.ticks_ms()
    t1 = time.ticks_diff(ahora, imu_ms)
    t2 = time.ticks_diff(ahora, gps_ms)
    t3 = time.ticks_diff(ahora, intervalo)

    acc = mpu.read_accel_data()
    gyro = mpu.read_gyro_data()

    acc_corr, gyro_corr = filtrar_imu(acc, gyro)

    try:
        if t1 >= intervalo_imu:
            imu_ms = ahora
            mensaje = "IMU,{},{:.8f},{:.8f},{:.8f},{:.8f},{:.8f},{:.8f},{:.8f}\n".format(
                imu_ms, *acc_corr, *gyro_corr)
            sys.stdout.buffer.write(mensaje.encode())

        if t2 >= intervalo_gps:
            gps_ms = ahora
            while gps_serial.any():
                data = gps_serial.read()
                for byte in data:
                    stat = gps.update(chr(byte))
                    if stat is not None:
                        num_sats = gps.satellites_in_use
                        hdop = gps.hdop
                        lat = gps.latitude_string()
                        lon = gps.longitude_string()
                        speed = gps.speed_string()
                        direc = gps.compass_direction()
                        if lat != ultima_lat or lon != ultima_lon:
                            ultima_lat = lat
                            ultima_lon = lon
                            mensaje = "GPS,{},{:.8f},{:.8f},{},{},{},{}\n".format(gps_ms, lat, lon, num_sats, hdop, speed, direc)
                            sys.stdout.buffer.write(mensaje.encode())
                            intervalo = ahora
                        elif time.ticks_diff(ahora, intervalo) >= 800:
                            mensaje = "GPS,{},{:.8f},{:.8f},{},{},{},{}\n".format(gps_ms, ultima_lat, ultima_lon, num_sats, hdop, speed, direc)
                            sys.stdout.buffer.write(mensaje.encode())
                            intervalo = ahora
```

Anexo 11: Código para el modelo Random Forest.

```python
csv_files = [
    "C:/Users/LENOVO/ProyectMicropython/TR11combinado.csv",
    "C:/Users/LENOVO/ProyectMicropython/TR22combinado.csv",
    "C:/Users/LENOVO/ProyectMicropython/TR3combinado.csv",
    "C:/Users/LENOVO/ProyectMicropython/TR4combinado.csv",
    "C:/Users/LENOVO/ProyectMicropython/TR5combinado.csv",
    "C:/Users/LENOVO/ProyectMicropython/TR6combinado.csv",
    "C:/Users/LENOVO/ProyectMicropython/TR7combinado.csv",
    "C:/Users/LENOVO/ProyectMicropython/TR8combinado.csv",
    "C:/Users/LENOVO/ProyectMicropython/TR9combinado.csv",
    "C:/Users/LENOVO/ProyectMicropython/TR10combinado.csv",
    "C:/Users/LENOVO/ProyectMicropython/TR12combinado.csv"
]
dfs = []
for file in csv_files:
    df = pd.read_csv(file)
    if {"x_r", "y_r", "x_est", "y_est"}.issubset(df.columns):
        df["error_x"] = df["x_r"] - df["x_est"]
        df["error_y"] = df["y_r"] - df["y_est"]
        dfs.append(df)
combined_df = pd.concat(dfs, ignore_index=True)
print(f"\nDatos combinados: {combined_df.shape[0]} muestras totales")

features = ["x_est", "y_est", "v_est", "ort_est",
            "accel_imu", "gyro_imu", "v_gps", "ort_gps", "num_sat", "hdop"]
X = combined_df[features]
y = combined_df[["error_x", "error_y"]]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

rf = RandomForestRegressor(
    n_estimators=300,
    max_depth=20,
    min_samples_split=2,
    random_state=42,
)
rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)

rmse_x = np.sqrt(mean_squared_error(y_test["error_x"], y_pred[:, 0]))
rmse_y = np.sqrt(mean_squared_error(y_test["error_y"], y_pred[:, 1]))
print(f"RandomForest RMSE X: {rmse_x:.4f} m | RMSE Y: {rmse_y:.4f} m")


joblib.dump(rf, "OP1modelo_correccion.pkl")
print("\nModelo guardado como 'OPmodelo_correccion.pkl'")
```

Anexo 12: Código para el modelo LSTM.

```python
X_seq, y_seq = create_sequences(X_norm, y, sequence_length)

X_train_seq, X_test_seq, y_train_seq, y_test_seq = train_test_split(
    X_seq, y_seq, test_size=0.2, random_state=42
)

model = Sequential([
    LSTM(128, activation='tanh', return_sequences=True, input_shape=(sequence_length, X_seq.shape[2])),
    Dropout(0.2),
    LSTM(64, activation='tanh'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dense(2)
])

model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mse'
)

print("Entrenando modelo LSTM...")

early_stop = EarlyStopping(
    monitor='val_loss',
    patience=10,
    restore_best_weights=True
)

history = model.fit(
    X_train_seq, y_train_seq,
    epochs=150,
    batch_size=64,
    validation_split=0.1,
    callbacks=[early_stop]
)

y_pred_seq = model.predict(X_test_seq)

rmse_x = np.sqrt(np.mean((y_test_seq[:, 0] - y_pred_seq[:, 0]) ** 2))
rmse_y = np.sqrt(np.mean((y_test_seq[:, 1] - y_pred_seq[:, 1]) ** 2))

print(f"\nLSTM RMSE en X: {rmse_x:.4f} metros | RMSE en Y: {rmse_y:.4f} metros")

model.save("lstm1_correccion_ekf.keras")
np.savez("normalizadores1_lstm.npz", mean=X_mean, std=X_std)
print("Modelo y normalizadores guardados correctamente.")
```

Anexo 13: Código para el Filtro Extendido de Kalman.

```python
class ExtendedKalmanFilter:
    def predict(self, a_x, g_z, Q):
        F = np.eye(6)
        F[0, 2] = np.cos(theta_new) * self.dt
        F[0, 3] = -v_new * np.sin(theta_new) * self.dt
        F[1, 2] = np.sin(theta_new) * self.dt
        F[1, 3] = v_new * np.cos(theta_new) * self.dt
        F[3, 5] = -self.dt

        self.P = F @ self.P @ F.T + Q

    def update(self, z, R_base):

        H = np.array([
            [1, 0, 0, 0, 0, 0],
            [0, 1, 0, 0, 0, 0],
            [0, 0, 1, 0, 0, 0],
            [0, 0, 0, 1, 0, 0]
        ])

        y = z.reshape(4, 1) - H @ self.x

        S = H @ self.P @ H.T + R_base

        distancia = np.sqrt(y.T @ np.linalg.inv(S) @ y).item()
        umbral_M = 10.0
        if distancia > umbral_M:
            return

        innovation_norm = np.linalg.norm(y[:2])
        umbral = 10.0
        if innovation_norm > umbral:
            factor = (innovation_norm / umbral) ** 2
            R = R_base * factor
        else:
            R = R_base

        S = H @ self.P @ H.T + R

        K = self.P @ H.T @ np.linalg.inv(S)

        self.x = self.x + K @ y
        I = np.eye(6)
        self.P = (I - K @ H) @ self.P


    def get_state(self):
        return self.x.flatten()
```

Anexo 14: Código para el diseño de la página web.

```html
<!DOCTYPE html>
<html lang="es">
<head>
    <meta charset="UTF-8">
    <title>Posicionamiento Vehicular</title>
    <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
    <link rel="stylesheet" href="https://unpkg.com/leaflet/dist/leaflet.css" />
    <script src="https://unpkg.com/leaflet/dist/leaflet.js"></script>
    <style>
        body { margin: 0; font-family: Arial, sans-serif; background: #f0f2f5; }
        #container { display: flex; height: 100vh; }
        #map { flex: 1; }
        #trayectoria { width: 50%; height: 100vh; display: none; background: white; }
        #btnTrayectoria { position: absolute; top: 80px; left: 10px; z-index: 1000; padding: 10px 20px; background: white; color: black; cursor: p
    </style>
</head>
<body>

<button id="btnTrayectoria">2D</button>
<div id="container">
    <div id="map"></div>
    <div id="trayectoria">
        <div id="plot" style="width:100vh; height:100vh;"></div>
    </div>
</div>

<script>
var map = L.map('map').setView([0.0, 0.0], 15);
    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
        maxZoom: 20,
        attribution: '© OpenStreetMap contributors'
    }).addTo(map);

    var polylines = {
        gps: L.polyline([], {color: 'black'}).addTo(map),
        ekf: L.polyline([], {color: 'red'}).addTo(map),
        rf: L.polyline([], {color: 'blue'}).addTo(map),
        lstm: L.polyline([], {color: 'green'}).addTo(map)
    };
    var traceData = [
        {x: [], y: [], mode: 'markers', marker: {color: 'black', symbol: "circle"}, name: 'GPS'},
        {x: [], y: [], mode: 'markers', marker: {color: 'red', symbol: "circle"}, name: 'EKF'},
        {x: [], y: [], mode: 'markers', marker: {color: 'blue', symbol: "circle"}, name: 'RF'},
        {x: [], y: [], mode: 'markers', marker: {color: 'green', symbol: "circle"}, name: 'LSTM'}
    ];
    var layout = {margin: {l: 50, r: 50, t: 50, b: 50}, title:"Trayectoria", xaxis:{title:'X (m)'}, yaxis:{title:'Y (m)'}};
    Plotly.newPlot('plot', traceData, layout);
    var centrado = true;
```

Anexo 15: Código principal del sistema.

```
if posicion[0] == 'IMU':
    #if len(posicion) < 8:
        #continue
    try:
        timestamp = float(posicion[1]) / 1000.0
        accel_imu = float(posicion[2])
        ay = float(posicion[3])
        az = float(posicion[4])
        gx = float(posicion[5])
        gy = float(posicion[6])
        gyro_imu = float(posicion[7])
    except ValueError:
        continue

    if lt_imu is None:
        lt_imu = timestamp
        continue

    dt_imu = timestamp - lt_imu
    """if not (0.001 < dt_imu < 0.2):
        #ekf.dt = 0.0
        lt_imu = timestamp
        continue"""

    ekf.dt = dt_imu
    lt_imu = timestamp

    alpha_vel = normalize(abs(accel_imu), 0, 3.0)
    alpha_gyro = normalize(abs(gyro_imu), 0, 2.0)
    alpha_Q = 0.6*alpha_vel + 0.4*alpha_gyro

    Q_dyn = Q_min + (Q_max - Q_min) * alpha_Q

    ekf.predict(accel_imu, gyro_imu, Q_dyn)
```

```
elif posicion[0] == 'GPS':
    if len(posicion) < 8:
        continue
    try:
        timestamp = float(posicion[1]) / 1000.0
        lat = float(posicion[2])
        lon = float(posicion[3])
        num_sat= float(posicion[4])
        hdop = float(posicion[5])
        v_gps = float(posicion[6])
        ort_gps = float(posicion[7])
    except ValueError:
        continue

    if abs(lat) < 0.1 or abs(lat) < 0.1:
        continue

    if lat0 is None:
        lat0, lon0 = lat, lon

    x_gps, y_gps = transformar_xy(lat, lon, lat0, lon0)
    yaw_gps = normalizar_angulo(ort_gps)

    alpha_hdop = normalize(hdop, 0.5, 5.0)
    alpha_vel = normalize(v_gps, 0, 25)

    if last_yaw is not None:
        delta_theta = normalizar_angulo(yaw_gps - last_yaw)
    else:
        delta_theta = 0.0
    last_yaw = yaw_gps
    alpha_theta = normalize(abs(delta_theta), 0, np.pi/2)

    peso_hdop, peso_vel, peso_theta = 0.5, 0.3, 0.2
    alpha_total = peso_hdop*alpha_hdop + peso_vel*alpha_vel + peso_theta*alpha_theta

    R_dyn = R_min + (R_max - R_min) * alpha_total

    if v_gps >= 0:
        z_gps = np.array([x_gps, y_gps, v_gps, yaw_gps])
        ekf.update(z_gps, R_dyn)
    else:
        puntos_malos_x.append(x_gps)
        puntos_malos_y.append(y_gps)

    x_est, y_est, v_est, ort_est,_,_ = ekf.get_state()
    lat_est, lon_est = transformar_latlon(x_est, y_est, lat0, lon0)
```