UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA MECÁNICA

DESARROLLO DE UN GEMELO DIGITAL DE NISSAN LEAF ELÉCTRICO BASADO EN MODELOS Y DATOS OPERACIONALES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL MECÁNICO

CLAUDIO ANDRÉS LABRIN ESPINOZA

PROFESOR GUÍA:
WILLIAMS RODRIGO CALDERON MUÑOZ

MIEMBROS DE LA COMISIÓN:
JORGE REYES MARAMBIO
PAULINA RAMIREZ DEL BARRIO

SANTIAGO DE CHILE
2021

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE: Ingeniero Civil Mecánico
POR: Claudio Andrés Labrin Espinoza
FECHA: 07/06/2021
PROFESOR GUÍA: Williams Rodrigo Calderón Muñoz

DESARROLLO DE UN GEMELO DIGITAL DE NISSAN LEAF ELÉCTRICO BASADO EN MODELOS Y DATOS OPERACIONALES

En el último tiempo tanto la electro movilidad como las herramientas digitales que trae la Cuarta Revolución Industrial han crecido enormemente aumentando el interés de diversos grupos industriales y económicos que, tratando de combatir los efectos del cambio climático por medio del reemplazo de vehículos de motores térmicos por vehículos eléctricos es que, se crea la necesidad de tener equipos cada vez más competitivos tanto en performance, durabilidad e impacto ambiental. Para ello es necesario llevar un estricto seguimiento de los vehículos y sus componentes para así estudiar de mejor manera el cómo mejorarlos y mantenerlos. Por lo que nace la idea de crear un gemelo digital de un vehículo eléctrico para extraer datos de vital importancia para el cuidado de este y el uso eficiente de recursos. Dicho esto, el objetivo principal de este trabajo es desarrollar un modelo electromecánico integral de un Nissan Leaf 2018 en base a múltiples modelos y datos operacionales para obtener consumo energético y datos o cifras relevantes para el cuidado de la batería y que sirva como punto de partida o que pueda ser integrado en un gemelo digital.

El modelo o programa realizado se basa en 3 modelos: Modelo dinámico longitudinal, modelo termodinámico equivalente en base a la perdida de potencia y el modelo OCV-SOC de circuito equivalente. Este programa puede ser utilizado de dos formas: Ingresando datos de GPS o aplicando un ruteador. Para el primer caso, además de agregar los datos de GPS se debe ingresar la temperatura inicial de la batería y la temperatura ambiente para que el programa entregue datos de consumo energético (potencia, SOC, energía y rendimiento), temperatura de batería y voltaje de batería en cada instante de medición mientras que para el segundo caso se deben ingresar las coordenadas de inicio y fin de un viaje, la temperatura ambiente y la temperatura inicial para que el programa entregue dos rutas alternativas una con el menor tiempo de llegada y otra con la menor distancia recorrida y en ambas entregará los datos de consumo energético, la temperatura de batería y el voltaje para cada ruta.

Para este trabajo se utilizó el software de Python junto con sus diferentes bibliotecas y librerías, los datos obtenidos fueron comparados con datos extraídos de la ECU del vehículo en viajes ya realizados dando diferencias o errores menores a un 20% en todos los parámetros medidos (SOC, temperatura y voltaje). Estas diferencias pueden tener múltiples factores que pueden ser minimizados con un hardware más sofisticado y con un ruteador con datos de tráfico en vivo.

Finalmente se concluye que se cumplen los objetivos y que el programa entrega datos fidedignos que aportarán en gran medida al cuidado del vehículo y que el desafío radica en crear un programa que utilice el modelo descrito en el trabajo y que cuente con una actualización dinámica y activa de datos, dicho de otro modo, realizar el gemelo digital del Nissan Leaf en base a este modelo electromecánico.

Agradecimientos

"Si veo más lejos es porque he subido a hombros de gigantes"

Quiero agradecer a todos los que se involucraron conmigo a lo largo de toda mi carrera académica y sobre todo en la etapa universitaria en especial a mi familia mis padres Armando Labrin y Norma Espinoza, a mis hermanos Diego y Benjamín; a todos mis tíos en especial a mi tía Mónica y mi tío Lolo; a mis abuelos tata Pancho, mami Chela, abuela Isabel y tata Armando (Q.E.P.D.); a mi novia Visnja Covacich; a mis primos; a mis amigos que tengo del colegio y de toda la vida; a los amigos que hice en la universidad por la sección, la carrera, la biblioteca y el futbol; a todos y cada uno de los profesores que he tenido a lo largo de mi vida; a mis compañeros y amigos del trabajo y a toda la institución Universidad de Chile. Finalmente quiero agradecer y destacar la paciencia y el apoyo que tuvieron y brindaron mis profesores guía y coguía Williams Calderón y Jorge reyes y la Secretaria Docente del departamento de ingeniería mecánica Claudia Villareal.

Sin su apoyo nada de esto y nada de lo que venga sería posible, gracias totales.

Tabla de contenido

Índice de tablas

Índice de ilustraciones

1 Introducción

Hoy en día, debido al impacto ambiental que ha generado la humanidad a lo largo de la historia, el fuerte crecimiento de la economía, la industria y la población mundial, es de vital importancia tener un desarrollo sustentable y ecológico. Bajo esta premisa es que se ha optado, durante este último tiempo, por el desarrollo de vehículos eléctricos que reemplacen a los vehículos de combustión, los cuales emiten una fracción importante de gases nocivos y de efecto invernadero.

Por otro lado, muchas veces la manufactura de prototipos, la ejecución de pruebas y el chequeo de salud de un dispositivo o sistema genera, también, un impacto ambiental y un gasto económico considerable, sin embargo, con la tecnología actual es posible replicar estos procesos por medio de herramientas digitales obteniendo resultados confiables, los cuales pueden validarse por medio de diferentes pruebas de fiabilidad. Un gemelo digital, consiste en una representación dinámica y continuamente actualizada de algún producto, sistema o proceso, lo que permite analizar en tiempo real el comportamiento, la condición de su gemelo físico y hacer proyecciones de rendimiento y desgaste de los diferentes partes o componentes de su análogo real.

De acuerdo con lo descrito es que este trabajo titulado Desarrollo de un Gemelo Digital de Nissan Leaf Eléctrico basado en modelos y datos operacionales, consiste en el desarrollo de un programa con 3 modelos electromecánicos y un ruteador para un vehículo eléctrico, específicamente para un Nissan Leaf, el cual resulta ser uno de los vehículos eléctricos de mayor circulación en Chile y el mundo. El objetivo principal de este trabajo es ser el punto de partida para la construcción e implementación de un gemelo digital, integrando los modelos de: Consumo energético basado en la potencia, termodinámico basado en la potencia y de voltaje basado en SOC y temperatura utilizados en este trabajo.

1.1 Motivación

Las principales razones para realizar el trabajo son, el auge de la electro movilidad como medida combativa contra el problema medioambiental que se vive hoy en día, la vulgarización del conocimiento en general y aportar al bienestar de los conductores de vehículos eléctricos en general.

Finalmente, la motivación principal y especifica es que se pueda tener un modelo confiable que entregue información suficiente como para hacer una gestión más eficiente y eficaz con respecto al consumo energético y el estado de la batería del vehículo eléctrico, de esta forma se tendrá un consumo adecuado del recurso lo que a su vez puede traducirse en un mayor cuidado para la batería del vehículo y, a una pequeña escala, para el medioambiente. Así, por ejemplo, dependiendo del contexto, se podrá elegir entre una ruta más larga que tarda menos tiempo o una más corta en distancia, pero con un tiempo estimado de llegada mayor, en base al consumo energético y/o la temperatura que podría llegar a alcanzar la batería.

1.2 Objetivos

1.2.1 Objetivo general

Desarrollar modelo electromecánico que podría ser integrado en un gemelo digital de un Nissan Leaf eléctrico, para obtener información del consumo eléctrico y datos relevantes de la batería del vehículo en base a modelos y datos operacionales.

1.2.2 Objetivos específicos

Determinar datos operacionales relevantes del vehículo.

Realizar modelo especifico de batería que simulen la potencia consumida o requerida por el vehículo en base a sus características y datos operacionales

Realizar modelos específicos de batería que simulen la temperatura y el voltaje de esta en base a sus características y datos operacionales

Integrar los modelos para determinar datos de utilidad, específicamente SOC, potencia, temperatura y voltaje de batería, conforme a los datos operacionales del vehículo y datos geográficos a través de simulación computacional.

Analizar datos para la verificación y validación de capacidad del modelo.

Proyectar rutas alternativas indicando datos relevantes de la ruta y del vehículo para aportar de manera activa en la elección de una u otra.

1.3 Alcances

- El trabajo de título consiste en la realización de un programa que integre distintos modelos que puedan ser utilizados en un gemelo digital de un vehículo eléctrico, específicamente un Nissan Leaf del año 2018
- Modelo de consumo de energía del vehículo basado en la potencia mecánica utilizada o requerida
- Modelo termodinámico basado en las perdidas de energía de la batería y sus características físicas
- Modelo de tensión de circuito abierto OCV (sigla en inglés de Open Circuit Voltage) no lineal basado en SOC y temperatura
- La implementación y validación del modelo se limitará a datos entregados por el Centro de Energía, extraídos de un Nissan Leaf, la operación de este vehículo está integrada en el proyecto de 5000 kilómetros eléctricos de la Agencia de Sostenibilidad Energética (ASE). Estos datos son extraídos a través de un smartphone y un dispositivo de diagnóstico, este dispositivo es conectado al puerto de diagnóstico (OBD2) y extrae datos directamente desde la ECU (sigla en inglés de Engine Control Unit) del vehículo los cuales provienen de los distintos sensores con los que cuenta el vehículo y son procesados y luego descargados a través de la aplicación Leaf Spy Pro, cabe hacer notar que los datos que no puede generar el vehículo, son entregados por los sensores o la base de datos del teléfono que tiene la App
- La única condición climática que se toma en cuenta es la temperatura ambiente y se considera constante a lo largo de un mismo viaje
- No se toman en cuenta los procesos de carga
- El programa consta de dos partes: Datos de viajes descargados y proyecciones de futuros viajes en mapa. Esta última se limita a viajes dentro de una misma región

2 Antecedentes y discusión bibliográfica

2.1 Gemelo digital

El término "gemelo digital" (DT: digital twin) fue acuñado por Michael Grieves en la Universidad de Michigan en el año 2011 y se refiere a un conjunto de construcciones de información virtual que describe completamente un producto manufacturado físico potencial o real desde el nivel micro atómico hasta el nivel macro geométrico [1]. A diferencia de otros tipos de modelación, los gemelos digitales exigen una representación dinámica y continuamente actualizada del producto, dispositivo, proceso o sistema, es decir, la transferencia de información entre el modelo y el dispositivo es mucho más estricta y, muchas veces, en tiempo real, impulsada por el contexto. De esta forma se tiene una alta fidelidad entre la parte real y virtual. De hecho, en su punto óptimo, cualquier información que pueda obtenerse de la inspección de un producto manufacturado físico puede obtenerse de su gemelo digital. La figura 1 presenta el concepto de gemelo digital en el contexto de la ingeniería de sistemas basada en modelos (MBSE).

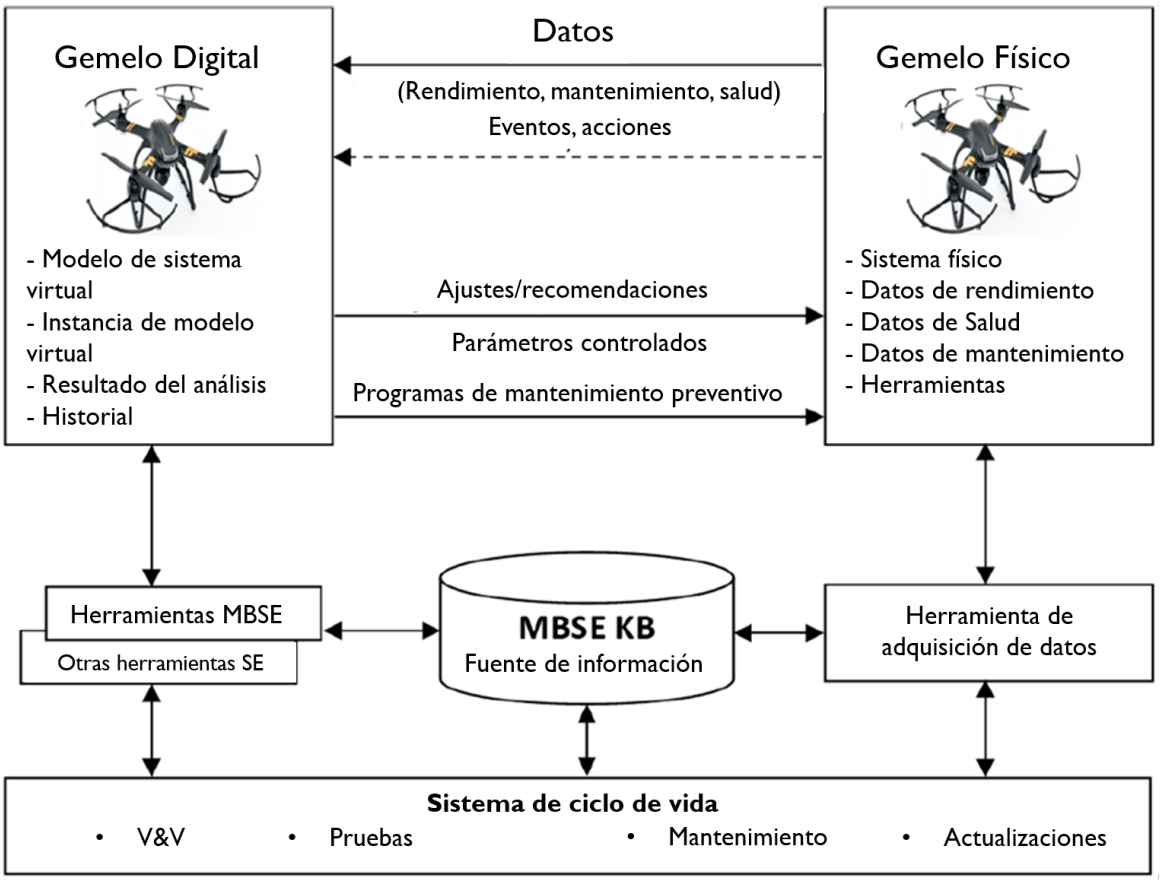

Ilustración 2.1 Esquema de gemelo digital [2]

Los gemelos digitales pueden ser de dos tipos: Prototipo de gemelo digital (DTP) e instancia de gemelo digital (DTI). Ambos deben ser operados en un Ambiente de Gemelo Digital (DTE).

Prototipo de gemelo digital (DTP): Gemelo digital que describe el artefacto físico prototípico. Contiene los conjuntos de información necesarios para describir y producir una versión física que duplique o gemele la versión virtual. Estos conjuntos de información podrían incluir: Modelo 3D completamente detallado, lista de materiales, lista de procesos, factura de servicios y factura de disposición.

Instancia de gemelo digital (DTI): Gemelo digital que describe un producto físico específico ya fabricado correspondiente al que el gemelo digital permanece vinculado durante toda la vida útil de este. Dependiendo de los casos de uso que se requieran para ello, este tipo de DT puede contener: Un modelo 3D completamente anotado con dimensiones y tolerancias generales (GDT) que describe la geometría de la instancia física y sus componentes, una lista de materiales que enumera los componentes actuales y todos los componentes anteriores, una lista de procesos que enumera las operaciones que se realizaron para crear esta instancia física junto con los resultados de las mediciones y pruebas de la instancia, un registro de servicio que describe los servicios prestados en el pasado y los componentes sustituidos y los estados operacionales capturados de los datos de los sensores pasados y actuales. Este tipo de DT se utiliza en general para ejecutar o proyectar pruebas en diferentes escenarios.

Agregado de gemelo digital (DTA): Es la agregación de todos los DTI. A diferencia del DTI, el DTA puede no ser una estructura de datos independiente. Puede ser una construcción de computadora que tiene acceso a todos los DTIs y los consulta ya sea ad-hoc o pro-activamente.

Entorno de gemelos digitales (DTE): Este es un espacio integrado de aplicaciones de física de dominio múltiple para operar un DT para una variedad de propósitos. Como lo son, por ejemplo: Predicción e interrogación.

Es importante mencionar que aprovechar la tecnología de los gemelos digitales para facilitar la evaluación del rendimiento del sistema y los riesgos técnicos, se logra al mismo tiempo una mejora de aproximadamente el 10 % en la eficacia del sistema [3].

Principales usos de un gemelo digital:

Validar el modelo del sistema con datos del mundo real: Los datos del entorno operativo y las interacciones del sistema con ese entorno pueden incorporarse al gemelo digital para validar sus modelos, hacer evaluaciones y predicciones.

Proporcionar apoyo a las decisiones y alertas a los usuarios: Después de incorporar los datos operacionales, de mantenimiento o de salud, el gemelo digital puede emplearse en un modo de análisis hipotético para producir información de apoyo a las decisiones y alertas adaptadas a los operadores/usuarios del sistema físico.

Predecir los cambios en el sistema físico a lo largo del tiempo: El análisis basado en la simulación de los datos operacionales, de mantenimiento y de salud del gemelo físico puede facilitarla optimización de las operaciones (incluida la satisfacción de las necesidades y la identificación de las causas fundamentales), mejorar la planificación para imprevistos y predecir el rendimiento del sistema. El gemelo digital también puede incorporarse al bucle de control para predecir los cambios en el sistema físico y ajustar/modificar los parámetros del sistema físico para hacer frente a las contingencias

Descubrir nuevas oportunidades de aplicación y flujos de ingresos: Con un gemelo digital, se pueden evaluar diferentes versiones de los sistemas para determinar qué configuraciones proporcionan mejores características costo-efectivas. El aprendizaje automático y otras tecnologías de ciencia de los datos pueden facilitar el análisis oportuno de importantes volúmenes de datos generados, proporcionando así una visión de los posibles nuevos usos y corrientes de ingresos.

2.2 Electro movilidad

La electro movilidad hace referencia al uso de medios de transporte que son impulsados por uno o más motores eléctricos. Existen cuatro principales tecnologías impulsadas por este tipo de máquinas y tienen la particularidad de emitir nula o bajas emisiones de sustancias contaminantes en comparación a sus homólogos impulsados íntegramente por un motor térmico de combustión

2.2.1 Tipos de vehículos eléctricos

El primer vehículo eléctrico fue construido por Gustave Trouvé en 1881, consistía en un triciclo con una batería de plomo-ácido que podía alcanzar velocidades de 15 [km/h] y una autonomía de 16[km]. En 1984 se crea el primer vehículo eléctrico comercial, que fue utilizado como taxi en Filadelfia, Pensilvania; éste tenía una autonomía de 40 [km] y alcanzaba velocidades de hasta 32[km/h] con dos motores de 1.5 [HP].

A pesar de que hubo un desarrollo de vehículos eléctricos a finales del siglo XIX, éstos empezaron a desaparecer y comienzan a ganar terreno los vehículos impulsados por gasolina con motores de combustión interna. Principalmente por los altos costos de producción de vehículos eléctricos y que sus pares a gasolina tenían una mucho mayor autonomía con el combustible, hoy en día el panorama es distinto gracias al desarrollo tecnológico, la electro movilidad está en crecimiento y hoy existen 4 grandes categorías de vehículos eléctricos:

Eléctricos Puros (BEV): Presentan uno o más motores eléctricos alimentados exclusivamente por una batería, generalmente de ion-litio, la cual puede cargarse en regímenes de carga lenta y rápida, dependiendo de la potencia suministrada. También pueden contar con sistemas de recuperación de energía.

Híbrido enchufable (PHEV): Dualidad entre un motor de combustión y uno o más motores eléctricos. Es posible cargar la batería a través de un conector externo además de los sistemas de recuperación.

Híbrido (HEV): Dualidad entre un motor térmico y uno o más motores eléctricos. La batería que alimenta el o los motores eléctricos es cargada cuando el vehículo se encuentra en movimiento, en particular en regímenes de regeneración. No posee un conector externo para cargar la batería como es el caso del híbrido enchufable.

Celdas de combustible (FVC): Este tipo de tecnologías utilizan el hidrógeno como fuente de energía para alimentar el motor eléctrico. Este elemento actúa como carburante en una reacción química en la que el hidrógeno se oxida y cede electrones, que son la corriente eléctrica que circula a través de la pila de combustible que posteriormente alimenta el o los motores eléctricos.

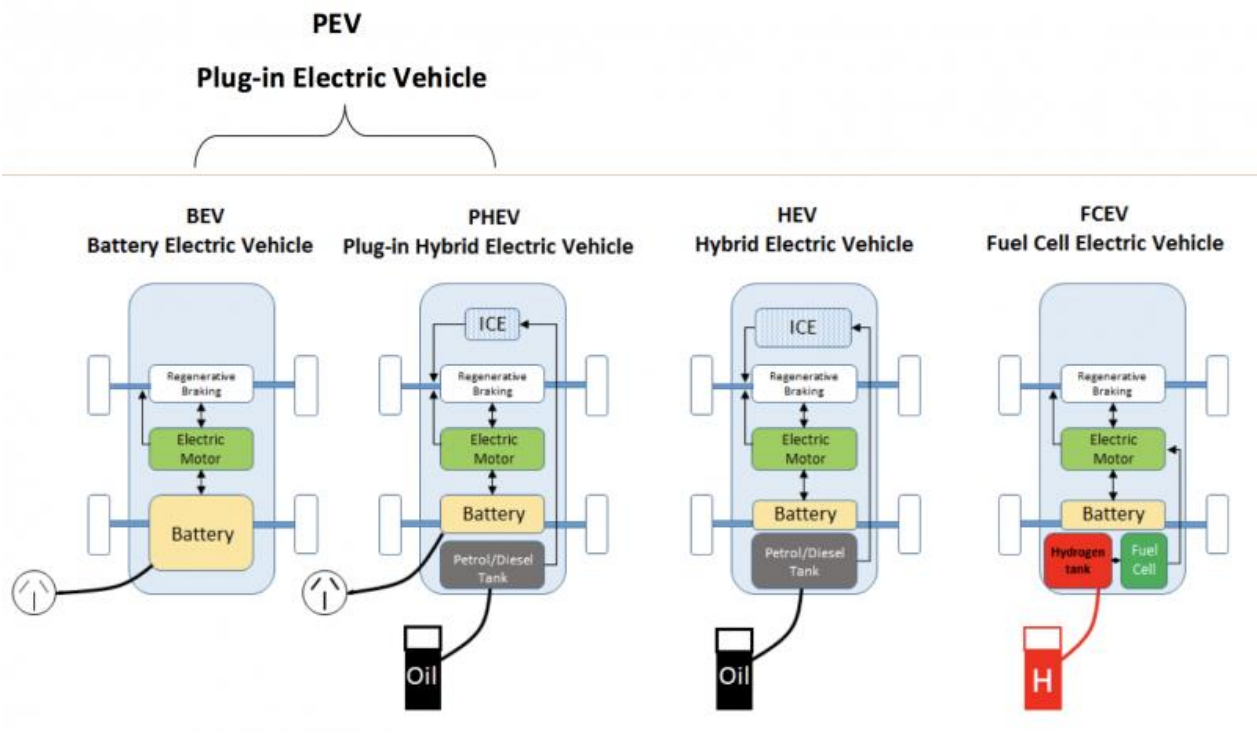

Ilustración 2.2 Tipos de vehiculos electricos

2.2.1 Partes de un vehículo eléctrico puro

A continuación, se definen las partes más importantes y que distinguen a un auto eléctrico de un automóvil convencional.

Cargador: Esta parte del vehículo es la encargada de absorber la electricidad de forma alterna directamente desde la red externa y la transforma en corriente continua, la cual almacena en la batería. Existen cargadores de carga rápida y de carga lenta, en función de la velocidad de carga y el tiempo que tarda en cargar la batería del coche eléctrico

Batería: Es el sistema de almacenamiento de la energía destinada a alimentar al motor eléctrico, ya sea conectada directamente o mediante un inversor

Inversor: Son los encargados de transformar la corriente continua en corriente alterna, cuando se trata motores de corriente alterna.

Motor eléctrico: El motor de un coche eléctrico puede ser de corriente alterna o corriente continua. Un vehículo eléctrico puede tener uno o varios motores eléctricos, dependiendo de su diseño y las prestaciones que se quieran conseguir. Pueden ser síncrono o asíncrono

Controlador: Son los encargados de comprobar el correcto funcionamiento por eficiencia y seguridad y regulan la energía que recibe o recarga el motor

2.2.2 Motor eléctrico

El motor eléctrico es un dispositivo que convierte la energía eléctrica en energía mecánica de rotación por medio de la acción de los campos magnéticos. Son máquinas eléctricas rotatorias compuestas por un estator, un rotor y su carcasa. De él depende la eficiencia, la autonomía y las prestaciones. En el mercado existen diferentes tipos de motores eléctricos. El estátor es la parte fija de la maquina rotativa, pudiendo ser desde electroimanes hasta chapas magnéticas, que acoge en su interior al rotor, la parte móvil. Todo ello es envuelto por la carcasa metálica. Según su alimentación mediante corriente alterna o continua y su arquitectura, se pueden dividir en las siguientes categorías:

Motor Asíncrono o de inducción AC: Su principal característica es que el giro del rotor no corresponde a la velocidad de giro del campo magnético producido por el estátor. Este motor está formado por un rotor que puede ser de tipo jaula de ardilla o bobinado. En el estátor (anillo cilíndrico de chapa magnética) se encuentran las bobinas inductoras que son trifásicas, desfasadas entre sí a 120º. Entre las ventajas encontramos la alta eficiencia, coste bajo, fiabilidad, bajo ruido y vibraciones y par constante. En cambio, sus contras son su baja densidad de potencia, el bajo par en el arranque y el riesgo de sobrecarga. Es uno de los motores más utilizados en la industria del VE.

Motor Síncrono de imanes permanentes AC: Con una velocidad de giro constante, siendo igual el giro del rotor que la velocidad del campo magnético creado por el estátor, el motor síncrono de imanes permanentes puede ser de dos tipos; de flujo radial o de flujo axial, dependiendo de la posición del campo magnético de inducción, que puede ser perpendicular o paralelo al eje de giro del rotor. Son más usados los de flujo radial. En cambio, los de flujo axial permiten ser integrados directamente en la rueda del vehículo, optimizando el espacio en el vehículo y simplificando los acoplamientos mecánicos entre motor y rueda, son los conocidos como «in-wheel motor». Las ventajas de este tipo de motor son su alto rendimiento, un control de velocidad sencillo, bajo ruido, vibración, tamaño y peso. Aunque tienen un alto coste, junto con los motores asíncronos, son los más extendidos dentro de los VE e híbridos.

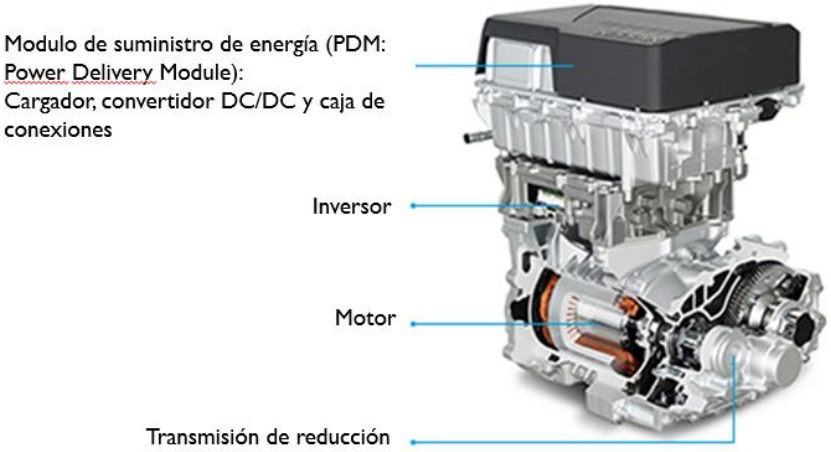

Ilustración 2.3 Motor de Nissan Leaf

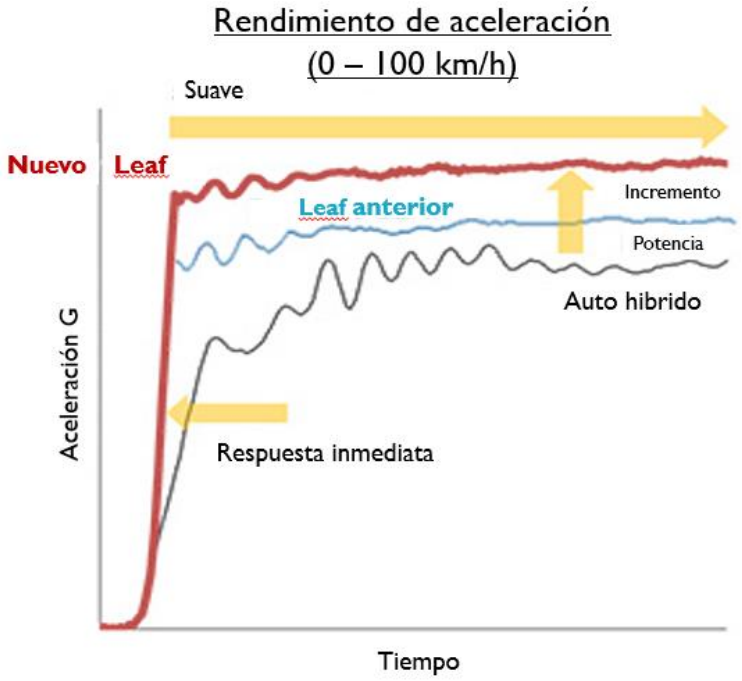

Ilustración 2.4 Perdormance de aceleración de motor de Nissan leaf

Motor Síncrono de reluctancia conmutada o variable AC: La corriente es conmutada entre las bobinas de cada fase del estátor hasta crear un campo magnético que gira. El rotor, que está hecho con un material magnético con polos salientes, son influenciados por el campo magnético, atrayéndose y creando un par que mantiene el rotor moviéndose a velocidad síncrona. Estos

motores no necesitan imanes permanentes ni escobillas, y tienen a favor su elevado par, robustez y bajo coste, mientras que en contra tiene su baja potencia y la complejidad de su diseño.

Motor sin escobillas de imanes permanentes DC: Conocidos como «brushless», estos motores poseen imanes permanentes situados en el rotor que funcionan mediante la alimentación secuencial de cada una de las fases del estátor. Pueden ser «inrunner», mayor velocidad de giro y menor par, o «outrunner» menor velocidad y mayor par. Aunque son usados mayormente en vehículos híbridos, los motores «brushless» ofrecen algunas ventajas para su uso en VE, su bajo ruido y rozamiento, robustez y ausencia de mantenimiento. Por ahora son motores poco experimentados, que tienen un precio elevado y poca potencia.

2.2.3 Baterías

La batería de un coche eléctrico es un dispositivo acumulador de energía (ESD, del inglés Energy Storage Device), que transforma en energía química la electricidad aportada en una carga (o a través del sistema de regeneración de energía), para liberarla más tarde, de nuevo como electricidad.

Las baterías, en general, están compuestas por una combinación de celdas las cuales se disponen en serio o paralelo, dependiendo del voltaje específico de salida o de la corriente. De esta forma, generalmente una batería es un conjunto de celdas dispuestas de forma inteligente para dar solución a algún desafío presentado. Sin embargo, en el presente trabajo se mencionará ambos conceptos como idénticos y si es necesario se especificará cuando se mencione un conjunto de estas. La batería es un componente clave en un coche eléctrico, ya que determina la autonomía, la entrega de energía al motor y afecta de forma notable al peso y al diseño del vehículo. Además, es la parte principal de un sofisticado sistema que además de las propias celdas, incluye su contenedor, refrigeración, cableado y gestión electrónica.

Cabe destacar que las baterías que montan los coches eléctricos actuales son muy diferentes de las que empleaban los primeros modelos eléctricos hace un siglo. Las baterías pueden clasificarse en dos tipos, las del tipo primarias y secundarias. La diferencia entre ambas es que las del tipo primarias, debido a su naturaleza no pueden ser recargadas cuando su energía se agota y las del tipo secundarias si pueden ser cargadas. También se pueden clasificar según los materiales con los cuales se fabrican y los más destacados son: **Plomo-ácido, Níquel-Cadmio, Níquel-hidruro metálico e Ion-Litio**.

Batería de Ion-Litio: Una batería de Ion-Litio es un ESD de tipo secundaria, el cual transforma la energía química almacenada en su estructura, en energía eléctrica a través de una reacción de óxido-reducción. La reacción química producida es reversible, en la cual las distintas partes de la batería, conocidas como, ánodo, cátodo y electrolito, tienen diferentes funciones que permiten el flujo de iones y electrones.

Acerca de los componentes principales de la batería antes mencionados, se tiene que el ánodo o electrodo negativo, es el elemento reductor, por lo tanto, entrega electrones al circuito externo y se oxida en la reacción electroquímica. En cambio, el cátodo o electrodo positivo, es el elemento oxidante, de esta forma absorbe los electrones del circuito externo y se reduce en la reacción electroquímica. Por otra parte, el electrolito, es el conductor iónico, el cual provee de un medio para la transferencia de carga, como iones, entre el ánodo y el cátodo. En la Figura, se muestran dos esquemas relativos al funcionamiento de un batería en sus procesos de descarga y carga respectivamente.

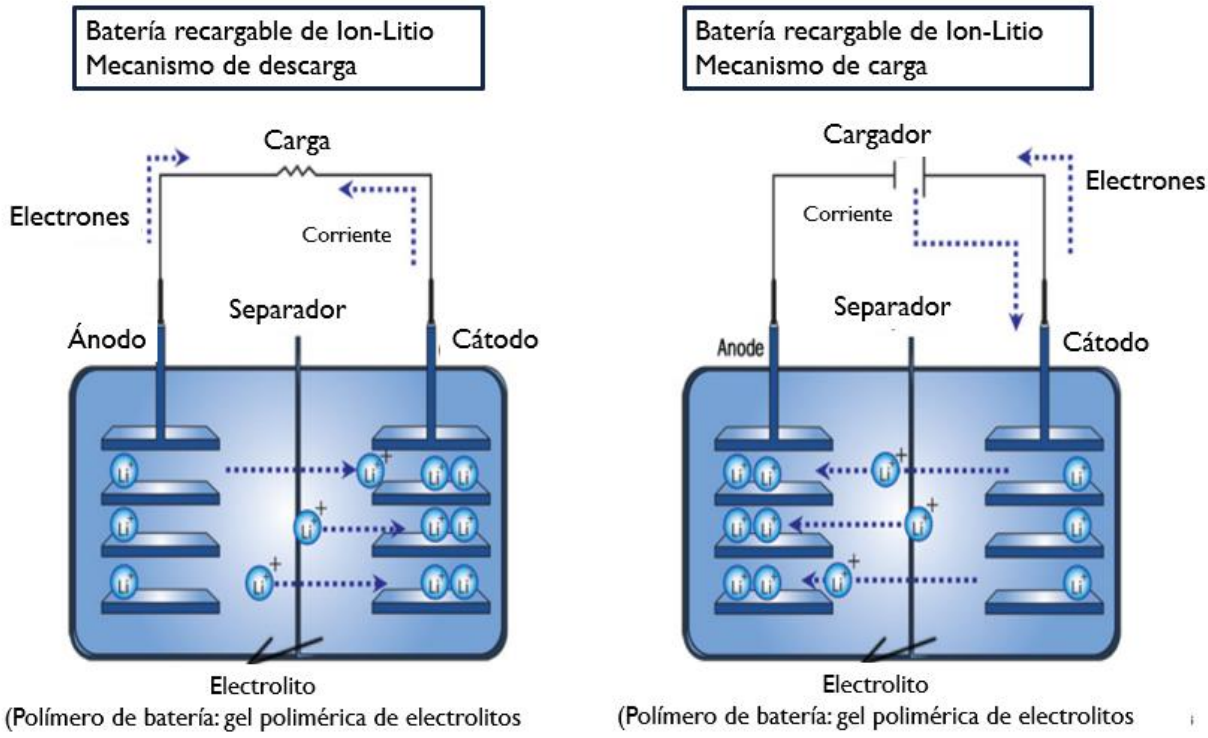

Ilustración 2.5 Carga y descarga de batería de litio

2.2.4 Degradación de batería: Conceptos básicos

Estado de Carga (en inglés, State of Charge, SoC): El estado de carga es un indicador de la cantidad de carga remanente en la batería en relación con la carga almacenada cuando la batería se encuentra completamente cargada. A diferencia de otros sistemas en los cuales se puede medir directamente la cantidad de combustible para determinar la energía total almacenada del sistema, en una batería de ion-litio la energía química almacenada no puede ser obtenida directamente, sino que debe ser estimada a partir de otros valores. Existen diferentes tipos de métodos que intentan

estimar el estado de carga de una batería de ion-litio como, por ejemplo: Método químico, Medida de gravedad específica, impedancia interna, contador de coulomb, estimación basada en voltaje e intensidad, Filtrado de Kalman y método de presión entre otros.

Estado de Salud (en inglés, State of Health, SoH: El estado de salud es un indicador de la capacidad máxima de carga de una batería bajo condiciones de carga y descarga nominales. Generalmente dicho indicador se compara respecto a la capacidad máxima de la misma batería cuando está nueva, bajo condiciones de carga y descarga nominales. Es por esto, que generalmente el estado de salud se expresa como un porcentaje. Existen variados métodos de estimación del estado de salud de baterías de ion-litio como, por ejemplo: Método de Resistencia-impedancia-conductancia interna, midiendo capacidad de batería y el voltaje, estudiando la auto descarga, número de ciclos o la edad de la batería entre otros.

C-rate: El C-rate, corresponde a la tasa a la cual una batería es cargada o descargada relativa a su capacidad nominal. Se define como la corriente a través de la batería dividida por el consumo de corriente teórico bajo el cual la batería entregaría su capacidad nominal en una hora. Se conoce que varios factores afectan en la degradación de las baterías de ion-litio, tales como la sobrecarga, el conservar las baterías en un elevado estado de carga por un prolongado tiempo, descargas profundas y la temperatura, entre otros. Conjuntamente, se conoce que el C-rate es uno de los factores fundamentales en la degradación de las baterías.

Profundidad de Descarga (en inglés, Depth of Discharge, DoD: La profundidad de descarga, es el porcentaje de la capacidad de la batería que ha sido descargada respecto de su máximo.

Resistencia Interna: La resistencia interna, corresponde a la resistencia eléctrica total dentro de una batería. Es diferente al cargar y descargar, además de cambiar según las condiciones de operación de la batería

Protocolo de Carga, Corriente Constante - Voltaje Constante (en inglés, Constant Current - Constant Voltage CC-CV): La carga de las baterías de ion-litio, comúnmente es realizada en dos fases, utilizando el protocolo CC-CV. El cual consiste en una primera fase de carga a corriente constante, generalmente establecida a 0.5C, y se mantiene esta fase hasta que el voltaje de la batería alcance su voltaje máximo, con lo cual se marca el inicio de la segunda fase. Esta segunda fase se realiza a voltaje constante, es decir, se mantiene el voltaje máximo de la batería ya alcanzado y la corriente comienza a descender hasta un umbral de corriente mínimo, en el cual se establece que la batería ya se encuentra completamente cargada. La corriente y el método de carga tiene un gran impacto en el tiempo de vida de una batería de ion-litio. Altas tasas de corriente de carga aceleran considerablemente la degradación de la batería

Inicio de vida (en inglés, Beginning of Life, BoL): Corresponde al instante inicial en el que una batería comienza a ser utilizada por primera vez. De esta forma se determinan las condiciones de resistencia inicial, capacidad de carga inicial, que pasan a ser una referencia a medida que la batería va envejeciendo.

Fin de vida (en inglés, End of Life (BoL): corresponde al tiempo final de la vida de una batería. Este momento se determina generalmente cuando al realizar una descarga a 1C, la capacidad de carga total entregada es de un 80 % o menos, respecto de la nominal

2.3 Vehículo eléctrico puro: Nissan Leaf

2.3.1 Características generales de Nissan Leaf

El vehículo eléctrico a modelar es un Nissan Leaf 2018 como el que se muestra a continuación.

Ilustración 2.6 Nissan Leaf 2018

Este vehículo es uno de los vehículos particulares eléctricos puros más populares en Chile y el mundo. Cuenta con una batería de ion-Litio laminada y en la siguiente tabla se muestran los datos de interés de este vehículo, extraídos de la ficha técnica.

Tabla 2.1 Ficha técnicas Nissan Leaf

Nissan Leaf 2018	
Dato	**Especificación**
Tipo de motor	Síncrono de imanes permanentes
Potencia de motor	80 [kW] (109 [CV])
Par motor	254 [Nm]
Tipo de batería	Ion-Litio
Energía de batería	40 [kWh]
Rendimiento	5.5 [km/kWh] (180 [kWh/100[km]])
Autonomía nominal	240 [km]
Masa	1580 [kg]
Voltaje nominal	360 [V]
Numero de celdas	96
Velocidad máxima	144 [km/h]

2.3.2 Caracterización de batería de Nissan Leaf

En el caso de la batería, el gemelo digital puede ser un modelo de sistema multifísico y multiescalar que también contiene datos históricos. Los modelos livianos pueden ser modelos de circuitos agrupados y equivalentes del paquete de baterías. Los datos históricos pueden consistir en datos de medición para el paquete de baterías específico, incluyendo: **Temperatura** y **estado de carga.**

La batería presente en el Nissan Leaf tiene una capacidad de 40 [kWh], está compuesta por 192 celdas agrupadas en 24 módulos de 8 celdas cada uno. Dentro de cada módulo, la conexión de celdas es la siguiente: 4 pares de celdas conectadas en serie y cada par conectado en paralelo. Dando una configuración global de 96 pares de celdas conectadas en serie y cada par conectado en paralelo.

Ilustración 2.7 Organización de las celdas de batería del Nissan Leaf

Cada módulo de batería tiene una masa aproximada de 8.7[kg] y sus medidas se pueden simplificar en 300[mm]x222[mm]x68[mm].

De esta forma se pueden establecer las siguientes relaciones para los voltajes nominales de celda y batería y para las capacidades de corriente nominales en función de la cantidad de celdas conectadas en serie (N_s) y paralelo (N_p) respectivamente:

$$V_{nom,bat} = N_s \cdot V_{nom,celda} \tag{2.1}$$

$$I_{nom,bat} = N_p \cdot I_{nom,celda} \tag{2.2}$$

2.4 Modelo de consumo energético en base a la potencia requerida

Para la realización de un gemelo digital es necesario crear modelos o representaciones matemáticas de las distintas componentes físicas que representen los viajes realizados por el vehículo y que, a la vez, sea capaz de hacer proyecciones. Distintos autores han hecho aproximaciones para poder estimar la potencia y la energía consumida por un vehículo en un viaje determinado utilizando "modelos hacia atrás". Los parámetros de los modelos varían según el objetivo del modelo, cambiando ciertos parámetros que tienen que ver con el vehículo, sin embargo, todos se basan en el mismo principio: calcular las fuerzas que el vehículo debe vencer para ponerse en movimiento en base a datos de velocidad, aceleración y pendiente del suelo. Con eso se puede calcular la potencia instantánea consumida y así la energía consumida por un vehículo. Este modelo se le denomina modelo dinámico longitudinal; dinámico pues se basa en las fuerzas y longitudinal ya que solo toma en cuenta el movimiento en una dirección.

2.4.1 Modelo dinámico longitudinal

El modelo dinámico longitudinal es un modelo basado en potencias CPEM (por sus siglas en ingles Comprehensive Power-based EV consumption Model), este requiere los datos de velocidad, elevación y características propias del vehículo. Este modelo es utilizado para estimar la energía consumida por un vehículo eléctrico y su rendimiento considerando también la recuperación de energía gracias al freno regenerativo con el que cuentan este tipo de vehículos [4]. Además, este modelo es capaz de estimar cuánto afecta el estilo de conducción en el consumo de energía [5]. Dentro de la misma literatura en que se basa el trabajo, los datos utilizados son probados de manera empírica llegando a un error cercano al 6%. Por otro lado, se plantea que los parámetros utilizados tienen un papel fundamental en el cálculo del consumo energético por lo que una buena elección implicara un error aún menor [6].

El modelo CPEM, como se dijo anteriormente se basa en el cálculo de las fuerzas necesarias para realizar un movimiento para así estimar la potencia instantánea consumida por el vehículo [7]

La formulación para el modelo general se basa en la Segunda Ley de Newton

$$\frac{\partial v}{\partial t} = \frac{\sum F_t - \sum F_{tr}}{\delta m} \tag{2.3}$$

Donde v es la velocidad del vehículo, $\sum F_t$ es la fuerza de tracción proporcionada por el motor del vehículo para provocar el movimiento de este, $\sum F_{tr}$ es la suma de todas las fuerzas que

provocan una resistencia al movimiento, δ es un factor de masa asociado a las partes móviles y rotatorias del vehículo y m es la masa del vehículo y el DCL mostrado en la Ilustración 2.8.

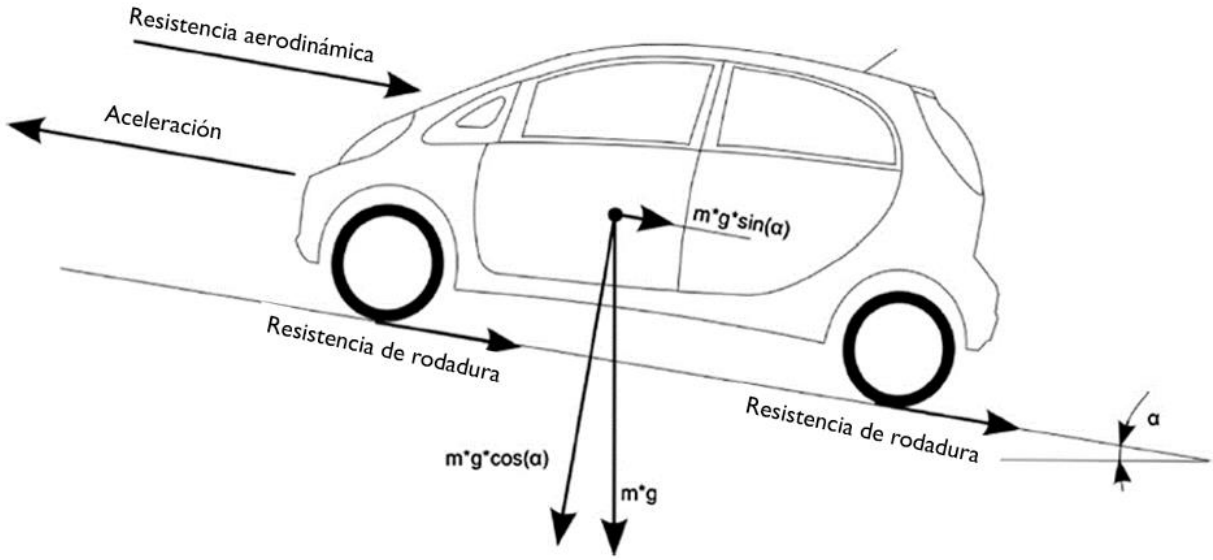

Ilustración 2.8 Esquema de fuerzas actuando sobre un vehículo que sube una pendiente

De esta forma y desarrollando la formula anterior, para saber la fuerza requerida para realizar el movimiento, quedará la siguiente formula:

$$F_{Traccion} = \frac{\partial v}{\partial t} \cdot \delta m + F_{Roce} + F_{Aerodinamico} + F_{pendiente} \tag{2.4}$$

El factor de masa toma valores entre 1.01 y 1.05, además la masa del vehículo es variable conforme al número de pasajeros por lo que se desprecia su valor. Es así entonces que el termino $\frac{\partial v}{\partial t} \cdot \delta m$ se puede escribir como masa por aceleración.

Si toda la fórmula es ponderada por la velocidad del vehículo se tendrá como resultado, la potencia requerida para cada instante de tiempo t y queda expresada como:

$$P_{ruedas}(t) = (ma(t) + mg\cos(\alpha)\frac{C_r}{1000}\cdot(c_1 v(t) + c_2) + \frac{1}{2}\rho_{aire}A_{ve}C_D v(t)^2 + mgsen(\alpha)) \cdot v(t)$$

(2.5)

Donde P es la potencia mecánica y el subíndice "ruedas" indica que es específicamente la potencia de las ruedas, es necesario tener en cuenta que esta potencia es la necesaria para realizar un movimiento descrito por lo tanto es distinta la potencia del motor y a la potencia entregada por la batería ya que hay perdidas de energía en el flujo. Además, las baterías no solo entregan potencia a las ruedas, sino que también a distintos elementos necesarios, auxiliares o accesorios del vehículo como luces, calefacción, etc. Otro aspecto que deriva de la formulación es la posibilidad de regeneración de energía (freno regenerativo) cuando se tienen una aceleración negativa, por lo tanto, se tiene un flujo de energía como el de la figura:

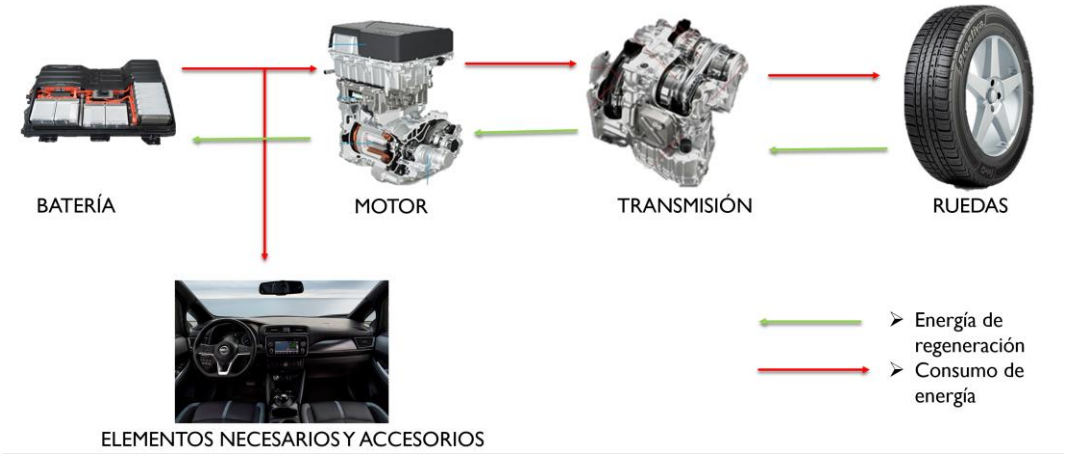

Ilustración 2.9 Esquema del flujo de energía en el vehículo

Todos los traspasos de energía implican una pérdida de energía que se estiman y se traducen en eficiencias. Por lo tanto, para estimar la potencia se utilizarán las siguientes formulas:

Si la potencia de la rueda es mayor que 0:

$$P_{bat-out} = \frac{P_{rueda}}{\eta_{trans}\eta_{bat}\eta_{motor}} + P_0$$

(2.6)

Si la potencia de la rueda es menor que 0 y el valor absoluto de la aceleración del vehículo es menor a 0.2 veces la aceleración de gravedad, esto ya que el vehículo en general frena con las ruedas conectadas con el motor (delanteras) para realizar la recarga de batería, pero se asume que

un frenado brusco implica una emergencia y por lo tanto el vehículo estaría frenando con las 4 ruedas [7], la fórmula de potencia de entrada a la batería será:

$$P_{bat-in} = \frac{P_{rueda}\,\eta_{reg}}{\eta_{trans}\eta_{bat}\eta_{motor}}$$

(2.7)

Con:

$$\eta_{reg} = (e^{\left(\frac{0.0411}{a(t)}\right)})^{-1}$$

(2.8)

Finalmente, para saber la energía utilizada durante un recorrido se debe integrar la potencia estimada de la batería a lo largo del tiempo

$$E_e = \int_0^T P_{Bat,i}\,\partial t$$

(2.9)

Para efectos del presente trabajo los datos son tomados de manera discreta por lo que el cálculo de la energía pasará desde una integral a una sumatoria, quedando de la siguiente forma:

$$E_e = \sum_{i=0}^{T} P_{Bat,i} \cdot \Delta t_i$$

(2.10)

2.4.2 Parámetros del modelo

Los parámetros utilizados son basados en trabajos realizados anteriormente [8], [9] con este mismo vehículo, fueron ajustados según pruebas que se detallaran más adelante y se muestran en la siguiente tabla.

Tabla 2.2 Parámetros utilizados en modelo de potencia mecánica

Parámetro	**Valor**

m: Masa del vehículo	1580[kg] varía según cantidad de pasajeros
g: Aceleración de gravedad	9.81[m/s^2] varía entre 9,764 y 9,834
c_1: Coeficiente de fricción de rodadura de neumático 1	0.0328 varía entre 0.03 y 0.06
c_2: Coeficiente de fricción de rodadura de neumático 2	4.575 varía entre 4 y 8
C_r: Coeficiente de fricción de rodadura pavimento	2.2806 varía entre 1 y 4
ρ_{aire}: Densidad del aire	1.25[kg/m^3] varía entre 1.1 y 1.3
C_D: Coeficiente de arrastre aerodinámico	0.28 varía entre 0.2 y 0.4
A_{ve}: Área o sección frontal del vehículo	2.76 [m^2]
R: Radio de la tierra	6371000 [m] varía entre 6366 y 6371

Las eficiencias asumidas para cada proceso de traspaso de energía son:

Tabla 2.3 Eficiencias utilizadas en modelo

η_{bat}: Eficiencia de batería	0.93 varía de 0.9 a 0.95
η_{motor}: Eficiencia de motor	0.91 varía de 0.9 a 0.95

η_{trans}: Eficiencia de transmisión	0.92 varía de 0.9 a 0.98

2.5 Modelo termodinámico en base a la perdida de potencia

El modelo termodinámico de batería utilizado define la temperatura interior de la batería como una función de la temperatura exterior y la perdida de energía de la batería, asociado a su resistencia eléctrica interna y/o su eficiencia [10]. La red térmica de capacitancia global de la batería y su entorno se define como:

$$C_{th} \cdot \frac{\partial T_{bat}}{\partial t} = \frac{1}{R_{th}} \cdot (T_{out} - T_{bat}) + E_J$$

(2.11)

Donde C_{th} es la capacitancia térmica de la batería, T_{bat} es la temperatura de la batería, t es tiempo, R_{th} es la resistencia térmica de la batería, T_{out} es la temperatura exterior y E_J es la energía liberada de la batería que no pudo ser utilizada o dicho de otra forma la perdida de energía. La pérdida de energía se calcula de la siguiente forma:

$$E_J = I_{bat}^2 \cdot R_{bat}$$

(2.12)

Con I_{bat} la intensidad de corriente y R_{bat} igual a la resistencia interna de la batería

2.5.1 Modelo termodinámico equivalente

El modelo termodinámico equivalente depende en gran medida del vehículo a modelar, en particular de su tipo de batería, de su sistema de intercambio de calor o regulación de temperatura y de la geometría u ordenamiento que tienen todas sus partes dentro del vehículo. Debido a que el Nissan Leaf no cuenta con un sistema de enfriamiento activo, se utiliza la formula planteada en 2.11 con temperatura exterior igual a temperatura ambiente y que luego de procesamientos matemáticos sencillos queda de la siguiente forma:

$$\Delta T_{bat} = (\frac{1}{R_{th}} \cdot (T_{amb} - T_{bat}) + P_J) \cdot \frac{1}{C_{th}} \cdot \Delta t$$

(2.13)

Donde ΔT_{bat} es la diferencia de temperatura de la batería entre un instante t_i y t_{i-1}, Δt es la diferencia de tiempo entre el instante t_i y t_{i-1}, T_{amb} es la temperatura ambiente y P_J es la pérdida de potencia en el instante t_i asociado a la eficiencia de la batería y se calcula de la siguiente manera:

$$P_j = P_{bat} \cdot \eta_{bat}$$

(2.14)

De esta forma se puede calcular la temperatura de la batería en cada instante de manera recursiva.

2.5.2 Parámetros del modelo

La capacitancia térmica es calculada en función del material de las baterías, como se dijo anteriormente el Nissan Leaf cuenta con sistemas pasivos de enfriamiento por lo que el paquete de baterías cuenta con una minimización de los materiales circundantes para maximizar las pérdidas térmicas. De esta forma se puede asumir que el calor especifico de la celda de batería es igual a la del módulo y al del paquete de baterías. El calor especifico de la celda se define como la suma de la capacidad de calor de cada uno de los componentes de la célula y sus valores fluctúan entre 700[J/kg K] y 1300[J/kg K] [11], [12]. La capacitancia térmica se calcula entonces de la siguiente manera:

$$C_{th} = c_{p,batt} \cdot m_{bat}$$

(2.15)

$$m_{batt} = N_{módulos} \cdot m_{módulo}$$

(2.16)

Donde $c_{p,batt}$ es el calor especifico de la batería y m_{bat} es la masa de la batería, la masa de la batería se considera como la masa del módulo de batería multiplicado por la cantidad de módulos.

La resistencia térmica es calculada en función del material de las baterías y de su geometría. Para el caso de la celda, se tiene un prisma rectangular por lo tanto la resistencia térmica se calcula de la siguiente manera:

$$R_{th,cell} = (\frac{h_{cell}}{2})/(2 \cdot S_{cell} \cdot k)$$

(2.17)

Donde h_{cell} es el grosor o la altura de la celda de batería, S_{cell} la superficie perpendicular a la altura y k el coeficiente de transferencia de calor que según la bibliografía se puede asumir como

$0.48[\frac{W}{mK}]$. Extrapolando esta idea y asumiendo que el módulo y el paquete de baterías es isotrópico se realiza la simplificación mostrada en la figura 2.10 para poder aplicar la misma formulación de las celdas de batería al pack de baterías.

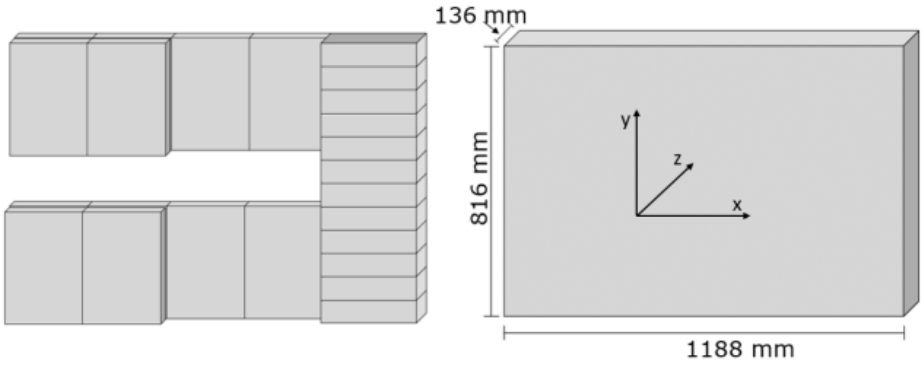

Ilustración 2.10 Distribucion del pack de baterias y su simplificación [10]

De esta forma se calcula la resistencia térmica para cada uno de los ejes dando los siguientes resultados: $R_{th,battZ} = 0.073$, $R_{th,battX} = 5.58$ y $R_{th,battY} = 2.63$. Se puede notar que la resistencia en el eje z mucho menor que en el x e y por lo cual representa un puente térmico hacia el exterior y por lo tanto se asume como la resistencia térmica de la batería en su totalidad.

Los parámetros del modelo se resumen en la siguiente tabla:

Tabla 2.4 parámetros utilizados en modelo termodinámico

Parámetro	**Valor**
m: Masa de la batería	208.8[kg]
C_{th}: capacitancia térmica	229680 [J/K]
$R_{th,battZ}$: Resistencia térmica	0.073 [K/W]

2.6 Modelo de voltaje de batería en función del SOC

El modelo utilizado para la estimación del voltaje de la batería es el modelo de circuito equivalente de primer orden. Este modelo está compuesto por una fuente de tensión que representa la tensión en circuito abierto OCV (por sus siglas en inglés open circuit voltage), una resistencia óhmica Ri que modela la resistencia interna y una red RC de una polarización que describe este fenómeno. La ramificación RC está compuesta por una capacitancia Cdf, que representa la polarización de los electrodos metálicos y la resistencia Rdf se produce por el contacto de los electrodos con el electrolito. Este modelo se muestra en la figura 2.11 donde Ibat es la corriente de carga o descarga dependiendo del caso y Vbat es la tensión en los terminales de la batería.

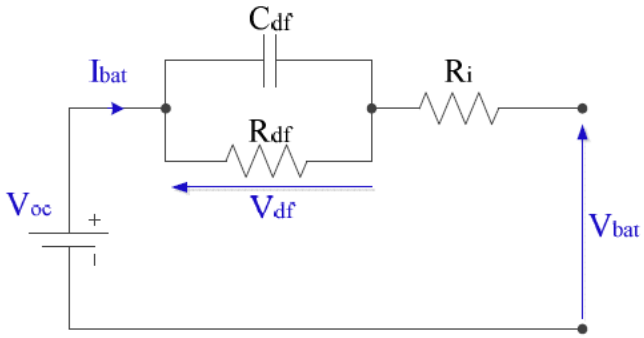

Ilustración 2.11Modelo de batería de primer orden

Luego la estimación del voltaje en circuito abierto de la batería se basa en la relación no lineal entre el SOC y la tensión de circuito abierto que en general sirve para representar los procesos electroquímicos y la termodinámica de la batería.

2.6.1 Modelo SOC-OCV

El modelo SOC-OCV plantea una relación matemática entre estas dos variables que pueden ser descritas o aproximadas en base a la experiencia y/o datos de laboratorio. Las ecuaciones OCV dadas a partir de datos experimentales son múltiples y se detallan algunos ejemplos en la siguiente tabla.

Tabla 2.5 Formulas de modelos OCV

Nombre	Formulación (SOC=x)
1. Lineal	$OCV(x) = p_1 x + p_0$

2. Polinomial de 6 grados	$OCV(x) = p_6x^6 + p_5x^5 + p_4x^4 + p_3x^3 + p_2x^2 + p_1x + p_0$
3. Modelo de Weng [13]	$OCV(x) = p_0 + p_1\frac{1}{1+e^{\propto_1(x-\beta_1)}} + p_2\frac{1}{1+e^{\propto_2(x-\beta_2)}} + p_3\frac{1}{1+e^{\propto_3(x-1)}} + p_4\frac{1}{1+e^{\propto_4 x}} + p_5x$
4. Doble exponencial	$OCV(x) = p_1e^{(x\propto_1)} + p_2e^{(x\propto_2)}$
5. Modelo de Baccouche [14]	$OCV(x) = p_0e^{(x\propto_1)} + p_1e^{(x\propto_2)} + p_2x^2$

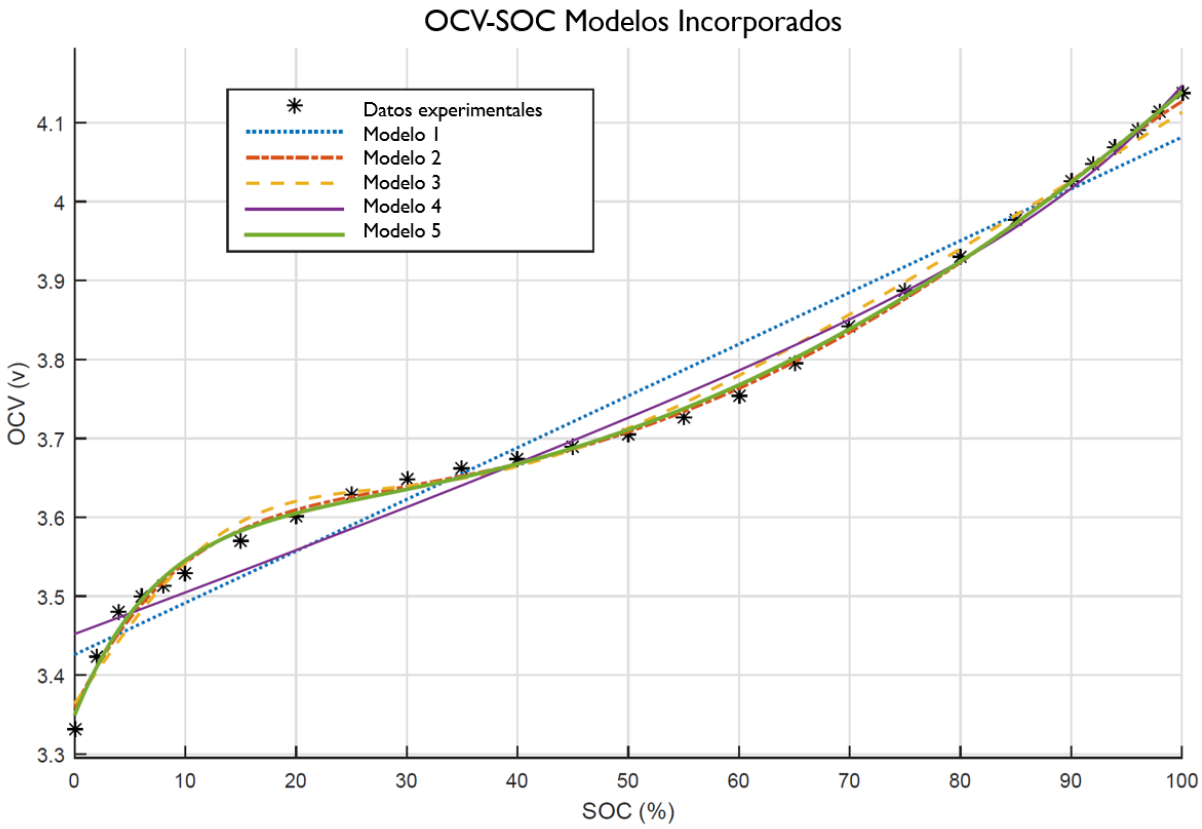

Ilustración 2.12 Resultados de prueba de modelos en una carga a 25°C [14]

Finalmente debido a su simplicidad, menor costo computacional gracias a la cantidad de parámetros y las operaciones matemáticas aplicadas y a su precisión o cercanía con los datos empíricos se utiliza el modelo 5 el cual, según la bibliografía, representa un menor error (RMSE igual 0.0105). Además, para efectos de este trabajo se asumirá que el voltaje de circuito abierto es igual al voltaje de la batería. Esta simplificación es valida siempre y cuando se tenga una buena eficiencia de la batería y se trabaje con modelos "livianos".

2.6.2 Parámetros del modelo

Como el comportamiento de las baterías varía en función de la temperatura, los parámetros de este modelo en particular también lo hacen. Los parámetros utilizados se especifican en la siguiente tabla:

Tabla 2.6 Parámetros del modelo OCV de Baccouche a diferentes temperaturas

Dato	5°C		15°C		25°C		45°C	
	Carga	Descarga	Carga	Descarga	Carga	Descarga	Carga	Descarga
p_0	3.743	3.804	3.629	3.599	3.637	3.604	3.584	3.55
p_1	-0.2756	-0.3487	-0.3191	-0.2933	-0.3091	-0.2803	-0.4868	-0.4573
p_2	$8.013 \cdot 10^{-5}$	$8.838 \cdot 10^{-5}$	$6.952 \cdot 10^{-5}$	$6.9 \cdot 10^{-5}$	$7.033 \cdot 10^{-5}$	$6.871 \cdot 10^{-5}$	$6.212 \cdot 10^{-5}$	$6.02 \cdot 10^{-5}$
$\propto_1$	-0.001155	-0.001618	-0.0005411	-0.0004687	-0.0005747	-0.0004523	-0.0001919	$-6.241 \cdot 10^{-5}$
$\propto_2$	-0.06674	-0.04724	-0.1493	-0.1434	-0.1366	-0.1341	-0.2181	-0.221

3 Metodología

Debido a que el objetivo es la realización de un modelo electro mecánico en base a datos operacionales la metodología base es la siguiente:

1. Revisión bibliográfica
   1.1. Definiciones
   1.2. Estudiar los diferentes modelos de batería existentes
   1.3. Encontrar parámetros bases claves para iniciar modelos
2. Recopilación de datos
   2.1. Datos entregados por el centro de energía
   2.2. Procesamiento y selección de datos
3. Realización de los modelos
   3.1. Se crea un modelo que como input necesita las condiciones geográficas y una marca de tiempo y como output entrega datos de potencia utilizada/recorrida, consumo energético, SOC, temperatura de batería y voltaje de batería
   3.2. Se comparan los datos calculados con los datos extraídos del vehículo
   3.3. Fase de entrenamiento
   3.4. Nuevo procesamiento de datos para comparación
   3.5. Extraer una base de datos geográficos para integración de modelo y poder hacer predicciones
4. Verificación y validación
   4.1. Limitaciones del modelo
   4.2. Comparación a secciones de viajes ya realizadas
   4.3. Comparación con software de ruteo
5. Análisis del modelo
   5.1. Potenciales usos
   5.2. Limitaciones
   5.3. Mejoras propuestas

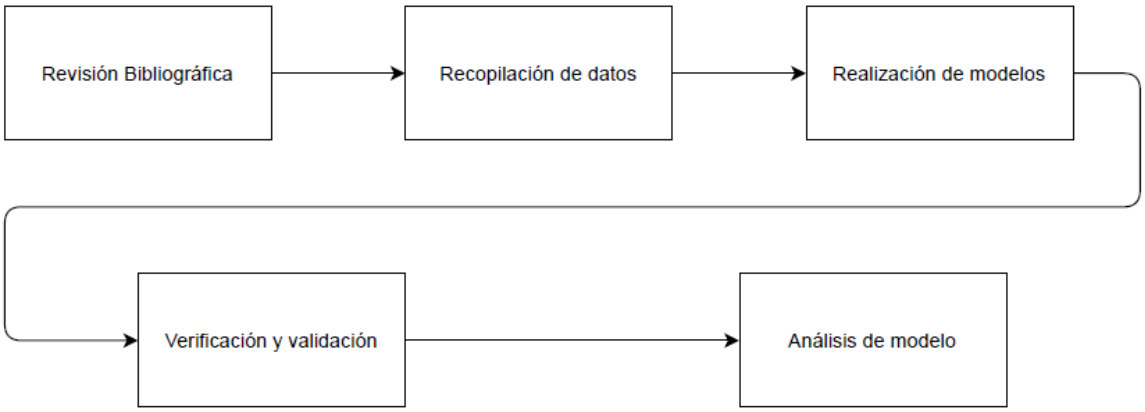

Ilustración 3.1 Diagrama de la estrategia

3.1 Recursos

Para poder llevar a cabo el trabajo se necesitaron distintos recursos, se describirá cada uno de los recursos más relevantes.

3.1.1 Datos y Leaf Spy Pro

La base de datos utilizada para la creación y entrenamiento del modelo fue entregada por Jorge Reyes, quien extrajo los datos con una aplicación para smartphones llamada LeafSpyPro. Esta aplicación funciona conectando un dispositivo de diagnóstico al puerto OBD del vehículo para que extraiga datos directamente desde la ECU, además agrega datos que extrae a través del teléfono celular, como lo son ubicación, marca de tiempo y altura entre otros. La base de datos extraída es bastante robusta por lo que no se usarán todos los datos. La definición y el formato en que se entregan se encuentran en la guía de ayuda de la aplicación [15],

3.1.2 Python

El lenguaje de programación Python ha ganado gran popularidad en el último tiempo debido a su gran versatilidad y por ser un lenguaje pensado en ser amigable con el usuario al ser orientada a objetos, es decir que permite al usuario generar nuevos tipos de datos para resolver los problemas. Además, cuenta con una comunidad activa que constantemente actualiza paquetes y librerías que son utilizadas en distintas áreas, dentro de las que se destaca la ciencia y tecnología [16][17].

Dentro de las librerías utilizadas en el trabajo destacan: Matplotlib, OSMnx, Pandas, Numpy y NetworK

3.2 Descripción de las etapas

3.2.1 Revisión bibliográfica: Parámetros e información inicial

Se encontró mucha información valiosa entorno a distintos papers y trabajos ya realizados que contaban con datos calculados y datos empíricos que servirían de punto de comparación y de partida para los modelos que se iban a realizar. De esta forma se decidió optar por un modelo dinámico longitudinal en base a la potencia requerida, un modelo termodinámico equivalente en base a la perdida de potencia y un modelo OCV-SOC para estimar el voltaje de la batería.

3.2.2 Recopilación de datos: Preprocesamiento

Al ser un modelo en base a una potencia necesaria para moverse de un lugar a otro, sería necesario trabajar con datos geográficos por lo tanto se buscó información pertinente y en base a la localización y marcas de tiempo se puede obtener: Altura del lugar y tiempos entre un punto y otro. Luego con la fórmula de Haversine calcular la distancia plana entre dos puntos geográficos. Con todos estos datos se podría sacar la velocidad y por ende también la aceleración entre dos puntos, para finalmente tener datos suficientes para calcular la potencia necesaria de las ruedas.

La fórmula de Haversine se define como:

$$\Delta d = 2R \cdot arcsen\left(\sqrt{sen^2\left(\frac{\emptyset_2 - \emptyset_1}{2}\right) + \cos(\emptyset_1)\cos(\emptyset_2)\, sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right)}\right) \tag{3.1}$$

Donde R es el radio de la tierra, $\emptyset$ latitud y φ longitud de cada punto.

Cómo la ubicación es entregada por el teléfono celular, está limitada a la cobertura telefónica por lo que muchas veces se capturan velocidades mayores a la máxima velocidad del Nissan Leaf, pendientes mayores a cualquier calle en el mundo y saltos inesperados, en estos casos se eliminaba el dato viciado y se volvía a calcular los datos de distancia velocidad y aceleración entorno a las nuevas coordenadas y marcas de tiempo.

Otro suceso inesperado es que dentro de un mismo dato de viaje se tenían datos de cargas, algunos dejaban registros y otros no lo que producía muchos errores en el modelo, es por ello que se confeccionó un programa para separar este tipo de eventos y solo se toma en cuenta los procesos de descarga y regeneración.

3.2.3 Realización de los modelos

Modelo dinámico longitudinal: Para llevar a cabo este objetivo en necesario calcular todos los datos para poder utilizar la fórmula 2.6, que luego nos permitirá obtener el consumo energético, para esto debemos:

- Determinar P_0 de la ecuación 2.6: esto se hizo tomando todos y cada uno de los datos de viajes compartidos, se filtro la velocidad y se calculó la potencia en función del amperaje y el voltaje del pack de baterías (datos obtenidos por telemetría), la varianza era bastante amplia y esto puede ser debido a múltiples factores como por ejemplo aire acondicionado, calefacción luces artefactos electrónicos, etc. Por lo tanto se escogió un valor cercano a la moda y al promedio de todos los datos y que a la vez fuera fiel a la bibliografía.
- Ingresar los datos de: tiempo, posición (coordenadas) y elevación
- Crear una lista o tuplas del mismo tamaño que la que contiene los datos antes mencionados, pero llenas de 0, esto para que el valor inicial de las variables calculadas sea 0, por ejemplo velocidad inicial y aceleración inicial siempre serán 0.
- Calcular la distancia plana entre dos puntos consecutivos con la fórmula de Haversine y la diferencia de altura de estos. De esta forma aplicando Pitágoras se puede obtener la distancia real entre dos puntos. Junto con eso será necesario calcular también la diferencia de tiempo que hay entre un punto y otro. De esta forma se puede obtener de manera discreta las velocidades en cada tramo de tiempo, y esto a su vez permitirá calcular la aceleración en base a las diferencias de velocidades durante un periodo de tiempo previamente calculado. Todo esto a partir del segundo dato como se mencionó en el punto anterior, es decir, la velocidad del punto 1 es 0 mientras que la del punto dos es igual a la distancia real entre el punto 1 y el punto 2 dividido por la diferencia de tiempo que hay entre el momento de estar en el punto 1 y el de estar en el punto 2.

$$v_i = \frac{\Delta d}{\Delta t} = \frac{\Delta d}{t_i - t_{i-1}} \tag{3.2}$$

$$a_i = \frac{\Delta v}{\Delta t} = \frac{v_1 - v_{i-1}}{t_i - t_{i-1}} \tag{3.3}$$

- Eliminar datos de velocidades mayores a 150 [km/h] y recalcular la velocidad y aceleración, buscar pendientes altas y ajustar ángulo a un máximo, esto último porque

muchas veces se puede tener que por datos geográficos se tiene un movimiento ínfimo en el plano, pero de gran envergadura en el eje vertical en un estacionamiento arriba o en el subterráneo de un edificio, donde para un mismo dato de coordenadas se puede tener dos datos de altura.

- Calcular la potencia de rueda en base a los datos antes mencionados.
- Calcular la eficiencia de regeneración en base a la aceleración
- Calcular la potencia de batería tomando en cuenta todas las eficiencias indicando también las condicionantes de uso (Potencia y aceleración negativa)
- Con el dato de potencia de batería, al multiplicarlo por el tiempo, podemos calcular la energía utilizada, es decir, se calcula la integral de la potencia de manera discreta para obtener la energía, finalmente se suma toda la energía utilizada para así poder estudiar el rendimiento y la utilización energética en términos de SOC

Modelo termodinámico equivalente: Para llevar a cabo este objetivo se considera la temperatura ambiente constante a lo largo de un viaje y se utiliza la formula 2.13 para calcular de manera recursiva la temperatura de la batería en cada instante de tiempo. Para el calculo de la perdida de potencia se utiliza la potencia calculada con el modelo antes mencionado.

Modelo SOC-OCV: Para llevar a cabo este objetivo se considera la temperatura calculada en el modelo anterior para la elección de parámetros para la fórmula de Baccouche y el SOC calculado con el modelo de potencia mecánica y para determinar si se esta en un proceso de carga o descarga se utilizará como discriminante el valor de la potencia calculado para ese instante, si la potencia es positiva se considera un proceso de descarga y para potencias negativas procesos de carga.

3.2.4 Verificación y validación

- Comparar todos los datos calculados versus los extraídos por telemetría, específicamente Energía utilizada, SOC, temperatura y voltaje.
- Comparar los datos de distancia, graficando cada punto en la página Kepler.org [18] y una ruta similar con el mapa de Google.
- Un aspecto importante al que se llegó en esta parte no es solo al ajuste de los parámetros para ir acercándose a los datos de telemetría, sino que también se decidió tomar en cuenta solo los ciclos de descarga por lo tanto cada vez que se detectaba una subida abrupta en el SOC los datos se dividían para tomar cada parte como un viaje distinto

3.2.5 Análisis del modelo

Se utiliza este modelo para datos GPS y otro con un ruteador en base a una biblioteca o librería de Python y se estudian los resultados obtenidos comparando estos datos con la ficha técnica, los registros de viajes del vehículo, datos de la web y la bibliografía para luego estudiar las diferencias y los factores que afectan en los cálculos para finalmente determinar las capacidades del modelo.

4 Desarrollo

El desarrollo del modelo integral es realizado con el lenguaje Python en la interfaz de Spyder, este trabaja mediante dos códigos distintos dependiendo del uso que se le quiera dar y además se agregó un tercer código que tiene integrado el modelo, como alternativa para combatir ciertas limitaciones que se nombrarán más adelante.

4.1 Programa para procesamiento de datos

Los datos entregados por el centro de energía venían en formato csv y en formato xlsx. Además, muchos de estos datos juntaban todos los viajes realizados durante el periodo de tiempo estudiado (en general, un día) involucrando cargas intermedias que a veces dejaban registro y otras no, simplemente había una recuperación abrupta de energía. Como el modelo solo involucra ciclos de descarga y regeneración muchas veces las pruebas daban lugar a errores que provocaron grandes contratiempos. Por lo tanto, lo que se hizo fue que por medio de un código de Python ingresando el nombre del documento de registro a trabajar, guarde copias que no involucran grandes cambios de energía, dicho de otra forma, se intentaba separar los registros para unificar eventos y posteriormente clasificarlos en viajes, viajes con carga y sin viajes. Además de esta forma guardan todos los archivos en el mismo formato xlsx. Una vez separados se les ingresó un formato condicional al valor del SOC para poder identificar rápidamente cuando se trataba de un ciclo de carga y no utilizarlo para las pruebas del modelo. Anexo A 8.1.1 y Anexo A 8.1.2

4.2 Modelo

El modelo realizado se basa en todas las fórmulas antes mencionadas, muchas veces se tenían picks de energía que eran asociados a los datos de GPS. Para evitar este problema lo que se hizo fue crear un loop calculando las velocidades, en base a dos coordenadas consecutivas y la diferencia de tiempo entre cada dato de coordenada, una vez que se terminaba el cálculo de velocidad de todos los puntos se revisa cuáles son mayores a 150[km/h] que, según la fecha técnica disponible, es imposible que el vehículo alcance esa velocidad y se eliminaba ese dato para recalcular las velocidades hasta llegar solo a velocidades menores al valor descrito.

En base a lo mismo el cálculo de algunas pendientes era demasiada alta para cualquier calle del planeta, en este caso no se filtró porque puede ser asociado a la tasa de refresco de ubicación y el lugar donde se encuentra, por ejemplo, dentro de un edificio se puede tener las mismas coordenadas en momentos distintos y con distinta elevación. Lo que se hizo para este caso es simplemente optar por un ángulo máximo de inclinación de esta forma no se pierde tanta información y se evita tener picos de energía asociado a la componente gravitatoria de la potencia requerida.

Muchas veces se tendrán problemas de cobertura telefónica por lo que algunos datos serán erróneos por ejemplo en túneles, lugares en carretera o las afuera de la ciudad lo que provoca pequeños errores en el cálculo de potencia, esto hace que la mayoría de las veces la energía utilizada calculada difiere hasta un 20% por sobre el dato de telemetría. Anexo A 8.1.3

El siguiente diagrama muestra el algoritmo utilizado, a grandes rasgos, para la creación del modelo integral.

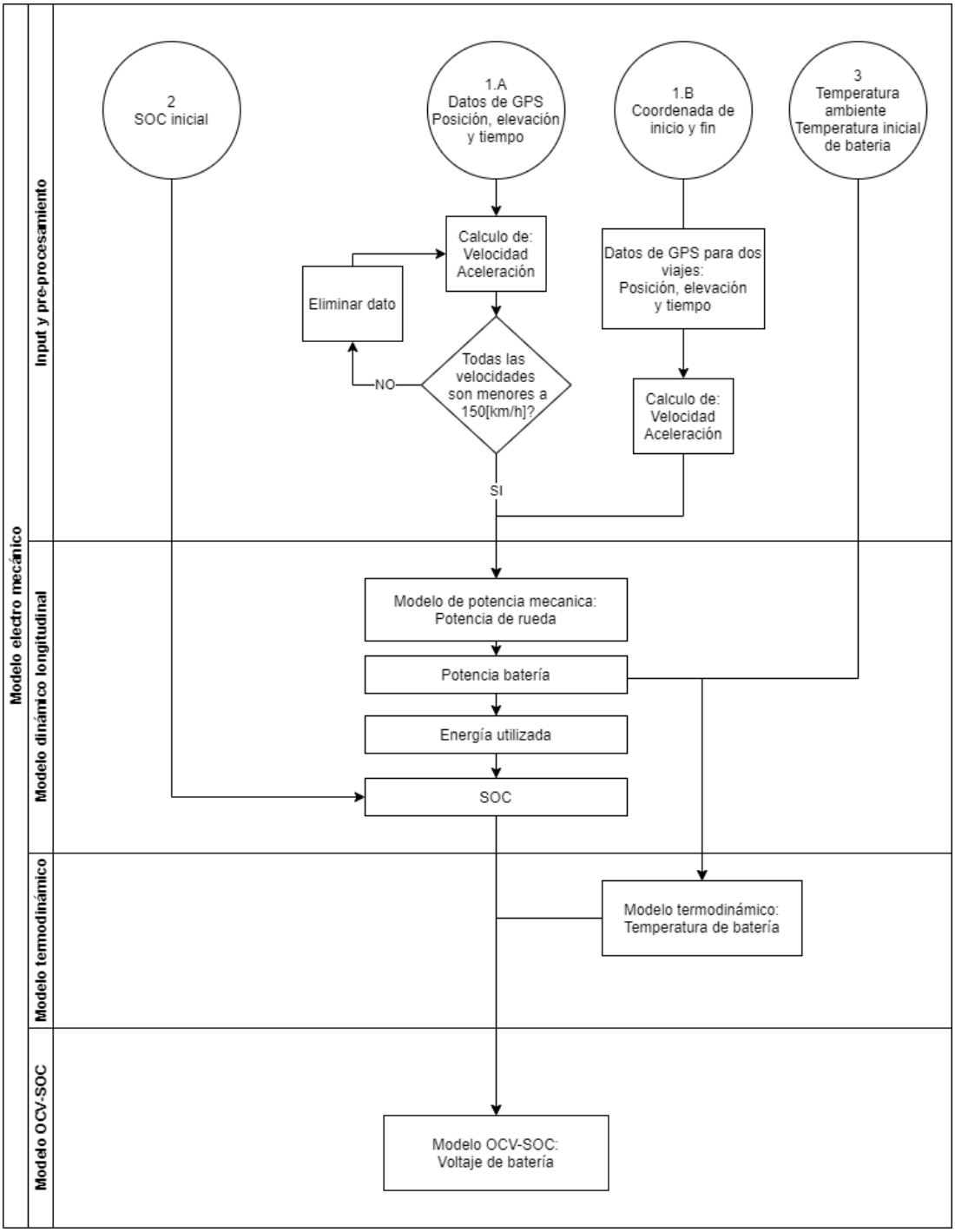

Ilustración 4.1Algoritmo del programa

El primer recuadro “Input y preprocesamiento” muestra en círculos los inputs necesarios para hacer funcionar el programa 1.A se utiliza cuando se tienen datos de GPS y 1.B cuando se utiliza el ruteador y cada rectángulo indicara una acción y/o el dato que se extrajo dando como output final: Potencia de rueda, potencia de batería, energía utilizada, rendimiento, SOC, temperatura de batería, voltaje de batería, distancia recorrida y tiempo de viaje.

4.2.1 Registro de viajes

El programa se base en datos de GPS de viajes ya realizados y en un ruteador que trata de replicar estos viajes o fracciones de viajes debido a las limitaciones de distancia y de puntos intermedios. Los programas tienen la capacidad de generar una tabla con los datos de tiempo y ubicación junto con todos los datos calculados: Velocidad, aceleración, potencia de rueda, potencia de batería, rendimiento, SOC, temperatura y voltaje similar al descargado por medio de Leaf Spy Pro para que puedan ser comparados.

4.2.2 Programación de viajes regionales

La programación de viajes a nivel regional se hace a través de un mapa abierto disponible en la biblioteca OSMnx de Python, junto con ello se integra una capa que agrega la elevación de del lugar en función de latitud y longitud, esta capa fue descargada en el sitio: www.diva-gis.org de manera gratuita. Para el trabajo se agregó la capa que se encuentra a lo largo de todo Chile.

Dentro del programa se utilizó un ruteador que, ingresando las coordenadas de inicio y fin, calcula dos rutas:

- Ruta 1 de tiempo: En base a las velocidades máximas y promedio utilizadas en los caminos que llevan de la locación de inicio a la final se escoge la que toma un menor tiempo
- Ruta 2 de distancia recorrida: En base a la distancia recorrida para llegar de la locación de inicio a la final se escoge la ruta que conlleva un menor recorrido independiente del tiempo que tome

Además, se deberá ingresar una temperatura ambiente y una temperatura inicial de batería de esta forma cada ruta, mediante el modelo electro mecánico, entregará información de: Potencia de rueda requerida, potencia de batería requerida, energía utilizada, SOC, temperatura de batería y voltaje de batería en distintos momentos de medición.

El costo informático o los recursos computacionales de utilizar este programa son altos ya que se deben descargar toda la base de dato asociada al lugar en el que se trazan las rutas, por lo que se recomienda utilizar para viajes dentro de una misma región o ciudad. Al ser estos datos guardados como gráficos al acercarse o hacer un acercamiento a alguna sección abrirá paso al detalle y no se perderá la calidad. Anexo A 8.1.4

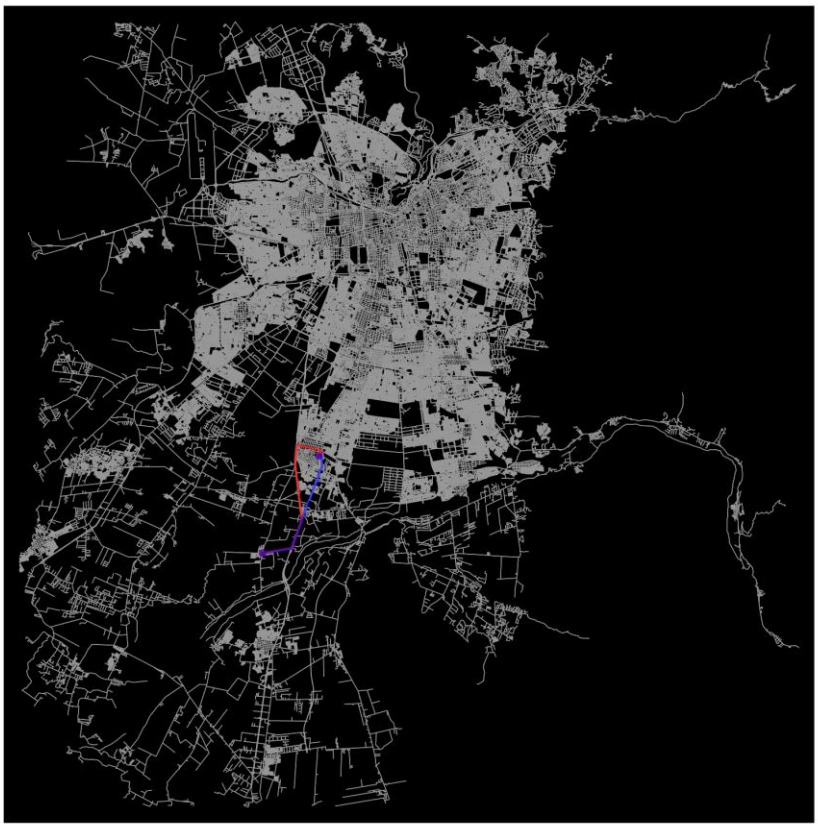

Ilustración 4.2Dos rutas azul más rápida y rojo menor distancia

Ilustración 4.3 Zoom de la imagen anterior sin pérdida de resolución

4.2.3 Alternativa viajes inter-regionales

Una vez que se descubrió esta limitante (solo viajes dentro de una misma región) se optó por buscar otra alternativa ampliando las herramientas a utilizar. Los elementos utilizados ahora serán Google Maps, Wikiloc y Python.

Wikiloc es una aplicación y pagina web para registrar o planificar viajes, esta aplicación permite descargar, de manera gratuita los viajes guardados en formato GPX. Como no es un ruteador lo que se debe hacer en primer lugar es crear la ruta de Google Maps y descargarla. Luego se sube esta ruta a Wikiloc para poder transformarla al formato antes nombrado, esto es necesario ya que al descargar los mapas directamente desde Google, estos no vienen con una marca de tiempo por lo que no se podrá calcular la velocidad y aceleración instantánea, Google si es capaz de entregar estos datos, pero es a través de una API pagada. Una vez que se tienen estos datos se aplica el mismo modelo para el cálculo de potencia.

5 Resultados y discusión

5.1 Resultados

El primer resultado de este trabajo es el modelo electromecánico del Nissan Leaf en si y el segundo resultado importante es el programa ruteador, ambos permiten estudiar el sistema de almacenamiento de energía en base a datos geográficos y marcas de tiempo.

Los resultados del modelo en particular se dividen en dos tipos: Programa para datos de GPS y programa ruteador

5.1.1 Modelo con datos de GPS

Modelo puesto a prueba comparado con viajes realizados utilizando las mismas marcas de tiempo y las mismas coordenadas registradas:

Ejemplo Viaje 1: Viaje interregional desde las cercanías de Talca hasta Chillan comprende cerca de 144 [km]

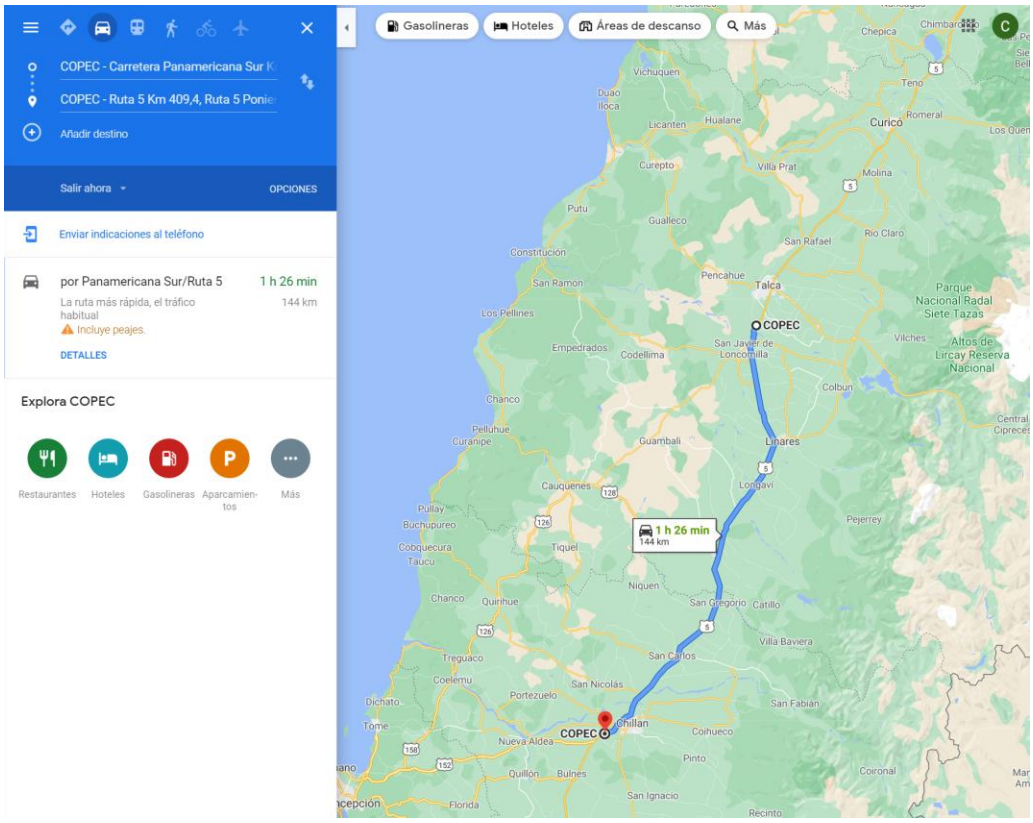

Ilustración 5.1 Mapa del recorrido del viaje 1 mapeado con Google

Se comienza con un SOC inicial de 95% y se termina con un 26% según los datos extraídos dando un rendimiento de 187 [Wh/km], una temperatura de batería inicial y final igual a 47.6°C y 47.5°C respectivamente.

Al aplicar el modelo los resultados son: Se considera una temperatura ambiente de 25°C dando como resultado un SOC final igual a14%, un rendimiento de 227 [Wh/km] y una temperatura final de 49°C. Las diferencias de los parámetros de temperatura y rendimiento medidos por telemetría y calculados por el modelo son de 3°C y 40 [Wh/km] respectivamente.

Además, se grafica el comportamiento del SOC, la temperatura y el voltaje en las ilustraciones 5.2, 5.3 y 5.4 respectivamente

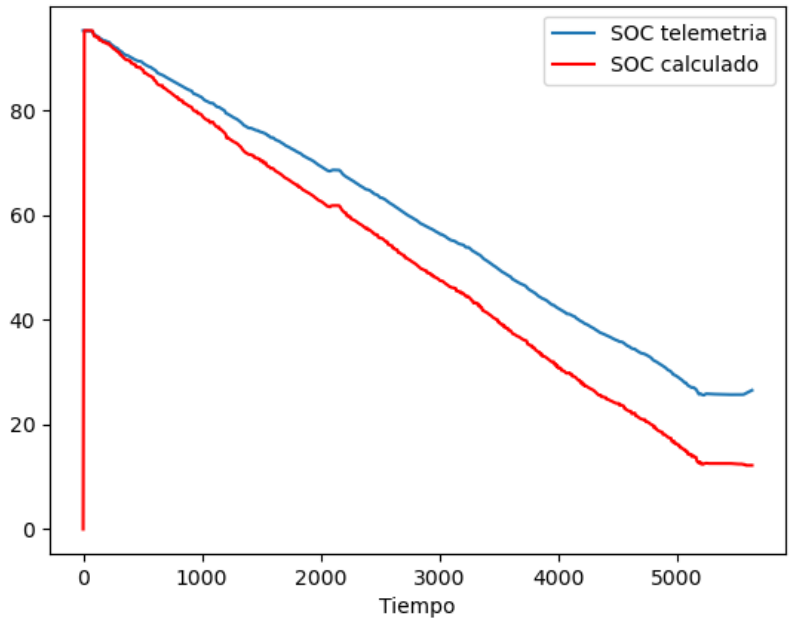

Ilustración 5.2 SOC con respecto al tiempo de viaje

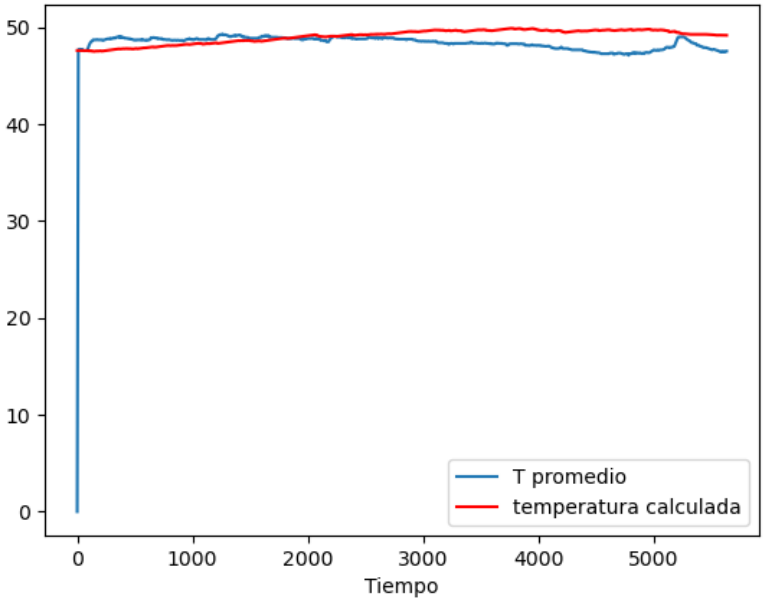

Ilustración 5.3 Temperatura con respecto al tiempo de viaje

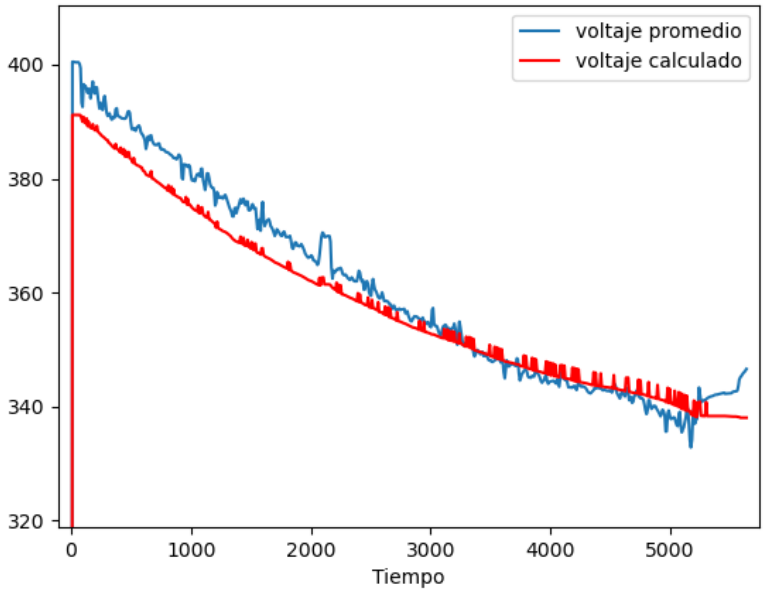

Ilustración 5.4 Voltaje de batería con respecto al tiempo de viaje

Estas diferencias se asocian a errores de GPS ya que como se muestra en la siguiente ilustración, para un mismo posicionamiento pueden verse múltiples elevaciones.

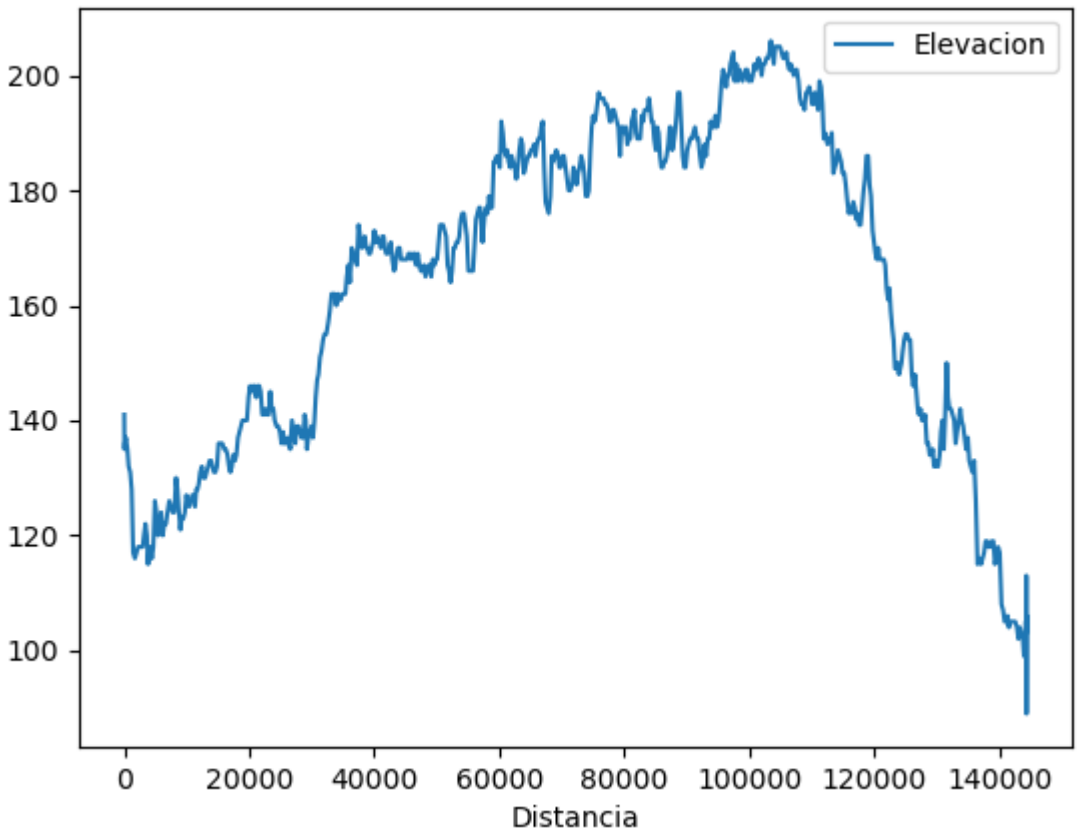

Ilustración 5.5 Elevación vs. Distancia plana recorrida

Ejemplo Viaje 2: Viaje de ida alrededor de la ciudad de Talcahuano que comprende alrededor de 33[km]

Ilustración 5.6 Mapa del recorrido viaje 2 mapeado con kepler

Se comienza con un SOC inicial de 53% y una temperatura de 24.7°C y se termina con un 28% de SOC y una temperatura de 26.2°C además según los datos extraídos da un rendimiento de 224 [Wh/km]

Al aplicar el modelo los resultados son: SOC final 25%, un rendimiento de 256 [Wh/km] y una temperatura final de 30.2°C. Todas las diferencias de los parámetros medidos por telemetría y los calculados por el modelo fueron menores a un 15%.

Nuevamente se grafica el comportamiento del SOC, la temperatura y el voltaje en las ilustraciones 5.7, 5.8 y 5.9 respectivamente. Al igual que el ejemplo anterior se ilustra la elevación con respecto al camino recorrido.

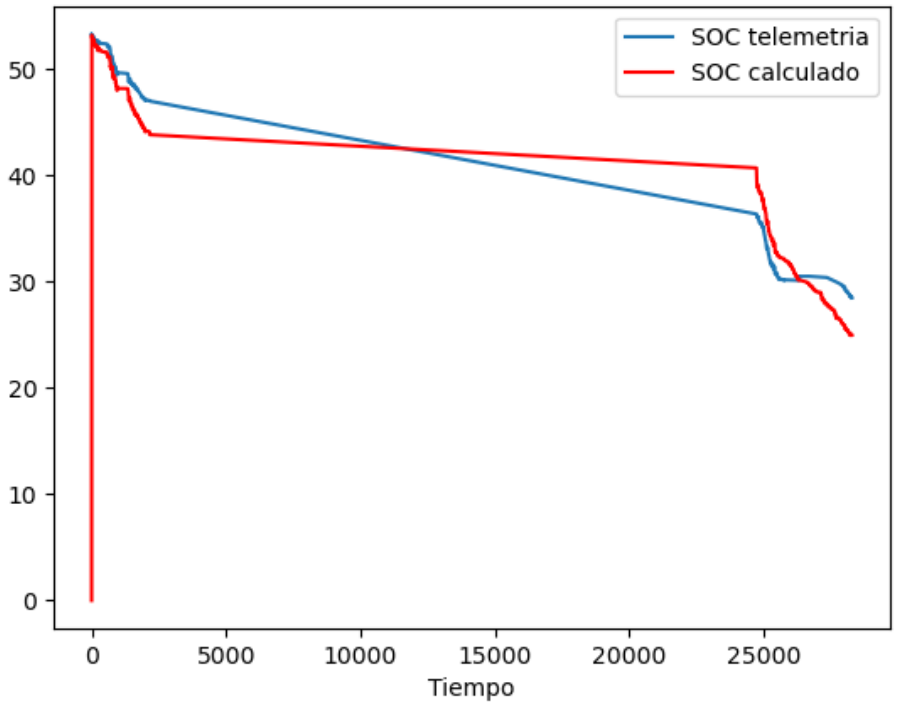

Ilustración 5.7 SOC con respecto al tiempo de viaje

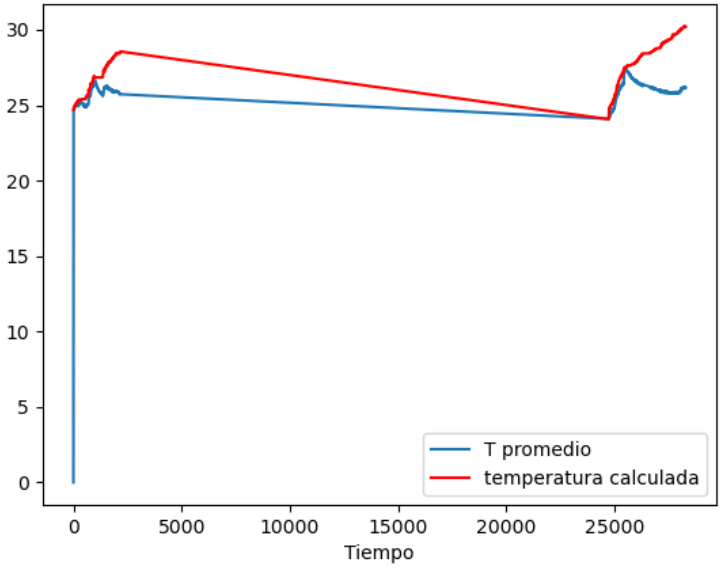

Ilustración 5.8 Temperatura con respecto al tiempo

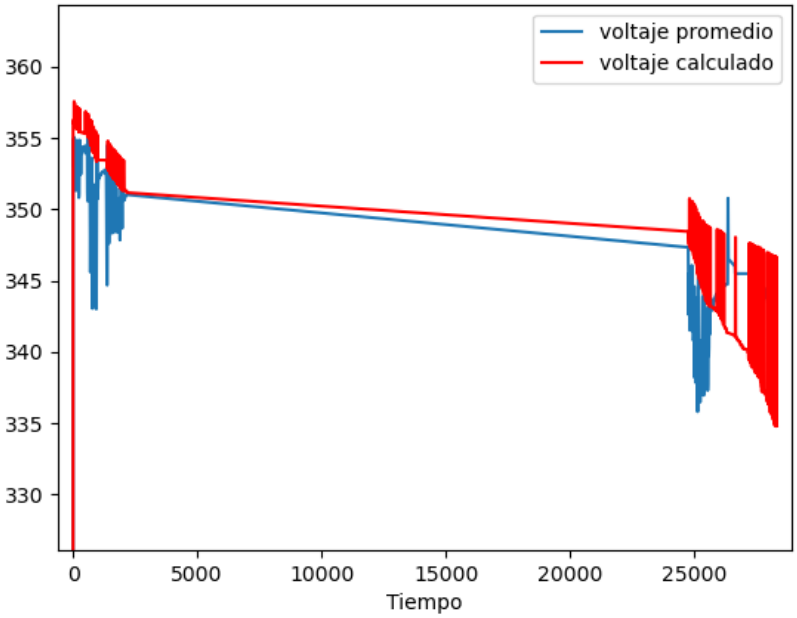

Ilustración 5.9 Voltaje con respecto al tiempo

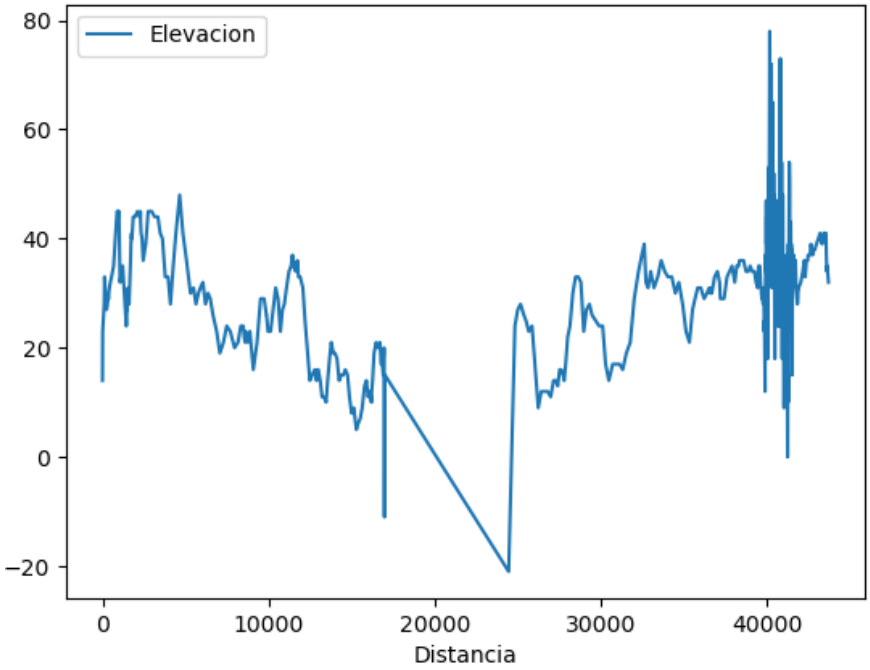

Ilustración 5.10 Elevación vs. Distancia plana recorrida

Finalmente se presentan dos tablas de resumen, una para viajes de ciudad y otra para viajes en carretera, de todos los viajes testeados con el programa.

Tabla 5.1 Máxima diferencia entre dato de telemetría y dato calculado para viajes en ciudad

Temperatura	SOC	Voltaje
Máx. diferencia Valor	Máx. diferencia Valor	Máx. diferencia Valor
8	12	12

Tabla 5.2 Máxima diferencia entre dato de telemetría y dato calculado para viajes en carretera

Temperatura	SOC	Voltaje
Máx. diferencia Valor	Máx. diferencia Valor	Máx. diferencia Valor
12	22	15

Como se puede observar tanto en los datos como en los gráficos, el consumo de energético siempre es mayor en el calculado versus el extraído por telemetría. Se considera que este error no es tan elevado para realizar grandes cambios ya que se asocia a errores de posicionamiento y/o altura que indique el teléfono celular que llena los datos del LeafSpyPro.

5.1.2 Modelo con ruteador

Ahora se presentarán los resultados del modelo con un mapa integrado, este programa tiene la capacidad de generar dos rutas desde un punto geográfico a otro, se necesita como input el SOC inicial, la región en la que se realizará el viaje, coordenadas de inicio y fin del viaje, la temperatura inicial de la batería y la temperatura ambiente al momento del viaje (se considera constante), con esto estimará y calculará la posición, elevación, velocidad, aceleración, consumo de energía, temperatura, voltaje y SOC del vehículo en cada instante . Cabe hacer notar que además de la limitación geográfica (solo viajes regionales) tampoco se pueden establecer puntos intermedios de ruta para llegar de un punto a otro.

Para estudiar la capacidad del modelo se establecieron rutas equivalentes a porciones o partes de viajes más grandes ya registradas por medio de LeafSpy y las rutas se compararon a las propuestas por el mapa de Google.

Ejemplo Viaje 2-A: Viaje en los alrededores de Concepción y Talcahuano, solo se toma una sección del viaje, el viaje toma alrededor de 13 minutos y un recorrido 14.3 km aproximadamente. Se registra un SOC inicial de 36.3 y un SOC final igual a 30.4 con un rendimiento de 162 [Wh/km] o 179 [Wh/km] dependiendo de cómo se mida (datos de telemetría). La temperatura inicial y final detectadas en este tramo son 24.2°C y 27.3°C respectivamente y los voltajes máximos y mínimos durante este recorrido son, respectivamente, 346V y 335V

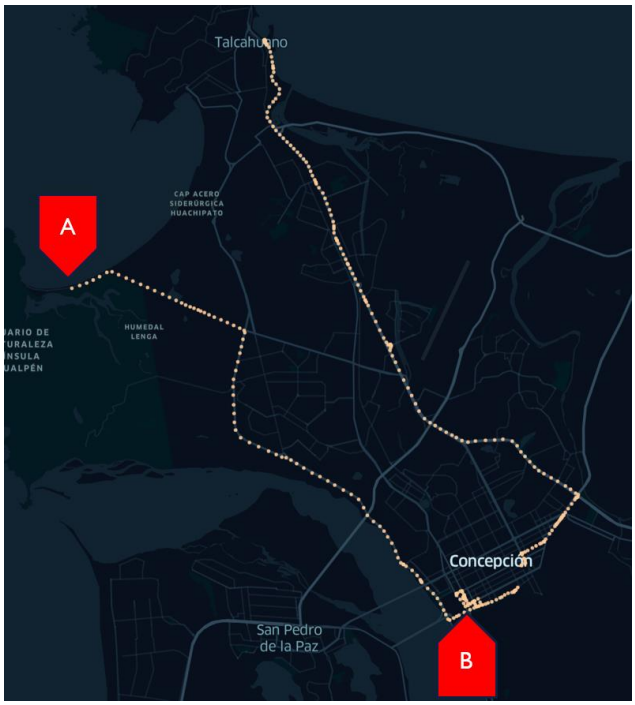

Ilustración 5.11 Parte del viaje 2 para estudiar el ruteador desde el punto A al punto B

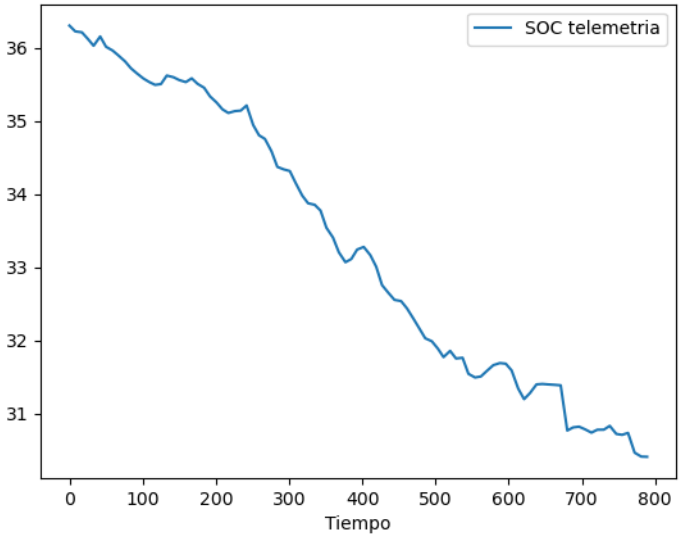

Ilustración 5.12 Dato de telemetría: SOC

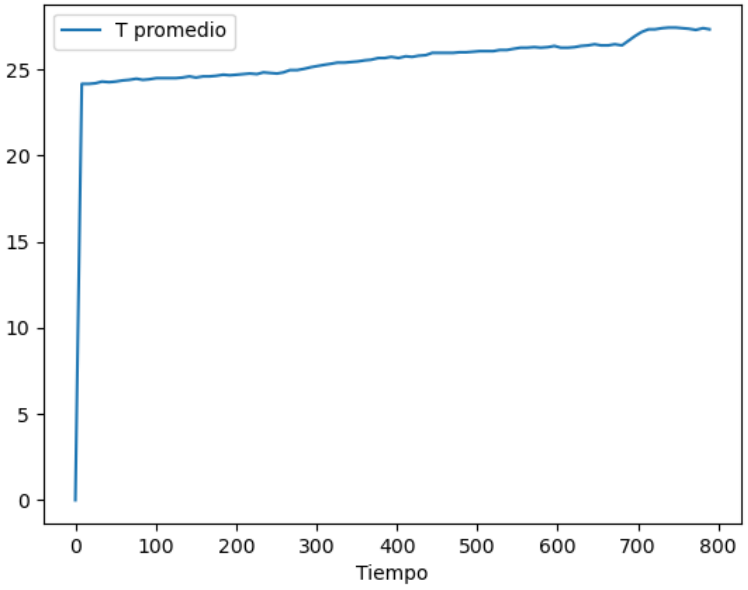

Ilustración 5.13 Dato de telemetría: Temperatura

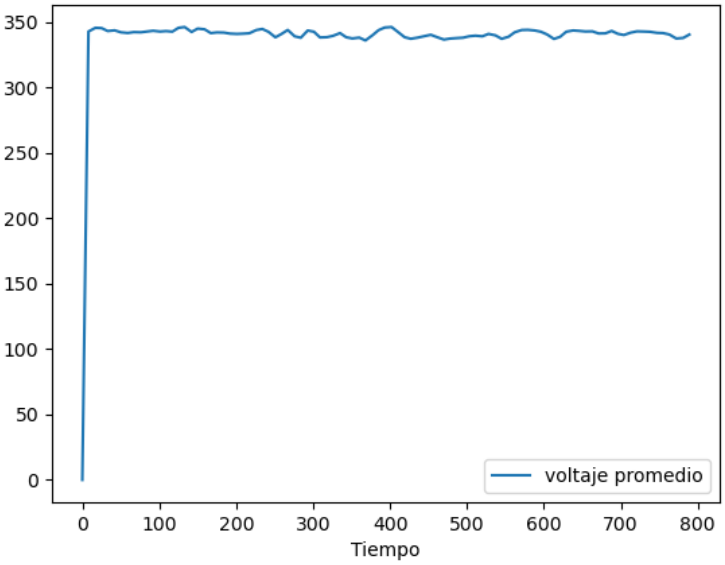

Ilustración 5.14 Dato de telemetría: Voltaje

Luego al programa se le deben ingresar lo siguiente: Las coordenadas de inicio y fin para la sección de viaje a comparar, SOC inicial igual a 36 y una temperatura ambiente e inicial de batería de 24°C y se obtiene lo siguiente

Ilustración 5.15 Ruteador

Color rojo se grafica la ruta más rápida, ruta 1, o la que implica un menor tiempo de viaje y en color azul la ruta 2 la cual implica el viaje con menor recorrido en distancia. Los resultados son los siguientes:

- Ruta 1 (Roja): 11.7 km y 11 minutos de viaje
  - SOC final: 32%
  - Temperatura final 25.7°C
  - Voltaje máximo igual a 350 V y mínimo 345 V
- Ruta 2 (Azul): 11.1 km y 12.8 minutos de viaje
  - SOC final: 31.7%
  - Temperatura final: 25.8°C
  - Voltaje máximo igual a 350 V y mínimo 344 V

Los resultados también pueden ser mostrados gráficamente:

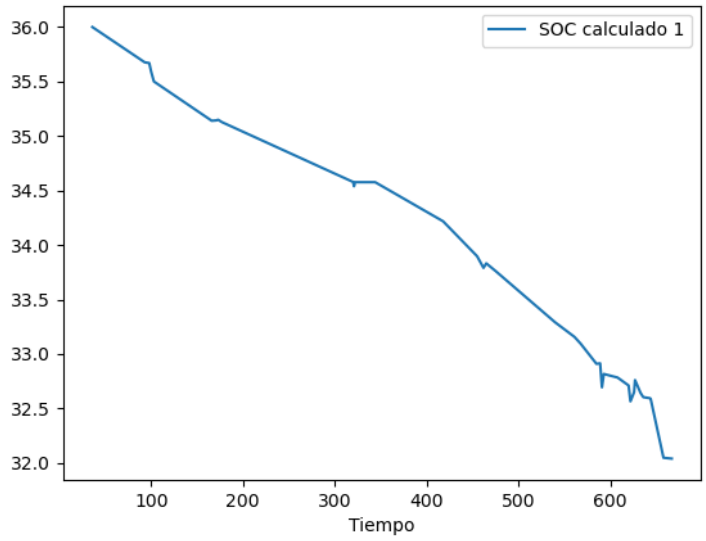

Ilustración 5.16 SOC calculado para ruta 1

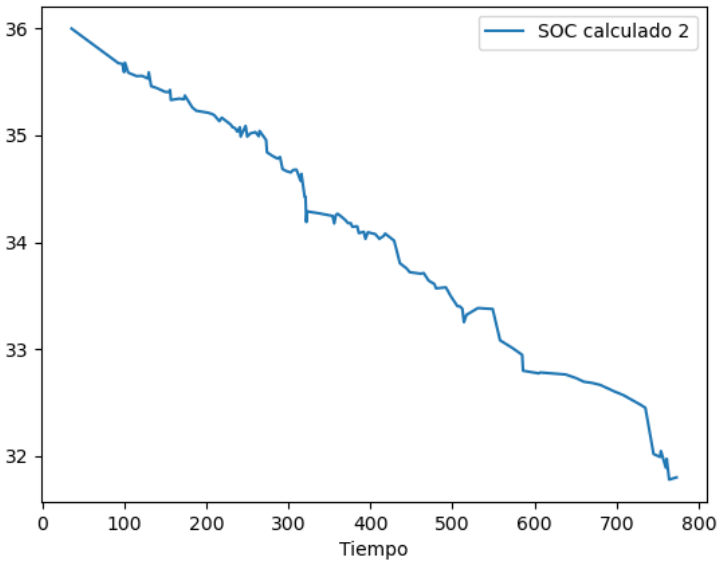

Ilustración 5.17 SOC calculado para ruta 2

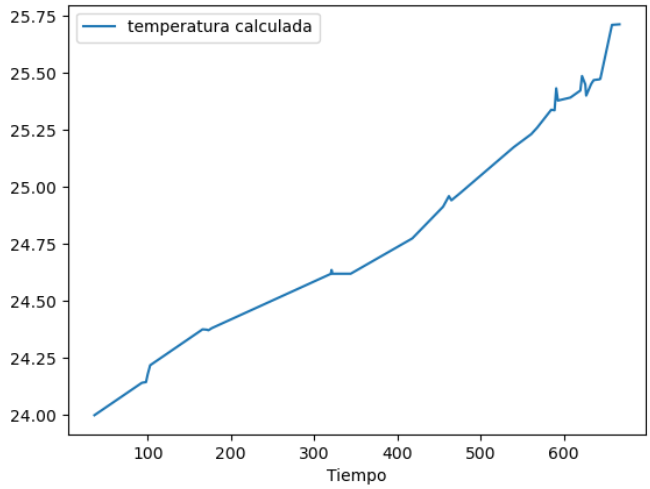

Ilustración 5.18 Temperatura calculada para ruta 1

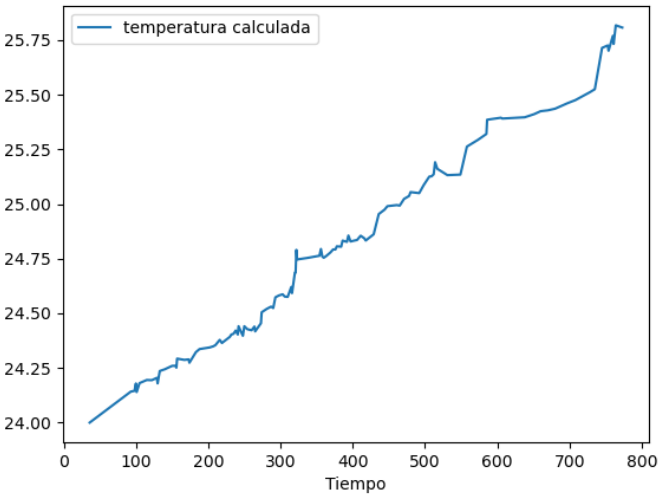

Ilustración 5.19 Temperatura calculada para ruta 2

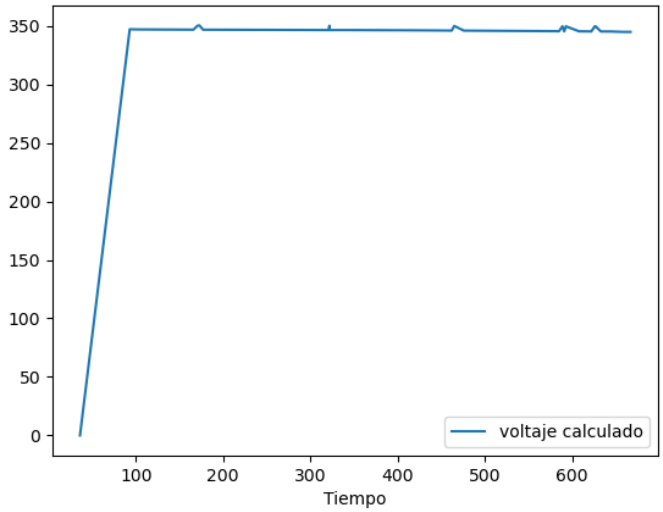

Ilustración 5.20 Voltaje calculado para ruta 1

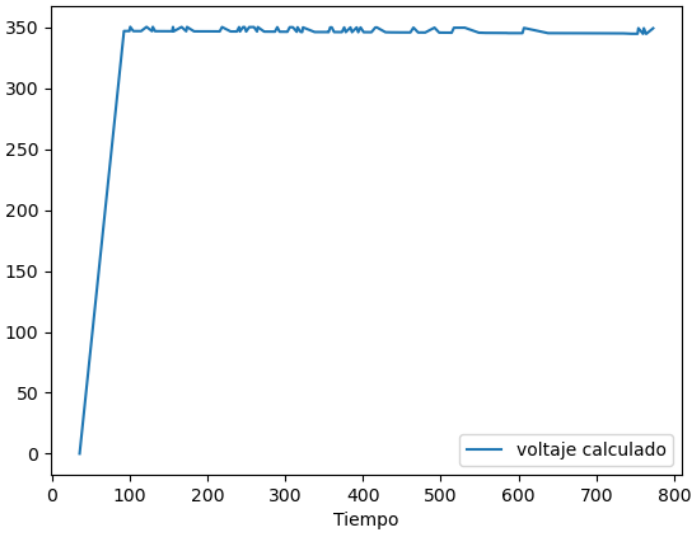

Ilustración 5.21 Voltaje calculado para ruta 2

La ruta estudiada es similar a la ruta1, como los viajes no son idénticos no se puden calcular las diferencias de datos punto a punto por lo que se tomarán los datos extremos como los puntos de comparación. Entonces las diferencias de SOC y temperatura finales son de 1 y 1.5°C y las diferencias de voltaje máximo y mínimo son de aproximadamente 4V y 10V respectivamente.

Diferencias lo suficientemente bajas como para mantener los parámetros ya que las diferencias pueden tener una explicación multifactorial que será discutida en la próxima sección.

Ejemplo Viaje 3: Viaje de ida a las cercanías de Melipilla desde Providencia nuevamente solo se toma una sección del viaje, el viaje toma alrededor de 1 hora y 6 minutos y un recorrido 82 km aproximadamente. Se registra un SOC inicial de 96.5 y un SOC final igual a 63.5 con un rendimiento de 161 [Wh/km] o 174 [Wh/km] dependiendo de cómo se mida (datos de telemetría). La temperatura inicial y final detectadas en este tramo son 17.1°C y 27.8°C respectivamente y los voltajes máximos y mínimos durante este recorrido son, respectivamente, 401V y 348V.

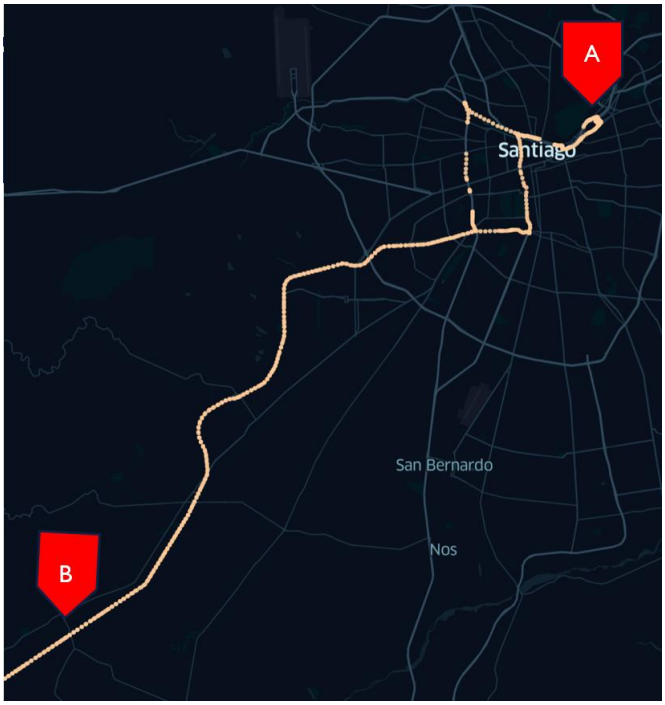

Ilustración 5.22 Sección de un viaje mostrado por kepler

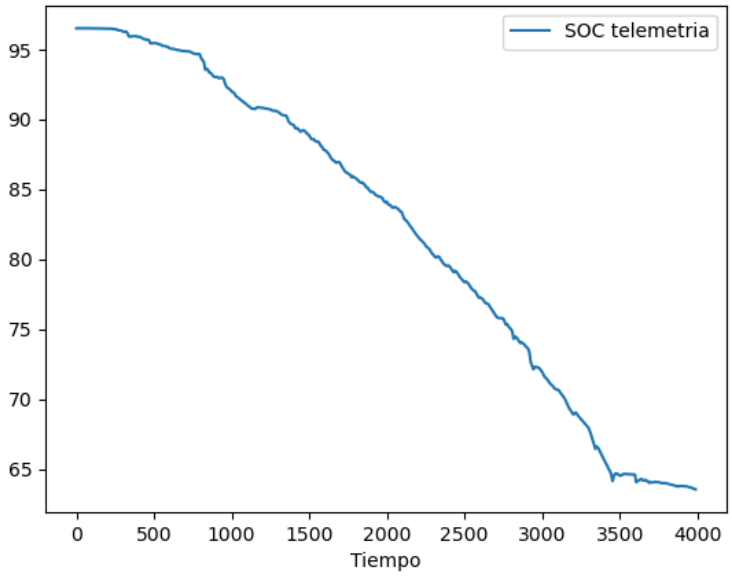

Ilustración 5.23 Datos de telemetría en ese tramo: SOC

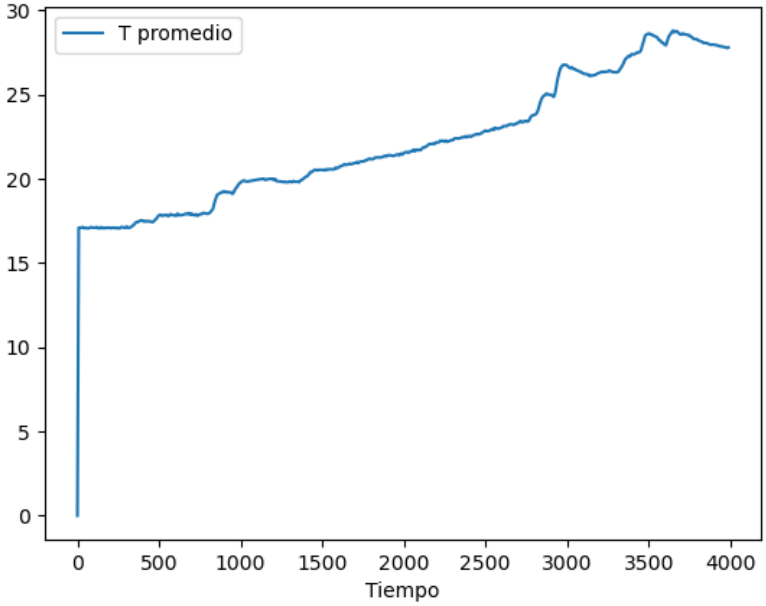

Ilustración 5.24 Datos de telemetría en ese tramo: Temperatura

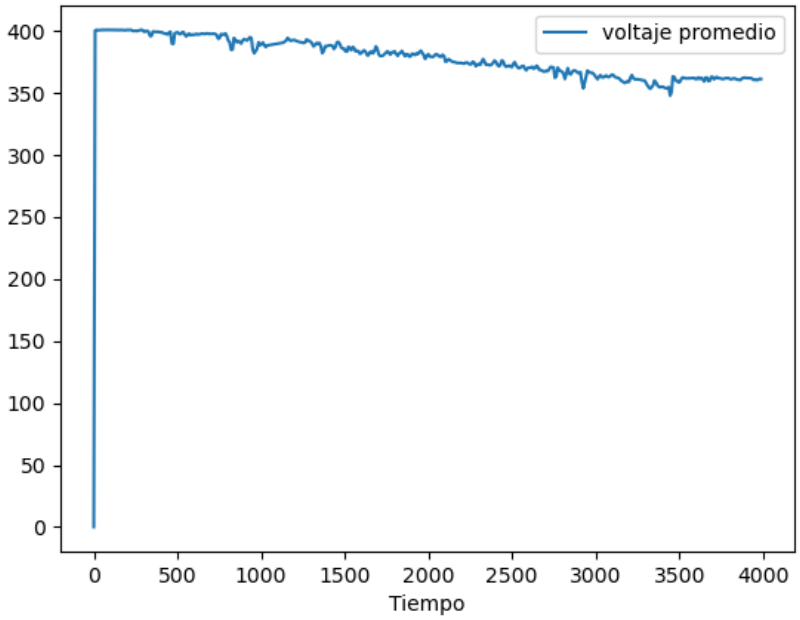

Ilustración 5.25 Datos de telemetría en ese tramo: Voltaje de batería

Se ingresan: Las coordenadas de inicio y fin para la sección de viaje a comparar, SOC inicial igual a 96 y una temperatura ambiente e inicial de batería de 17°C y se obtiene lo siguiente:

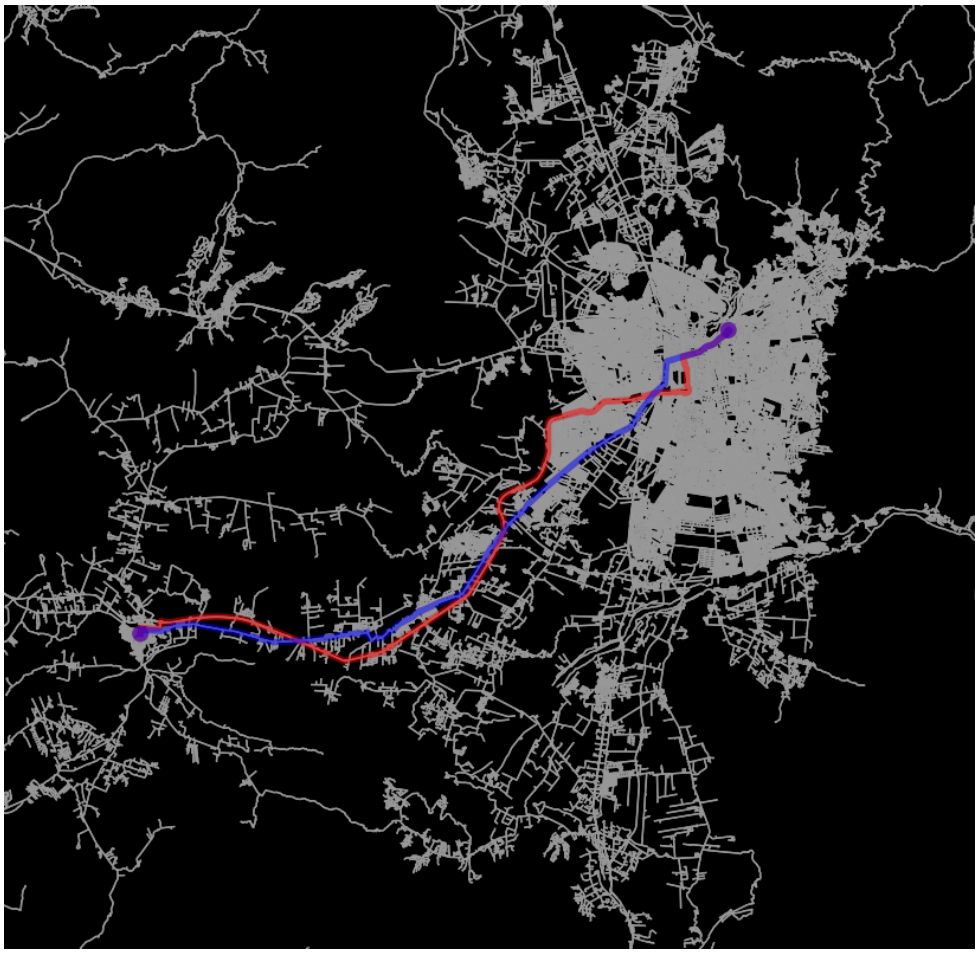

Ilustración 5.26 Ruteador de sección de viaje

Color rojo se grafica la ruta más rápida, ruta 1, o la que implica un menor tiempo de viaje y en color azul la ruta 2 la cual implica el viaje con menor recorrido en distancia. Los resultados son los siguientes:

- Ruta 1 (Roja): 77.6 km y 50 minutos de viaje
  - SOC final: 57.9%
  - Temperatura final 32.4°C
  - Voltaje máximo igual a 392 V y mínimo 359 V
- Ruta 2 (Azul): 70.7 km y 1 hora y 20 minutos de viaje
  - SOC final: 80%
  - Temperatura final: 23°C
  - Voltaje máximo igual a 392 V y mínimo 376 V

Los resultados también pueden ser mostrados gráficamente:

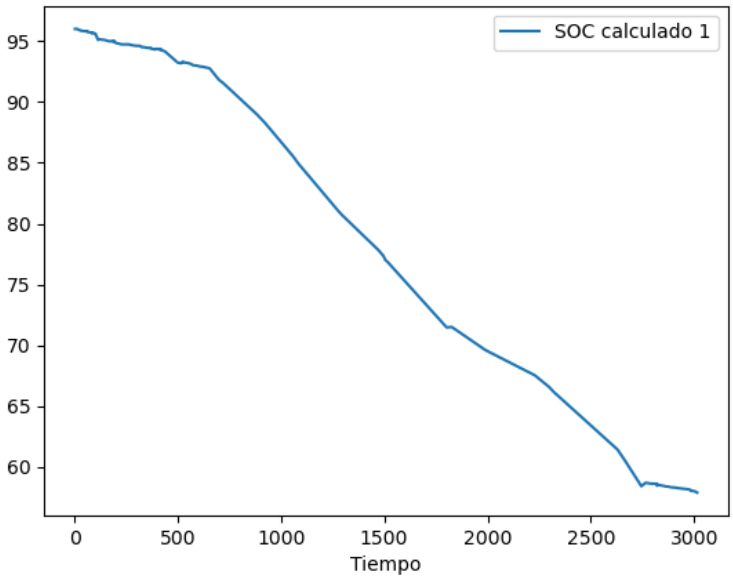

Ilustración 5.27 SOC calculado para ruta 1

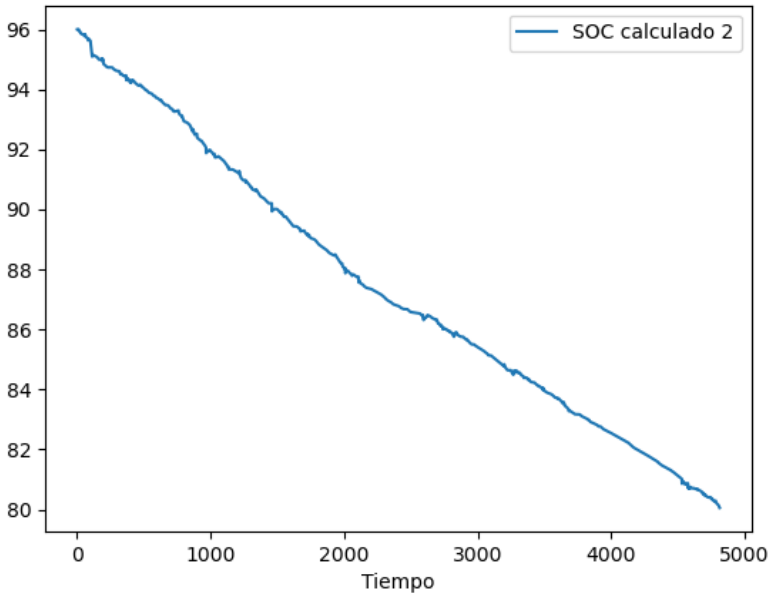

Ilustración 5.28 SOC calculado para ruta 2

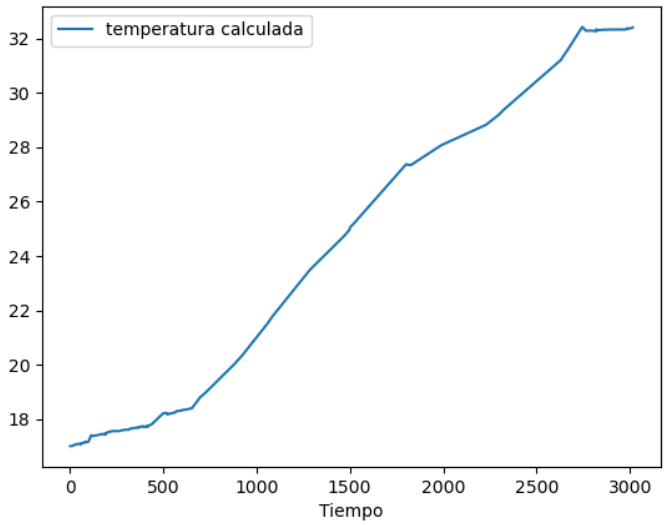

Ilustración 5.29 Temperatura calculada para ruta 1

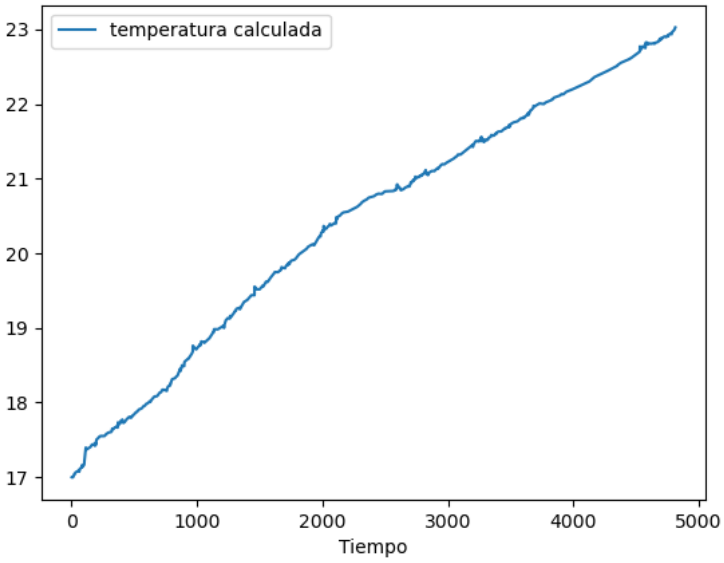

Ilustración 5.30 Temperatura calculada para ruta 2

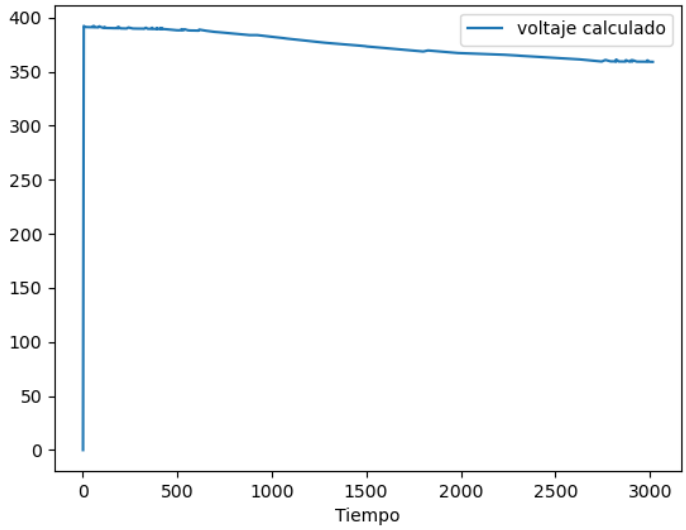

Ilustración 5.31 Voltaje calculado para ruta 1

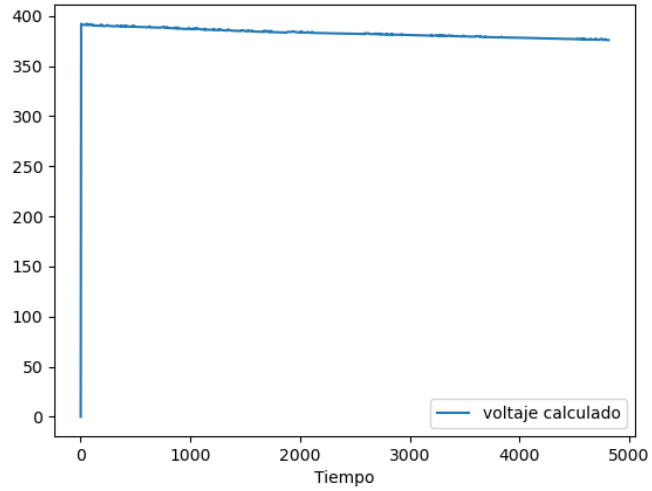

Ilustración 5.32 Voltaje calculado para ruta 2

La ruta estudiada es similar a la ruta1 donde las diferencias de SOC y temperatura finales son de 5.6 y 4.6°C y las diferencias de voltaje máximo y mínimo son de aproximadamente 18V y 11V respectivamente.

A modo de resumen, se expone la siguiente tabla, en ella se muestra la máxima diferencia de valores para los parámetros utilizados en los viajes de estudio (secciones de viajes registrados)

Tabla 5.3 Máxima diferencia entre dato de telemetría y dato calculado para viaje con ruta similar

Temperatura	SOC	Voltaje
Máx. diferencia Valor	Máx. diferencia Valor	Máx. diferencia Valor
8	10	21

Las diferencias presentadas están en grados Celsius, porcentaje y Volt respectivamente y pueden asociarse a múltiples factores de los cuales destacan: Viajes comparados no son idénticos en ruta (cambio de algunas calles), al tomar secciones de viaje las velocidades cercanas al inicio y final de la sección serán distintas a las planteadas por el ruteador ya que se asume deben partir desde 0 en cambio en la sección de viaje ya se tiene una velocidad inicial distinta de 0 y también pueden no terminar en velocidad 0 y finalmente todas las condiciones que pueden ir cambiando en la vida real.

5.2 Discusión

Como es de esperarse el consumo energético del vehículo está relacionado directamente con su uso, tanto en tiempo como en distancia recorrida y los factores más influyentes son: En la geografía las diferencias de altura y en las conductas de manejo la velocidad y las aceleraciones. Por lo tanto, las conductas operacionales juegan un rol fundamental en el consumo eficiente de energía.

El voltaje de la batería está relacionado directamente con el SOC por lo que se puede estimar rápidamente el SOC midiendo el voltaje de la batería por medio de un voltímetro.

La temperatura de la batería es calculada en base a la temperatura ambiente y las pérdidas de energía asociadas a su eficiencia, por lo tanto, también tienen una relación directa con su uso ya sea en ciclos de carga, descarga o regeneración. Como este vehículo en particular no cuenta con un sistema de enfriamiento activo, el sistema de protección contra las altas temperaturas es un límite máximo de potencia (distinto y más bajo a las posibilidades del motor) que se activa luego de alcanzar una temperatura cercana a los 45°C para que la temperatura de la batería no siga aumentando y no llegue a niveles peligrosos.

Las diferencias entre los datos de telemetría y los calculados por el modelo se deben a múltiples factores de las cuales se pueden nombrar:

- Condiciones climáticas: Viento, lluvias, altas/bajas temperaturas etc. Esto puede afectar la performance del vehículo y las condiciones en que se usa (uso de accesorios). En particular el uso de aire acondicionado en el vehículo implica un consumo energético basal mayor en el vehículo
- Estilo de conducción: Conductas abruptas como aceleraciones, frenados o giros bruscos, excesos de velocidad y/o ahorro de combustible inteligente en bajadas y frenadas por parte del conductor distintas a las estimadas o registradas por el programa
- Coordinación de GPS: Muchas veces puede haber datos viciados cuando el smartphone está calibrando su posición y altura. Además, puede haber diferencias de nivel que no son tomadas en cuenta en la capa descargada, como por ejemplo nuevos túneles o pasos sobre nivel lo que puede traducirse en mayor o menor consumo dependiendo del caso
- Condiciones de carretera y caminos: Desvíos, mal estado y distintos contratiempos que pueden afectar la conducción normal generando diferencias entre las posiciones estimadas y la realidad
- Horario: Horarios con mayor o menor cantidad de vehículos en carretera, provocando atochamientos u otros problemas
- Temperatura de batería: Afecta la entrega de potencia por parte de la batería no solo cambiando su eficiencia, también poniendo un límite de potencia máximo ya que, como se dijo anteriormente, el Nissan Leaf no cuenta con un sistema de enfriamiento activo por lo que limita la potencia entregada para no superar temperaturas criticas
- Condición del vehículo: Neumáticos desinflados, rodamientos desgastados, estado de la batería etc. Todos estos factores podrían influir en un sobre consumo
- Ocupantes: Afecta la masa del vehículo y por ende su potencia, este factor se puede agregar fácilmente al programa

Todos estos factores podrían llegar a ser integrados en un modelo o gemelo digital, pero muchas veces las opciones para adquirir estos datos son pagadas, como tráfico en vivo, por ejemplo, lo cual es una gran limitante a nivel académico.

Con respecto a la limitante de distancias máximas, se debe a que el programa necesita descargar todas las calles alrededor de una coordenada en particular, puede ser nombrando una región o el punto medio entre las coordenadas de inicio y fin de un viaje, por lo tanto, mientras mayor sea la distancia que haya entre los puntos inicial y final de un viaje mayor es la cantidad de calles que deben descargarse. De manera alternativa se trabajó con una alternativa que implica una mayor cantidad de pasos y la creación de una cuenta en una red social, este código se encuentra adjunto (Anexo A 8.1.5)

.

6 Propuesta de trabajo y conclusión

6.1 Conclusión

El cuidado del medioambiente es una tarea que debe primar en las personas debido a todo el acontecer actual (gases de efecto invernadero, toneladas de desechos y cambio climático) por lo tanto el uso eficiente de la energía es de vital importancia además, como fue visto en antecedentes, el Nissan Leaf es uno de los vehículos eléctricos más populares del mundo, y las baterías eléctricas desechadas, específicamente las de litio, son un problema aun sin solución ya que presentan un peligro físico y medioambiental, por lo que alargar su vida y mantener los cuidados es una de las tareas importantes para no generar una montaña radioactiva de desechos. Uno de los principales cuidados a nivel personal que se le puede dar a una batería eléctrica de un vehículo son: evitar las cargas rápidas, el sobrecalentamiento de las baterías y las descargas profundas.

Dicho esto, la realización de un gemelo digital de un vehículo eléctrico permitirá a su dueño tener un mejor control de su vehículo en base a una mayor visualización para finalmente darle un uso más eficiente. Además, contar con un ruteador permitirá tomar mejores decisiones con respecto a futuros viajes en base al SOC inicial y final, al tiempo que se tiene para realizar un viaje y/o el tiempo que se tiene para realizar una carga, también ayudará a disminuir la ansiedad de la autonomía, que es el temor a que el vehículo eléctrico no disponga de la carga suficiente para cumplir con su deber, ya que podría contar con una proyección que indicará si el carro puede o no realizar cierta tarea.

Otro aporte de contar con este programa es que permitirá detectar problemas de sobreconsumo en el vehículo, es decir, diferencias de SOC muy grandes entre lo planificado y lo realizado lo que será un llamado a realizar una revisión preventiva del vehículo para detectar cual o cuales son los factores que hacen que uno o más viajes gasten más energía de lo presupuestado.

Finalmente se puede decir que se cumple con los objetivos planteados ya que se cuenta con un modelo electromecánico capaz de entregar datos de vital importancia del vehículo estudiado con una diferencia en todas sus métricas menor al 20% (error simple). Este programa es lo suficientemente maleable para cambiar parámetros como masa cuando se tiene una cantidad de pasajeros conocida o temperatura cuando se tienen datos de esta y puede ser replicado o integrado fácilmente en algún otro programa que entregue datos de manera dinámica para así convertir el modelo o programa en un gemelo digital. Los datos operacionales relevantes además de los entregados por ficha técnica del vehículo son el posicionamiento y altura con una marca de tiempo, la temperatura inicial de la batería, la temperatura ambiente y la masa extra (cantidad de pasajero). El modelo utilizado para calcular el consumo energético es el modelo dinámico longitudinal, en base a esta información para calcular la temperatura del vehículo se utilizó el modelo termodinámico equivalente y finalmente, con los datos calculados previamente, para el cálculo del voltaje en batería se utilizó el modelo SOC-OVC. Todos estos modelos fueron integrados para poder obtener toda esta información en viajes ya realizados y en la planificación de viajes.

6.2 Propuestas de trabajo

- Se propone utilizar este modelo con datos de un GPS y altímetro más sofisticado para obtener datos más fidedignos y un mejor ajuste de parámetros
- Se propone utilizar este modelo con un ruteador que utilice menos recursos y así evitar las limitantes de distancia o viajes regionales y utilizar un ruteador que contenga datos de tráfico en vivo o estime tráfico según horas para tener nuevos tipos de resultados
- Se propone generar una tercera ruta al ruteador que indique la que conlleva un consumo energético menor
- Se propone utilizar este modelo en un gemelo digital de Nissan Leaf, es decir, utilizar el mismo algoritmo, formulas y parámetros para el cálculo de datos, pero con un programa que entregue datos de manera activa y dinámica para cumplir con la definición de gemelo digital

7 Bibliografía

[1] Michael Grieves, *Virtually Perfect: Driving Innovative and Lean Products Through Product Lifecycle Management*. 2011.

[2] A. Madni, C. Madni, and S. Lucero, "Leveraging Digital Twin Technology in Model-Based Systems Engineering," *Systems*, vol. 7, no. 1, p. 7, Jan. 2019, doi: 10.3390/systems7010007.

[3] Christy Pettey, "Prepare for the Impact of Digital Twins," *Smarter With Gartner*, 2017.

[4] C. Fiori, K. Ahn, and H. A. Rakha, "Power-based electric vehicle energy consumption model: Model development and validation," *Applied Energy*, vol. 168, pp. 257–268, Apr. 2016, doi: 10.1016/j.apenergy.2016.01.097.

[5] X. Wu, D. Freese, A. Cabrera, and W. A. Kitch, "Electric vehicles' energy consumption measurement and estimation," *Transportation Research Part D: Transport and Environment*, vol. 34, pp. 52–67, Jan. 2015, doi: 10.1016/j.trd.2014.10.007.

[6] J. Asamer, A. Graser, B. Heilmann, and M. Ruthmair, "Sensitivity analysis for energy demand estimation of electric vehicles," *Transportation Research Part D: Transport and Environment*, vol. 46, pp. 182–199, Jul. 2016, doi: 10.1016/j.trd.2016.03.017.

[7] M. Ehsani, Y. Gao, S. E. Gay, and A. Emadi, "Modern Electric, Hybrid Electric, and Fuel Cell Vehicles: Fundamentals, Theory, and Design."

[8] P. Espinoza, "Analisis de parametros para ajuste de modelo de potencia en vehiculos electricos," 2020.

[9] D. Carbacho, "Análisis del desempeño de un vehículo eléctrico sometido a diversas condiciones en la zona central y sur de Chile," 2020.

[10] L. Calearo, A. Thingvad, and M. Marinelli, "Modeling of Battery Electric Vehicles for Degradation Studies," 2019.

[11] E. Hosseinzadeh, R. Genieser, D. Worwood, A. Barai, J. Marco, and P. Jennings, "A systematic approach for electrochemical-thermal modelling of a large format lithium-ion battery for electric vehicle application," *Journal of Power Sources*, vol. 382, pp. 77–94, Apr. 2018, doi: 10.1016/j.jpowsour.2018.02.027.

[12] D. Worwood *et al.*, "A new approach to the internal thermal management of cylindrical battery cells for automotive applications," *Journal of Power Sources*, vol. 346, pp. 151–166, 2017, doi: 10.1016/j.jpowsour.2017.02.023.

[13] C. Weng, J. Sun, and H. Peng, “A unified open-circuit-voltage model of lithium-ion batteries for state-of-charge estimation and state-of-health monitoring,” *Journal of Power Sources*, vol. 258, pp. 228–237, Jul. 2014, doi: 10.1016/j.jpowsour.2014.02.026.

[14] I. Baccouche, S. Jemmali, B. Manai, N. Omar, and N. Essoukri Ben Amara, “Improved OCV model of a Li-ion NMC battery for online SOC estimation using the extended Kalman filter,” *Energies*, vol. 10, no. 6, Jun. 2017, doi: 10.3390/en10060764.

[15] “LeafSpy-Help-1.1.1”.

[16] “OSMnx,” *https://osmnx.readthedocs.io/en/stable/osmnx.html*, 2021.

[17] “NetworkX,” 2021.

[18] “Kepler,” *https://kepler.gl/demo*, 2021.

8 Anexos

8.1 Anexo A - Código

8.1.1 Código separador Excel

```
1.
   import numpy as np
2. import pandas as pd
3. import math as mt
4. import datetime
5. import matplotlib.pyplot as plt
6.
7. #COMIENZO Y ARREGLO DE DATOS: se quitan velocidades mayores a la que el
   auto puede llegar y se eliminan errores de gps
8.
9. datos1=pd.read_excel("C:\\Users\\claud\\Desktop\\memoria\\DATOS\\04_10_2
   019_9_34_59SEPARAR.xlsx")
10.      datos=datos1[datos1.Lat.notnull()]
11.      datos=datos1[datos1.Long.notnull()].reset_index(drop=True)
12.
13.      numfilas=len(datos)
14.
15.      delta_soc=pd.DataFrame(np.zeros((numfilas,1)),columns=["Delta
   SOC"])
16.
17.
18.      for k in range(1,numfilas):
19.          delta_soc.iloc[k,0]=datos["SOC"][k]-datos["SOC"][k-1]
20.
21.      particion=pd.DataFrame()
22.      particion=delta_soc[abs(delta_soc["Delta SOC"])>=100000]
23.      numparticiones=len(particion)
24.      particion=particion.rename_axis('aindex').reset_index()
25.
26.
27.      aux1=0
28.
29.      if numparticiones>1:
30.          for k in range(1,numparticiones):
31.              auxilio=datos[aux1:particion.iloc[k,0]]
32.              nombre=str(auxilio.iloc[0,0])
33.              nombre=nombre.replace(" ","_")
34.              nombre=nombre.replace(":","_")
35.              nombre=nombre.replace("-","_")
36.              nombre=nombre.replace("/","_")
37.              aux1=particion.iloc[k,0]
38.              auxilio.to_excel(r'C:\\Users\\claud\\Desktop\\memoria\\nue
   vosdatos\\'+nombre+'.xlsx', index = False, header=True)
39.          auxiliofinal=datos[particion.iloc[k,0]:numfilas+1]
40.          nombre=str(auxiliofinal.iloc[0,0])
41.          nombre=nombre.replace(" ","_")
```

```
            nombre=nombre.replace(":","_")
            nombre=nombre.replace("-","_")
            nombre=nombre.replace("/","_")
            auxiliofinal.to_excel(r'C:\\Users\\claud\\Desktop\\memoria\\nuevosdatos\\'+nombre+'.xlsx', index = False, header=True)


        elif numparticiones==0:
            nombre=str(datos.iloc[0,0])
            nombre=nombre.replace(" ","_")
            nombre=nombre.replace(":","_")
            nombre=nombre.replace("-","_")
            nombre=nombre.replace("/","_")
            datos.to_excel(r'C:\\Users\\claud\\Desktop\\memoria\\nuevosdatos\\'+nombre+'.xlsx', index = False, header=True)


        else:
            auxilio=datos[0:particion.iloc[0,0]]
            nombre=str(auxilio.iloc[0,0])
            nombre=nombre.replace(" ","_")
            nombre=nombre.replace(":","_")
            nombre=nombre.replace("-","_")
            nombre=nombre.replace("/","_")
            auxilio.to_excel(r'C:\\Users\\claud\\Desktop\\memoria\\nuevosdatos\\'+nombre+'.xlsx', index = False, header=True)
            auxilio1=datos[particion.iloc[0,0]:numfilas]
            nombre=str(auxilio1.iloc[0,0])
            nombre=nombre.replace(" ","_")
            nombre=nombre.replace(":","_")
            nombre=nombre.replace("-","_")
            nombre=nombre.replace("/","_")
            auxilio.to_excel(r'C:\\Users\\claud\\Desktop\\memoria\\nuevosdatos\\'+nombre+'.xlsx', index = False, header=True)

```

8.1.2 Código separador CSV

```
import numpy as np
import pandas as pd
import math as mt
import datetime
import matplotlib.pyplot as plt

#COMIENZO Y ARREGLO DE DATOS: se quitan velocidades mayores a la que el auto puede llegar y se eliminan errores de gps

datos1=pd.read_csv("C:\\Users\\claud\\Desktop\\memoria\\DATOS\\Log_JC315373_fd280.csv")
      datos=datos1[datos1.Lat.notnull()]
```

```
    datos=datos1[datos1.Long.notnull()].reset_index(drop=True)

    numfilas=len(datos)

    delta_soc=pd.DataFrame(np.zeros((numfilas,1)),columns=["Delta SOC"])


    for k in range(1,numfilas):
        delta_soc.iloc[k,0]=datos["SOC"][k]-datos["SOC"][k-1]

    particion=pd.DataFrame()
    particion=delta_soc[abs(delta_soc["Delta SOC"])>=100000]
    numparticiones=len(particion)
    particion=particion.rename_axis('aindex').reset_index()


    aux1=0

    if numparticiones>1:
        for k in range(1,numparticiones):
            auxilio=datos[aux1:particion.iloc[k,0]]
            nombre=str(auxilio.iloc[0,0])
            nombre=nombre.replace(" ","_")
            nombre=nombre.replace(":","_")
            nombre=nombre.replace("-","_")
            nombre=nombre.replace("/","_")
            aux1=particion.iloc[k,0]
            auxilio.to_excel(r'C:\\Users\\claud\\Desktop\\memoria\\nuevosdatos\\'+nombre+'.xlsx', index = False, header=True)
        auxiliofinal=datos[particion.iloc[k,0]:numfilas+1]
        nombre=str(auxiliofinal.iloc[0,0])
        nombre=nombre.replace(" ","_")
        nombre=nombre.replace(":","_")
        nombre=nombre.replace("-","_")
        nombre=nombre.replace("/","_")
        auxiliofinal.to_excel(r'C:\\Users\\claud\\Desktop\\memoria\\nuevosdatos\\'+nombre+'.xlsx', index = False, header=True)


    elif numparticiones==0:
        nombre=str(datos.iloc[0,0])
        nombre=nombre.replace(" ","_")
        nombre=nombre.replace(":","_")
        nombre=nombre.replace("-","_")
        nombre=nombre.replace("/","_")
        datos.to_excel(r'C:\\Users\\claud\\Desktop\\memoria\\nuevosdatos\\'+nombre+'.xlsx', index = False, header=True)


    else:
        auxilio=datos[0:particion.iloc[0,0]]
```

```
59.          nombre=str(auxilio.iloc[0,0])
60.          nombre=nombre.replace(" ","_")
61.          nombre=nombre.replace(":","_")
62.          nombre=nombre.replace("-","_")
63.          nombre=nombre.replace("/","_")
64.          auxilio.to_excel(r'C:\\Users\\claud\\Desktop\\memoria\\nuevosd
   atos\\'+nombre+'.xlsx', index = False, header=True)
65.          auxilio1=datos[particion.iloc[0,0]:numfilas]
66.          nombre=str(auxilio1.iloc[0,0])
67.          nombre=nombre.replace(" ","_")
68.          nombre=nombre.replace(":","_")
69.          nombre=nombre.replace("-","_")
70.          nombre=nombre.replace("/","_")
71.          auxilio.to_excel(r'C:\\Users\\claud\\Desktop\\memoria\\nuevosd
   atos\\'+nombre+'.xlsx', index = False, header=True)
72.
```

8.1.3 Código de modelo con datos GPS

```
1.
2. import numpy as np
3. import pandas as pd
4. import math as mt
5. import datetime
6. import matplotlib.pyplot as plt
7. from sklearn.linear_model import LinearRegression #Regresión Lineal con
   scikit-learn
8. from sklearn.preprocessing import PolynomialFeatures
9. from sklearn.metrics import mean_squared_error # importamos el cálculo
   del error cuadrático medio (MSE)
10.      from sklearn.metrics import r2_score
11.
12.
13.      #COMIENZO Y ARREGLO DE DATOS: se quitan velocidades mayores a la
   que el auto puede llegar y se eliminan errores de gps
14.      #datos1=pd.read_excel("C:\\Users\\claud\\Desktop\\23092019_142421e
   rror - copia.xlsx")
15.
16.      datos1=pd.read_excel("C:\\Users\\claud\\Desktop\\memoria\\nuevosda
   tos\\26_11_2019_18_11_54MEJOR.xlsx")
17.      #datos1=pd.read_excel("C:\\Users\\claud\\Desktop\\pruebbbaaa22.xls
   x")
18.      datos=datos1[datos1.Lat.notnull()]
19.      datos=datos1[datos1.Long.notnull()].reset_index(drop=True)
20.
21.      numfilas=len(datos)
22.
23.      rt=6371*1000 # metros
24.
25.
```

```
26.       distancia_plana1=pd.DataFrame(np.zeros((numfilas,1)),columns=["Dis
   tancia plana"])
27.       distancia_vertical1=pd.DataFrame(np.zeros((numfilas,1)),columns=["
   Distancia vertical"])
28.       distancia_real1=pd.DataFrame(np.zeros((numfilas,1)),columns=["Dist
   ancia real"])
29.       delta_t1=pd.DataFrame(np.zeros((numfilas,1)),columns=["Delta
   tiempo"])
30.       velocidad1=pd.DataFrame(np.zeros((numfilas,1)),columns=["Velocidad
   "])
31.       aceleracion1=pd.DataFrame(np.zeros((numfilas,1)),columns=["Acelera
   cion"])
32.       pen1=pd.DataFrame(np.zeros((numfilas,1)),columns=["Pendiente"])
33.
34.
35.       for k in range(numfilas):
36.           datos["Lat"][k]=(float(datos["Lat"][k].split()[0])-
   float(datos["Lat"][k].split()[1])/60)*mt.pi/180
37.           datos["Long"][k]=(float(datos["Long"][k].split()[0])-
   float(datos["Long"][k].split()[1])/60)*mt.pi/180
38.
39.       while True:
40.
41.           numfilas=len(datos)
42.           velocidad1=pd.DataFrame(np.zeros((numfilas,1)),columns=["Veloc
   idad"])
43.           if type(datos.iloc[1,0])==str:
44.               for k in range(1,numfilas):
45.                   delta_t1.iloc[k,0]=(datetime.datetime.strptime(datos.i
   loc[k,0],'%d/%m/%Y %H:%M:%S')-datetime.datetime.strptime(datos.iloc[k-
   1,0],'%d/%m/%Y %H:%M:%S')).total_seconds()
46.           else:
47.               for k in range(1,numfilas):
48.                   delta_t1.iloc[k,0]=(datos.iloc[k,0]-datos.iloc[k-
   1,0]).total_seconds()
49.
50.           for k in range(1,numfilas):
51.               resta_lat=(datos["Lat"][k]-datos["Lat"][k-1])/2
52.               resta_long=(datos["Long"][k]-datos["Long"][k-1])/2
53.               mult_cos=mt.cos(datos["Lat"][k])*mt.cos(datos["Lat"][k-1])
54.               distancia_plana1.iloc[k,0]=2*rt*mt.asin((mt.sin(resta_lat)
   **2+(mult_cos)*(mt.sin(resta_long)**2))**(1/2))
55.
56.           for k in range(1,numfilas):
57.               distancia_vertical1.iloc[k,0]=datos["Elv"][k]-
   datos["Elv"][k-1]
58.
59.           for k in range(1,numfilas):
60.               distancia_real1.iloc[k,0]=(distancia_plana1.iloc[k,0]**2+d
   istancia_vertical1.iloc[k,0]**2)**(1/2)
61.
62.           for k in range(1,numfilas):
```

```
63.                 pen1.iloc[k,0]=distancia_vertical1.iloc[k,0]/distancia_pla
   na1.iloc[k,0]
64.
65.             for k in range(1,numfilas):
66.                 velocidad1.iloc[k,0]=distancia_real1.iloc[k,0]/delta_t1.il
   oc[k,0]
67.
68.
69.             for k in range(1,numfilas):
70.                 aceleracion1.iloc[k,0]=(velocidad1.iloc[k,0]-
   velocidad1.iloc[k-1,0])/(delta_t1.iloc[k,0])
71.
72.             #sys.exit()
73.
74.             datos=pd.concat([datos,velocidad1],axis=1,)
75.
76.             if all(datos['Velocidad']<42):
77.                 break
78.
79.             else:
80.                 datos=datos[datos["Velocidad"]<=42].reset_index(drop=True)
81.                 datos=datos.drop(['Velocidad'], axis=1)
82.
83.
84.
85.         #%%
86.
87.         #NUEVO COMIENZO
88.
89.
90.         datos=datos.drop(['Velocidad'], axis=1)
91.
92.
93.         numfilas=len(datos)
94.
95.
96.         #PARAMETROS
97.         rt=6371*1000 # metros
98.         daire=1.25 # kg/m3
99.         af=2.76 # m2
100.        cd=0.28 # [0,2-0,4] dependiendo de la velocidad, es estima como
   constante
101.        g=9.81 # [9,764-9,834]
102.        m=1580 # kg
103.        cr=2.2806 # [1-4], 1,75 inicial segun cierta bibliografia
104.        c1=0.0328 # [0,03-0,06]
105.        c2=4.575 # [4-8]
106.        vnom=360
107.
108.        P0=300 #potencia de bateria solo por estar encendido moda 317
   promedio 447 mediana 288 maximo 31530 minimo 2.08 con filtros de
   velocidad=0 aceleracion=0 y dif soc entre 50 y 50
```

```
109.     #promedio parado 162 los datos anteriores son cuando v=0
110.     #Effect of Ambient Temperature on Electric Vehicles' Energy
   Consumption and Range: Model Definition and Sensitivity Analysis Based
   on Nissan Leaf Data
111.
112.     #eficiencias
113.     efic_bat=0.93 #90 a95% se asume por biblio  de la bateria sale a
   P0 y al motor
114.     efic_motor=0.91 #Hesham A Rakha, Kyoungho Ahn, Kevin Moran, Bart
   Saerens, and Eric Van Den Bulck. Virginia Tech Comprehensive Power-Based
   Fuel Consumption Model : Model development and testing. Transportation
   Research Part D, 16(7):492–503, 2011.
115.     efic_trans=0.92 #SAE International. 2011 Nissan Leaf vehicle
   overview; 2011.
116.     #Regenera: Yimin Gao, Liang Chu, and Mehrdad Ehsani. Design and
   control principles of hybrid braking system for EV, HEV and FCV. VPPC
   2007 - Proceedings of the 2007 IEEE Vehicle Power and Propulsion
   Conference, (1):384–391, 2007.
117.     #me convendrian numeros más grandes
118.     energia_vehiculo=vnom*datos.iloc[0,7]/10000 #Vnom por AHr
119.     energia_in=(energia_vehiculo*datos.iloc[0,6]/10000)/100 #Soc por
   energia inicial
120.     soc_inicial=datos.iloc[0,6]/10000
121.     energia_fin=(energia_vehiculo*datos.iloc[numfilas-1,6]/10000)/100
122.     soc_final=datos.iloc[numfilas-1,6]/10000
123.
124.
125.     #Definicion de variables a utilizar
126.     distancia_plana=pd.DataFrame(np.zeros((numfilas,1)),columns=["Dist
   ancia plana"])
127.     distancia_vertical=pd.DataFrame(np.zeros((numfilas,1)),columns=["D
   istancia vertical"])
128.     distancia_real=pd.DataFrame(np.zeros((numfilas,1)),columns=["Dista
   ncia real"])
129.     delta_t=pd.DataFrame(np.zeros((numfilas,1)),columns=["Delta
   tiempo"])
130.     delta_soc=pd.DataFrame(np.zeros((numfilas,1)),columns=["Delta
   SOC"])
131.     velocidad=pd.DataFrame(np.zeros((numfilas,1)),columns=["Velocidad"
   ])
132.     aceleracion=pd.DataFrame(np.zeros((numfilas,1)),columns=["Acelerac
   ion"])
133.     pen=pd.DataFrame(np.zeros((numfilas,1)),columns=["Pendiente"])
134.     alpha=pd.DataFrame(np.zeros((numfilas,1)),columns=["Angulo"])
135.     f_aer=pd.DataFrame(np.zeros((numfilas,1)),columns=["Fuerza
   Aerodinamica"])
136.     f_pendiente=pd.DataFrame(np.zeros((numfilas,1)),columns=["Fuerza
   de Pendiente"])
137.     f_rol=pd.DataFrame(np.zeros((numfilas,1)),columns=["Fuerza de
   Rolling"])
138.     f_mov=pd.DataFrame(np.zeros((numfilas,1)),columns=["Fuerza de
   Movimiento"])
```

```
139.     pot_rueda=pd.DataFrame(np.zeros((numfilas,1)),columns=["Potencia
   de Rueda"])
140.     pot_bat=pd.DataFrame(np.zeros((numfilas,1)),columns=["Potencia de
   Bateria"])
141.     efic_reg=pd.DataFrame(np.zeros((numfilas,1)),columns=["Eficiencia
   regeneracion"])
142.     energia_i=pd.DataFrame(np.zeros((numfilas,1)),columns=["Energia
   restante"])
143.     energia_gid=pd.DataFrame(np.zeros((numfilas,1)),columns=["Energia
   gid"])
144.
145.     potencia_basal=pd.DataFrame(np.zeros((numfilas,1)),columns=["Poten
   cia basal"])
146.     potencia_motorecu=pd.DataFrame(np.zeros((numfilas,1)),columns=["Po
   wer motor"])
147.     potencia_totalecu=pd.DataFrame(np.zeros((numfilas,1)),columns=["Po
   wer total"])
148.
149.
150.     p_aer=pd.DataFrame(np.zeros((numfilas,1)),columns=["potencia
   Aerodinamica"])
151.     p_pendiente=pd.DataFrame(np.zeros((numfilas,1)),columns=["potencia
   de Pendiente"])
152.     p_rol=pd.DataFrame(np.zeros((numfilas,1)),columns=["potencia de
   Rolling"])
153.     p_mov=pd.DataFrame(np.zeros((numfilas,1)),columns=["potencia de
   Movimiento"])
154.
155.
156.
157.     #Formulas
158.     if type(datos.iloc[1,0])==str:
159.         for k in range(1,numfilas):
160.             delta_t.iloc[k,0]=(datetime.datetime.strptime(datos.iloc[k
   ,0],'%d/%m/%Y %H:%M:%S')-datetime.datetime.strptime(datos.iloc[k-
   1,0],'%d/%m/%Y %H:%M:%S')).total_seconds()
161.     else:
162.         for k in range(1,numfilas):
163.             delta_t.iloc[k,0]=(datos.iloc[k,0]-datos.iloc[k-
   1,0]).total_seconds()
164.
165.
166.     for k in range(1,numfilas):
167.         resta_lat=(datos["Lat"][k]-datos["Lat"][k-1])/2
168.         resta_long=(datos["Long"][k]-datos["Long"][k-1])/2
169.         mult_cos=mt.cos(datos["Lat"][k])*mt.cos(datos["Lat"][k-1])
170.         distancia_plana.iloc[k,0]=2*rt*mt.asin((mt.sin(resta_lat)**2+(
   mult_cos)*(mt.sin(resta_long)**2))**(1/2))
171.
172.     for k in range(1,numfilas):
173.         distancia_vertical.iloc[k,0]=datos["Elv"][k]-datos["Elv"][k-1]
174.
```

```
    for k in range(1,numfilas):
        distancia_real.iloc[k,0]=(distancia_plana.iloc[k,0]**2+distanc
    ia_vertical.iloc[k,0]**2)**(1/2)

    for k in range(1,numfilas):
        velocidad.iloc[k,0]=distancia_real.iloc[k,0]/delta_t.iloc[k,0]

    for k in range(1,numfilas):
        aceleracion.iloc[k,0]=(velocidad.iloc[k,0]-velocidad.iloc[k-
    1,0])/(delta_t.iloc[k,0])


    for k in range(1,numfilas):
        pen.iloc[k,0]=distancia_vertical.iloc[k,0]/distancia_plana.ilo
    c[k,0]


    for k in range(1,numfilas):
        alpha.iloc[k,0]=mt.atan((distancia_vertical.iloc[k,0])/distanc
    ia_plana.iloc[k,0])
        if alpha.iloc[k,0] > 0.65:
            alpha.iloc[k,0]=0.65
        elif alpha.iloc[k,0] < -0.65:
            alpha.iloc[k,0]=-0.65
        else :
            alpha.iloc[k,0]=alpha.iloc[k,0]


    for k in range(1,numfilas):
        f_aer.iloc[k,0]=(1/2)*daire*af*cd*(velocidad.iloc[k,0]**2)

    for k in range(1,numfilas):
        p_aer.iloc[k,0]=f_aer.iloc[k,0]*velocidad.iloc[k,0]


    for k in range(1,numfilas):
        f_pendiente.iloc[k,0]=m*g*mt.sin(alpha.iloc[k,0])

    for k in range(1,numfilas):
        p_pendiente.iloc[k,0]=f_pendiente.iloc[k,0]*velocidad.iloc[k,0
    ]


    for k in range(1,numfilas):
        f_rol.iloc[k,0]=m*g*mt.cos(alpha.iloc[k,0])*(cr/1000)*(c1*velo
    cidad.iloc[k,0]+c2)

    for k in range(1,numfilas):
        p_rol.iloc[k,0]=f_rol.iloc[k,0]*velocidad.iloc[k,0]


    for k in range(1,numfilas):
```

```
221.         f_mov.iloc[k,0]=m*aceleracion.iloc[k,0]
222.
223.     for k in range(1,numfilas):
224.         p_mov.iloc[k,0]=f_mov.iloc[k,0]*velocidad.iloc[k,0]
225.
226.
227.     for k in range(1,numfilas):
228.         pot_rueda.iloc[k,0]=(f_mov.iloc[k,0]+f_rol.iloc[k,0]+f_pendien
    te.iloc[k,0]+f_aer.iloc[k,0])*velocidad.iloc[k,0]
229.
230.     for k in range(1,numfilas):
231.         pot_bat.iloc[k,0]=datos["Pack Volts"][k]*datos["Pack Amps"][k]
232.
233.
234.     for k in range(1,numfilas):
235.         if aceleracion.iloc[k,0]>=0:
236.             efic_reg.iloc[k,0]=0
237.         else:
238.             efic_reg.iloc[k,0]=mt.exp(0.0411/(abs(aceleracion.iloc[k,0
    ])))**(-1)
239.
240.     #pór que me dan mas de 1?
241.
242.     for k in range(1,numfilas):
243.         energia_i.iloc[k,0]=energia_vehiculo*(datos.iloc[k,6]/10000)/1
    00
244.
245.
246.     for k in range(1,numfilas):
247.         delta_soc.iloc[k,0]=datos["SOC"][k]-datos["SOC"][k-1]
248.
249.
250.     pot_batcalculada=pd.DataFrame(np.zeros((numfilas,1)),columns=["Pot
    encia de Bateria calculada"])
251.
252.     aux1=1
253.     aux2=1
254.     aux3=1
255.     #
256.     #if datos["Elv"][0]-datos["Elv"][numfilas-1]<0:
257.     #    aux3=0.9
258.     #else:
259.     #    aux3=0.8
260.
261.     for k in range(1,numfilas):
262.         if pot_rueda.iloc[k,0]>=0:
263.             aux1=1
264.             aux2=0
265.         else:
266.             aux1=0
267.             aux2=1
```

```
268.         pot_batcalculada.iloc[k,0]=(((aux1*pot_rueda.iloc[k,0]+aux2*po
   t_rueda.iloc[k,0]*efic_reg.iloc[k,0])/(efic_bat*efic_motor*efic_trans))+
   P0)*aux3
269.
270.
271.     energiacal_1=pd.DataFrame(np.zeros((numfilas,1)),columns=["Energia
   1"])
272.     energiacal_2=pd.DataFrame(np.zeros((numfilas,1)),columns=["Energia
   2"])
273.
274.     for k in range(1,numfilas):
275.         energiacal_1.iloc[k,0]=pot_batcalculada.iloc[k,0]*delta_t.iloc
   [k,0]
276.
277.     for k in range(1,numfilas):
278.         energiacal_2.iloc[k,0]=pot_bat.iloc[k,0]*delta_t.iloc[k,0]
279.
280.
281.     for k in range(1,numfilas):
282.         energia_gid.iloc[k,0]=77.5*(datos.iloc[k,5])
283.     #energia que queda en bateria
284.
285.     for k in range(1,numfilas):
286.         potencia_basal.iloc[k,0]=(datos.iloc[k,136])*100+(datos.iloc[k
   ,137])*250+(datos.iloc[k,139])*50+(datos.iloc[k,140])*250
287.
288.     for k in range(1,numfilas):
289.         potencia_motorecu.iloc[k,0]=(datos.iloc[k,135])
290.
291.     for k in range(1,numfilas):
292.         potencia_totalecu.iloc[k,0]=(potencia_motorecu.iloc[k,0])+(pot
   encia_basal.iloc[k,0])
293.
294.     distancia_rec=  datos.iloc[numfilas-1,123] - datos.iloc[1,123]
295.
296.     energiautilizada1=energiacal_1['Energia 1'].sum()
297.     energiautilizada2=energiacal_2['Energia 2'].sum()
298.     energiautilizada3=energia_in-energia_fin
299.
300.
301.
302.     seno=pd.DataFrame(np.zeros((numfilas,1)),columns=["SENO"])
303.
304.     for k in range(1,numfilas):
305.         seno.iloc[k,0]=mt.sin(alpha.iloc[k,0])
306.
307.
308.     distancia_recorrida=(distancia_real.sum())
309.
310.     rendimiento1=energiautilizada1/(3600*(distancia_recorrida/1000))
   #Wh/km
```

```
    rendimiento2=energiautilizada2/(3600*(distancia_recorrida/1000))
#Wh/km
    rendimiento3=energiautilizada3/(distancia_recorrida/1000)
#Wh/km



    soc_cal=pd.DataFrame(np.zeros((numfilas,1)),columns=["SOC
calculado"])
    energiacal_1=energiacal_1['Energia 1'].fillna(0)

    sumatoria=0

    for k in range(1,numfilas):
        sumatoria=sumatoria + energiacal_1[k]/3600
        soc_cal.iloc[k,0]=((energia_vehiculo*(soc_inicial/100) -
sumatoria)/energia_vehiculo)*100

    soc_finalcalculado=((energia_vehiculo*(soc_inicial/100) -
energiautilizada1/3600)/energia_vehiculo)*100


    #
    temperatura_prom=pd.DataFrame(np.zeros((numfilas,1)),columns=["T
promedio"])
    temperatura=[]
    for k in range(1,numfilas):
        temperatura=[datos.iloc[k,16],datos.iloc[k,18],None,datos.iloc
[k,22]]
        temperatura=[i for i in temperatura if i is not None]
        temperatura_prom.iloc[k,0]=np.mean(temperatura)



    diftemperatura_prom=pd.DataFrame(np.zeros((numfilas,1)),columns=["
delta T promedio"])
    for k in range(1,numfilas):
        diftemperatura_prom.iloc[k,0]=temperatura_prom.iloc[k,0]-
temperatura_prom.iloc[k-1,0]


    difpotencia_prom=pd.DataFrame(np.zeros((numfilas,1)),columns=["del
ta Potencia"])
    for k in range(1,numfilas):
        difpotencia_prom.iloc[k,0]=(pot_rueda.iloc[k,0])-
(pot_rueda.iloc[k-1,0])

    difpotencia_prom=difpotencia_prom.fillna(0)

    tiempo_seg=pd.DataFrame(np.zeros((numfilas,1)),columns=["Tiempo"])
    time=0

```

```
    for k in range(1,numfilas):
        tiempo_seg.iloc[k,0]=(time+delta_t.iloc[k,0])
        time=tiempo_seg.iloc[k,0]

    soc_tele=pd.DataFrame(np.zeros((numfilas,1)),columns=["SOC
  telemetria"])
    for k in range(0,numfilas):
        soc_tele.iloc[k,0]=(datos["SOC"][k])/10000


    aa=pd.concat([tiempo_seg,soc_tele,soc_cal],axis=1,)
    plt.figure(1)

    ax=plt.gca()

    aa.plot(kind='line',x='Tiempo',y='SOC telemetria',ax=ax)
    aa.plot(kind='line',x='Tiempo',y='SOC
  calculado',color='red',ax=ax)


    #Calculo temperatura

    rth=0.0731 #resistencia termica
    cth=229680 #capacidad termica
    t0=47.6 #temperatura de la batería si es que la teniamos de antes
  si no se sabe completar con t ambiente
    tamb=25

    temp_cal=pd.DataFrame(np.zeros((numfilas,1)),columns=["temperatura
  calculada"])
    temp_cal.iloc[0,0]=t0

    diftemp_cal=pd.DataFrame(np.zeros((numfilas,1)),columns=["delta t
  calculado"])
    pi=0

    for k in range(1,numfilas):
        if t0>=40:
            pi=min(10000,pot_batcalculada.iloc[k,0])
        else:
            pi=pot_batcalculada.iloc[k,0]
        if (np.isnan(pot_batcalculada.iloc[k,0])):
            diftemp_cal.iloc[k,0]=0
        else:
            diftemp_cal.iloc[k,0]=((1/rth)*(tamb-temp_cal.iloc[k-
  1,0])+pi*(1-efic_bat))*(delta_t.iloc[k,0]/cth)
        temp_cal.iloc[k,0]=temp_cal.iloc[k-
  1,0]+diftemp_cal.iloc[k,0]


    cc=pd.concat([tiempo_seg,temperatura_prom,temp_cal],axis=1,)
    plt.figure(2)
```

```
397.     cx=plt.gca()
398.
399.     cc.plot(kind='line',x='Tiempo',y='T promedio',ax=cx)
400.     cc.plot(kind='line',x='Tiempo',y='temperatura
   calculada',color='red',ax=cx)
401.
402.
403.     #calculo del voltaje
404.     # 0 10   10 20  20  35   35 infinito
405.     #Improved OCV Model of a Li-Ion NMC Battery for Online SOC
   Estimation Using the Extended Kalman Filter
406.
407.     packvolt=pd.DataFrame(np.zeros((numfilas,1)),columns=["voltaje
   promedio"])
408.     for k in range(1,numfilas):
409.         packvolt.iloc[k,0]=(datos.iloc[k,8]+datos.iloc[k-1,8])/2
410.
411.
412.
413.     p0c=    [3.734          ,  3.629          ,  3.637          , 3.584
   ]
414.     p0d=    [3.804          ,  3.599          ,  3.604          , 3.55]
415.     p1c=    [-0.2756        ,  -0.3191        ,  -0.3091        ,  -
   0.4868]
416.     p1d=    [-0.3487        ,  -0.2933        ,  -2.2803        ,  -
   0.4573]
417.     p2c=    [8.013*0.00001  ,  6.952*0.00001  ,  7.033*0.00001  ,  6.21
   2*0.00001]
418.     p2d=    [8.838*0.00001  ,  6.9*0.00001    ,  6.871*0.00001  ,  6.02
   *0.00001]
419.     alfa1c=[-0.001155       ,  -0.0005411     ,  -0.0005747     ,  -
   0.0001919]
420.     alfa1d=[-0.001618       ,  -0.0004687     ,  -0.0004523     ,  -
   6.241*0.00001]
421.     alfa2c=[-0.06674        , -0.1493         ,  -0.1366        ,  -
   0.2181]
422.     alfa2d=[-0.04724        , -0.1434         ,  -0.1341        ,  -
   0.221]
423.
424.
425.     aux4=100
426.
427.     volt_cal=pd.DataFrame(np.zeros((numfilas,1)),columns=["voltaje
   calculado"])
428.
429.
430.     for k in range(1,numfilas):
431.         if temp_cal.iloc[k-1,0]<10:
432.             aux4=0
433.         elif temp_cal.iloc[k-1,0]>=10 and temp_cal.iloc[k-1,0]<20:
434.             aux4=1
435.         elif temp_cal.iloc[k-1,0]>35:
```

```
436.             aux4=3
437.         else:
438.             aux4=2
439.         if pot_batcalculada.iloc[k,0]<0:
440.             volt_cal.iloc[k,0]=(p0c[aux4]*mt.exp(alfa1c[aux4]*soc_cal.
   iloc[k,0])+p1c[aux4]*mt.exp(alfa2c[aux4]*soc_cal.iloc[k,0])+p2c[aux4]*so
   c_cal.iloc[k,0]**2)*96 #*(3.75/3.65)
441.         else:
442.             volt_cal.iloc[k,0]=(p0d[aux4]*mt.exp(alfa1d[aux4]*soc_cal.
   iloc[k,0])+p1d[aux4]*mt.exp(alfa2d[aux4]*soc_cal.iloc[k,0])+p2d[aux4]*so
   c_cal.iloc[k,0]**2)*96 #*(3.75/3.65)
443.
444.
445.
446.
447.     bb=pd.concat([tiempo_seg,packvolt,volt_cal],axis=1,)
448.     plt.figure(3)
449.     bx=plt.gca()
450.
451.     bb.plot(kind='line',x='Tiempo',y='voltaje promedio',ax=bx)
452.     bb.plot(kind='line',x='Tiempo',y='voltaje
   calculado',color='red',ax=bx)
453.
454.
455.
456.
457.     dd=pd.concat([tiempo_seg,pot_bat,pot_batcalculada],axis=1,)
458.     plt.figure(4)
459.     dx=plt.gca()
460.
461.     dd.plot(kind='scatter',x='Tiempo',y='Potencia de Bateria',ax=dx)
462.     dd.plot(kind='scatter',x='Tiempo',y='Potencia de Bateria
   calculada',color='red',ax=dx)
463.
464.
465.
466.     dif_temperature=pd.DataFrame(np.zeros((numfilas,1)),columns=["dif
   temperatura"])
467.     dif_volta=pd.DataFrame(np.zeros((numfilas,1)),columns=["dif
   voltaje"])
468.     dif_power=pd.DataFrame(np.zeros((numfilas,1)),columns=["dif
   potencia"])
469.     dif_soc=pd.DataFrame(np.zeros((numfilas,1)),columns=["dif soc"])
470.
471.
472.     for k in range(1,numfilas):
473.         dif_temperature.iloc[k,0]=min([abs(cc.iloc[k,1]-
   cc.iloc[k,2]),abs(cc.iloc[k,1]-cc.iloc[k-1,2])])
474.         dif_volta.iloc[k,0]=min([abs(bb.iloc[k,1]-
   bb.iloc[k,2]),abs(bb.iloc[k,1]-bb.iloc[k-1,2])])
475.         dif_power.iloc[k,0]=min([abs(dd.iloc[k,1]-
   dd.iloc[k,2]),abs(dd.iloc[k,1]-dd.iloc[k-1,2])])
```

```
476.          dif_soc.iloc[k,0]=min([abs(aa.iloc[k,1]-
   aa.iloc[k,2]),abs(aa.iloc[k,1]-aa.iloc[k-1,2])])
477.
478.      diferencias=pd.concat([tiempo_seg,dif_temperature,dif_volta,dif_po
   wer,dif_soc],axis=1)
479.
480.
481.      distancia_metro=pd.DataFrame(np.zeros((numfilas,1)),columns=["Dist
   ancia"])
482.      metro=0
483.
484.      for k in range(1,numfilas):
485.          distancia_metro.iloc[k,0]=(metro+distancia_real.iloc[k,0])
486.          metro=distancia_metro.iloc[k,0]
487.
488.
489.      altura=pd.DataFrame(np.zeros((numfilas,1)),columns=["Elevacion"])
490.      for k in range(0,numfilas):
491.          altura.iloc[k,0]=datos.iloc[k,3]
492.
493.
494.      ee=pd.concat([distancia_metro,tiempo_seg,altura],axis=1,)
495.      plt.figure(5)
496.      ex=plt.gca()
497.
498.      ee.plot(kind='line',x='Distancia',y='Elevacion',ax=ex)
499.
500.
501.      plt.figure(6)
502.      ex=plt.gca()
503.
504.      ee.plot(kind='line',x='Tiempo',y='Elevacion',ax=ex)
505.
```

8.1.4 Código de modelo con ruteador

```
1.
2. import pandas as pd
3. from IPython.display import display
4. import geopandas as gpd
5. from shapely.geometry import Point, LineString
6. import plotly_express as px
7. import networkx as nx
8. import osmnx as ox
9. import requests
10.      import urllib
11.      import numpy as np
12.      import math as mt
13.      import matplotlib.pyplot as plt
```

```
14.
15.      vnom=350 #360 se usa a veces en bibliografia de nissan leaf
16.      soc_inicial=96
17.      energia_vehiculo=40000 #deberia ir elm ultimo valor obtenido a
   traves del leaf spy siendo el resultado del ultimo (Ahr*Vnom)/10000  Ej:
   1096564*360/10000=39476.304
18.      Energia_i=energia_vehiculo*soc_inicial/100
19.
20.      rt=6371*1000 # metros
21.
22.      # start = (-33.42355467,-70.61493883)
23.      # end = (-33.68365467,-71.21784)
24.      #VIAJE DE PROVIDENCIA A MELIPILLA
25.
26.
27.      # start = (-33.42355467,-70.61493883)
28.      # end = (-33.019527,-71.564214)
29.      #viñA
30.
31.
32.      end = (-36.834762,-73.05486967)
33.      start = (-36.76613767,-73.15967783)
34.      #cONCE
35.
36.
37.      # start = (-35.532235,-71.687948)
38.      # end = (-36.634364,-72.194452)
39.      #mewjorchillan talca
40.
41.      t0=17 #temperatura de la batería si es que la teniamos de antes si
   no se sabe completar con t ambiente
42.      tamb=17
43.      # end = (-33.602389,-70.6970735)
44.      # start = (-33.669920, -70.743906)
45.      # MICASAlh
46.
47.      medio=((start[0]+end[0])/2,(start[1]+end[1])/2)
48.
49.      ox.config(use_cache=True, log_console=True)
50.
51.
52.
53.      def create_graph(loc, dist, transport_mode, loc_type="address"):
54.          """Transport mode = 'walk', 'bike', 'drive', 'drive_service',
   'all', 'all_private', 'none'"""
55.          if loc_type == "address":
56.              G = ox.graph_from_address(loc, dist=dist, network_type=tra
   nsport_mode)
57.          elif loc_type == "points":
58.              G = ox.graph_from_point(loc, dist=dist, network_type=trans
   port_mode )
59.          return G
```

```
60.
61.      G=create_graph("Concepcion chile",8000,"drive")
62.      # G=create_graph("Region metropolitana chile",80000,"drive")
63.
64.      #plotea region
65.      #plt.figure(1)
66.      fig1, ax = ox.plot_graph(G)
67.
68.
69.      G = ox.add_edge_speeds(G)
70.      G = ox.add_edge_travel_times(G)
71.      raster_path="./eleva/CHL1_alt.vrt"
72.      G = ox.add_node_elevations_raster(G,raster_path,cpus=1)
73.
74.
75.      start_node = ox.get_nearest_node(G, start)
76.      end_node = ox.get_nearest_node(G, end)
77.      # Ruta mas corta en tiempo y Ruta mas corta en distancia
78.
79.
80.      route1 = nx.shortest_path(G, start_node, end_node, weight='travel_
   time')
81.      route2 = nx.shortest_path(G, start_node, end_node, weight='length'
   )
82.
83.
84.
85.      # #Plotea la region y lñas rutas
86.      # #plt.figure(2)
87.      fig2, bx = ox.plot_graph_routes(G, [route1,route2], route_colors=[
   'r','b'], route_linewidth=1, node_size=0, bgcolor='k',show=False,close=F
   alse)
88.      # #plt.show(block=True)
89.      # #plt.ioff()
90.      # # plt.gcf().show
91.
92.
93.      node_start1 = []
94.      node_end1 = []
95.      X_to1 = []
96.      Y_to1 = []
97.      X_from1 = []
98.      Y_from1 = []
99.      length1 = []
100.     travel_time1 = []
101.     elevation_from1 = []
102.     elevation_to1 = []
103.     difh1=[]
104.
105.     node_start2 = []
106.     node_end2 = []
107.     X_to2 = []
```

```
    Y_to2 = []
    X_from2 = []
    Y_from2 = []
    length2 = []
    travel_time2 = []
    elevation_from2 = []
    elevation_to2 = []
    difh2=[]


    for u, v in zip(route1[:-1], route1[1:]):
        node_start1.append(u)
        node_end1.append(v)
        length1.append(round(G.edges[(u, v, 0)]['length']))
        travel_time1.append(round(G.edges[(u, v, 0)]['travel_time']))
        X_from1.append(G.nodes[u]['x'])
        Y_from1.append(G.nodes[u]['y'])
        X_to1.append(G.nodes[v]['x'])
        Y_to1.append(G.nodes[v]['y'])
        elevation_from1.append(G.nodes[u]['elevation'])
        elevation_to1.append(G.nodes[v]['elevation'])



    for u, v in zip(route2[:-1], route2[1:]):
        node_start2.append(u)
        node_end2.append(v)
        length2.append(round(G.edges[(u, v, 0)]['length']))
        travel_time2.append(round(G.edges[(u, v, 0)]['travel_time']))
        X_from2.append(G.nodes[u]['x'])
        Y_from2.append(G.nodes[u]['y'])
        X_to2.append(G.nodes[v]['x'])
        Y_to2.append(G.nodes[v]['y'])
        elevation_from2.append(G.nodes[u]['elevation'])
        elevation_to2.append(G.nodes[v]['elevation'])

    numfilas1=len(elevation_to1)
    numfilas2=len(elevation_to2)

    for k in range(0,numfilas1):
        difh1.append(elevation_to1[k]-elevation_from1[k])

    for k in range(0,numfilas2):
        difh2.append(elevation_to2[k]-elevation_from2[k])



    df1 = pd.DataFrame(list(zip(node_start1, node_end1, X_from1, Y_fro
   m1, X_to1, Y_to1, length1, travel_time1, difh1)),columns=["node_start","
   node_end","X_from","Y_from","X_to","Y_to","Distancia Horizontal
   1","Delta Tiempo1","Delta h1"])
    df1.head()
```

```
157.
158.
159.     df2 = pd.DataFrame(list(zip(node_start2, node_end2, X_from2, Y_fro
   m2, X_to2, Y_to2, length2, travel_time2, difh2)),columns=["node_start","
   node_end","X_from","Y_from","X_to","Y_to","Dist Horizontal 2","Delta
   Tiempo2","Delta h2"])
160.     df2.head()
161.
162.
163.
164.     distanciareal1=[]
165.     for k in range(0,numfilas1):
166.         distanciareal1.append(((difh1[k])**2+(length1[k])**2)**(1/2))
167.
168.     distanciareal2=[]
169.     for k in range(0,numfilas2):
170.         distanciareal2.append(((difh2[k])**2+(length2[k])**2)**(1/2))
171.
172.
173.     #PARAMETROS
174.     #potencia
175.     daire=1.25 # kg/m3
176.     af=2.76 # m2
177.     cd=0.28 # [0,2-0,4] dependiendo de la velocidad, es estima como
   constante
178.     g=9.81 # [9,764-9,834]
179.     m=1580 # kg
180.     cr=2.2806 # [1-4], 1,75 inicial segun cierta bibliografia
181.     c1=0.0328 # [0,03-0,06]
182.     c2=4.575 # [4-8]
183.     vnom=360
184.
185.     P0=300 #potencia de bateria solo por estar encendido moda 317
   promedio 447 mediana 288 maximo 31530 minimo 2.08 con filtros de
   velocidad=0 aceleracion=0 y dif soc entre 50 y 50
186.
187.     #eficiencias
188.     efic_bat=0.93 #90 a95% se asume por biblio  de la bateria sale a
   P0 y al motor
189.     efic_motor=0.91 #Hesham A Rakha, Kyoungho Ahn, Kevin Moran, Bart
   Saerens, and Eric Van Den Bulck. Virginia Tech Comprehensive Power-Based
   Fuel Consumption Model : Model development and testing. Transportation
   Research Part D, 16(7):492–503, 2011.
190.     efic_trans=0.92 #SAE International. 2011 Nissan Leaf vehicle
   overview; 2011.
191.     #Regenera: Yimin Gao, Liang Chu, and Mehrdad Ehsani. Design and
   control principles of hybrid braking system for EV, HEV and FCV. VPPC
   2007 - Proceedings of the 2007 IEEE Vehicle Power and Propulsion
   Conference, (1):384–391, 2007.
192.     #me convendrian numeros más grandes
193.
194.     #temperatura
```

```
195.     rth=0.0731 #resistencia termica
196.     cth=229680 #capacidad termica
197.
198.
199.     #calculo del voltaje
200.     # 0 10   10 20  20  35   35 infinito
201.     #Improved OCV Model of a Li-Ion NMC Battery for Online SOC
   Estimation Using the Extended Kalman Filter
202.
203.     p0c=    [3.734          ,  3.629          ,  3.637          , 3.584
   ]
204.     p0d=    [3.804          ,  3.599          ,  3.604          , 3.55]
205.     p1c=    [-0.2756        ,  -0.3191        ,  -0.3091        ,  -
   0.4868]
206.     p1d=    [-0.3487        ,  -0.2933        ,  -2.2803        ,  -
   0.4573]
207.     p2c=    [8.013*0.00001  ,  6.952*0.00001  ,  7.033*0.00001  ,  6.21
   2*0.00001]
208.     p2d=    [8.838*0.00001  ,  6.9*0.00001    ,  6.871*0.00001  ,  6.02
   *0.00001]
209.     alfa1c=[-0.001155       ,  -0.0005411     ,  -0.0005747     ,  -
   0.0001919]
210.     alfa1d=[-0.001618       ,  -0.0004687     ,  -0.0004523     ,  -
   6.241*0.00001]
211.     alfa2c=[-0.06674        , -0.1493         ,  -0.1366        ,  -
   0.2181]
212.     alfa2d=[-0.04724        , -0.1434         ,  -0.1341        ,  -
   0.221]
213.
214.
215.
216.     velocidad1=[]
217.     velocidad2=[]
218.
219.     for k in range(0,numfilas1):
220.         velocidad1.append(distanciareal1[k]/travel_time1[k])
221.
222.     for k in range(0,numfilas2):
223.         velocidad2.append(distanciareal2[k]/travel_time2[k])
224.
225.     aceleracion1=[]
226.     aceleracion2=[]
227.     aceleracion1.append(velocidad1[0]/travel_time1[0])
228.     aceleracion2.append(velocidad2[0]/travel_time2[0])
229.
230.     for k in range(1,numfilas1):
231.         aceleracion1.append((velocidad1[k]-velocidad1[k-
   1])/travel_time1[k])
232.
233.     for k in range(1,numfilas2):
234.         aceleracion2.append((velocidad2[k]-velocidad2[k-
   1])/travel_time2[k])
```

```
235.
236.
237.     alpha1=[]
238.     alpha2=[]
239.
240.     for k in range(0,numfilas1):
241.         alpha1.append(mt.atan((difh1[k])/length1[k]))
242.         if alpha1[k] > 0.65:
243.             alpha1[k]=0.65
244.         elif alpha1[k] < -0.65:
245.             alpha1[k]=-0.65
246.         else :
247.             alpha1[k]=alpha1[k]
248.
249.     for k in range(0,numfilas2):
250.         alpha2.append(mt.atan((difh2[k])/length2[k]))
251.         if alpha2[k] > 0.65:
252.             alpha2[k]=0.65
253.         elif alpha2[k] < -0.65:
254.             alpha2[k]=-0.65
255.         else :
256.             alpha2[k]=alpha2[k]
257.
258.     f_aer1=[]
259.     for k in range(0,numfilas1):
260.         f_aer1.append((1/2)*daire*af*cd*(velocidad1[k]**2))
261.
262.     p_aer1=[]
263.     for k in range(0,numfilas1):
264.         p_aer1.append(f_aer1[k]*velocidad1[k])
265.
266.     f_pendiente1=[]
267.     for k in range(0,numfilas1):
268.         f_pendiente1.append(m*g*mt.sin(alpha1[k]))
269.
270.     p_pendiente1=[]
271.     for k in range(0,numfilas1):
272.         p_pendiente1.append(f_pendiente1[k]*velocidad1[k])
273.
274.     f_rol1=[]
275.     for k in range(0,numfilas1):
276.         f_rol1.append(m*g*mt.cos(alpha1[k])*(cr/1000)*(c1*velocidad1[k
    ]+c2))
277.
278.     p_rol1=[]
279.     for k in range(0,numfilas1):
280.         p_rol1.append(f_rol1[k]*velocidad1[k])
281.
282.     f_mov1=[]
283.     for k in range(0,numfilas1):
284.         f_mov1.append(m*aceleracion1[k])
285.
```

```
    p_mov1=[]
    for k in range(0,numfilas1):
        p_mov1.append(f_mov1[k]*velocidad1[k])

    pot_rueda1=[]
    for k in range(0,numfilas1):
        pot_rueda1.append((f_mov1[k]+f_rol1[k]+f_pendiente1[k]+f_aer1[k])*velocidad1[k])

    efic_reg1=[]
    for k in range(0,numfilas1):
        if aceleracion1[k]>=0:
            efic_reg1.append(0)
        else:
            if aceleracion1[k]>=0:
                efic_reg1.append(0)
            else:
                efic_reg1.append(mt.exp(0.0411/(abs(aceleracion1[k])))**(-1))


    pot_batcalculada1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Potencia de Bateria calculada"])

    aux1=1
    aux2=1
    aux3=1

    for k in range(0,numfilas1):
        if pot_rueda1[k]>=0:
            aux1=1
            aux2=0
        else:
            aux1=0
            aux2=1
        pot_batcalculada1.iloc[k,0]=(((aux1*pot_rueda1[k]+aux2*pot_rueda1[k]*efic_reg1[k])/(efic_bat*efic_motor*efic_trans))+P0)*aux3


    f_aer2=[]
    for k in range(0,numfilas2):
        f_aer2.append((1/2)*daire*af*cd*(velocidad2[k]**2))

    p_aer2=[]
    for k in range(0,numfilas2):
        p_aer2.append(f_aer2[k]*velocidad2[k])

    f_pendiente2=[]
    for k in range(0,numfilas2):
        f_pendiente2.append(m*g*mt.sin(alpha2[k]))
```

```
334.
335.     p_pendiente2=[]
336.     for k in range(0,numfilas2):
337.         p_pendiente2.append(f_pendiente2[k]*velocidad2[k])
338.
339.     f_rol2=[]
340.     for k in range(0,numfilas2):
341.         f_rol2.append(m*g*mt.cos(alpha2[k])*(cr/1000)*(c1*velocidad2[k
    ]+c2))
342.
343.     p_rol2=[]
344.     for k in range(0,numfilas2):
345.         p_rol2.append(f_rol2[k]*velocidad2[k])
346.
347.     f_mov2=[]
348.     for k in range(0,numfilas2):
349.         f_mov2.append(m*aceleracion2[k])
350.
351.     p_mov2=[]
352.     for k in range(0,numfilas2):
353.         p_mov2.append(f_mov2[k]*velocidad2[k])
354.
355.     pot_rueda2=[]
356.     for k in range(0,numfilas2):
357.         pot_rueda2.append((f_mov2[k]+f_rol2[k]+f_pendiente2[k]+f_aer2[
    k])*velocidad2[k])
358.
359.     efic_reg2=[]
360.     for k in range(0,numfilas2):
361.         if aceleracion2[k]>=0:
362.             efic_reg2.append(0)
363.         else:
364.             if round(abs(aceleracion2[k]),4)==0:
365.                 efic_reg2.append(0)
366.             else:
367.                 efic_reg2.append(mt.exp(0.0411/(abs(aceleracion2[k])))
    **(-1))
368.
369.
370.
371.     pot_batcalculada2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["P
    otencia de Bateria calculada"])
372.
373.     aux1=1
374.     aux2=1
375.     aux3=1
376.
377.     for k in range(0,numfilas2):
378.         if pot_rueda2[k]>=0:
379.             aux1=1
380.             aux2=0
381.         else:
```

```
382.             aux1=0
383.             aux2=1
384.         pot_batcalculada2.iloc[k,0]=(((aux1*pot_rueda2[k]+aux2*pot_rue
   da2[k]*efic_reg2[k])/(efic_bat*efic_motor*efic_trans))+P0)*aux3
385.
386.
387.     energiacal_1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Energi
   a 1"])
388.     energiacal_2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Energi
   a 2"])
389.
390.
391.     for k in range(1,numfilas1):
392.         energiacal_1.iloc[k,0]=pot_batcalculada1.iloc[k,0]*travel_time
   1[k]
393.
394.     for k in range(1,numfilas2):
395.         energiacal_2.iloc[k,0]=pot_batcalculada2.iloc[k,0]*travel_time
   2[k]
396.
397.
398.     energiautilizada1=energiacal_1['Energia 1'].sum()
399.     energiautilizada2=energiacal_2['Energia 2'].sum()
400.
401.     distancia_recorrida1=(sum(distanciareal1))
402.     distancia_recorrida2=(sum(distanciareal2))
403.     tiempo_total1=(sum(travel_time1))
404.     tiempo_total2=(sum(travel_time2))
405.
406.
407.     rendimiento1=energiautilizada1/(3600*(distancia_recorrida1/1000))
    #Wh/km
408.     rendimiento2=energiautilizada2/(3600*(distancia_recorrida2/1000))
    #Wh/km
409.
410.
411.     soc_cal1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["SOC
   calculado 1"])
412.     soc_cal2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["SOC
   calculado 2"])
413.
414.     energiacal_1=energiacal_1['Energia 1'].fillna(0)
415.     energiacal_2=energiacal_2['Energia 2'].fillna(0)
416.
417.
418.     sumatoria1=0
419.
420.     for k in range(0,numfilas1):
421.         sumatoria1=sumatoria1 + energiacal_1[k]/3600
422.         soc_cal1.iloc[k,0]=((energia_vehiculo*(soc_inicial/100) -
   sumatoria1)/energia_vehiculo)*100
423.
```

```
424.     soc_finalcalculado1=((energia_vehiculo*(soc_inicial/100) -
   energiautilizada1/3600)/energia_vehiculo)*100
425.
426.
427.
428.     sumatoria2=0
429.
430.     for k in range(0,numfilas2):
431.         sumatoria2=sumatoria2 + energiacal_2[k]/3600
432.         soc_cal2.iloc[k,0]=((energia_vehiculo*(soc_inicial/100) -
   sumatoria2)/energia_vehiculo)*100
433.
434.     soc_finalcalculado2=((energia_vehiculo*(soc_inicial/100) -
   energiautilizada2/3600)/energia_vehiculo)*100
435.
436.     print("SOC inicial:",soc_inicial)
437.     print("Ruta 1 (más rápida: roja):")
438.     print("Distancia:",distancia_recorrida1/1000,"km")
439.     print("Tiempo:",int(tiempo_total1/3600),"hr
   y",(tiempo_total1/3600-int(tiempo_total1/3600))*60,"min")
440.     print("Energia utilizada:",energiautilizada1/3600,"Wh")
441.     print("SOC final",soc_finalcalculado1)
442.
443.
444.
445.     print("SOC inicial:",soc_inicial)
446.     print("Ruta 2 (más corta: azul):")
447.     print("Distancia:",distancia_recorrida2/1000,"km")
448.     print("Tiempo:",int(tiempo_total2/3600),"hr
   y",(tiempo_total2/3600-int(tiempo_total2/3600))*60,"min")
449.     print("Energia utilizada:",energiautilizada2/3600,"Wh")
450.     print("SOC final",soc_finalcalculado2)
451.
452.
453.
454.
455.     tiempo_seg1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Tiempo"
   ])
456.     time1=0
457.
458.     for k in range(0,numfilas1):
459.         tiempo_seg1.iloc[k,0]=(time1+travel_time1[k])
460.         time1=tiempo_seg1.iloc[k,0]
461.
462.     recorrido1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Distanci
   a"])
463.     metros1=0
464.
465.     for k in range(0,numfilas1):
466.         recorrido1.iloc[k,0]=(metros1+length1[k])
467.         metros1=recorrido1.iloc[k,0]
468.
```

```
469.
470.
471.     tiempo_seg2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Tiempo"
   ])
472.     time2=0
473.
474.     for k in range(0,numfilas2):
475.         tiempo_seg2.iloc[k,0]=(time2+travel_time2[k])
476.         time2=tiempo_seg2.iloc[k,0]
477.
478.     recorrido2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Distanci
   a"])
479.     metros2=0
480.
481.     for k in range(0,numfilas2):
482.         recorrido2.iloc[k,0]=(metros2+length2[k])
483.         metros2=recorrido2.iloc[k,0]
484.
485.
486.     temp_cal1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["temperatu
   ra calculada"])
487.     temp_cal1.iloc[0,0]=t0
488.
489.     diftemp_cal1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["delta
   t calculado"])
490.     pi1=0
491.
492.     for k in range(1,numfilas1):
493.         if t0>=40:
494.             pi1=min(10000,pot_batcalculada1.iloc[k,0])
495.         else:
496.             pi1=pot_batcalculada1.iloc[k,0]
497.         if (np.isnan(pot_batcalculada1.iloc[k,0])):
498.             diftemp_cal1.iloc[k,0]=0
499.         else:
500.             diftemp_cal1.iloc[k,0]=((1/rth)*(tamb-temp_cal1.iloc[k-
   1,0])+pi1*(1-efic_bat))*(travel_time1[k]/cth)
501.         temp_cal1.iloc[k,0]=temp_cal1.iloc[k-
   1,0]+diftemp_cal1.iloc[k,0]
502.
503.
504.     temp_cal2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["temperatu
   ra calculada"])
505.     temp_cal2.iloc[0,0]=t0
506.
507.     diftemp_cal2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["delta
   t calculado"])
508.     pi2=0
509.
510.     for k in range(1,numfilas2):
511.         if t0>=40:
512.             pi2=min(10000,pot_batcalculada2.iloc[k,0])
```

```
513.         else:
514.             pi2=pot_batcalculada2.iloc[k,0]
515.         if (np.isnan(pot_batcalculada2.iloc[k,0])):
516.             diftemp_cal2.iloc[k,0]=0
517.         else:
518.             diftemp_cal2.iloc[k,0]=((1/rth)*(tamb-temp_cal2.iloc[k-
    1,0])+pi2*(1-efic_bat))*(travel_time2[k]/cth)
519.         temp_cal2.iloc[k,0]=temp_cal2.iloc[k-
    1,0]+diftemp_cal2.iloc[k,0]
520.
521.
522.
523.     aux41=100
524.
525.     volt_cal1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["voltaje
    calculado"])
526.
527.
528.     for k in range(1,numfilas1):
529.         if temp_cal1.iloc[k-1,0]<10:
530.             aux41=0
531.         elif temp_cal1.iloc[k-1,0]>=10 and temp_cal1.iloc[k-1,0]<20:
532.             aux41=1
533.         elif temp_cal1.iloc[k-1,0]>35:
534.             aux41=3
535.         else:
536.             aux41=2
537.         if pot_batcalculada1.iloc[k,0]<0:
538.             volt_cal1.iloc[k,0]=(p0c[aux41]*mt.exp(alfa1c[aux41]*soc_c
    al1.iloc[k,0])+p1c[aux41]*mt.exp(alfa2c[aux41]*soc_cal1.iloc[k,0])+p2c[a
    ux41]*soc_cal1.iloc[k,0]**2)*96
539.         else:
540.             volt_cal1.iloc[k,0]=(p0d[aux41]*mt.exp(alfa1d[aux41]*soc_c
    al1.iloc[k,0])+p1d[aux41]*mt.exp(alfa2d[aux41]*soc_cal1.iloc[k,0])+p2d[a
    ux41]*soc_cal1.iloc[k,0]**2)*96
541.
542.
543.     aux42=100
544.
545.     volt_cal2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["voltaje
    calculado"])
546.
547.
548.     for k in range(1,numfilas2):
549.         if temp_cal2.iloc[k-1,0]<10:
550.             aux42=0
551.         elif temp_cal2.iloc[k-1,0]>=10 and temp_cal2.iloc[k-1,0]<20:
552.             aux42=1
553.         elif temp_cal2.iloc[k-1,0]>35:
554.             aux42=3
555.         else:
556.             aux42=2
```

```
557.         if pot_batcalculada2.iloc[k,0]<0:
558.             volt_cal2.iloc[k,0]=(p0c[aux42]*mt.exp(alfa1c[aux42]*soc_c
   al2.iloc[k,0])+p1c[aux42]*mt.exp(alfa2c[aux42]*soc_cal2.iloc[k,0])+p2c[a
   ux42]*soc_cal2.iloc[k,0]**2)*96
559.         else:
560.             volt_cal2.iloc[k,0]=(p0d[aux42]*mt.exp(alfa1d[aux42]*soc_c
   al2.iloc[k,0])+p1d[aux42]*mt.exp(alfa2d[aux42]*soc_cal2.iloc[k,0])+p2d[a
   ux42]*soc_cal2.iloc[k,0]**2)*96
561.
562.
563.     informacionderuta1=pd.concat([tiempo_seg1,pot_batcalculada1,soc_ca
   l1,temp_cal1,volt_cal1],axis=1,)
564.     informacionderuta2=pd.concat([tiempo_seg2,pot_batcalculada2,soc_ca
   l2,temp_cal2,volt_cal2],axis=1,)
565.
566.     elevacion1=pd.DataFrame(list(zip(elevation_to1)),columns=["elevaci
   on"])
567.     elevacion2=pd.DataFrame(list(zip(elevation_to2)),columns=["elevaci
   on"])
568.
569.     mapa1=pd.concat([tiempo_seg1,recorrido1,elevacion1],axis=1,)
570.     mapa2=pd.concat([tiempo_seg2,recorrido2,elevacion2],axis=1,)
571.
572.
573.     plt.figure(3)
574.     bx=plt.gca()
575.
576.     mapa1.plot(kind='line',x='Distancia',y='elevacion',ax=bx)
577.
578.     plt.figure(4)
579.     bx=plt.gca()
580.
581.     mapa2.plot(kind='line',x='Distancia',y='elevacion',ax=bx)
582.
583.     plt.figure(5)
584.     bx=plt.gca()
585.
586.     informacionderuta1.plot(kind='line',x='Tiempo',y='SOC calculado
   1',ax=bx)
587.
588.     plt.figure(6)
589.     bx=plt.gca()
590.
591.     informacionderuta2.plot(kind='line',x='Tiempo',y='SOC calculado
   2',ax=bx)
592.
593.
594.     plt.figure(7)
595.     bx=plt.gca()
596.
597.     informacionderuta1.plot(kind='line',x='Tiempo',y='temperatura
   calculada',ax=bx)
```

```
598.
599.      plt.figure(8)
600.      bx=plt.gca()
601.
602.      informacionderuta2.plot(kind='line',x='Tiempo',y='temperatura
   calculada',ax=bx)
603.
604.
605.      plt.figure(9)
606.      bx=plt.gca()
607.
608.      informacionderuta1.plot(kind='line',x='Tiempo',y='voltaje
   calculado',ax=bx)
609.
610.      plt.figure(10)
611.      bx=plt.gca()
612.
613.      informacionderuta2.plot(kind='line',x='Tiempo',y='voltaje
   calculado',ax=bx)
614.
615.      plt.figure(11)
616.      bx=plt.gca()
617.
618.      informacionderuta1.plot(kind='scatter',x='Tiempo',y='Potencia de
   Bateria calculada',ax=bx)
619.
620.      plt.figure(12)
621.      bx=plt.gca()
622.
623.      informacionderuta2.plot(kind='scatter',x='Tiempo',y='Potencia de
   Bateria calculada',ax=bx)
```

8.1.5 Código proyeccion alternativa dos rutas interregionales

```
1.

2. import numpy as np
3. import pandas as pd
4. import math as mt
5. import datetime
6. import matplotlib.pyplot as plt
7. import gpxpy
8. import gpxpy.gpx
9. import pandas as pd
10.      from pandas.api.types import is_string_dtype
11.      from pandas.api.types import is_numeric_dtype
12.      import time
13.      import datetime
14.      from datetime import datetime
```

```
15.      import difflib
16.      import sys
17.      import numpy as np
18.
19.      vnom=360
20.      soc_inicial=100
21.      energia_vehiculo=40000 #deberia ir elm ultimo valor obtenido a
   traves del leaf spy siendo el resultado del ultimo (Ahr*Vnom)/10000  Ej:
   1096564*360/10000=39476.304
22.      Energia_i=energia_vehiculo*soc_inicial/100
23.
24.      rt=6371*1000 # metros
25.
26.      path="C:\\Users\\claud\\Desktop\\"
27.      data1=[]
28.      gps1 = gpxpy.parse(open('C:\\Users\\claud\\Desktop\\ruta1.gpx'))
29.
30.      gpx_file1 = open('C:\\Users\\claud\\Desktop\\ruta1.gpx', 'r')
31.
32.      gpx1 = gpxpy.parse(gpx_file1)
33.
34.      print("{} track(s)".format(len(gpx1.tracks)))
35.      track1 = gpx1.tracks[0]
36.
37.      print("{} segment(s)".format(len(track1.segments)))
38.      segment1 = track1.segments[0]
39.
40.      print("{} point(s)".format(len(segment1.points)))
41.
42.      data1 = []
43.      segment_length1 = segment1.length_3d()
44.      for point_idx, point in enumerate(segment1.points):
45.          data1.append([point.longitude, point.latitude,
46.                        point.elevation, point.time, segment1.get_speed(
   point_idx)])
47.
48.      from pandas import DataFrame
49.
50.      columns = ['Longitude', 'Latitude', 'Altitude', 'Time', 'Speed']
51.      df1 = DataFrame(data1, columns=columns)
52.      df1.head()
53.      datos11=df1
54.      datos1 = df1
55.      numfilas1=len(datos1)
56.
57.
58.      data2=[]
59.      gps2 = gpxpy.parse(open('C:\\Users\\claud\\Desktop\\ruta2.gpx'))
60.
61.      gpx_file2 = open('C:\\Users\\claud\\Desktop\\ruta2.gpx', 'r')
62.
63.      gpx2 = gpxpy.parse(gpx_file2)
```

```
64.
65.      print("{} track(s)".format(len(gpx2.tracks)))
66.      track2 = gpx2.tracks[0]
67.
68.      print("{} segment(s)".format(len(track2.segments)))
69.      segment2 = track2.segments[0]
70.
71.      print("{} point(s)".format(len(segment2.points)))
72.
73.      data2 = []
74.      segment_length2 = segment2.length_3d()
75.      for point_idx, point in enumerate(segment2.points):
76.          data2.append([point.longitude, point.latitude,
77.                        point.elevation, point.time, segment2.get_speed(p
   oint_idx)])
78.
79.      from pandas import DataFrame
80.
81.      columns = ['Longitude', 'Latitude', 'Altitude', 'Time', 'Speed']
82.      df2 = DataFrame(data2, columns=columns)
83.      df2.head()
84.      datos22=df2
85.      datos2 = df2
86.      numfilas2=len(datos2)
87.
88.      #%%
89.
90.      distancia_plana11=pd.DataFrame(np.zeros((numfilas1,1)),columns=["D
   istancia plana"])
91.      distancia_vertical11=pd.DataFrame(np.zeros((numfilas1,1)),columns=
   ["Distancia vertical"])
92.      distancia_real11=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Di
   stancia real"])
93.      delta_t11=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Delta
   tiempo"])
94.      velocidad11=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Velocid
   ad"])
95.      aceleracion11=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Acele
   racion"])
96.      pen11=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Pendiente"])
97.
98.
99.      for k in range(numfilas1):
100.         datos1["Latitude"][k]=(datos1["Latitude"][k])*mt.pi/180
101.         datos1["Longitude"][k]=(datos1["Longitude"][k])*mt.pi/180
102.
103.     while True:
104.
105.         numfilas1=len(datos1)
106.         velocidad11=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Vel
   ocidad1"])
107.         for k in range(1,numfilas1):
```

```
108.             delta_t11.iloc[k,0]=(datos1.iloc[k,3]-datos1.iloc[k-
   1,3]).total_seconds()
109.
110.         for k in range(1,numfilas1):
111.             resta_lat1=(datos1["Latitude"][k]-datos1["Latitude"][k-
   1])/2
112.             resta_long1=(datos1["Longitude"][k]-datos1["Longitude"][k-
   1])/2
113.             mult_cos1=mt.cos(datos1["Latitude"][k])*mt.cos(datos1["Lat
   itude"][k-1])
114.             distancia_plana11.iloc[k,0]=2*rt*mt.asin((mt.sin(resta_lat
   1)**2+(mult_cos1)*(mt.sin(resta_long1)**2))**(1/2))
115.
116.         for k in range(1,numfilas1):
117.             distancia_vertical11.iloc[k,0]=datos1["Altitude"][k]-
   datos1["Altitude"][k-1]
118.
119.         for k in range(1,numfilas1):
120.             distancia_real11.iloc[k,0]=(distancia_plana11.iloc[k,0]**2
   +distancia_vertical11.iloc[k,0]**2)**(1/2)
121.
122.         for k in range(1,numfilas1):
123.             pen11.iloc[k,0]=distancia_vertical11.iloc[k,0]/distancia_p
   lana11.iloc[k,0]
124.
125.         for k in range(1,numfilas1):
126.             velocidad11.iloc[k,0]=distancia_real11.iloc[k,0]/delta_t11
   .iloc[k,0]
127.
128.
129.         for k in range(1,numfilas1):
130.             aceleracion11.iloc[k,0]=(velocidad11.iloc[k,0]-
   velocidad11.iloc[k-1,0])/(delta_t11.iloc[k,0])
131.
132.         #sys.exit()
133.
134.         datos1=pd.concat([datos1,velocidad11],axis=1,)
135.
136.         if all(datos1['Velocidad1']<42):
137.             break
138.
139.         else:
140.             datos1=datos1[datos1["Velocidad1"]<=42].reset_index(drop=T
   rue)
141.             datos1=datos1.drop(['Velocidad1'], axis=1)
142.
143.
144.     datos1=datos1.drop(['Velocidad1'], axis=1)
145.
146.     datos1=datos1.reindex(columns=['Time','Latitude','Longitude','Alti
   tude','Speed'])
147.
```

```
148.     datos1.rename(columns={'Time':'Date/Time','Latitude':'Lat','Longit
   ude':'Long','Altitude':'Elv'},inplace=True)
149.
150.
151.     distancia_plana22=pd.DataFrame(np.zeros((numfilas2,1)),columns=["D
   istancia plana"])
152.     distancia_vertical22=pd.DataFrame(np.zeros((numfilas2,1)),columns=
   ["Distancia vertical"])
153.     distancia_real22=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Di
   stancia real"])
154.     delta_t22=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Delta
   tiempo"])
155.     velocidad22=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Velocid
   ad"])
156.     aceleracion22=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Acele
   racion"])
157.     pen22=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Pendiente"])
158.
159.
160.     for k in range(numfilas2):
161.         datos2["Latitude"][k]=(datos2["Latitude"][k])*mt.pi/180
162.         datos2["Longitude"][k]=(datos2["Longitude"][k])*mt.pi/180
163.
164.     while True:
165.
166.         numfilas2=len(datos2)
167.         velocidad22=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Vel
   ocidad2"])
168.         for k in range(1,numfilas2):
169.             delta_t22.iloc[k,0]=(datos2.iloc[k,3]-datos2.iloc[k-
   1,3]).total_seconds()
170.
171.         for k in range(1,numfilas2):
172.             resta_lat2=(datos2["Latitude"][k]-datos2["Latitude"][k-
   1])/2
173.             resta_long2=(datos2["Longitude"][k]-datos2["Longitude"][k-
   1])/2
174.             mult_cos2=mt.cos(datos2["Latitude"][k])*mt.cos(datos2["Lat
   itude"][k-1])
175.             distancia_plana22.iloc[k,0]=2*rt*mt.asin((mt.sin(resta_lat
   2)**2+(mult_cos2)*(mt.sin(resta_long2)**2))**(1/2))
176.
177.         for k in range(1,numfilas2):
178.             distancia_vertical22.iloc[k,0]=datos2["Altitude"][k]-
   datos2["Altitude"][k-1]
179.
180.         for k in range(1,numfilas2):
181.             distancia_real22.iloc[k,0]=(distancia_plana22.iloc[k,0]**2
   +distancia_vertical22.iloc[k,0]**2)**(1/2)
182.
183.         for k in range(1,numfilas2):
```

```
184.             pen22.iloc[k,0]=distancia_vertical22.iloc[k,0]/distancia_p
   lana22.iloc[k,0]
185.
186.         for k in range(1,numfilas2):
187.             velocidad22.iloc[k,0]=distancia_real22.iloc[k,0]/delta_t22
   .iloc[k,0]
188.
189.
190.         for k in range(1,numfilas2):
191.             aceleracion22.iloc[k,0]=(velocidad22.iloc[k,0]-
   velocidad22.iloc[k-1,0])/(delta_t22.iloc[k,0])
192.
193.         #sys.exit()
194.
195.         datos2=pd.concat([datos2,velocidad22],axis=1,)
196.
197.         if all(datos2['Velocidad2']<42):
198.             break
199.
200.         else:
201.             datos2=datos2[datos2["Velocidad2"]<=42].reset_index(drop=T
   rue)
202.             datos2=datos2.drop(['Velocidad2'], axis=1)
203.
204.
205.     datos2=datos2.drop(['Velocidad2'], axis=1)
206.
207.     datos2=datos2.reindex(columns=['Time','Latitude','Longitude','Alti
   tude','Speed'])
208.
209.     datos2.rename(columns={'Time':'Date/Time','Latitude':'Lat','Longit
   ude':'Long','Altitude':'Elv'},inplace=True)
210.
211.
212.
213.     #%%
214.
215.     numfilas1=len(datos1)
216.     numfilas2=len(datos2)
217.
218.
219.     #NUEVOCOMIENZO
220.     #PARAMETROS
221.     rt=6371*1000 # metros
222.     daire=1.25 # kg/m3
223.     af=2.76 # m2
224.     cd=0.28 # [0,2-0,4] dependiendo de la velocidad, es estima como
   constante
225.     g=9.81 # [9,764-9,834]
226.     m=1580 # kg
227.     cr=2.2806 # [1-4], 1,75 inicial segun cierta bibliografia
228.     c1=0.0328 # [0,03-0,06]
```

```
229.     c2=4.575 # [4-8]
230.     vnom=360
231.
232.     P0=300 #potencia de bateria solo por estar encendido moda 317
   promedio 447 mediana 288 maximo 31530 minimo 2.08 con filtros de
   velocidad=0 aceleracion=0 y dif soc entre 50 y 50
233.
234.
235.     #eficiencias
236.     efic_bat=0.93 #90 a95% se asume por biblio  de la bateria sale a
   P0 y al motor
237.     efic_motor=0.91 #Hesham A Rakha, Kyoungho Ahn, Kevin Moran, Bart
   Saerens, and Eric Van Den Bulck. Virginia Tech Comprehensive Power-Based
   Fuel Consumption Model : Model development and testing. Transportation
   Research Part D, 16(7):492–503, 2011.
238.     efic_trans=0.92 #SAE International. 2011 Nissan Leaf vehicle
   overview; 2011.
239.     #Regenera: Yimin Gao, Liang Chu, and Mehrdad Ehsani. Design and
   control principles of hybrid braking system for EV, HEV and FCV. VPPC
   2007 - Proceedings of the 2007 IEEE Vehicle Power and Propulsion
   Conference, (1):384–391, 2007.
240.     #me convendrian numeros más grandes
241.     #energia_vehiculo=vnom*datos.iloc[0,7]/10000 #Vnom por AHr
242.     #energia_in=(energia_vehiculo*datos.iloc[0,6]/10000)/100 #Soc por
   energia inicial
243.     #soc_inicial=datos.iloc[0,6]/10000
244.     #energia_fin=(energia_vehiculo*datos.iloc[numfilas-1,6]/10000)/100
245.
246.
247.     #Definicion de variables a utilizar
248.     distancia_plana1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Di
   stancia plana"])
249.     distancia_vertical1=pd.DataFrame(np.zeros((numfilas1,1)),columns=[
   "Distancia vertical"])
250.     distancia_real1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Dis
   tancia real"])
251.     delta_t1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Delta
   tiempo"])
252.     delta_soc1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Delta
   SOC"])#ojo con este
253.     velocidad1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Velocida
   d"])
254.     aceleracion1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Aceler
   acion"])
255.     pen1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Pendiente"])
256.     alpha1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Angulo"])
257.     f_aer1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Fuerza
   Aerodinamica"])
258.     f_pendiente1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Fuerza
   de Pendiente"])
259.     f_rol1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Fuerza de
   Rolling"])
```

```
260.     f_mov1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Fuerza de
   Movimiento"])
261.     pot_rueda1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Potencia
   de Rueda"])
262.     efic_reg1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Eficienci
   a regeneracion"])
263.     energia_i1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Energia
   restante"])
264.
265.     p_aer1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["potencia
   Aerodinamica"])
266.     p_pendiente1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["potenc
   ia de Pendiente"])
267.     p_rol1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["potencia de
   Rolling"])
268.     p_mov1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["potencia de
   Movimiento"])
269.
270.
271.
272.     distancia_plana2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Di
   stancia plana"])
273.     distancia_vertical2=pd.DataFrame(np.zeros((numfilas2,1)),columns=[
   "Distancia vertical"])
274.     distancia_real2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Dis
   tancia real"])
275.     delta_t2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Delta
   tiempo"])
276.     delta_soc2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Delta
   SOC"])#ojo con este
277.     velocidad2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Velocida
   d"])
278.     aceleracion2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Aceler
   acion"])
279.     pen2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Pendiente"])
280.     alpha2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Angulo"])
281.     f_aer2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Fuerza
   Aerodinamica"])
282.     f_pendiente2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Fuerza
   de Pendiente"])
283.     f_rol2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Fuerza de
   Rolling"])
284.     f_mov2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Fuerza de
   Movimiento"])
285.     pot_rueda2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Potencia
   de Rueda"])
286.     efic_reg2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Eficienci
   a regeneracion"])
287.     energia_i2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Energia
   restante"])
288.
```

```
289.     p_aer2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["potencia
   Aerodinamica"])
290.     p_pendiente2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["potenc
   ia de Pendiente"])
291.     p_rol2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["potencia de
   Rolling"])
292.     p_mov2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["potencia de
   Movimiento"])
293.
294.     #Formulas
295.
296.     for k in range(1,numfilas1):
297.         delta_t1.iloc[k,0]=(datos1.iloc[k,0]-datos1.iloc[k-
   1,0]).total_seconds()
298.
299.     for k in range(1,numfilas2):
300.         delta_t2.iloc[k,0]=(datos2.iloc[k,0]-datos2.iloc[k-
   1,0]).total_seconds()
301.
302.
303.     for k in range(1,numfilas1):
304.         resta_lat1=(datos1["Lat"][k]-datos1["Lat"][k-1])/2
305.         resta_long1=(datos1["Long"][k]-datos1["Long"][k-1])/2
306.         mult_cos1=mt.cos(datos1["Lat"][k])*mt.cos(datos1["Lat"][k-1])
307.         distancia_plana1.iloc[k,0]=2*rt*mt.asin((mt.sin(resta_lat1)**2
   +(mult_cos1)*(mt.sin(resta_long1)**2))**(1/2))
308.
309.     for k in range(1,numfilas2):
310.         resta_lat2=(datos2["Lat"][k]-datos2["Lat"][k-1])/2
311.         resta_long2=(datos2["Long"][k]-datos2["Long"][k-1])/2
312.         mult_cos2=mt.cos(datos2["Lat"][k])*mt.cos(datos2["Lat"][k-1])
313.         distancia_plana2.iloc[k,0]=2*rt*mt.asin((mt.sin(resta_lat2)**2
   +(mult_cos2)*(mt.sin(resta_long2)**2))**(1/2))
314.
315.
316.     for k in range(1,numfilas1):
317.         distancia_vertical1.iloc[k,0]=datos1["Elv"][k]-
   datos1["Elv"][k-1]
318.
319.     for k in range(1,numfilas2):
320.         distancia_vertical2.iloc[k,0]=datos2["Elv"][k]-
   datos2["Elv"][k-1]
321.
322.
323.     for k in range(1,numfilas1):
324.         distancia_real1.iloc[k,0]=(distancia_plana1.iloc[k,0]**2+dista
   ncia_vertical1.iloc[k,0]**2)**(1/2)
325.
326.     for k in range(1,numfilas2):
327.         distancia_real2.iloc[k,0]=(distancia_plana2.iloc[k,0]**2+dista
   ncia_vertical2.iloc[k,0]**2)**(1/2)
328.
```

```
329.
330.     for k in range(1,numfilas1):
331.         velocidad1.iloc[k,0]=distancia_real1.iloc[k,0]/delta_t1.iloc[k
   ,0]
332.
333.     for k in range(1,numfilas2):
334.         velocidad2.iloc[k,0]=distancia_real2.iloc[k,0]/delta_t2.iloc[k
   ,0]
335.
336.
337.     for k in range(1,numfilas1):
338.         aceleracion1.iloc[k,0]=(velocidad1.iloc[k,0]-
   velocidad1.iloc[k-1,0])/(delta_t1.iloc[k,0])
339.
340.     for k in range(1,numfilas2):
341.         aceleracion2.iloc[k,0]=(velocidad2.iloc[k,0]-
   velocidad2.iloc[k-1,0])/(delta_t2.iloc[k,0])
342.
343.
344.     for k in range(1,numfilas1):
345.         pen1.iloc[k,0]=distancia_vertical1.iloc[k,0]/distancia_plana1.
   iloc[k,0]
346.
347.     for k in range(1,numfilas2):
348.         pen2.iloc[k,0]=distancia_vertical2.iloc[k,0]/distancia_plana2.
   iloc[k,0]
349.
350.
351.     for k in range(1,numfilas1):
352.         alpha1.iloc[k,0]=mt.atan((distancia_vertical1.iloc[k,0])/dista
   ncia_plana1.iloc[k,0])
353.         if alpha1.iloc[k,0] > 0.65:
354.             alpha1.iloc[k,0]=0.65
355.         elif alpha1.iloc[k,0] < -0.65:
356.             alpha1.iloc[k,0]=-0.65
357.         else :
358.             alpha1.iloc[k,0]=alpha1.iloc[k,0]
359.
360.     for k in range(1,numfilas2):
361.         alpha2.iloc[k,0]=mt.atan((distancia_vertical2.iloc[k,0])/dista
   ncia_plana2.iloc[k,0])
362.         if alpha2.iloc[k,0] > 0.65:
363.             alpha2.iloc[k,0]=0.65
364.         elif alpha2.iloc[k,0] < -0.65:
365.             alpha2.iloc[k,0]=-0.65
366.         else :
367.             alpha2.iloc[k,0]=alpha2.iloc[k,0]
368.
369.
370.     for k in range(1,numfilas1):
371.         f_aer1.iloc[k,0]=(1/2)*daire*af*cd*(velocidad1.iloc[k,0]**2)
372.
```

```
    for k in range(1,numfilas2):
        f_aer2.iloc[k,0]=(1/2)*daire*af*cd*(velocidad2.iloc[k,0]**2)


    for k in range(1,numfilas1):
        p_aer1.iloc[k,0]=f_aer1.iloc[k,0]*velocidad1.iloc[k,0]

    for k in range(1,numfilas2):
        p_aer2.iloc[k,0]=f_aer2.iloc[k,0]*velocidad2.iloc[k,0]


    for k in range(1,numfilas1):
        f_pendiente1.iloc[k,0]=m*g*mt.sin(alpha1.iloc[k,0])

    for k in range(1,numfilas2):
        f_pendiente2.iloc[k,0]=m*g*mt.sin(alpha2.iloc[k,0])


    for k in range(1,numfilas1):
        p_pendiente1.iloc[k,0]=f_pendiente1.iloc[k,0]*velocidad1.iloc[
    k,0]

    for k in range(1,numfilas2):
        p_pendiente2.iloc[k,0]=f_pendiente2.iloc[k,0]*velocidad2.iloc[
    k,0]


    for k in range(1,numfilas1):
        f_rol1.iloc[k,0]=m*g*mt.cos(alpha1.iloc[k,0])*(cr/1000)*(c1*ve
    locidad1.iloc[k,0]+c2)

    for k in range(1,numfilas2):
        f_rol2.iloc[k,0]=m*g*mt.cos(alpha2.iloc[k,0])*(cr/1000)*(c1*ve
    locidad2.iloc[k,0]+c2)


    for k in range(1,numfilas1):
        p_rol1.iloc[k,0]=f_rol1.iloc[k,0]*velocidad1.iloc[k,0]

    for k in range(1,numfilas2):
        p_rol2.iloc[k,0]=f_rol2.iloc[k,0]*velocidad2.iloc[k,0]


    for k in range(1,numfilas1):
        f_mov1.iloc[k,0]=m*aceleracion1.iloc[k,0]

    for k in range(1,numfilas2):
        f_mov2.iloc[k,0]=m*aceleracion2.iloc[k,0]


    for k in range(1,numfilas1):
        p_mov1.iloc[k,0]=f_mov1.iloc[k,0]*velocidad1.iloc[k,0]
```

```
421.
422.     for k in range(1,numfilas2):
423.         p_mov2.iloc[k,0]=f_mov2.iloc[k,0]*velocidad2.iloc[k,0]
424.
425.
426.     for k in range(1,numfilas1):
427.         pot_rueda1.iloc[k,0]=(f_mov1.iloc[k,0]+f_rol1.iloc[k,0]+f_pend
   iente1.iloc[k,0]+f_aer1.iloc[k,0])*velocidad1.iloc[k,0]
428.
429.     for k in range(1,numfilas2):
430.         pot_rueda2.iloc[k,0]=(f_mov2.iloc[k,0]+f_rol2.iloc[k,0]+f_pend
   iente2.iloc[k,0]+f_aer2.iloc[k,0])*velocidad2.iloc[k,0]
431.
432.
433.     for k in range(1,numfilas1):
434.         if aceleracion1.iloc[k,0]>=0:
435.             efic_reg1.iloc[k,0]=0
436.         else:
437.             efic_reg1.iloc[k,0]=mt.exp(0.0411/(abs(aceleracion1.iloc[k
   ,0])))**(-1)
438.
439.     for k in range(1,numfilas2):
440.         if aceleracion2.iloc[k,0]>=0:
441.             efic_reg2.iloc[k,0]=0
442.         else:
443.             efic_reg2.iloc[k,0]=mt.exp(0.0411/(abs(aceleracion2.iloc[k
   ,0])))**(-1)
444.
445.
446.     #for k in range(1,numfilas):
447.     #    delta_soc.iloc[k,0]=datos["SOC"][k]-datos["SOC"][k-1]
448.
449.
450.
451.
452.     pot_batcalculada1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["P
   otencia de Bateria calculada"])
453.     pot_batcalculada2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["P
   otencia de Bateria calculada"])
454.
455.
456.     aux1=1
457.     aux2=1
458.     aux3=1
459.
460.     #if datos["Elv"][0]-datos["Elv"][numfilas-1]<0:
461.     #    aux3=0.9
462.     #else:
463.     #    aux3=0.8
464.
465.     for k in range(1,numfilas1):
466.         if pot_rueda1.iloc[k,0]>=0:
```

```
467.             aux1=1
468.             aux2=0
469.         else:
470.             aux1=0
471.             aux2=1
472.         pot_batcalculada1.iloc[k,0]=(((aux1*pot_rueda1.iloc[k,0]+aux2*
   pot_rueda1.iloc[k,0]*efic_reg1.iloc[k,0])/(efic_bat*efic_motor*efic_tran
   s))+P0)*aux3
473.
474.     for k in range(1,numfilas2):
475.         if pot_rueda2.iloc[k,0]>=0:
476.             aux1=1
477.             aux2=0
478.         else:
479.             aux1=0
480.             aux2=1
481.         pot_batcalculada2.iloc[k,0]=(((aux1*pot_rueda2.iloc[k,0]+aux2*
   pot_rueda2.iloc[k,0]*efic_reg2.iloc[k,0])/(efic_bat*efic_motor*efic_tran
   s))+P0)*aux3
482.
483.
484.
485.     energiacal_1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["Energi
   a 1"])
486.
487.     energiacal_2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["Energi
   a 2"])
488.
489.
490.     for k in range(1,numfilas1):
491.         energiacal_1.iloc[k,0]=pot_batcalculada1.iloc[k,0]*delta_t1.il
   oc[k,0]
492.
493.     for k in range(1,numfilas2):
494.         energiacal_2.iloc[k,0]=pot_batcalculada2.iloc[k,0]*delta_t2.il
   oc[k,0]
495.
496.
497.     energiautilizada1=energiacal_1['Energia 1'].sum()
498.     energiautilizada2=energiacal_2['Energia 2'].sum()
499.
500.
501.     distancia_recorrida1=(distancia_real1.sum())
502.     distancia_recorrida2=(distancia_real2.sum())
503.
504.
505.     rendimiento1=energiautilizada1/(3600*(distancia_recorrida1/1000))
    #Wh/km
506.     rendimiento2=energiautilizada2/(3600*(distancia_recorrida2/1000))
    #Wh/km
507.
508.
```

```
509.     soc_cal1=pd.DataFrame(np.zeros((numfilas1,1)),columns=["SOC
   calculado"])
510.     energiacal_1=energiacal_1['Energia 1'].fillna(0)
511.
512.     soc_cal2=pd.DataFrame(np.zeros((numfilas2,1)),columns=["SOC
   calculado"])
513.     energiacal_2=energiacal_2['Energia 2'].fillna(0)
514.
515.
516.     sumatoria1=0
517.
518.     for k in range(1,numfilas1):
519.         sumatoria1=sumatoria1 + energiacal_1[k]/3600
520.         soc_cal1.iloc[k,0]=((energia_vehiculo*(soc_inicial/100) -
   sumatoria1)/energia_vehiculo)*100
521.
522.     sumatoria2=0
523.
524.     for k in range(1,numfilas2):
525.         sumatoria2=sumatoria2 + energiacal_2[k]/3600
526.         soc_cal2.iloc[k,0]=((energia_vehiculo*(soc_inicial/100) -
   sumatoria2)/energia_vehiculo)*100
527.
528.
529.     soc_finalcalculado1=((energia_vehiculo*(soc_inicial/100) -
   energiautilizada1/3600)/energia_vehiculo)*100
530.
531.     soc_finalcalculado2=((energia_vehiculo*(soc_inicial/100) -
   energiautilizada2/3600)/energia_vehiculo)*100
532.
```