Universidad Carlos III de Madrid
Escuela Politécnica Superior

Ingeniería de Telecomunicación

Proyecto Fin de Carrera

Sistema de Estimación de Autonomía en Vehículos de Tracción Eléctrica mediante Redes Neuronales

Autor: Daniel Saiz Llarena

Tutor/Director: Jorge Pleite Guerra

Co-Director: Israel González Carrasco

Octubre 2014

Resumen

Diseño e implementación de una aplicación para dispositivos móviles Android con funcionalidades de visualización de mapas, navegación guiada paso a paso, visionado de datos de telemetría del vehículo (tanto en tiempo real como en forma de resumen para cada ruta realizada) y estimación inteligente del consumo para una ruta determinada (utilizando para ello técnicas de *Machine Learning* como son las Redes Neuronales Artificiales, las cuales permiten que el sistema aprenda del comportamiento del usuario).

Abstract

Design and implementation of a mobile application for Android devices with map-display, step by step guided navigation, vehicle's telemetry data display (both real-time and for consulting each realized route) and intelligent estimating of consumption for a given route (using Machine Learning techniques such as Artificial Neural Networks, which allow the system to learn from user's behavior).

Índice

Índice de figuras

Índice de tablas

Nomenclatura

ANN	Artificial Neural Network
API	Application Programming Interface
APP	Application
BBDD	Base de datos
BCU	Bultaco Control Unit
BP	Backpropagation
DB	Database
DOD	Depth of Discharge
FIR	Finite impulse response
GPS	Global Positioning System (Sistema de posicionameinto global)
IA	Inteligencia Artificial
IDE	Integrated development environment
ISS	Internet Information Services
ML	Machine Learning (Aprendizaje máquina)
MLP	Multilayer Perceptron
MPC	Modelo de Predicció de Consumo
MVC	Modelo-Vista-Controlador (Model-View-Controller)
null	Sin asignación
OSM	Open Street Maps
PC	Personal Computer
POI	Point Of Interest
RN	Red Neuronal
SDK	Software Development Kit
SE	Square Error
SOC	State of charge
SRTM	Shuttle Radar Topography Mission

UC3M	Universidad Carlos III de Madrid
WIP	Work In Progress

1 Introducción

1.1 Problema

Este proyecto se plantea como proyecto de investigación en conjunto con la empresa Bultaco Motors a través de la cátedra que ha establecido con la Universidad Carlos III de Madrid y pretende aplicarse en un escenario real para funcionar conjuntamente con la motocicleta 100% eléctrica que han desarrollado.

Dadas las características de este tipo de vehículos (baja autonomía, largo tiempo de recarga, baja densidad de puntos de recarga) se hace necesario buscar formas de afinar la estimación de consumo o autonomía para así fortalecer la confianza del usuario y que no se vea limitado por el interrogante de si se quedará sin energía a mitad de una ruta.

El problema que se pretende abordar consiste en el desarrollo de una aplicación para dispositivos móviles que provea de visualización de mapas, navegación guiada GPS, conexión a una motocicleta para lectura de telemetría y GPS, visualización de datos de telemetría en tiempo real, grabación de rutas y su posterior visualización así como el desarrollo de un sistema inteligente de estimación de autonomía que aprenda de las rutas guardadas y por tanto se adapte a las diferentes características de la ruta, así como a la conducción del piloto.

1.2 Objetivos del proyecto

A pesar de que la aplicación se ha desarrollado prácticamente en su totalidad, este proyecto se centrará principalmente en el desarrollo innovador que supone la aplicación de un sistema de aprendizaje máquina para proporcionar estimaciones de consumo adaptadas al usuario y a la ruta a realizar.

El objetivo por tanto no es sólo construir todo el sistema (aplicación con mapas, navegación, conexión con la motocicleta, visionado de telemetría, etc.) sino tratar de investigar la viabilidad de implantar un sistema basado en técnicas de *machine learning* como son las Redes Neuronales para realizar estimaciones de consumo de una determinada ruta, todo ello desarrollado en un dispositivo móvil.

1.3 Estructura del documento

El presente documento se ha confeccionado en base a una estructura bien definida y organizada en función tanto de las características del contenido como de su actuación temporal en el ciclo de vida del proyecto.

En primer lugar se realizará una **descripción del entorno** en el que se trabaja, dando una visión más detallada del problema que Bultaco Motors pretende resolver a través de este proyecto.

Posteriormente se dará una visión general del **estado del arte** en este campo, desde las diferentes técnicas planteadas a la hora de realizar una estimación de la autonomía del

vehículo eléctrico más eficiente, hasta el estudio de la teoría implicada en el desarrollo de este sistema (tanto las características y funcionamiento de las Redes Neuronales como la caracterización del consumo de energía en un vehículo eléctrico).

Una vez analizado el entorno, caracterizado el problema y estudiadas las técnicas que se pretenden aplicar, se procederá al **diseño del sistema** propiamente dicho. En dicho apartado se detallará en primer lugar un **diseño conceptual**, que tratará de dar una visión de alto nivel de la implementación del sistema. En segundo lugar se tratará el **diseño de sistema de inteligencia artificial**, otorgando especial detalle a este apartado por ser el elemento más innovador en este proyecto. Para finalizar se procederá a realizar un **diseño del software** implicado, esto es, un diseño más a bajo nivel sobre cómo se llevará a cabo la implementación del sistema.

Una vez concluida la tarea de diseño, se pasará a describir todo lo relacionado con el **desarrollo** del proyecto. Este apartado incluye detalles como la **metodología de desarrollo** empleada, la **planificación y presupuesto** del proyecto, cómo se ha **gestionado el código** y los pasos a la hora de elegir la **plataforma de desarrollo**. Así mismo, se ha profundizado en la propia **implementación del sistema**, realizando un análisis de la aplicación de la metodología elegida a la plataforma de desarrollo en cuestión, la estructura del código y todas las modificaciones y nuevas implementaciones desarrolladas.

Construido ya el sistema, se realizarán una serie pruebas para comprobar los **resultados** derivados del funcionamiento del sistema. En este apartado se detallará el tipo de pruebas realizadas, centrándose éstas en el sistema de inteligencia artificial y en su aplicabilidad para crear un Modelo de Predicción de Consumo óptimo. Se detallarán también las pertinentes conclusiones sobre los resultados obtenidos.

A continuación se ha creado un apartado a modo de resumen del proyecto, denominado **conclusiones**. En este apartado se dará una visión del conjunto de objetivos logrados, una visión global de todo lo aprendido a través de la realización de este proyecto, la cantidad de trabajo aportado al mismo, así como un análisis de las diferentes líneas de trabajo que se podrían seguir como continuación o como derivados del proyecto.

Se añade también un apartado con la **bibliografía** empleada y citada a lo largo de este documento, ya que es una base muy importante dado el carácter innovador de este sistema.

Por último se han añadido una serie de **anexos** a este documento con información adicional acerca de la telemetría que se recibe de la motocicleta, el manual de usuario de la aplicación o la librería de comunicación con la centralita de la motocicleta.

2 Descripción del entorno

2.1 Introducción

Con motivo de la *Cátedra UC3M LGN Tech Design*, formando parte del desarrollo de la misma como becario de desarrollo móvil, he realizado diferentes actividades de acuerdo al plan de desarrollo que la empresa *LGN Tech Design SL*, rebautizada como *Bultaco Motors,* ha establecido con la universidad.

El trabajo realizado se ha centrado en el diseño e implementación de una aplicación para *Smartphones* que provea, entre otras funcionalidades, navegación guiada *offline* y visionado de telemetrías en tiempo real, así como en ofrecer al usuario predicciones de consumo adaptadas su modo de conducción y a las características de la ruta: ***Biker Manager***.

2.2 Bultaco Motors

En el año 2014 Bultaco retorna a la actividad industrial de la mano de una empresa de I+D creada por un grupo de ingenieros de la **Universidad Carlos III de Madrid (UC3M)** con el apoyo del Vivero de Empresas del Parque Científico UC3M. Dicha iniciativa surge a partir de un proyecto fin de carrera desarrollado con los investigadores del Grupo MAqLAB de la Universidad Carlos III de Madrid (UC3M) y su desarrollo tecnológico en la empresa *LGN Tech Design*, creada en el Vivero de Empresas del Parque Científico UC3M [1].

La empresa inicia la andadura con dos modelos basados en un **sistema de propulsión eléctrica**: *Bultaco Rapitán* y *Bultaco Rapitán Sport*.

La *Bultaco Rapitán* (Ilustración 1 - Bultaco Rapitán) y *Bultaco Rapitán Sport* (Ilustración 2 - Bultaco Rapitán Sport) se caracterizan por un peso muy contenido, 189 Kg, y un propulsor de corriente alterna que genera una potencia equivalente a 54 CV, lo que permite una velocidad punta de 145 Km/h. Su exclusivo sistema de recuperación de energía asegura que las baterías ofrezcan autonomía suficiente al máximo de rendimiento: 200 km en ciudad, 110 km por autopista y 140 km en condiciones mixtas. El tiempo de recarga es de 3,5-5 h, con la opción de recarga rápida en 45'-1 h.

Ilustración 1 - Bultaco Rapitán

Ilustración 2 - Bultaco Rapitán Sport

El software de **la unidad de control (*BCU*, *Bultaco Control Unit*)** es extraordinariamente capaz, permitiendo gestionar la potencia del motor, la recuperación de energía, la recarga de la batería y la conexión con dispositivos móviles. Mediante la aplicación ***Biker Manager***, desarrollada en este proyecto, el usuario podrá saber con este tipo de dispositivos datos como la carga de la batería, la autonomía según el tipo de conducción que realiza o la situación de la moto, entre otros muchos.

2.3 Problema a resolver

2.3.1 Visión general

Este proyecto pretende resolver el problema planteado por Bultaco de implementar una aplicación móvil que provea una experiencia enriquecida al usuario a través de diferentes funcionalidades. No obstante, dado que dentro de dichas funcionalidades existe una que destaca sobre las demás por la innovación inherente a ella, se tratará con especial detalle.

Dicha funcionalidad consiste en la elaboración de un **sistema de estimación de consumo inteligente** basado en el comportamiento particular de cada usuario.

Debido a esta diferenciación en el diseño, a lo largo de este documento se realizará un especial énfasis en el diseño de dicho sistema de estimación. No obstante, también se pretende dar una visión global de todo el conjunto de la aplicación desarrollada, por lo que no se obviará ningún elemento del diseño.

2.3.2 El vehículo eléctrico

Los vehículos de tracción eléctrica[1] presentan múltiples ventajas en comparación con los propulsados por el clásico motor de combustión interna. Entre ellas se pueden destacar el hecho de que poseen un mayor rendimiento energético [2], ofrecen un par constante desde el inicio de la marcha [3], y una contaminación directa nula [4].

Actualmente el limitante en cuanto a desarrollo de vehículos eléctricos a gran escala es la autonomía de éstos, generalmente muy inferior a su homólogo de combustión (debido a que la energía específica es mucho menor que en los carburantes tradicionales [5]). Puesto que el desarrollo de nuevas tecnologías de almacenamiento de electricidad más allá de la batería de ion-litio aún se encuentra en fases muy prematuras [6], es necesario buscar tecnologías que permitan alcanzar un mejor rendimiento.

Además de las diferentes soluciones mecánicas y eléctricas que Bultaco ha desarrollado (como el sistema de frenado regenerativo o el sistema de suspensión *DUAL LINK Evolution*), se hace necesario desarrollar nuevas tecnologías asociadas a este nuevo paradigma que supone el vehículo eléctrico.

En dicho paradigma, el modo en que un usuario recarga su vehículo cambia totalmente. En este nuevo escenario, el usuario típicamente saldrá de su casa con la máxima cantidad de

[1] En este contexto se refiere a **vehículos de tracción eléctrica** a aquellos cuya tecnología de almacenamiento de energía está basada en baterías (típicamente de ion-litio) por ser la tecnología implementada por Bultaco, así como por otras grandes fuerzas de la industria como Tesla Motors Inc. Existen diferentes tecnologías como la pila de combustible de hidrógeno [36], pero que energéticamente son menos eficientes [37] [4].

energía posible (dado que puede recargar la motocicleta en una regleta convencional del garaje de su casa). Esto, junto con el hecho de que los tiempos de carga sean tan elevados, hace que una buena **estimación del consumo** para una determinada ruta sea crítica.

2.3.3 Biker Manager

En base al problema de autonomía planteado, se diseñará e implementará una aplicación para dispositivos móviles que provea una solución al mismo. No obstante, además de dicho sistema de estimación, y como ya se ha comentado anteriormente, se implementarán numerosas funcionalidades adicionales en base a las especificaciones de Bultaco. Dichas funcionalidades se listan a continuación:

- Conexión con la motocicleta a través de Bluetooth/USB, de tal forma que permita mostrar telemetrías en tiempo real.
- Almacenar dichas telemetrías asociándolas a rutas realizadas, de tal forma que el usuario pueda tener una vista/resumen de dichas magnitudes en los trayectos que haya realizado.
- Histórico de avisos/testigos que la motocicleta ha encendido (derivados típicamente de algún error de funcionamiento).
- Acceso directo a comandos de emergencia: llamada de emergencia, obtención de la localización de la moto o bloqueo de la moto a distancia.
- Proveer de consulta de mapas así como de navegación guiada (todo ello de manera totalmente *offline*).
- **Estimación de consumo inteligente**, basada en el aprendizaje del comportamiento de cada usuario. Realizará una estimación de viabilidad de rutas en base al comportamiento registrado en rutas anteriores y a las características de la ruta a realizar.

El conjunto de especificaciones técnicas, incluyendo aquellas como la telemetría que se recibirá (tipos de datos y sus magnitudes) a través de la trama que enviará la motocicleta, así como la caracterización del funcionamiento de ésta se encuentran detalladas en el Anexo 9.1.

2.4 Conclusión

Se desarrollará una aplicación móvil para la empresa *Bultaco Motors* que, además de numerosas funcionalidades destinadas a ofrecer una experiencia enriquecida al usuario, sea capaz de aprender del uso de la motocicleta y ofrecer estimaciones de consumo adaptadas a cada usuario.

Para ello se implementará un sistema de navegación guiada que, una vez calculada la ruta, consulte el **Modelo de Predicción de Consumo** y ofrezca una estimación de la viabilidad (tanto del trayecto de ida como el de ida y vuelta). Idealmente además, ofrecería al usuario una serie de puntos de recarga por los que podría parar en caso de que la ruta no fuera viable.

3 Estado del arte

3.1 Introducción

En la búsqueda de una solución lo más óptima posible para el problema que se pretende resolver, se hace necesario realizar cierto trabajo de investigación que permita saber las diferentes aproximaciones que se han realizado a día de hoy en la materia que nos ocupa.

A continuación se realizará una evaluación de dichas aproximaciones para así establecer un punto de partida en el que basar el desarrollo de este trabajo.

3.2 Alternativas: evaluación y selección

Tradicionalmente, el consumo de combustible de un vehículo es calculado por el ordenador de a bordo del mismo. Dicho cálculo se realiza a partir de la distancia recorrida por el vehículo y la cantidad teórica de combustible consumida por éste (normalmente basado en la duración de los pulsos de inyección de combustible) [7].

En base a dicho cálculo, típicamente se ofrece como lectura al usuario tanto el consumo instantáneo como el consumo medio. Es a partir de dicho cálculo, junto con la medida del nivel de combustible, como dicho ordenador calcula la autonomía en distancia del vehículo (pudiendo aplicar algún tipo de filtro FIR paso bajo a dicha función que promedie el resultado con el histórico).

Aunque el resultado obtenido en dichos cálculos pueda ser suficiente para el vehículo tradicional de explosión, no es así para su homólogo eléctrico en el que la autonomía juega un papel tan crucial (al menos hasta que el número de puntos de recarga se equipare al número de gasolineras).

En base a dicho problema, se han diseñado diferentes aproximaciones con el objetivo de proveer una estimación más precisa. En este texto se analizarán varias de ellas con el objetivo de evaluar sus resultados y comprobar así tanto su posible aplicación a este proyecto, como la posible incorporación de ideas o mecanismos al sistema a diseñar.

3.2.1 Evaluación

3.2.1.1 Comportamiento no-lineal de las baterías

Al contrario que un tanque de combustible en un tradicional vehículo de explosión, la cantidad de energía que puede extraerse de una batería no tiene un comportamiento lineal. Estos sistemas de almacenamiento de energía tienen un comportamiento sumamente complejo y fuertemente no-lineal.

En una batería, la cantidad de energía que puede ser extraída depende de la tasa de descarga de corriente. Cuanto mayor es la corriente entregada, menor es la carga (y la energía) que puede ser extraída. Cuando una batería es descargada a altas tasas, aunque parte de la energía almacenada no está disponible a dichas tasas, ésta no ha desaparecido sino que puede ser extraída posteriormente a tasas menores. En comparación con el

tanque de combustible comentado anteriormente, éste podría ser análogo a una batería en cuanto a la capacidad de almacenar energía, solo que dicha capacidad variaría dinámicamente en función de las tasas de descarga. Dicha característica se puede observar mejor en base a la Figura 1 - No linealidad de las baterías.

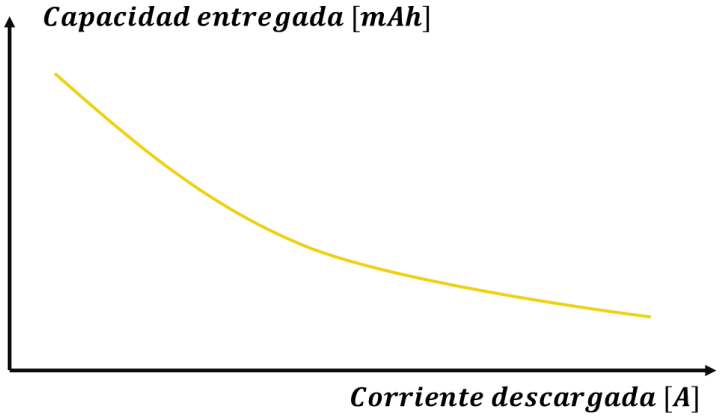


Figura 1 - No linealidad de las baterías

Debido a este fenómeno, la tarea de estimar la autonomía de un vehículo eléctrico no es trivial, ya que para conocer la cantidad de energía disponible será necesario conocer el histórico de las tasas de descarga, así como una posible estimación de las tasas que se realizarán en el futuro.

En base a este comportamiento, Ceraolo y Pede en su artículo **Techniques for Estimating the Residual Range of an Electric Vehicle** [8] diseñan un sistema que pretende estimar así la cantidad de carga/energía restante en la batería de un vehículo eléctrico. Para ello emplean diferentes medidas en conjunción con un modelo numérico, obteniendo el SOC (*State of Charge*) de la batería y calculando a partir de éste la carga residual. Una vez obtenida dicha carga residual se calcula la autonomía en base al ratio de consumo por kilómetro. Dicho ratio se asume constante junto con el estilo de conducción del piloto, además será en base a la experiencia de éste como podrá estimar el consumo que tendrá su vehículo (sabiendo que si realiza una conducción más deportiva, el consumo será mayor y por tanto la autonomía menor).

Es decir, el objetivo más que realizar una buena estimación de autonomía, se centra en ofrecer un modelo adecuado a través del cual obtener la energía restante en la batería y una vez aplicado éste, calcular la autonomía en base un dato de consumo estático (kWh/km).

3.2.1.2 Estimación según modelos estocásticos

Otro punto de vista es el que toman Javier A. Oliva, Christoph Weihrauch y Torsten Bertran en su artículo **A Model-Based Approach for Predicting the Remaining Driving Range y Electric Vehicles** [9]. En dicho documento, se propone una predicción del consumo con un modelo quasi-estático en el que las variables más dinámicas (velocidad, aceleración e inclinación de la vía) se estiman mediante cadenas de Markov cuyas probabilidades de transición se determinan en base a datos históricos de conducción así como patrones estándar de conducción para el caso de la velocidad y la aceleración; y en base a los perfiles de altitud de carreteras reales para el caso de la inclinación. De esta manera

consigue “estimar” el perfil de conducción, obteniendo una autonomía en forma de densidad de probabilidad (en lugar de realizar una predicción determinista).

Es decir, en lugar de realizar la estimación directa de la autonomía/consumo, se hace una estimación del comportamiento del conductor. Dicho comportamiento vendrá definido por una función de densidad de probabilidad que podrá analizarse para obtener la probabilidad de que la autonomía esté dentro de un determinado rango.

3.2.1.3 Modelado del vehículo eléctrico

En el artículo **Range Estimation for Electric Vehicles** de Oliver Zirn, Michael ahlborn, Raul Heyne y Christian Vetter [10] se propone la implementación de un modelo de coche eléctrico basado en Simulink, el cual considera diferentes características de la ruta como la altitud, el límite de velocidad, el tipo de pavimento y las condiciones meteorológicas.

Una vez construido el modelo permite estudiar diferentes situaciones y así podría calcular el consumo de una determinada ruta si dispone de información suficiente de los diferentes parámetros que emplea en el modelo.

3.2.1.4 Ruta con menor consumo energético

Myriam Neaimeh, Graeme A. Hill, Yvonne Hübner y Phil T. Blythe en su artículo **Routing Systems to Extend the Driving Range of Electric Vehicles** [11], detallan un sistema que pretende encontrar la ruta óptima hasta un destino en función de la energía que se consumirá en dicha ruta. Es decir, en lugar de utilizar el algoritmo de optimización típico que emplean los navegadores GPS que busca la ruta más corta o más rápida, este sistema busca aquella ruta cuyo consumo energético será menor. Para ello básicamente realiza una regresión lineal con datos reales y una vez construida la utiliza para evaluar el consumo energético para diferentes velocidades e inclinaciones.

Para construir dicha recta de regresión utiliza básicamente la velocidad típica de la vía y la inclinación de la carretera en cuestión. Tras un análisis tramo a tramo de las diferentes rutas posibles, elige aquella cuyo cálculo de consumo energético es menor.

3.2.1.5 Energía consumida en ruta conocida

En el artículo **Electric Vehicle Energy Usage Modeling, Simulation and Testing for Range Estimation** de Robbie Hurlbrink, Lee Winslow y Robert Prins [12] plantean un modelo de la energía consumida en una determinada ruta (ya conocida) mediante un balance de las fuerzas involucradas en el movimiento del vehículo.

3.2.1.6 Cálculo de autonomía instantánea

Wenjia Wang, Yuhe Zhang, Yuichi Kobayashi y Keisuke Shirai proponen en su artículo **Remaining Driving Range Estimation of Electric Vehicle** [13] un cálculo periódico de la autonomía del vehículo, proporcionando así una estimación actualizada en todo momento. Para ellos toma en consideración diferentes parámetros como la localización de éste, la energía restante de la batería, las características de la ruta o el tráfico en tiempo real.

El objetivo es calcular el punto en el que se quedaría sin energía tomando varios puntos de referencia (en una circunferencia de una determinada distancia) y calculando el coste energético de llegar a cada uno de dichos puntos.

3.2.2 Selección

Se ha visto que una característica deseable del sistema a construir sería que fuese capaz de analizar la tasa de descarga de la batería puesto que de esta forma se tendría mayor certeza de la energía acumulada (debido al comportamiento no-lineal de ésta).

A pesar de que se pudiera analizar dicha característica, sería un ingrediente ideal para el sistema algún tipo de mecanismo que pudiera asociarse al tipo de comportamiento que tiene un determinado conductor. Es decir, que el sistema se adapte a su tipo de conducción.

Así mismo, sería conveniente disponer de un sistema de modelado del comportamiento de un vehículo eléctrico que permita así realizar estimaciones en situaciones reales. De esta manera se podrían ofrecer estimaciones de consumo en base a una serie de parámetros como pueden ser la inclinación de la carretera, el tipo de calzada, el tráfico, las condiciones meteorológicas, etc.

Se conseguiría juntar por tanto con un análisis cartográfico de las diferentes alternativas de navegación y, utilizando las características de éstas, proveer una estimación de consumo en conjunto con el modelo de vehículo creado. Es decir, la autonomía calculada vendría dada por el modelo de vehículo creado, que además también estará adaptado al comportamiento del conductor, y por las características de la ruta (además de poder añadir otros factores externos como las condiciones meteorológicas o el tráfico).

Además de todos los factores comentados, sería también deseable añadir cierta periodicidad al cálculo de dicha autonomía, de tal forma que se actualice en tiempo real y no realice únicamente una predicción inicial, sino que en función de la energía restante se recalcule conforme se avanza en la ruta.

3.2.3 Conclusión

Se buscará construir un sistema que se acerque a construir un modelo del vehículo eléctrico lo más preciso posible, pero que además del comportamiento intrínseco al propio vehículo, tenga en cuenta el tipo de conducción del piloto.

Una vez construido dicho sistema se podrá utilizar para simular su comportamiento en diferentes condiciones (podrá tener diferentes parámetros de entrada en función de la disponibilidad de éstos o de la dificultad de su implementación), como por ejemplo dar una estimación del consumo en una determinada ruta y que ofrezca información acerca de la viabilidad de la misma con la energía disponible.

3.3 Resumen de la teoría

Para la construcción del modelo de predicción de consumo se emplearán métodos de **aprendizaje máquina** (***Machine Learning***). Estos métodos son capaces de extraer características de un conjunto de datos sin necesidad de conocer la física que los ha generado.

En concreto para este proyecto se empleará un sistema basado en **Redes Neuronales (*ANN – Artificial Neural Networks*)**, más concretamente un **Perceptrón Multi-capa (*MLP – Multi-layer Perceptron*)**.

La implementación de dicho sistema permitirá al usuario obtener una estimación de consumo en base a las características de la ruta que desee hacer, así como de sus características como piloto. Es decir, el sistema aprende del comportamiento usuario-moto para así realizar estimaciones adaptadas a éste.

En los siguientes apartados se realizará una introducción a la tecnología de Inteligencia Artificial empleada, así como un análisis de las características del problema. De igual modo, se establecerán las bases en las que se fundamenta la elección de esta tecnología para la resolución de este problema en particular.

3.3.1 Redes de Neuronas

En el contexto presente, se utiliza el término Redes Neuronales o Redes de Neuronas en referencia a las **Redes Neuronales Artificiales** (RNA, *Artificial Neural Networks – ANN*). Es decir, se refiere a la recreación matemática de los sistemas nerviosos biológicos donde la neurona es el elemento fundamental: las redes neuronales presentes en el cerebro.

3.3.1.1 ¿Qué es una Red Neuronal?

A pesar de que no exista una definición universalmente aceptada del concepto de Red Neuronal, generalmente los profesionales afines aceptarían definir una Red Neuronal como una red de varios simples procesadores (unidades o neuronas), cada uno de los cuales tiene una pequeña cantidad de memoria. Dichas unidades están conectadas a través de canales de comunicación (conexiones) que típicamente tienen asociado un valor numérico, que puede estar codificado de varias formas. Las unidades operan únicamente en base a sus datos locales y las entradas que reciben a través de las conexiones.

Algunas Redes Neuronales son modelos de redes neuronales biológicas, pero no en todos los casos es así. No obstante, históricamente la mayor inspiración de esta técnica viene del deseo de construir sistemas artificiales capaces de realizar cálculos en cierto modo *inteligentes*, similares a los que típicamente realiza un cerebro humano (además de mejorar la comprensión de éste).

La mayoría de Redes Neuronales tienen algún tipo de regla de **entrenamiento** a través de la cual los pesos de las conexiones se ajustan a un conjunto de datos dado. En otras palabras, las redes **aprenden** de los ejemplos. Si dicho entrenamiento se produce bajo ciertas condiciones, estos sistemas pueden presentar cierto tipo de generalización más allá del conjunto de datos con el que se ha entrenado. Es decir, será capaz de producir resultados bastante aproximados para nuevos datos de entrada.

3.3.1.2 ¿Qué puede hacer una Red Neuronal?

Las aplicaciones prácticas que puede tener una Red Neuronal emplean a menudo **aprendizaje supervisado**[2]. Para este tipo de aprendizaje, es necesario proveer al sistema de

[2] Existen otros tipos de aprendizaje como el **no supervisado**, donde no se aporta información acerca de los datos de salida y es el propio sistema el que *clasifica* los datos de entrada.

un conjunto de datos de entrenamiento en el que se incluyan tanto las entradas del sistema como las salidas correspondientes (el valor deseado o *target*). Después del entrenamiento, se pueden introducir los datos de entrada y la red calculará una salida que se aproxime al valor deseado.

En la práctica, las Redes Neuronales son especialmente útiles tanto para problemas de **clasificación** como de **aproximación/mapeo de funciones** que sean tolerantes de cierta imprecisión y que tengan gran cantidad de datos disponibles. Prácticamente cualquier función vectorial de dimensión finita sobre un conjunto compacto puede ser aproximada con cierta precisión por una Red Neuronal de tipo *feedforward*[3] si se tienen suficientes datos y capacidad de cómputo.

3.3.1.3 Estructura de una Red Neuronal

Atendiendo al comportamiento matemático que se implementa en el modelo, una neurona actúa en función de un determinado número de entradas, ponderadas a través de los pesos de la sinapsis que interconecta las capas. Se calcula la salida de dicha neurona como la suma de dichas cantidades ponderadas, aplicando finalmente una **función de activación** que introducirá la **no-linealidad** a la red. En la Figura 2 - Perceptrón simple, se puede observar la estructura básica que tendrá cada neurona del sistema.

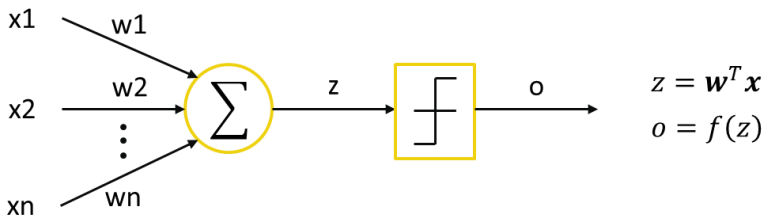


Figura 2 - Perceptrón simple

Dicha función será típicamente la función escalón en un **Perceptrón**, actuando así como un umbral cuya salida será 1 en caso de superar dicho umbral, o 0 en caso contrario.

Sin embargo, cuando se habla de Redes Neuronales, se refiere a un conjunto de estos elementos básicos. En dicha estructura, que se puede ver en la Figura 3 - Estructura general Red neuronal, la función de activación no podrá ser la función escalón, sino que tendrá que ser de tipo *sigmoide* para así poder realizar el entrenamiento en base a algún tipo de algoritmo de descenso por gradiente, esto es, tiene que ser una función continua y derivable.

En cuanto a la estructura típica que tiene una Red Neuronal, ésta se puede clasificar en función del número de capas. Se dice que una Red Neuronal tiene una única capa cuando no existen capas intermedias que conecten las entradas con las salidas. En caso contrario, se dice que es una Red Neuronal multi-capa, la cual constará de una o varias capas ocultas con un determinado número de neuronas.

[3] En las redes *feedforward* las conexiones se realizan en un único sentido (desde la entrada hacia la salida). Existen otros tipos de redes según el tipo de conexión como son las redes recurrentes o las parcialmente recurrentes.

A pesar de que las entradas del sistema reciben también la nomenclatura de neuronas, éstas no se comportan como tal, ya que no tienen pesos asociados ni función de activación. Es decir, la definición de neurona comenzaría con la sinapsis de la capa anterior (pesos de las conexiones) así como la función de activación realizada a la combinación lineal de dichas entradas ponderadas.

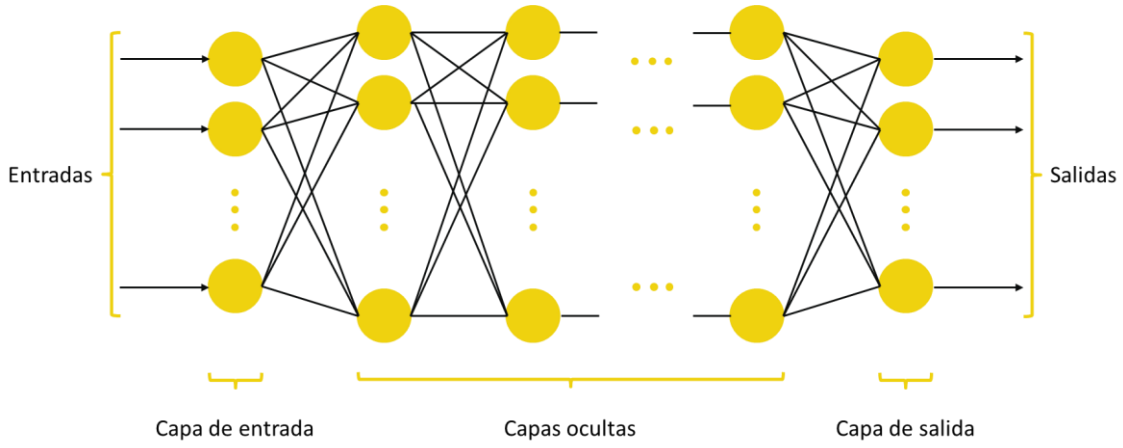


Figura 3 - Estructura general Red neuronal

Desde el punto de vista de la **Inteligencia Artificial (IA)**[4], en el área referente al **aprendizaje automático** (aprendizaje máquina o *machine learning*), el objetivo es desarrollar técnicas que permitan a los sistemas aprender, entendiendo dicho concepto como la capacidad de **generalizar comportamientos** a partir de una información no estructurada proporcionada en forma de ejemplos. Es decir, se emplea algún tipo de algoritmo que permita reducir el error en una serie de datos de entrenamiento.

Se emplea un conjunto de algoritmos de *descenso por gradiente* para alterar los parámetros del sistema con el objetivo de reducir algún tipo de medida de error y de esta forma se *ajuste* al modelo o función que generó los datos.

3.3.1.4 Aprendizaje

El aprendizaje, o mecanismo a través del cual se *entrena* la Red Neuronal, no es más que un **algoritmo de búsqueda**. Dicho algoritmo típicamente lo que pretende es buscar el mínimo de una función de coste dada, es decir, trata de **minimizar el coste**[5] que el diseñador establezca. En base a dichas premisas, se pueden clasificar los algoritmos de búsqueda en función de si dicha búsqueda consiste en una búsqueda global (mínimo global de la función de coste) o en una búsqueda local (mínimo local). Es decir, se tiene una función $\boldsymbol{C_w}$, cuya variación con $\boldsymbol{w}$ es arbitraria como se muestra en la Figura 4 - Función de coste y de la cual se pretende encontrar un mínimo.

Como no siempre es posible obtener una solución cerrada, es necesario aplicar algún tipo de algoritmo de búsqueda. Incluso cuando sea accesible dicha solución cerrada, puede ser

[4] Área multidisciplinaria que, a través de ciencias como la informática, la lógica y la filosofía, estudia la creación y diseño de entidades capaces de razonar por sí mismas utilizando como paradigma la inteligencia humana [14].
[5] Puede tratarse como el coste asociado a una decisión o como una medida de la diferencia entre la señal que se desea estimar (deseada) y la estimada (obtenida).

interesante aplicar una búsqueda para conseguir **generalización** ya que la minimización para las muestras no equivale a la minimización para todo el espacio de observación, pudiendo producirse sobreajuste, lo cual se puede evitar:

- **Regularizando** la solución.
- Eligiendo un coste adecuado.
- Entrenando hasta un cierto límite (lo que es posible mediante la detención de un **algoritmo iterativo** de búsqueda). Técnica ampliamente conocida como ***early-stopping***.

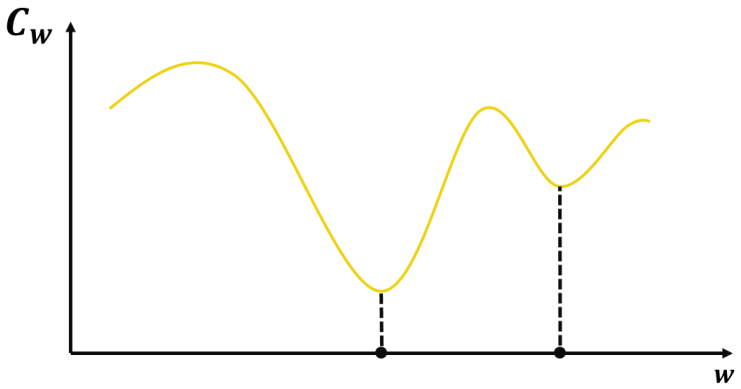


Figura 4 - Función de coste

En general, los métodos de búsqueda globales son computacionalmente muy costosos. Es por ello que únicamente se aplican en casos en que merezca la pena perseguir el mínimo absoluto. Si basta con un buen mínimo relativo, suele ser suficiente repetir búsquedas locales con distintas inicializaciones y seleccionar la mejor solución. Aunque ambos mecanismos tienen fundamentos diferentes, éstos incluyen un mecanismo de **exploración** del espacio de soluciones ($\boldsymbol{w}$) y uno de **explotación** de los resultados favorables de dicha exploración.

Unos de los algoritmos de búsqueda local más ampliamente utilizados son los denominados **algoritmos de gradiente**. El planteamiento es el siguiente:

Si se trabajase mediante procedimientos analíticos, es decir, conociendo $\bar{C}(\boldsymbol{w})$ (el coste medio, que incluye la promediación sobre todo el espacio muestral), y $\boldsymbol{g}(\boldsymbol{w}) = \nabla_{\boldsymbol{w}}\bar{C}(\boldsymbol{w})$ es su gradiente, entonces proceder de la forma (1)

$$\boldsymbol{w}^{nuevo} = \boldsymbol{w}^{anterior} - \eta \boldsymbol{g}\left(\boldsymbol{w}^{anterior}\right), \quad \eta > 0 \qquad (1)$$

conducirá a un mínimo local, si η es suficientemente pequeño.

Este procedimiento se conoce como **algoritmo del descenso más pendiente** (*steepset descent*) y requiere conocer $\bar{C}(\boldsymbol{w})$, lo que no es habitual.

En situaciones muestrales, se puede trabajar con el coste para las muestras. Aunque es posible proceder para la totalidad de éstas, es frecuente y recomendable hacerlo secuencialmente, esto es: muestra a muestra, repitiendo el ciclo tantas veces como se precise, y extendiendo de este modo el conjunto de muestras. Se procederá entonces de la

forma (2), que es el **algoritmo (secuencial) de gradiente**. Donde $\boldsymbol{g}^{(k)}$ es el gradiente del coste evaluado para la k-ésima muestra.

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \boldsymbol{g}^{(k)}\left(\boldsymbol{w}^{(k)}\right) \quad (2)$$

Para el caso de coste cuadrático y estimador lineal, obtenemos las expresiones de (3), el cual se conoce como el algoritmo **LMS (*Least Mean Squares*)** o de **Widrow-Hoff**.

$$\begin{gathered} \hat{s} = \boldsymbol{w}^T \boldsymbol{x} \quad (valor\ estimado) \\ C(\boldsymbol{w}) = \frac{1}{2} E\{(s - \boldsymbol{w}^T \boldsymbol{x})^2\} \\ \boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \eta \left(d^{(k)} - \boldsymbol{w}^{(k)^T} \boldsymbol{x}^{(k)}\right) \boldsymbol{x}^{(k)} \end{gathered} \quad (3)$$

Como se puede observar, el valor estimado no es más que la combinación lineal de una serie de valores ponderados por un conjunto de pesos, es decir, es la salida del perceptrón antes de emplear la función de activación. Cuando dicha función está presente, dado que ésta es continua, se hallará el nuevo peso a través de la ecuación general (4).

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} + \eta\left(d^{(k)} - f(\hat{s})\right) f'(\hat{s}) \boldsymbol{x}^{(k)} \quad (4)$$

Por lo tanto, se trata de un algoritmo que ajusta los pesos secuencialmente tratando de corregir una medida del error entre la salida deseada y la obtenida.

Dicho algoritmo se ha adaptado para su uso en Redes Neuronales, conformando lo que se conoce como el **algoritmo de retropropagación (BP – *Backpropagation*)**. Dicha adaptación es necesaria puesto que en dicho contexto existen varias capas, es decir varias sinapsis con sus correspondientes pesos a actualizar. La aplicación de la ecuación es inmediata para la capa de salida donde se puede acceder a la salida del sistema y así calcular el error, pero para las capas intermedias es necesario retropropagar el error de dicha capa de salida, y así poder ajustar los pesos del resto de capas.

3.3.2 Caracterización del problema

Se pretende dar solución al problema que supone la estimación de la autonomía de un vehículo eléctrico. Será necesario por tanto examinar la física envuelta en el consumo de energía de un vehículo de estas características. Para dicho análisis se toma como referencia el artículo elaborado por *Solar Journey USA* y que lleva por título *EV Energy Consumption* [15].

Además de analizar la física involucrada en el consumo de energía de un vehículo eléctrico, es necesario considerar el comportamiento no-lineal de la entrega de energía de las baterías de éste en función de la cantidad de energía demandada.

En la caracterización del modelo de consumo de energía de un vehículo eléctrico se tendrán en cuenta diferentes aspectos de la dinámica del vehículo. En concreto se analizará el consumo a velocidad constante, la energía involucrada en los cambios de velocidad, las pérdidas inherentes al funcionamiento del sistema, así como las características de la vía.

3.3.2.1 Consumo a velocidad constante

Existen cuatro elementos principales que analizar en lo que respecta al consumo cuando el vehículo se encuentra a velocidad constante:

1. Aerodinámica.
2. Sistema de transmisión.
3. Neumáticos.
4. Sistema auxiliar.

Las pérdidas debido a la **resistencia aerodinámica** son especialmente importantes a altas velocidades. La fuerza de fricción del aire en un objeto es un vector que apunta en la dirección contraria al movimiento del objeto y cuya magnitud viene definida por

$$F_D = \frac{1}{2}\rho V^2 A C_d \quad (5)$$

Donde ρ es la densidad del aire, V es la velocidad del vehículo, A es el área frontal de éste y C_d es el coeficiente de arrastre (*drag coeficient*).

Se simplifica la expresión agrupando coeficientes, calculando así la pérdida de potencia debido a la aerodinámica como

$$P_{Aer} = \alpha_{Aer} \cdot V^3 \quad (6)$$

Para el vehículo analizado en el artículo, un *Testa Roadster*, $\alpha_{Aer} = 3.45 \cdot 10^{-4}$ (para la potencia en kW).

En cuanto a las **pérdidas del sistema de transmisión**, son las debidas a la conversión de energía en la batería en el par de las ruedas del vehículo. Dichas pérdidas son muy difíciles de derivar de ecuaciones físicas, debido a que el rendimiento de varios sistemas del vehículo tendría que estar modelado individualmente. Se considera por tanto el polinomio propuesto en el artículo original

$$P_{Dr} = \alpha_{Dr} V^3 + \beta_{Dr} V^2 + \gamma_{Dr} V + C_{Dr} \quad (7)$$

Donde C_{Dr} es el uso de potencia del sistema de transmisión cuando el vehículo no se mueve.

Para el vehículo analizado en el artículo, $\alpha_{Dr} = 4 \cdot 10^{-6}$, $\beta_{Dr} = 5 \cdot 10^{-4}$, $\gamma_{Dr} = 0.0293$ y $C_{Dr} = 0.375\ kW$ (para la potencia en kW).

Si se analiza la pérdida de potencia en los **neumáticos**, se puede ver que la principal causa de la resistencia a la rodadura (*rolling resistance*) es el efecto de histéresis: la energía necesaria para deformar la forma del neumático es mayor que la energía de recuperación. Por esta razón, es aconsejable comprobar la presión de los neumáticos, puesto que un neumático blando tiene más histéresis que uno duro.

La potencia requerida para superar la resistencia de rodadura es función de la fuerza normal N (pero llevado por la rueda) y el coeficiente de resistencia a la rodadura, y es proporcional a la velocidad

$$P_{Tire} = C_{rr} N V \quad (8)$$

Para cada rueda habrá una pérdida diferente puesto que el peso será diferente (en función de la distribución de pesos).

Para el vehículo analizado en el artículo, $C_{rr} = 0.0075$ y la pérdida varía entre 2 y 2.8 kW.

Las **pérdidas en el sistema auxiliar** engloban el resto de pérdidas eléctricas en el vehículo. Algún ejemplo serían las relacionadas con el usuario: climatizador, luces y sistema de audio, así como sistemas para regular la temperatura de la batería. En el caso de una motocicleta, se podría reducir a la expresión

$$P_{Anc} = P_{Bat.Management} + P_{lights} \quad (9)$$

Para el vehículo analizado en el artículo, $P_{Anc} = 0.18\ kW$.

3.3.2.2 Consumo para variaciones de velocidad

Unas condiciones reales de conducción incluyen aceleraciones y frenadas, que incrementan y decrementan la energía cinética (respectivamente). Se mostrará a continuación cómo una conducción agresiva afecta al consumo de energía y a la autonomía.

La energía cinética en el vehículo se conserva como energía cinética lineal y rotacional. La energía cinética lineal está en el movimiento total del vehículo en su dirección y la energía cinética rotacional se conserva en los elementos rotativos del vehículo (principalmente las ruedas).

$$E_{Kin\,Lin} = \frac{1}{2} m V^2$$
$$E_{Kin\,Rot} = \frac{1}{2} I \omega^2 \quad (10)$$

Donde m es la masa del vehículo, V su velocidad, I el momento de inercia y ω la velocidad angular.

Típicamente la energía rotacional es sólo el 5-10% del total de energía en el vehículo. Como es más fácil hallar la masa que el momento de inercia de todos los elementos rotativos del vehículo, se empleará la expresión

$$E_{Kin} = 1{,}05 \cdot E_{Kin\,Lin} \quad (11)$$

Incluso aunque el vehículo sea capaz de regenerar energía en el frenado (recupera cierta parte de la energía cinética y la almacena en las baterías), la mayoría de energía es disipada como calor en los discos de freno cuando se frena con fuerza.

En el artículo analizado asume una eficiencia en la regeneración del 30% cuando la frenada es fuerte (80 mph – 50 mph) y una eficiencia del 50 % cuando la frenada es suave (54 mph – 50 mph).

Como se puede observar en las ecuaciones, la energía necesaria para aumentar la velocidad es proporcional al cuadrado de dicho aumento, por lo que una conducción agresiva, con grandes variaciones de velocidad, provocará un gran consumo de energía.

3.3.2.3 Otros factores de influencia

3.3.2.3.1 Ruta

Las variaciones de altura influirán en el consumo de energía del vehículo, acortando la autonomía de éste en rutas que atraviesen puertos de montaña u otro tipo de carreteras con orografía abrupta. En la Figura 5 - Balance de fuerzas del vehículo se puede ver el balance de fuerzas del vehículo a partir del cual se realizará el análisis.

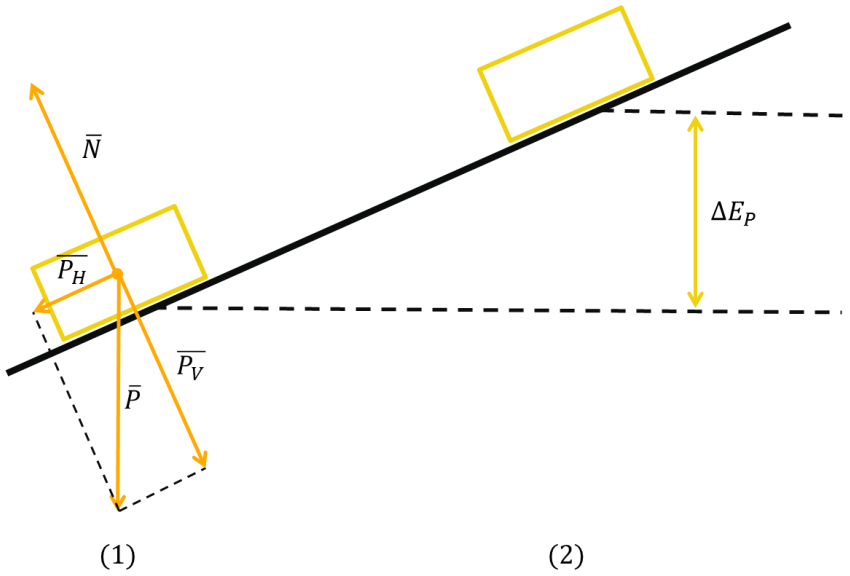


Figura 5 - Balance de fuerzas del vehículo

Por una parte se producirá un gasto de energía extra debido al incremento de la energía potencial producido al variar la altura del vehículo.

$$\Delta E_P = mg(h_2 - h_1) \quad (12)$$

Por otro lado, la inclinación de la vía tendrá que ser también considerada como un término adicional de fuerza en dirección contraria al desplazamiento del vehículo. Así mismo, al disminuir la fuerza *normal*, la resistencia de rodadura disminuirá en cierta medida.

$$\begin{aligned} P_H &= P \cdot sen\alpha \\ P_V &= P \cdot cos\alpha \end{aligned} \quad (13)$$

3.3.2.3.2 SOC (State-Of-Charge)

El estado de carga de la batería (o *State-Of-Charge*) es un valor absoluto basado en la capacidad de la batería cuando está nueva (no un porcentaje de la capacidad actual, lo que puede ser un error de hasta el 20% dependiendo de la edad de la batería).

Obviamente, la autonomía dependerá de la energía que hay en la batería

$$\begin{aligned} Carga[C] &= SOC \cdot Capacidad[C] \\ Energía[J] &= Tensión[V] \cdot Carga[C] \end{aligned} \quad (14)$$

3.3.2.3.3 Edad de las baterías

Las baterías se degradan a lo largo del tiempo en función del DOD (*Depth of Discharge*) y el número de ciclos de carga. También les afecta la exposición a temperaturas extremas.

3.3.2.3.4 Temperatura

Una baja temperatura ambiente reduce la capacidad máxima de carga de la batería.

3.4 Conclusión

Como se ha visto, utilizando un mecanismo de aprendizaje como es una Red Neuronal, se puede construir un sistema que modele el comportamiento de un vehículo eléctrico. Pero además del modelado del propio vehículo, dadas las características de este mecanismo, el modelo generado también incluirá las características del propio piloto, ya que serán los datos históricos generados por éste los que definirán el modelo en sí.

Idealmente dicho modelo será capaz de caracterizar los diferentes aspectos que se han analizado en esta sección y que los diferentes autores analizados pretendían modelar de una forma u otra como la no linealidad de las baterías, el comportamiento del piloto, etc.

De esta forma, será capaz de ofrecer una estimación de consumo para, por ejemplo, una determinada ruta con diferentes velocidades y diferentes ángulos de inclinación. Se podría incluso añadir otra información adicional como usan algunos de los autores analizados como son las condiciones meteorológicas (ya que la temperatura afecta de manera muy importante al rendimiento del motor eléctrico) o de tráfico.

Si además dicha estimación se realiza de forma periódica según el vehículo avanza a lo largo de una ruta, actualizando así su valor en función del valor real de energía disponible, se obtendría un nivel de precisión difícilmente alcanzable mediante otras metodologías.

Después de plantear el problema y una posible solución a éste, y habiendo aunado el conjunto de alternativas propuestas por otros autores, quedará analizar el modo de implementar dicho sistema, elegir una tecnología para dicha implementación y realizar las pertinentes pruebas de rendimiento que permitan establecer la viabilidad del sistema propuesto como solución al problema de estimación de consumo en un vehículo eléctrico.

4 Diseño

4.1 Introducción

En primer lugar se propondrá una aproximación conceptual al diseño del sistema completo. Posteriormente, se realizará un análisis detallado del diseño que se pretende adoptar en cuanto al sistema de Inteligencia Artificial se refiere. Finalmente, dado que se trata de un desarrollo software, se dará un enfoque específico a éste.

4.2 Diseño conceptual

Se pretende construir un sistema complejo que proporcione al usuario diferentes tipos de información asociados con su vehículo. Para ofrecer una visión global del diseño del sistema se presentarán diferentes esquemas con pequeñas variaciones en función del nivel de abstracción.

Tomando el mayor nivel de abstracción posible distinguimos tres elementos bien diferenciados en el esquema de funcionamiento. Dicho esquema se puede observar en la Figura 6 - Diagrama de bloques general.

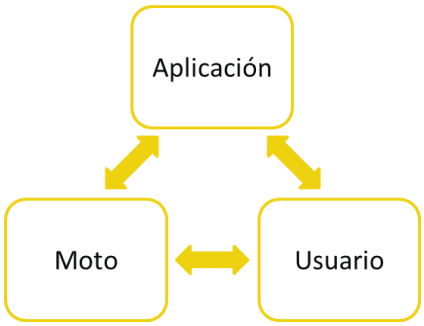


Figura 6 - Diagrama de bloques general

Por una parte tenemos al **Usuario**, el cual interactúa directamente con la moto a través de los controles mecánicos de ésta, así como a través de las lecturas disponibles en el *display*.

A su vez, la **Moto** se comunica con la **Aplicación** de *Smartphone*, recibiendo peticiones de telemetría y devolviéndolas a éste para que el usuario pueda verlas directamente en su dispositivo móvil. Éste, además de mostrar telemetrías, permite al usuario realizar diferentes tareas como buscar nuevos destinos y rutas, consultar antiguas rutas, iniciar la navegación u obtener una estimación de consumo (entre otras funciones).

La comunicación entre el dispositivo móvil y la motocicleta se debería poder realizar a través de dos mecanismos diferentes: *Bluetooth* o *USB*.

Profundizando más en el sistema, se puede establecer un diagrama de bloques más elaborado que permita distinguir los elementos básicos dentro de la implementación. Este esquema se puede observar en la Figura 7 - Diagrama de bloques general detallado.

Se puede ver que la entidad anteriormente denominada Aplicación está ahora formada por un sub-conjunto de entidades. Entre dichas entidades, destaca la denominada **APP**, la cual

hace referencia al elemento de interacción usuario-*Smartphone*, siendo el resto de bloques secundarios en cuanto a dicha comunicación se refiere.

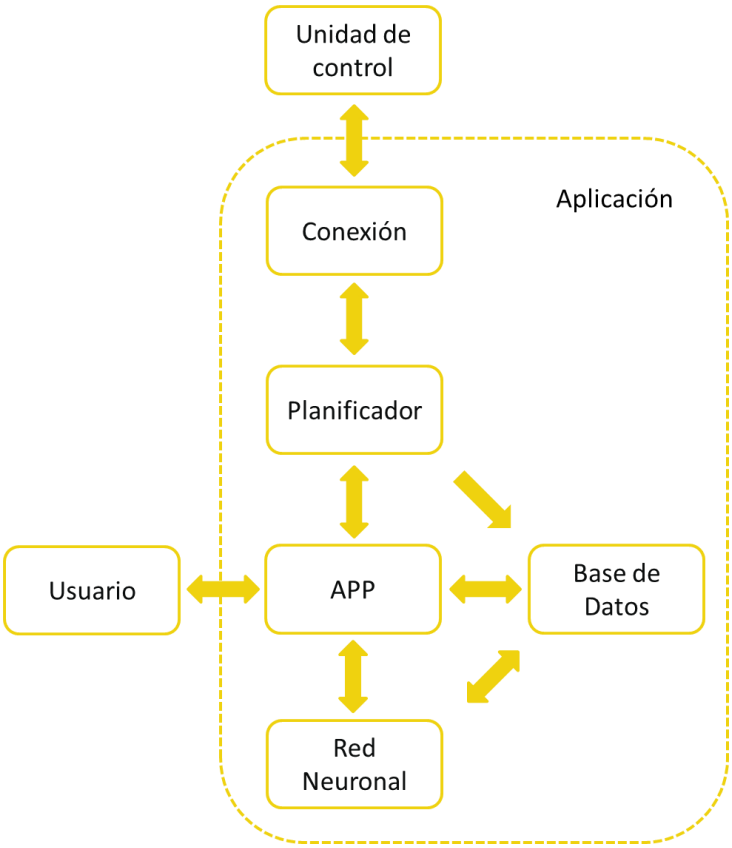


Figura 7 - Diagrama de bloques general detallado

A continuación se describen con más detalle cada una de dichas entidades.

La entidad referida como **APP** hace referencia a la parte de la aplicación referida a la comunicación usuario-*Smartphone*. Dicha interacción engloba diferentes aspectos como la navegación, consulta de mapas o visualización de resúmenes de rutas (entre otros).

Se podría decir que se refiere a las funcionalidades que el usuario percibe directamente, quedando el resto en un "segundo plano" de ejecución. Las funcionalidades accesibles son:

- Consulta de mapas y navegación guiada: el usuario puede utilizar la aplicación como una aplicación de GPS común (tipo TomTom o Garmin), pudiendo buscar un destino y obtener un guiado paso a paso (mediante imágenes y voz) hasta él.
  Si el usuario ha activado previamente el guardado de la ruta, se le proporcionará una estimación de la viabilidad de dicha ruta en función de la energía disponible en la batería de la motocicleta.
- Monitorización y guardado de rutas y telemetrías de la moto: una vez seleccionada una ruta, el usuario puede elegir guardar dicha ruta y sus parámetros de telemetría asociados. Esto le permitirá al usuario visionar datos de telemetría de la moto en tiempo real, así como visualizar posteriormente un resumen de las características de las rutas realizadas. Además, le permitirá tener acceso a un histórico de eventos con los errores asociados a la moto que se han producido en el transcurso de la ruta (que se reciben a través de las tramas de telemetría).

- Comandos de acceso directo: el usuario podrá acceder directamente a una serie de comandos de emergencia como son la realización de una llamada de emergencia, la obtención de la posición de la motocicleta y el bloqueo a distancia de la misma. Dichas acciones hacen uso de elementos inherentes al sistema como son la realización de llamadas o el envío de SMS.
- Configuración de usuario: como usuario de una motocicleta Bultaco, el usuario podrá registrarse como tal a través de la aplicación móvil. Parte de dicha información de usuario podría ser empleada para el sistema de estimación (como el peso o la altura).

Una vez que el usuario acciona el botón de grabación de ruta, la entidad **APP** pone en marcha el **Planificador**, el cual se encargará de realizar peticiones periódicas a la **Unidad de Control** a través de la **Conexión**. Además, el Planificador se encargará de guardar la información de telemetría en la **Base de Datos**, así como proporcionar el último valor leído a la APP para poder dar lectura de ello.

Cuando el usuario selecciona un destino en la aplicación de navegación, se procederá al cálculo de la ruta óptima. Una vez que se ha obtenido dicha ruta, la entidad **APP** se comunica con la **Red Neuronal**, pasándole como parámetro la ruta calculada y obteniendo una estimación de la viabilidad tanto para el trayecto de ida como para el trayecto de ida y vuelta. Para el cálculo de dicha viabilidad, la Red Neuronal toma la última información acerca de la energía disponible en la batería (dicha información estaría disponible a través de la entidad APP) y junto con la estimación de la energía necesaria para la ruta, da el resultado consecuente. En caso de que la ruta no fuera viable, podría dar al usuario la opción de elegir un punto intermedio para poder recargar la motocicleta.

Para visualizar el conjunto de rutas que se han realizado, la **APP** realiza una consulta a la **Base de Datos**. De esta forma el usuario puede consultar tanto el conjunto de rutas que ha realizado como las características de cada ruta en concreto, visualizando datos como la distancia total, energía total consumida y regenerada, velocidad media y máxima, etc. Así mismo, podrá ver en varias gráficas la evolución de magnitudes como la velocidad y la altura en función de la distancia recorrida.

Para que la **Red Neuronal** pueda realizar estimaciones, previamente ha tenido que ser entrenada con datos reales. Para ello, y tras haber realizado varias rutas, el usuario accionará dicho entrenamiento a través del menú de la aplicación. En dicho escenario entonces, la **APP** se comunicará con la Red Neuronal para que inicie su entrenamiento. Ésta se comunicará con la **Base de Datos** para obtener la información sobre la que se construirá el modelo de predicción de consumo. Una vez que se ha creado dicho modelo, se guardará en el dispositivo para que pueda usarse en futuras estimaciones.

Para que se pueda establecer la comunicación entre la entidad etiquetada como **Conexión** y la **Unidad de Control** de la motocicleta, y suponiendo que dicha comunicación se realizará a través de *Bluetooth*, previamente se tendrá que establecer la sincronización entre el *Smartphone* y ésta. Una vez que ambos dispositivos estén sincronizados, la conexión y desconexión será realizada por el **Planificador** siempre que sea necesario.

4.3 Diseño Inteligencia Artificial

Como se ha comentado anteriormente, el sistema de Inteligencia Artificial estará basado en una **Red Neuronal**. Introducido ya el mecanismo de funcionamiento de esta tecnología en el apartado 3.3.1, a continuación se especificará la estructura elegida y los parámetros a utilizar, así como otras consideraciones de diseño.

4.3.1 Modelo de predicción de consumo

El sistema de Inteligencia Artificial definirá un **Modelo de Predicción de Consumo (MPC)**, el cual estimará la viabilidad de la ruta en base al esquema de funcionamiento de la Figura 8 - Modelo de Predicción de Consumo.

Figura 8 - Modelo de Predicción de Consumo

Es decir, a partir de la ruta obtenida, se realizará un análisis de sus características y se determinará una estimación de su viabilidad (teniendo en cuenta, como se ha comentado anteriormente, la energía disponible en la batería en dicho instante).

4.3.2 Entradas y salidas de la Red Neuronal

A pesar de los numerosos parámetros que influyen en el consumo del vehículo (como se ha detallado en el apartado 3.3.2), se han elegido 4 de ellos para tratar de elaborar el modelo de predicción de consumo. Dichos parámetros han de estar disponibles en la información obtenida de la ruta, y es por ello que se ha limitado a dicho número. Los parámetros elegidos serán:

- **Distancia**.
- **Velocidad media**.
- **Incremento de altitud**.
- **Incremento de velocidad**.

Por tanto, dichos parámetros actuarán como entradas de la Red Neuronal. La salida que se buscará obtener será una medida del consumo de energía en función de dichas variables de entrada, por lo que se empleará el **incremento de energía**.

Puesto que tratando la ruta como un sistema completo se perdería gran cantidad de información, se dividirá en **pequeños bloques**, cada uno con una medida de distancia, velocidad media, incremento de altitud e incremento de velocidad. En caso de que la Red Neuronal se encuentre en modo de aprendizaje (aprendiendo de los datos almacenados), dichos bloques deberán también tener información acerca del incremento de energía (ya sea un incremento positivo o negativo en función de si se ha consumido o generado energía).

Por tanto, el funcionamiento de la Red Neuronal se basará en dichos bloques, por lo que a la hora de dar una estimación, lo hará para un bloque determinado, como se muestra en la Figura 9 - Red Neuronal: entradas y salidas.

Figura 9 - Red Neuronal: entradas y salidas

Se calculará por tanto el incremento total de energía de una ruta como la suma de los incrementos de energía estimados para cada bloque. Dicha configuración se puede ver en la Figura 10 - Estimación por bloques de ruta.

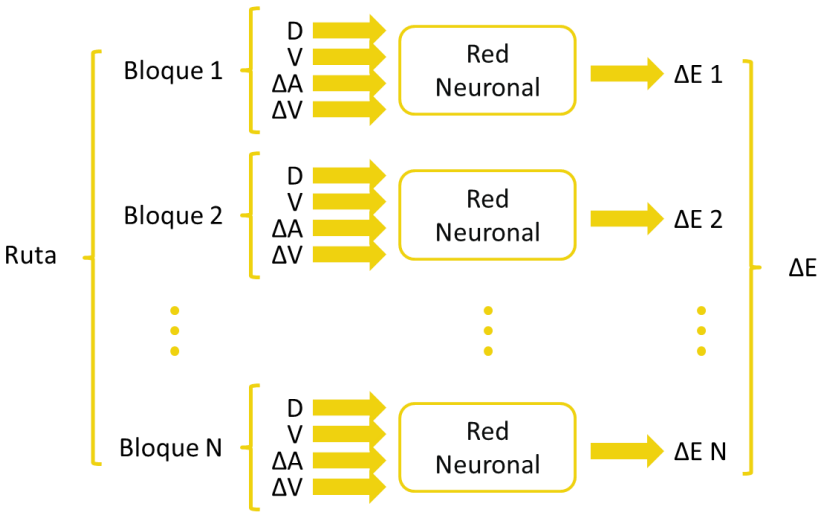


Figura 10 - Estimación por bloques de ruta

Una vez obtenida la estimación de energía que se consumirá para dicha ruta, se calculará la estimación de energía del camino de vuelta, para lo cual se generarán nuevos bloques de acuerdo al sentido inverso de la marcha.

De este modo, a la hora de construir el Modelo de Predicción de Consumo antes expuesto, se tomará lectura de la energía que dispone la motocicleta y mediante un sencillo cálculo determinará si la ruta es viable o no (para el trayecto de ida y para el trayecto de ida y vuelta). El cálculo a realizar se detalla en las ecuaciones (15) y (16).

$$\Delta E_{total} = \sum_i \Delta E_i \;\Rightarrow\; \begin{cases} \Delta E_{ida} \\ \Delta E_{ida\ y\ vuelta} = \Delta E_{ida} + \Delta E_{vuelta} \end{cases} \quad (15)$$

$$\Delta E_{ida} > E_{disponible} \;\Rightarrow\; Ruta\ no\ viable$$
$$\Delta E_{ida} < E_{disponible} \;\Rightarrow\; \begin{cases} \Delta E_{ida\ y\ vuelta} < E_{disponible} \;\Rightarrow\; Ruta\ viable\ (ida\ y\ vuelta) \\ \Delta E_{ida\ y\ vuelta} > E_{disponible} \;\Rightarrow\; Ruta\ viable\ (sólo\ ida) \end{cases} \quad (16)$$

Estando ya definidas las entradas y salidas, se diseñará a continuación la estructura a implementar.

4.3.3 Estructura de la Red Neuronal

La **estructura de la Red Neuronal** se establece en base al número de capas que la componen, así como el número de neuronas en cada una de dichas capas. Se tiene una capa de entrada, una o varias capas intermedias y una capa de salida.

El número de neuronas de la capa de entrada viene determinado por el número de parámetros que son relevantes para el problema a resolver, en este caso cuatro. El número de neuronas de la capa de salida vendrá determinado por las magnitudes que deseemos obtener, en este caso una.

En cuanto al **número de capas ocultas**, en un Perceptrón Multi-capa con cualquiera de una amplia variedad de funciones de activación continuas y no lineales en las capas ocultas, con una única capa oculta es suficiente para que sea un aproximador universal [16] [17]. Por tanto se optará por establecer **una capa oculta** (ver [18] para ampliar información) al menos para comenzar a realizar pruebas.

El **número de neuronas ocultas** depende de varios factores y no es una relación sencilla. Las principales variables que influyen en determinar este valor son [18]:

- Número de neuronas de entrada y salida.
- Número de datos para entrenamiento.
- La cantidad de ruido en los *target* (valor de salida deseado).
- La complejidad de la función a aprender.
- La arquitectura.
- El tipo de función de activación de las neuronas ocultas.
- El algoritmo de entrenamiento.
- Regularización.

No obstante, en la mayoría de situaciones no hay forma de determinar el número de neuronas ocultas sin entrenar varias redes y estimar el error de generalización de cada una de ellas. Se tomará como valor inicial (a falta de realizar pruebas con varias configuraciones) el propuesto por L. Berke y P. Hajela [19], los cuales sugieren que el número de nodos de la capa oculta debería ser un valor intermedio entre la media y la suma de las entradas y las salidas, esto es, un valor entre 2.5 y 5, por lo que se elegirán **4 neuronas** para la capa oculta.

Por tanto la estructura elegida (a falta de variaciones derivadas de los resultados sobre el error de generalización) quedará según el esquema de la Figura 11 - Estructura Red Neuronal.

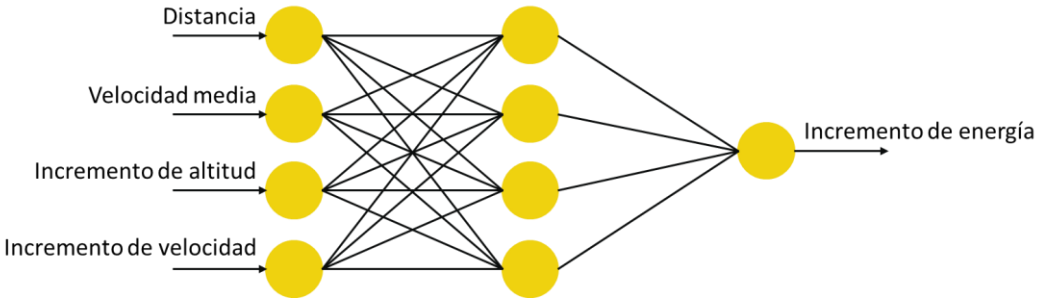


Figura 11 - Estructura Red Neuronal

4.3.4 Función de activación

Además de estos elementos, es necesario definir la función de activación que se empleará. Esto es necesario puesto que es a través de dicha función como **se introduce la no-linealidad** a la red, ya que de otro modo al ser una función lineal de funciones lineales, obtendríamos como resultado otra función lineal. Dicha función tendrá también que ser derivable para así poder utilizar el **algoritmo de retropropagación** (*BP – Backpropagation*) comentado en el apartado 3.3.1.

Unas de las funciones más comunes son las conocidas como *funciones sigmoides*, entre las que podemos encontrar la ***función logística***, la *tangente hiperbólica* o la *función gaussiana* [18]. Elegiremos por sencillez la primera de ellas, la cual se detalla en la Figura 12 - Función logística.

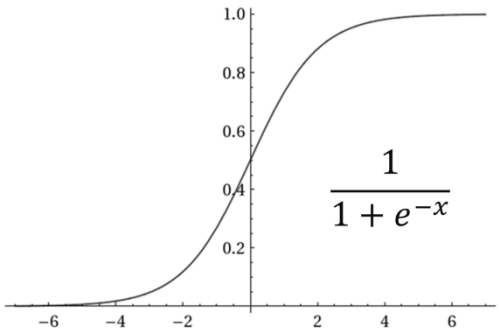


Figura 12 - Función logística

Como también se empleará dicha función de activación para la capa de salida, es necesario realizar un **escalado o normalización** de los datos de salida a la salida de dicha función. Para escalar la variable de salida dentro de un intervalo $[\lambda_1, \lambda_2]$ que se corresponda con el rango de valores de la función empleada se empleará la Ecuación (17). Dicha expresión, para el caso de la *función logística*, se reduce a la expresión de la Ecuación (18).

$$x_i = \lambda_1 + (\lambda_2 - \lambda_1)\frac{z_i - Z_{mín}}{Z_{máx} - Z_{mín}} \quad (17)$$

$$X\epsilon[0,1] \;\Rightarrow\; x_i = \frac{z_i - Z_{mín}}{Z_{máx} - Z_{mín}} \quad (18)$$

4.3.5 Generalización

Durante el entrenamiento, las salidas de una Red Neuronal supervisada se aproximan a los valores deseados dadas una serie de entradas de conjunto de entrenamiento. Cuando se

habla de **generalización**, lo que se busca es obtener salidas de la Red Neuronal que se aproximan a los valores deseados cuando las entradas no forman parte del conjunto de entrenamiento. Típicamente existen tres condiciones necesarias para obtener una buena generalización [18]:

1. Las entradas de la red tienen que tener la suficiente información en relación con la salida deseada, de tal forma que pueda existir una función matemática que relacione correctamente entradas y salidas con un mínimo grado de precisión.
2. La función que relaciona las entradas con las salidas tiene que ser, en cierto modo, *suave*. Esto es, tiene que ser continua y tener ciertas restricciones en la primera derivada sobre la mayoría del espacio de entrada.
3. El conjunto de datos de entrenamiento tiene que ser lo suficientemente grande, así como ser una muestra lo suficiente representativa del conjunto de datos objetivo.

El conjunto de dichas condiciones se cumplirá para este caso particular. Como se ha visto en el apartado 4.3.2, las entradas y salidas elegidas no se han elegido al azar, sino que guardan una estrecha relación en términos de la física que define el problema. Así mismo, el conjunto de datos se presume suficientemente grande, quizás incluso excesivo. Es probable que sea necesaria la búsqueda de mecanismos de selección del conjunto de entrenamiento de entre los datos de muestra, asegurando que dicho conjunto sea una muestra suficientemente representativa.

Sin embargo, conseguir un sistema con una buena generalización no depende únicamente de las características de los datos disponibles, sino también del tipo de entrenamiento que se lleve a cabo. En concreto, es necesario evitar el llamado **sobreajuste (*overfitting*)** sobre el conjunto de datos de entrenamiento, lo que llevaría a un modelo ajustado en exceso a dicho conjunto y por tanto perdería capacidad de generalización (capacidad para estimar cuando los datos de entrada no son conocidos puesto que no formaban parte del conjunto con el que se ha entrenado). Existen **varias técnicas o mecanismos** que pretenden evitar dicho sobreajuste a la hora de entrenar la red.

Se puede por ejemplo **añadir ruido** a las muestras con las que se va a entrenar (***jitter***), lo cual en este caso puede que no interese ya que se presume que el número de datos disponibles será lo suficientemente grande y representativo (aunque tampoco se descarta su uso).

Otra opción podría ser emplear el mecanismo llamado ***early-stopping***, que consiste en lo siguiente: se dividen los datos disponibles en dos conjuntos denominados de **entrenamiento (*train*)** y de **validación (*validation*)**, típicamente se selecciona un número grande de neuronas ocultas y un valor pequeño del valor inicial aleatorio de los pesos así como una tasa de aprendizaje lenta. En base a dichas premisas, se inicia el entrenamiento, calculando periódicamente el error de validación (esto es, el error en el conjunto de validación). Se detendrá el error cuando dicho error comience a ascender (señal de que se está perdiendo generalización a través de un sobreajuste). Dicho comportamiento se puede observar en la Figura 13 - *Early-stopping*. Con este método además se podría conseguir un tiempo de ejecución mucho menor (aunque para esto habría que tener cuidado, ya que se podría parar la ejecución en un mínimo local de la función de error de validación y no en el mínimo global).

Esta última técnica podría resultar ventajosa para el problema que se pretende abordar, ya que la ejecución del entrenamiento se realizará en un dispositivo móvil y por tanto el tiempo de ejecución se podría dilatar más allá de lo que el usuario estaría dispuesto a esperar. No obstante, se introduce aquí como elemento de diseño a falta de una validación del mecanismo en las pruebas consecuentes, que determinarán si se incluirá o no finalmente en la implementación final.

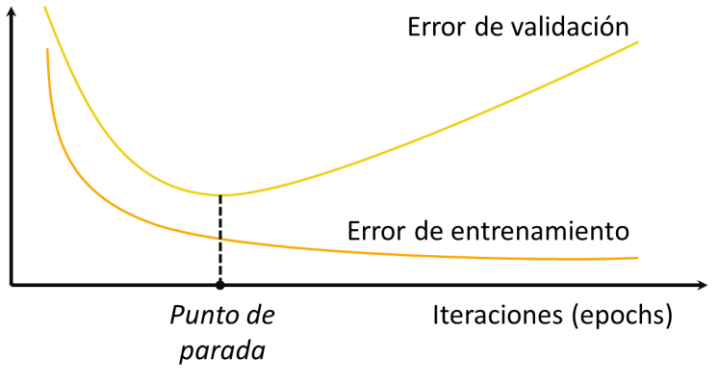


Figura 13 - *Early-stopping*

Otra de las opciones disponibles para garantizar en cierta medida la generalización son los métodos de **regularización** (***regularization***), esto es, intentar minimizar una función objetivo que es la suma de la función de error total y una función de regularización. La función de regularización es una función de los pesos o de la función de salida. Por ejemplo, en el **decaimiento de pesos** (***weight decay***), la función de regularización es la suma del cuadrado de los pesos, penalizando así los pesos con valores elevados.

Si no se usa ninguna función de regularización, la función objetivo es igual a la función de error total o media (la función que se pretende minimizar a través de sucesivas iteraciones con el algoritmo de descenso por gradiente).

En principio no se integrará ningún método de este tipo en la implementación, no obstante se introduce en el diseño por su interés práctico, y a partir de los resultados que se obtengan se determinará o no si optar por incluir este tipo de técnicas.

4.3.6 Algoritmo de aprendizaje

Como se ha comentado anteriormente en este documento, el algoritmo empleado para el entrenamiento de la Red Neuronal es el denominado *Backpropagation*. La intención de dicho algoritmo es modificar los pesos de la Red Neuronal a través de la minimización de un coste establecido, esto es:

- Si el valor de salida es correcto con respecto a los datos de entrenamiento, no se hace nada.
- Si el valor de salida es muy alto, se deberá disminuir el valor de los pesos asociados a dicha salida.
- Si el valor de salida es muy bajo, se deberá aumentar el valor de los pesos asociados a dicha salida.

4.3.7 Comprobación de resultados

Una vez se ha entrenado la Red Neuronal y con el objetivo de cuantificar la precisión con la que ésta ha modelado el comportamiento de los datos de entrenamiento, es necesario aplicar algún tipo de método de verificación de sus propiedades.

En este caso se plantea el uso de dos métodos diferentes que proporcionarán mayor certidumbre al modelo generado por el sistema. El primero de ellos consiste en utilizar parte de los datos recolectados y etiquetados como **conjunto de *test***, el cual servirá para obtener una medida del error estimado que tiene las salidas del modelo (error cuadrático medio). Típicamente se emplea este método junto con el comentado anteriormente *early-stopping*, de tal forma que se divide el conjunto de datos disponibles en entrenamiento (70%), validación (20%) y test (10%).

El segundo de dichos métodos consiste en calcular para cada salida el **coeficiente de correlación lineal** r (coeficiente de correlación de *Pearson*) que existe entre los valores reales y los obtenidos por la Red Neuronal. Dicho coeficiente permite obtener el grado de similitud y de precisión de la respuesta. Una correlación significativa se situaría en valores entre 0.7 y 1. El cálculo de dicho coeficiente se realiza según la ecuación

$$\mathrm{r} = \frac{\frac{\sum_i (x_i - \bar{x})\left(d_i - \bar{d}\right)}{N}}{\sqrt{\frac{\sum_i (x_i - \bar{x})^2}{N}}\sqrt{\frac{\sum_i (d_i - \bar{d})^2}{N}}} \qquad (19)$$

Donde el numerador se corresponde con la varianza y el denominador con el producto de las desviaciones típicas de las variables x (valor obtenido) y d (valor deseado).

4.4 Diseño del Software

Una vez establecidas las bases de funcionamiento del sistema, se hace necesario acotar las incertidumbres sobre la implementación a través de un esbozo básico del comportamiento de la aplicación. Se detallarán a continuación algunos elementos de diseño referentes al software de la aplicación.

4.4.1 Patrón de Diseño: MVC

A la hora de abordar un proyecto de desarrollo de software es buena práctica adoptar algún tipo de patrón de diseño que facilite tanto la planificación como la implementación del mismo (además de su mantenimiento). Dichos patrones son soluciones para problemas típicos y recurrentes [20].

Existen numerosos patrones en función de su propósito:

- **Patrones creacionales**: utilizados para instanciar objetos, y así separar la implementación del cliente de la de los objetos que se utilizan. Con ellos se intenta **separar la lógica de creación** de objetos y encapsularla.
- **Patrones de comportamiento**: se utilizan a la hora de definir cómo las clases y objetos interaccionan entre ellos.

- **Patrones estructurales**: utilizados para crear clases u objetos que incluidos dentro de estructuras más complejas.

En el sistema desarrollado se adoptará el patrón creacional denominado **MVC (*Model-View-Contoller*, Modelo-Vista-Controlador)**, el cual plantea la separación del problema en tres capas: la capa **Modelo**, que representa el tratamiento de datos; la capa **Controlador**, que conoce los métodos y atributos del modelo, recibe y realiza lo que el usuario quiere hacer (la lógica de la aplicación); y la capa **Vista**, que muestra un aspecto del modelo y es utilizada por la capa anterior para interaccionar con el usuario, es decir: la interfaz gráfica.

Este patrón de diseño permite una mayor portabilidad de la aplicación, así como facilitar tanto el desarrollo como el mantenimiento. Esto es debido a la separación que se realiza, de tal forma que aunque se cambie el sistema de almacenamiento de datos, únicamente habría que modificar la capa Modelo (o al menos idealmente). Así mismo, si se quisiera realizar un cambio de plataforma únicamente sería necesario modificar la capa Controlador. Y por último, si se quisiera realizar un cambio de apariencia bastaría con modificar la capa Vista. En la Figura 14 - MVC: Modelo Vista Controlador se puede ver un esquema básico de este patrón.

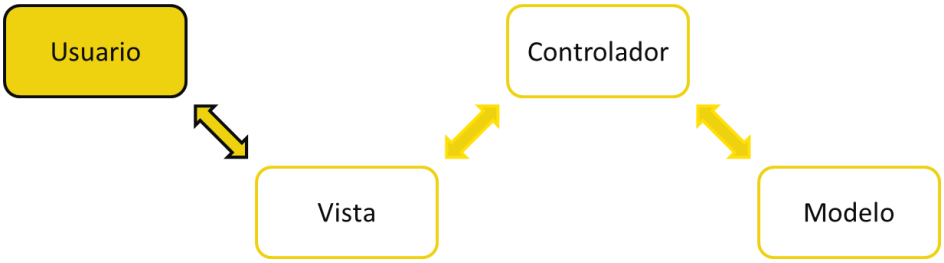


Figura 14 - MVC: Modelo Vista Controlador

En el esquemático se puede observar la interacción entre las diferentes capas, actuando siempre entre la interfaz y la base de datos la entidad Controlador.

En base a este planteamiento es como se desarrolla el diseño del sistema. En este estado del proyecto aún no está definida la plataforma en la que se va a desarrollar, así que se realizará el diseño de la Vista, el Controlador y el Modelo únicamente de forma teórica: atendiendo al diseño del mismo (el qué) y no al desarrollo (el cómo). En lo que respecta a la Vista, es decir a la interfaz gráfica, dependerá en gran medida de la tecnología usada y es por ello que se tendrá en consideración en fases más avanzadas del proyecto (en el desarrollo de la implementación).

4.4.2 Modelo de datos

Como se ha especificado anteriormente en este documento, el usuario podrá acceder a un histórico de las rutas realizadas, consultando parámetros integrados como la distancia total, velocidad media, etc. Además, la disponibilidad de dichos datos es crítica puesto que es imprescindible para poder realizar el entrenamiento de la Red Neuronal.

En base a estas premisas, se hace necesario diseñar un modelo de datos que permita almacenar toda la información necesaria. A continuación se realizará un diseño de la Base de Datos (BBDD) necesario para el almacenamiento de los datos en el sistema.

Para el diseño conceptual de la BBDD se empleará el Modelo Entidad Relación (Modelo E/R). En la Figura 15 - Modelo Base de Datos se muestra dicho esquemático.

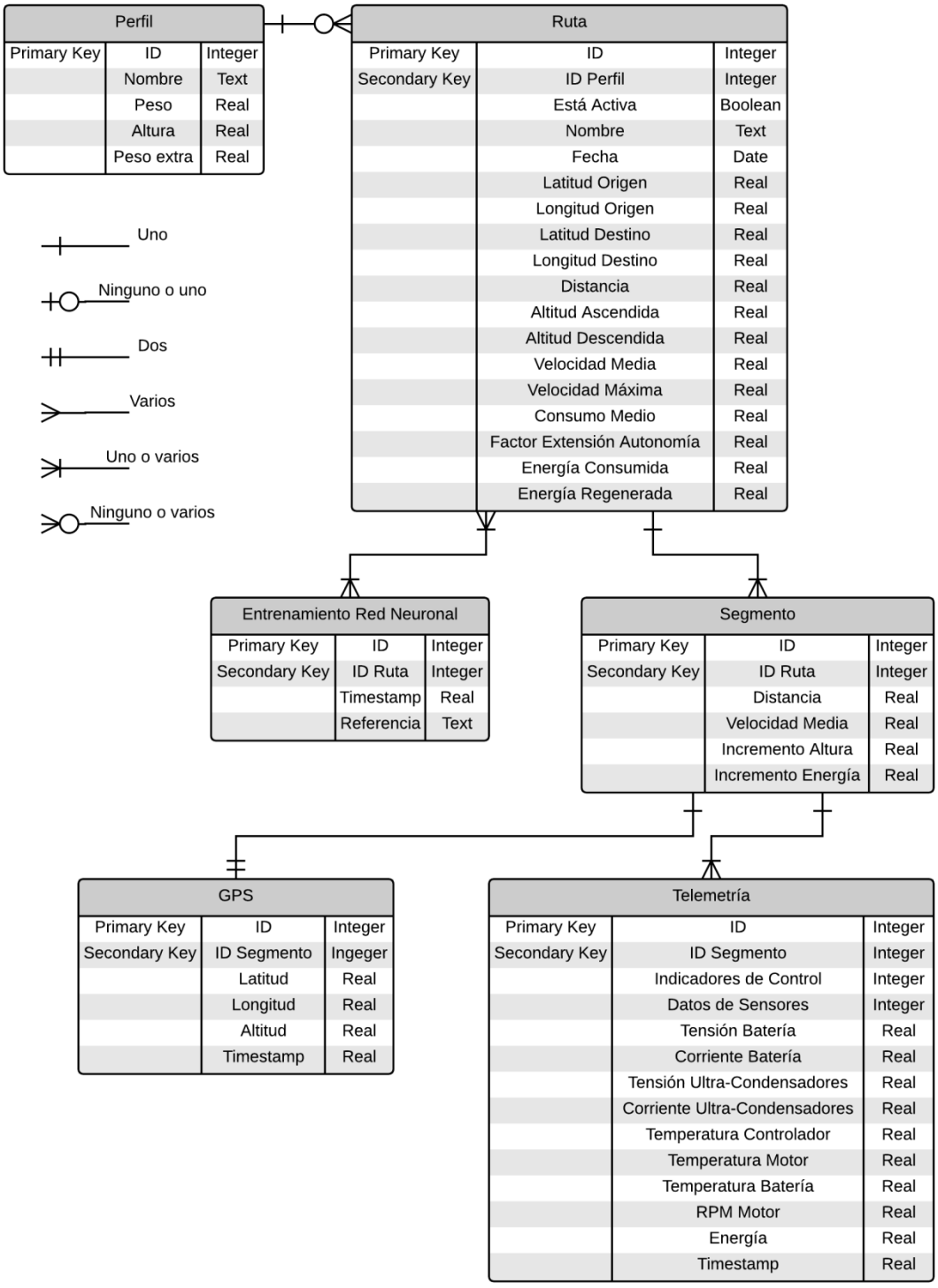


Figura 15 - Modelo Base de Datos

En dicho esquemático se pueden ver 6 entidades diferentes. Dichas entidades estarán asociadas con sus correspondientes tablas en el modelo relacional, por lo que se ha incluido el tipo de clave (primaria o secundaria). Además, se ha especificado el tipo de

datos necesario para su almacenamiento, que dependiendo de la implementación podrá variar.

4.4.2.1 Perfil

El perfil corresponde al usuario de la motocicleta. En la versión inicial se ha considerado que el perfil será único por cada dispositivo móvil, no obstante el diseño de la base de datos se ha considerado para que no exista dicha restricción y así en un futuro no tener problemas de compatibilidad (o al menos minimizar el número de éstos).

En dicho planteamiento multi-usuario se pueden plantear diferentes escenarios para el entrenamiento de la Red Neuronal. Por un lado podría hacerse una diferenciación de las rutas que hacen los diferentes usuarios y crear un modelo de predicción de consumo para cada uno de ellos. Otro enfoque consistiría en emplear la diferencia peso-altura para crear un modelo que discriminase en función de ello, no obstante esto eliminaría la adaptación al estilo de conducción de cada piloto, creando un sistema menos preciso.

En el modelo desarrollado para esta versión se tiene en cuenta tanto el peso del piloto como el peso extra (maletas o acompañantes) puesto que éstos influyen directamente en el consumo. No obstante, para que la Red Neuronal sea capaz de discriminar en base a estos parámetros deberán existir en el histórico de rutas variaciones de ambos, puesto que si es siempre el mismo la Red Neuronal obviará dicho valor.

4.4.2.2 Ruta

Como se ha comentado anteriormente, será necesario mantener un histórico de rutas tanto para la consulta por parte del usuario como para el entrenamiento de la Red Neuronal. Es el primero de dichos propósitos el que hace necesaria la inclusión de ciertas magnitudes integradas de la ruta, así como otros campos destinados a la identificación de ésta.

Esta doble visión es la que propicia la creación del campo *Está Activa*, el cual permite especificar cuándo una ruta puede ser o no visible para el usuario pero sin embargo conservar los datos de telemetría y GPS de cara a entrenar la Red Neuronal. Es decir, existirá la posibilidad de eliminar la ruta del histórico de ruta, pero sin embargo conservar toda la información y utilizarla así para la creación del modelo de predicción.

Se puede observar que cada ruta estará compuesta por varios tramos o **segmentos**, cada uno de los cuales estará definido típicamente en base a dos **posiciones GPS** (no obstante se mantiene la opción de poder agrupar más de dos). Cada uno de dichos segmentos tiene asociado ciertas magnitudes como la distancia, la velocidad media, el incremento de altitud o el incremento de energía, todas ellas en consecuencia con el diseño de la Red Neuronal visto en apartados anteriores.

Cada segmento tendrá asociado varios **datos de telemetría** (aproximadamente 10). A la hora de agrupar los datos de telemetría dentro de un segmento, se comprobará que su *timestamp* esté dentro del rango que definen los *timestamp* de las posiciones de inicio y fin de dicho tramo.

4.4.2.3 Entrenamiento Red Neuronal

Con el objetivo de poder seleccionar de entre las rutas disponibles para realizar el entrenamiento de la Red Neuronal, se ha añadido una entidad adicional, de tal forma que permita identificar qué modelo o modelos se han generado con cada ruta (en caso de que se permita crear más de un modelo, o para propósitos de análisis de rendimiento de varios de éstos).

4.4.3 Controlador: lógica del sistema

Dentro de toda la lógica del sistema, existen algunos procedimientos o rutinas con más importancia que otros (desde el punto de vista del diseño) por lo que serán únicamente los casos más significativos los analizados en esta sección.

4.4.3.1 Conexión y petición de datos

Como se ha especificado anteriormente en este documento, el sistema deberá de proveer conexión con la motocicleta. Será a través de dicha conexión como se recibirán datos de telemetría y GPS. Dicho procedimiento está ilustrado en la Figura 16 - Diagrama de flujo 1 y en la Figura 17 - Diagrama de flujo 2.

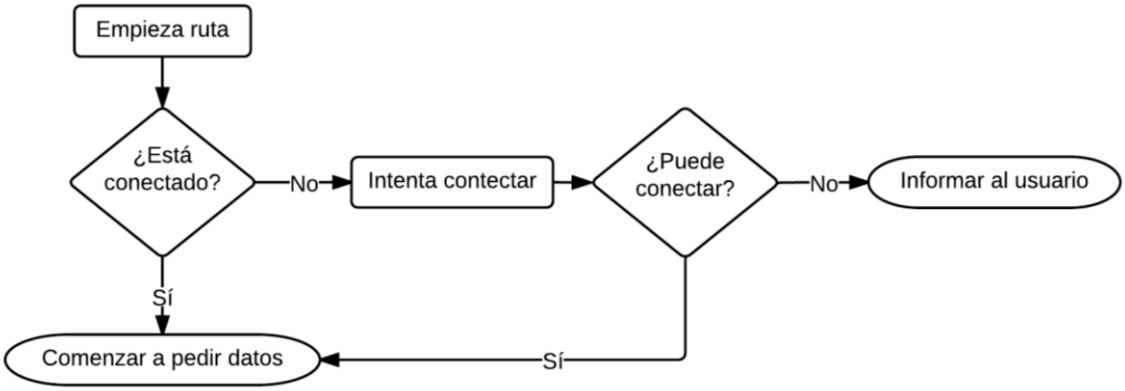


Figura 16 - Diagrama de flujo 1

Como se aprecia en la figura, se ha diseñado de tal forma que antes de comenzar cualquier ruta el *Smartphone* se encuentre conectado a la motocicleta. Esto es necesario puesto que será a partir de dicha conexión como el dispositivo móvil obtenga información de posicionamiento y telemetría, necesarios para poder grabar la ruta y así añadirla al histórico (que como se ha comentado, servirá tanto para propósitos de consulta por parte del usuario como para entrenar el sistema de inteligencia artificial).

En la segunda de las figuras citadas se puede ver más en detalle la rutina *Comenzar a pedir datos*. Analizando un poco más el funcionamiento diseñado, cabe destacar el procedimiento o rutina *Actualizar magnitudes a mostrar*; cuyo objetivo es desencadenar el proceso a través del cual la aplicación será capaz de leer las magnitudes de telemetría en tiempo real y poder así mostrarlas en la pantalla del dispositivo. Es decir, dado el funcionamiento planteado, dicha tarea será simplemente una lectura de datos, ya que es la rutina ilustrada en la figura la encargada de actualizar esos datos con información en tiempo real.

El proceso que se acaba de describir estará activo hasta que el usuario desee detener la ruta, momento en el cual se desencadenará el proceso descrito en la Figura 18 - Diagrama de flujo 3, al final del cual la ruta quedará guardada en memoria con todos los datos descritos en el apartado anterior incluidos.

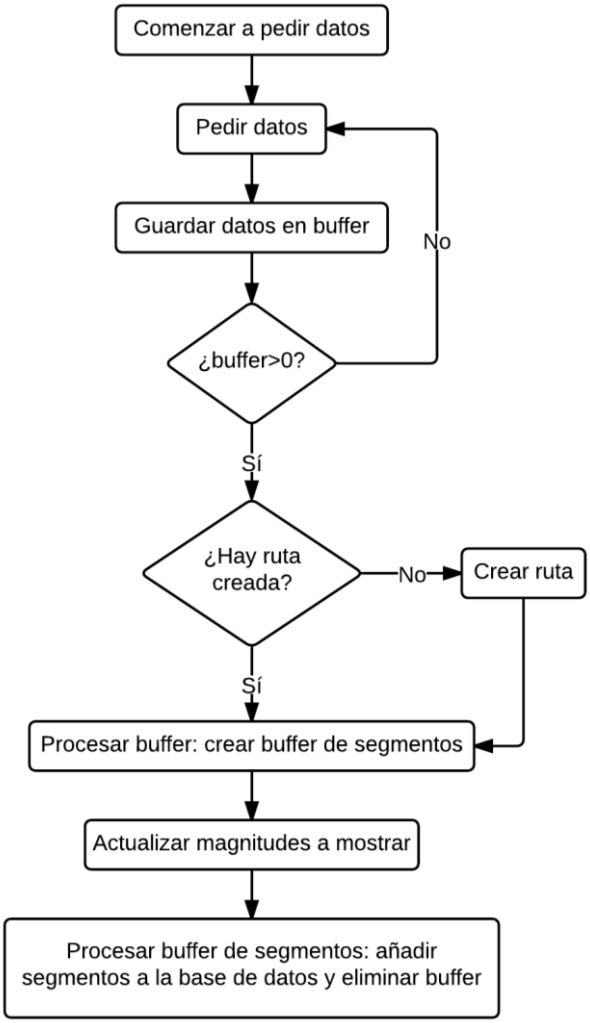


Figura 17 - Diagrama de flujo 2

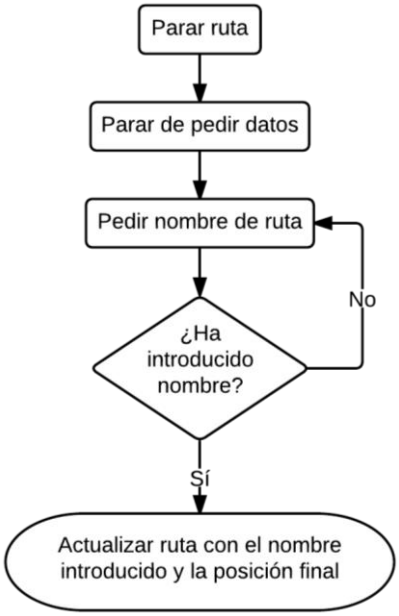


Figura 18 - Diagrama de flujo 3

4.4.3.2 Red Neuronal

En cuanto al funcionamiento de la Red Neuronal con el resto del sistema, se describirán a continuación una serie de rutinas que previsiblemente será necesario implementar en el sistema final y que determinarán el tratamiento de las estimaciones en el conjunto de la aplicación.

Se puede ver en la Figura 19 - Diagrama de flujo 4 y en la Figura 20 - Diagrama de flujo 5 de qué forma está previsto "cargar" el modelo de predicción en el sistema y en qué puntos de ejecución se comprobará si dicho modelo ha sido creado (se entiende *ANN=null* como la comprobación que determina si existe algún modelo de predicción, es decir si la Red Neuronal se ha entrenado).

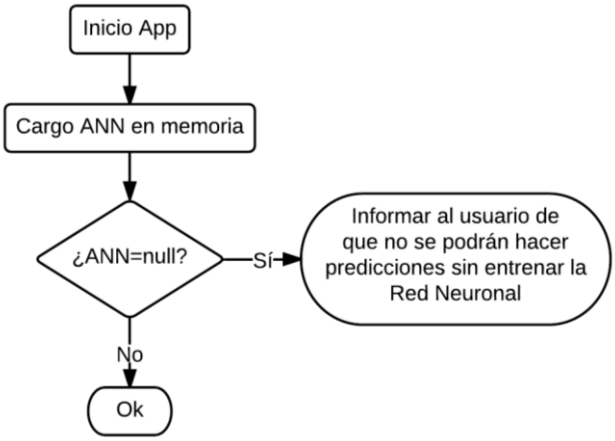


Figura 19 - Diagrama de flujo 4

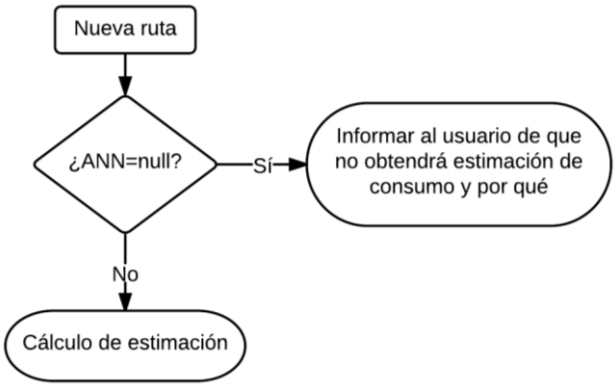


Figura 20 - Diagrama de flujo 5

En este punto es cuando se hace necesario definir en qué momento se procederá al entrenamiento de la Red Neuronal para así poder disponer de un modelo de predicción que permita realizar estimaciones de viabilidad de rutas. Dicho proceso se puede ver en la Figura 21 - Diagrama de flujo 6. En el momento en que se ejecuta la rutina *Establecer como ANN del sistema*, la decisión *ANN=null*, dará como resultado un valor afirmativo. Posteriormente se guarda dicho modelo para que pueda ser cargado a memoria en futuras ejecuciones de la aplicación.

Se puede observar la iteración que se comentaba en el apartado 3.3.1.4 a través de la cual el sistema obtiene un modelo con un mínimo de garantías de funcionamiento (según los diferentes métodos de evaluación comentados en el apartado 4.3.7).

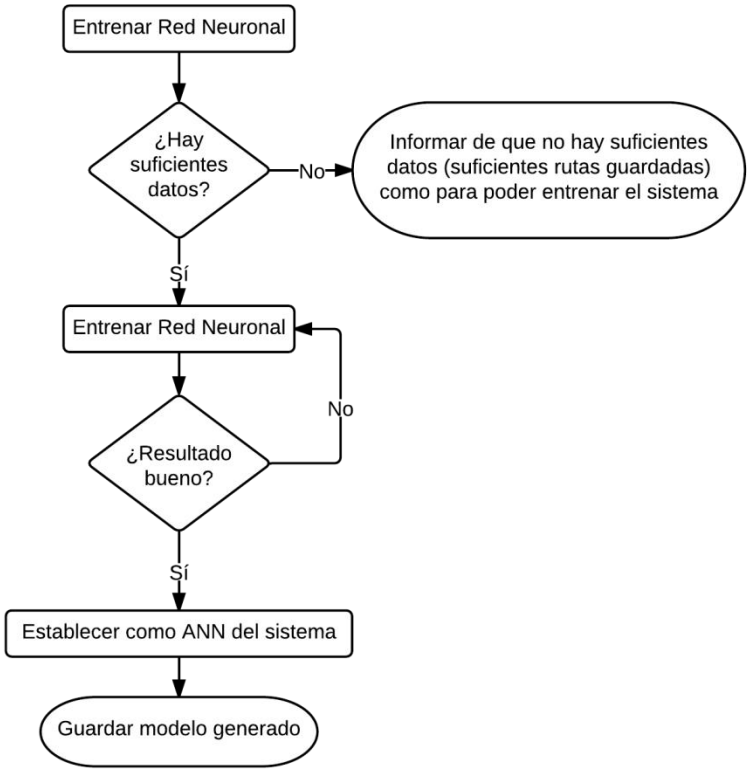


Figura 21 - Diagrama de flujo 6

En cuanto a la tarea de estimación (la rutina encargada de, dada una ruta, calcular su viabilidad), ésta se puede ver en el diagrama de flujo definido en la Figura 22 - Diagrama de flujo 7. En este caso, dado que es un proceso que interfiere con el funcionamiento gráfico, se ha incluido el procedimiento a través del cual se informa al usuario de que se está realizando la estimación (*Iniciar Spinner*, el cual mostrará un diálogo de información con una imagen en movimiento de tipo *spinner*), dado que dicha tarea tendrá que implementarse de manera que se ejecute de forma paralela al hilo principal de ejecución.

En dicho esquemático también se puede observar que la obtención de los datos de ruta se ha subdividido en dos tareas. Esto es debido a que previsiblemente los datos de altura no suelen estar disponibles en los datos cartográficos, pero esto puede variar en función de la implementación que se realice (principalmente en función de la tecnología usada).

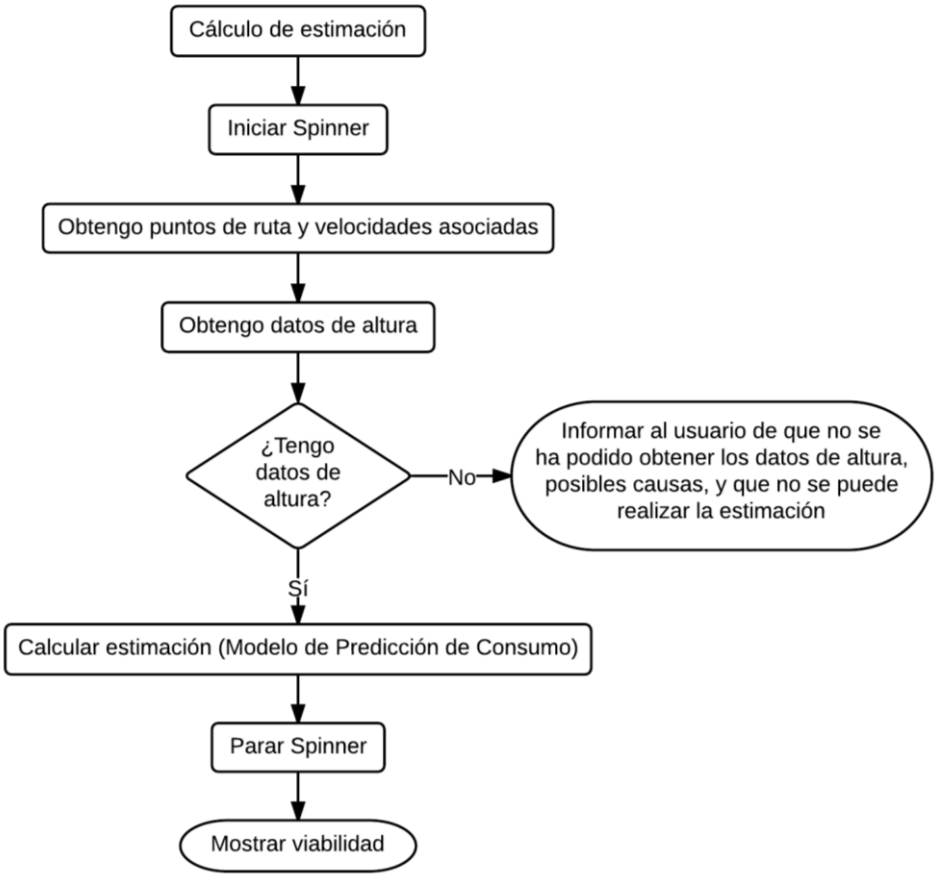


Figura 22 - Diagrama de flujo 7

4.4.4 Vista: Interfaz gráfica

En base a las especificaciones se ha realizado un diseño básico de la interfaz de aplicación. Como a nivel de funcionamiento ya se han comentado los diferentes aspectos con los que contará, será en este momento de desarrollo del proyecto donde se aporte más definición al aspecto que pueda tener la aplicación.

Como se ha introducido al inicio de este documento, existen una serie de especificaciones que son de obligado cumplimiento en cuanto a la interfaz se refiere. Además del sistema de estimación de viabilidad (Modelo de Predicción de Consumo), existen numerosas

características que aportará la aplicación. Entre ellas destacan la **navegación guiada** y la consulta de un **histórico de rutas**.

En la Figura 23 - Wireframe de la aplicación se puede ver un *wireframe* con las principales pantallas de las que dispondrá la aplicación. La dependencia de la plataforma restringe el alcance del diseño que se puede realizar, por lo que se tomarán como referencia de diseño de interfaz únicamente las 3 posibles pantallas mostradas.

4.4.4.1 Navegación guiada: datos de telemetría

El sistema de navegación guiada que se ha de implementar, además del cálculo de ruta y la estimación de viabilidad, debe mostrar una serie de magnitudes físicas asociadas al vehículo y la ruta. El usuario además podrá seleccionar cuál de ellas mostrar. Dichas magnitudes se definen a continuación:

- **Distancia recorrida** (km).
- **Autonomía en distancia**: número de kilómetros que se podrán recorrer con la energía disponible.
- **Energía consumida** (kJ).
- **Energía regenerada** (kJ).
- **Consumo neto medio**: media del consumo del vehículo (kJ/km).
- **Factor de extensión de autonomía**: ratio entre la energía regenerada y la consumida (%).
- **Temperatura del motor** (ºC).
- **Temperatura de la electrónica de tracción** (ºC).
- **Temperatura de batería** (ºC).
- **Velocidad media** (km/h).
- **Velocidad máxima** (km/h).
- **Excedente de batería**: porcentaje de la energía de la batería que presumiblemente sobrará al finalizar la ruta planificada.

Además de dichos parámetros deberá mostrar los característicos de este tipo de sistemas de guiado como la **distancia restante** para alcanzar el destino o la **velocidad instantánea**.

El conjunto de las magnitudes antes numeradas estará disponible para visualización en una pantalla dedicada si el usuario así lo desea (porque ya conoce la trayectoria a seguir, por ejemplo). En dicha pantalla se mostrará de igual manera el indicador con el nombre de la magnitud y su valor correspondiente además de un indicador gráfico de tipo barra de progreso.

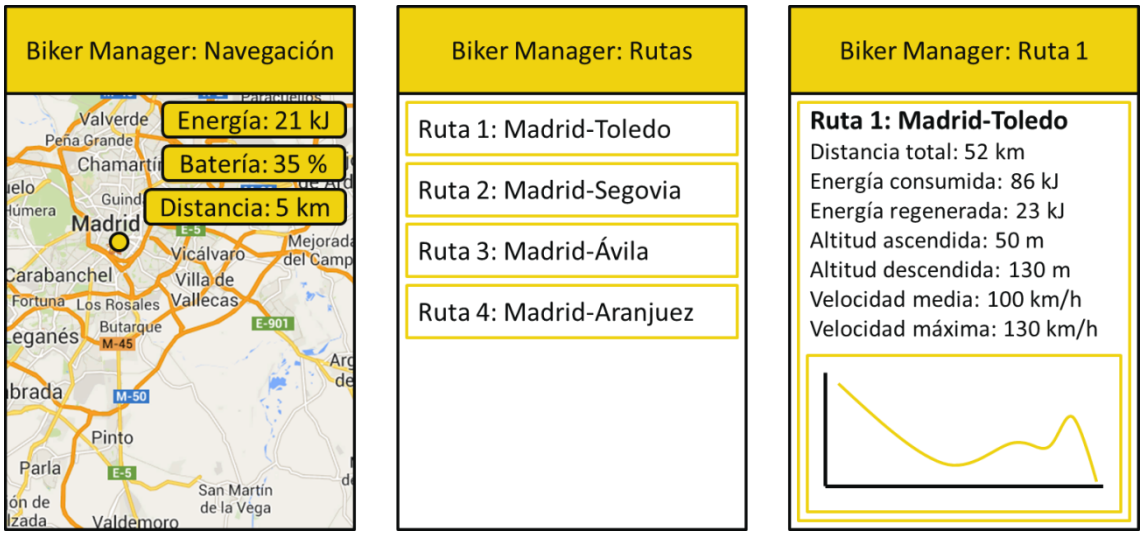


Figura 23 - Wireframe de la aplicación

4.4.4.2 Histórico de rutas

El usuario tiene la opción de consultar las rutas que ha realizado, las cuales tendrán gran cantidad de información acerca de la misma. Además de una lista en la que podrá consultar todas y cada una de las rutas que ha realizado, podrá seleccionar cualquiera de ellas y consultar así las magnitudes que se citan a continuación:

- **Fecha y hora de inicio**.
- **Duración total de la ruta**: formato hh:mm:ss (horas, minutos y segundos).
- **Distancia total de la ruta** (km).
- **Altura total ascendida** (m).
- **Altura total descendida** (m).
- **Energía total consumida** (kJ).
- **Energía total regenerada** (kJ).
- **Gráfica Velocidad** (km/h) – **Distancia** (km).
- **Gráfica Altitud** (m) – **Distancia** (km).

Además de las citadas magnitudes, se ha establecido que se guardarán todos los datos correspondientes a la telemetría, por lo que existe la posibilidad de añadir más datos en esta sección. El conjunto de datos de telemetría disponibles se puede consultar en el Anexo 9.1.

4.5 Conclusión

Se ha visto cuál será el enfoque del sistema de estimación, detallando el diseño inicial del mismo, además de su integración con el sistema completo. El conjunto de la aplicación ofrecerá por tanto un sistema de navegación con cálculo de rutas y estimación de viabilidad de éstas basado en las magnitudes leídas de la motocicleta y un Modelo de Predicción de Consumo en base a un sistema de estimación inteligente basado en Redes Neuronales Artificiales.

El diseño del sistema deseado comprende gran cantidad de detalles, muchos de los cuales se han especificado en esta sección del documento. No obstante, existe también un gran

número de características que dada su dependencia de la implementación no será hasta la siguiente sección, el apartado 5, donde se detalle su estructura.

5 Desarrollo

5.1 Introducción

En todo proyecto es necesaria cierta planificación y es sumamente recomendable adoptar algún tipo de metodología de gestión del proyecto para mantener la consistencia del desarrollo y no perder el enfoque en el trabajo a realizar. En esta sección se dará una pequeña visión de las posibilidades en este aspecto, profundizando en la metodología utilizada y en cómo se ha llevado a cabo de forma práctica.

Así mismo, se hará un análisis tanto de las elecciones relativas al desarrollo como un análisis y posterior elección de una plataforma de desarrollo que se adapte a los requerimientos del proyecto.

Posteriormente se comentará con mayor detalle el proceso de implementación llevado a cabo en base al diseño realizado.

5.2 Metodología de desarrollo

En este contexto se puede definir *Metodología de desarrollo* como el **conjunto de procedimientos, técnicas, herramientas y soporte documental que deben seguirse para el desarrollo de un proyecto**.

Existen numerosos enfoques a la hora de afrontar el desarrollo de cualquier tipo de proyecto. Algunas metodologías típicas son el desarrollo en cascada (*waterfall*), el modelo de prototipos (*prototyping*), el desarrollo iterativo incremental (*iterative and incremental development*), el desarrollo en espiral (*spiral development*), el desarrollo rápido de aplicaciones (*rapid application development*) y el desarrollo ágil de software (*agile software development*) [21].

En esta sección se realizará un pequeño análisis de las características de las principales o más conocidas metodologías, centrando el análisis en la opción elegida para este proyecto: la metodología **Kanban**.

5.2.1 Análisis de alternativas

Como se ha introducido en esta sección, existen numerosas alternativas disponibles a la hora de elegir una metodología de desarrollo. En concreto, en lo que respecta al desarrollo de software, el número es ciertamente más elevado que en el desarrollo de proyectos típicos de ingeniería.

En la tarea de elegir qué tipo de metodología se adaptará mejor al desarrollo de este proyecto, será necesario realizar un análisis que permita comparar y así elegir con ciertas garantías qué tipo de metodología usar.

Las metodologías clásicas como el desarrollo en cascada o espiral requieren que todos o parte de los requisitos sean estables, algo típico en proyectos de gran envergadura y duración. Al ser muy estructurados suelen ser también poco flexibles. Además de ello,

suelen suponer una gran carga administrativa y de documentación, centrando el esfuerzo en tareas no asociadas al propio desarrollo.

En contraposición a dichas metodologías tradicionales, donde todo tiene que estar estrictamente planificado, existen otro tipo de tecnologías denominadas **metodologías ágiles**. Estas metodologías se basan principalmente en dos aspectos:

- Retraso de las decisiones.
- Planificación adaptativa.

El **retraso de las decisiones** reduce el número de decisiones de alta inversión que se toman. Reduce también el número de cambios en el proyecto, y el coste de éstos en caso de ser necesarios.

La **planificación adaptativa** consiste en la toma de decisiones a lo largo del proyecto, desarrollando así una planificación a corto plazo que permite tener pequeñas evoluciones del producto para mostrar al cliente.

En el cuadro comparativo de la Tabla 1 - Metodologías Tradicionales vs Ágiles se pueden ver las principales diferencias entre las metodologías ágiles y las metodologías tradicionales [22].

Metodologías Tradicionales	Metodologías Ágiles
Rigidez ante los cambios	Flexibilidad ante los cambios
Los clientes interactúan con el equipo de desarrollo mediante reuniones	Los clientes forman parte del equipo de desarrollo
Grupos de gran tamaño y típicamente distribuidos en lugares diferentes	Grupos pequeños (sobre 10 personas) en el mismo lugar
Dependencia de la arquitectura de software mediante modelos	Menor dependencia de la arquitectura de software
Poco *feedback*, por lo que se extiende el tiempo de entrega	Continuo *Feedback*, acortando así el tiempo de entrega
Mínimos roles	Diversidad de roles
Basadas en normas de estándares de desarrollo	Basadas en heurísticas a partir de prácticas de producción de código
Procesos muy controlados por políticas y normas	Procesos menos controlados, pocas políticas y normas
Seguimiento estricto del plan inicial de desarrollo	Capacidad de respuesta ante los cambios

Tabla 1 - Metodologías Tradicionales vs Ágiles

A raíz del análisis de las diferentes características mostradas, se tomará como referente algún tipo de metodología ágil, por ser más adecuada a las características de este proyecto.

Dentro de las metodologías ágiles de desarrollo de software se pueden encontrar varias alternativas [23]: *Adaptive Software Development (ASD)*, *Agile Modeling*, *Agile Unified Process (AUP)*, *Crystal Methods (Crystal Clear)*, *Disciplined Agile Delivery*, *Dynamic Systems Development Method (DSDM)*, *Extreme Programming (XP)*, *Feature Driven Development (FDD)*, *Lean software development*, *Kanban (development)*, *Scrum* y *Scrum-ban*.

La bibliografía disponible con respecto a dichas metodologías es muy amplia y no se realizará un análisis de éstas, queda a disposición del lector ampliar dicha información mediante otros medios.

Para la realización de este proyecto se ha elegido quizás una de las menos comunes: la metodología **Kanban**. Dicha elección se justifica en base a las siguientes razones:

- Familiarización con dicha metodología por experiencia en otros proyectos, lo que proporciona una enorme ventaja en su utilización.
- La disponibilidad de herramientas conocidas que permiten adaptarse al desarrollo del proyecto y a su flujo de trabajo.

Es por ello que, dada la experiencia anterior en trabajar con esta metodología, será la elegida para el desarrollo del proyecto, permitiendo así un inicio rápido del proyecto (al hilo de la filosofía que sustentan este tipo de metodologías).

5.2.2 Kanban

5.2.2.1 ¿Qué es Kanban?

El Kanban es un sistema de planificación *just-in-time* desarrollado por Toyota para gestionar el flujo de trabajo en las fábricas de coches. Kanban se basan en el desarrollo incremental, dividiendo el trabajo en partes. Una de las principales aportaciones es que utiliza técnicas visuales para ver la situación de cada tarea [24].

Las principales reglas de Kanban son las tres siguientes:

(1) Visualizar el trabajo y las fases del ciclo de producción o flujo de trabajo.
(2) Determinar el límite de "trabajo en curso" (o *WIP – Work In Progress*).
(3) Medir el tiempo en completar una tarea (lo que se conoce como *lead time*).

En cuanto a la **visualización**, tradicionalmente un Kanban consiste en una gran tabla, donde cada columna representa un estado en el proceso de fabricación. Cada columna contiene tarjetas que representan los elementos —coches en el caso de Toyota— en cada estado. La tabla o pizarra tiene tantas columnas como estados por los que puede pasar la tarea (ejemplo, en espera de ser desarrollada, en análisis, en diseño, etc.).

El **límite del WIP** propone limitar el número de tareas en curso y por tanto el número máximo de tareas que se pueden realizar en cada fase debe ser algo conocido. El objetivo de este límite es centrar el esfuerzo en las tareas planificadas de un determinado proceso y no empezar nuevas hasta finalizar éstas. De esta manera se avanza continuamente en el desarrollo sin dejar colgada ninguna tarea.

A la **medida del tiempo** que se tarda en terminar una tarea se le llama *lead time*. El *lead time* cuenta desde que se hace una petición hasta que se hace la entrega. Aunque la métrica más conocida del Kanban es el *lead time*, normalmente se suele utilizar también otra métrica importante: el *cycle time*. El *cycle time* mide desde que el trabajo sobre una tarea comienza hasta que termina. Con el *lead time* se mide lo que ven los clientes, lo que esperan, y con el *cycle time* se mide más el rendimiento del proceso.

5.2.2.2 Implementación de la metodología Kanban

Una vez elegida la metodología, el siguiente paso es adaptarla al flujo de trabajo que se va a seguir en el desarrollo del proyecto. En concreto, habrá que definir cómo se implementarán en la práctica las tres reglas expuestas en el apartado anterior.

En cuanto a la organización y visualización, se dividirá el tablón en cuatro columnas: Inbox, To Do, Doing, Done. En la Figura 24 - Tablón Kanban se puede ver la forma del tablón Kanban utilizado en este proyecto.

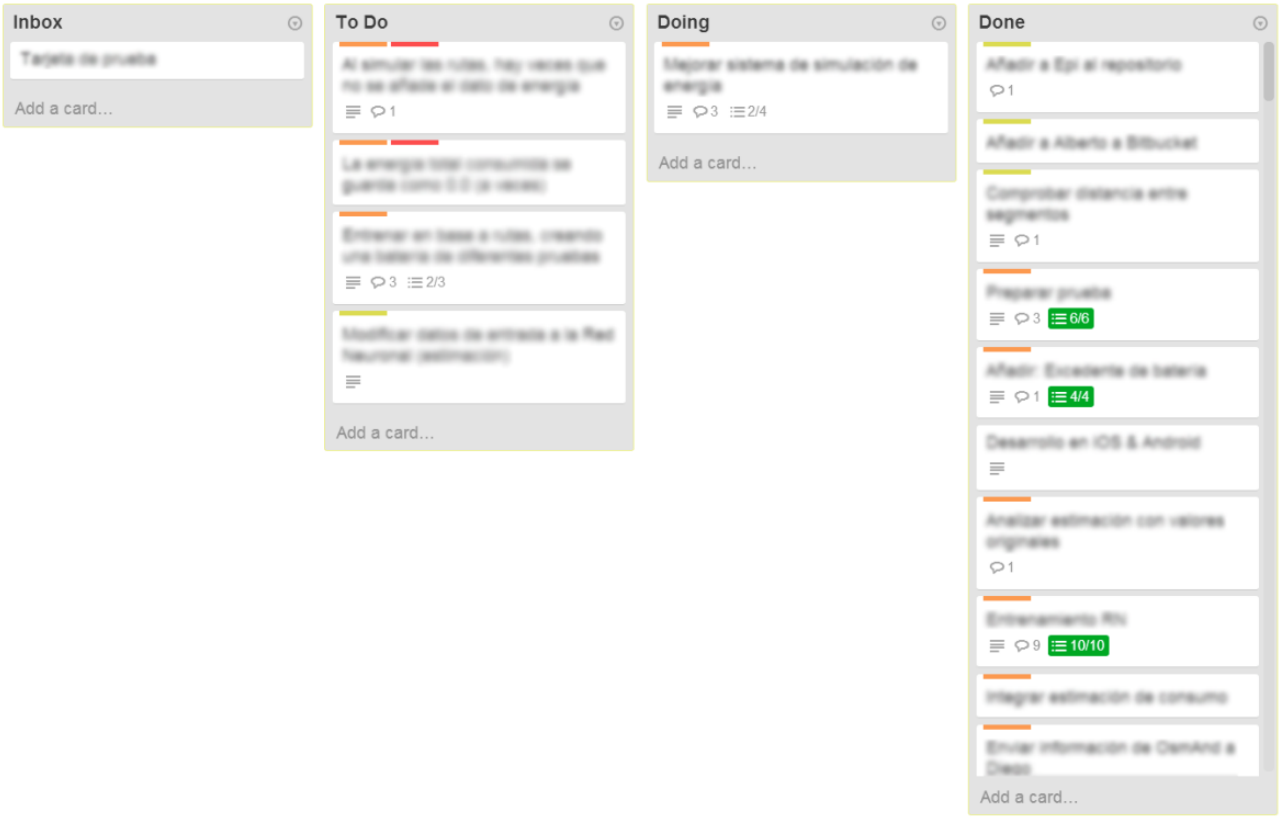


Figura 24 - Tablón Kanban

La columna **Inbox** servirá como medio de entrada al proyecto, ya sea como idea para implementar, posibles tareas a realizar o artículos o documentos que puedan ser útiles para el proyecto.

Una vez que se han evaluado las tareas presentes en Inbox y se ha decidido que han de ser realizadas, pasarían a la columna **To Do**. En dicha columna existirá un orden de prioridades según 3 colores: Naranja – Alta prioridad, Amarillo – Prioridad media, Verde – Baja prioridad. Además, existirá otro identificador con color rojo para identificar aquellas tareas que tengan que ver con fallos, *bugs* o errores que haya que subsanar.

En la columna **Doing** estarán presentes todas las tareas que se están ejecutando en ese momento, es decir lo que anteriormente se ha denominado *WIP*.

Al finalizar el proceso de ejecución de una tarea, ésta pasa a la columna **Done**. El objetivo de esta columna es disponer de una visión de conjunto sobre lo que hay que hacer, lo que se está haciendo y lo que ya se ha hecho.

Para la implementación de esta metodología se utiliza la herramienta web **Trello (https://trello.com/)**. Dicha herramienta no está diseñada para trabajar específicamente con Kanban pero se adapta muy bien a él, además de añadir diversas funcionalidades que ayudan a la gestión del proyecto como añadir listas de tareas, comentarios, descripciones, fechas límite, etiquetas, etc. a las tarjetas presentes en cada columna (las diferentes tareas).

5.3 Planificación y presupuesto

5.3.1 Planificación

Como ya se ha comentado en el apartado anterior, la planificación de este proyecto es bastante flexible debido a la metodología empleada. No obstante, todo proyecto debe estar acotado temporalmente y tener un objetivo final que cumplir. En base a esto se ha realizado una planificación en base a tareas básicas de la implementación.

El proyecto se realizará a lo largo de **9 meses** (~190 días), comenzando a principios de **Octubre de 2013** y finalizando al acabar **Junio de 2014**. En la Figura 25 - Diagrama de Gantt se puede ver un diagrama de Gantt básico que ilustra las principales fases de desarrollo del proyecto y su duración en el tiempo.

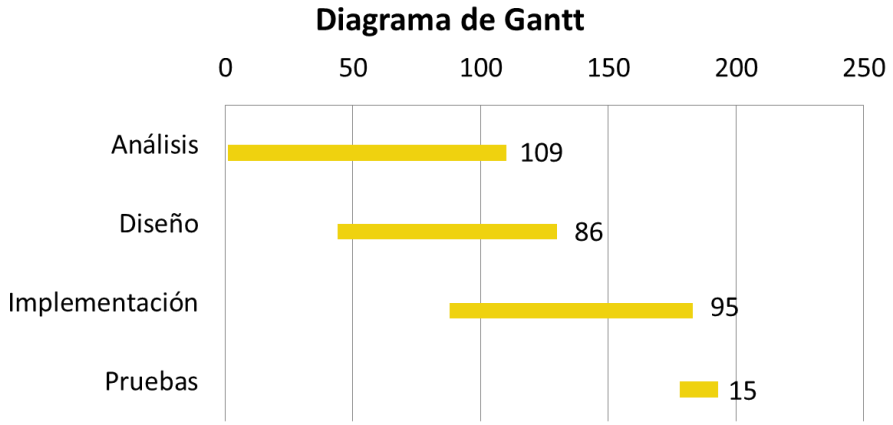


Figura 25 - Diagrama de Gantt

En dicho gráfico se puede observar el número total de días en el eje horizontal, mientras que al lado de cada barra se puede leer el número de días estimados para la realización de cada fase. En la Tabla 2 - Planificación del proyecto se puede observar con más detalle dicha planificación.

	Día comienzo	Día fin	Duración (días)
Análisis	1 Octubre 2013	28 Febrero 2014	109
Diseño	1 Diciembre 2013	31 Marzo 2014	86
Implementación	1 Febrero 2014	15 Junio 2014	95
Pruebas	10 Junio 2014	30 Junio 2014	15

Tabla 2 - Planificación del proyecto

5.3.2 Presupuesto

En cuanto al presupuesto del desarrollo, se ha estimado en base a la duración del proyecto, las personas implicadas en desarrollo del mismo así como el coste de material y licencias. En la Tabla 3 - Presupuesto se puede observar el presupuesto en cuestión.

Costes directos					
	Personal				
	Categoría	Dedicación		Coste hora	Coste total
	Ingeniero	760 horas		35 € / hora	26.600 €
	Equipos				
	Descripción	Coste (con IVA)	Dedicación	Periodo depreciación	Coste imputable
	PC desarrollo: Sony Vaio E (Intel Core i-5 4GB RAM)	968 €	9 meses	60 meses	120€
	Dispositivo desarrollo: LG Nexus 5	370 €	7 meses	24 meses	89,19 €
	Licencia MS Office	269 €	9 meses	60 meses	33,35 €
	Subcontratación de tareas				
	Descripción				Coste total
	Servidor local + Servicio Web *				4.300 €
				TOTAL:	**31.142,54 €**
Costes indirectos					
	Tasa CI				Coste total
	20%				6.228,51 €
				TOTAL:	**6.228,51 €**
			TOTAL:	37.682,47 €	

Tabla 3 - Presupuesto

* Las tareas subcontratadas según el presupuesto corresponden al diseño, implementación e implantación del servidor con la base de datos de alturas y el servicio web al que se accede a través del dispositivo móvil.

5.4 Gestión del código

En todo proyecto que involucre el desarrollo de software es más que recomendable utilizar algún tipo de **sistema de control de versiones**. Este tipo de herramientas registran los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo, de modo que se puedan recuperar versiones específicas más adelante.

En concreto es más que recomendable utilizar un sistema de control de versiones distribuido. Este sistema guarda los cambios realizados en un documento en una base de datos local, replicando cada uno de esos cambios en un servidor. De esta manera, se tiene una réplica idéntica del repositorio en el cliente (PC) y el servidor.

Uno de dichos sistemas de control de versiones distribuido es **Git**. Este sistema fue diseñado por Linus Torvals (quien a su vez diseñó y actualmente mantiene y desarrolla el kernel de Linux) y permite crear diferentes versiones de un mismo código fuente,

pudiendo revertir los cambios volviendo a versiones anteriores e incluso emplear varias líneas de desarrollo paralelas (para implementar características diferentes por parte de diferentes miembros de un equipo).

No obstante, este tipo de sistemas son extraordinariamente potentes cuando se aplica un flujo de trabajo optimizado. En este caso se utilizará **Gitflow**, sistema ideado por Vincent Driessen que define un estricto modelo de ramas diseñado alrededor de la publicación de un proyecto [25].

Aunando dicho sistema de control de versiones con un flujo de trabajo como el que propone Vincent se alcanza un nivel de productividad muy alto. Además, se han empleado dos herramientas de la empresa Altassian: **Bitbucket** (https://bitbucket.org/) que permite almacenar repositorios de código de forma gratuita en la nube, además de múltiples herramientas de administración a través de un *front-end* muy intuitivo (pudiendo crear equipos, añadir desarrolladores, etc.); y un cliente de escritorio llamado **SourceTree** (http://www.sourcetreeapp.com/) el cual está pensada no sólo para trabajar con Git sino también para aplicar Gitflow [26].

5.5 Elección de plataforma de desarrollo

En esta fase inicial de desarrollo del producto se ha optado por utilizar la plataforma **Android**. Entre las principales razones por las cuales se ha elegido dicha plataforma para comenzar el desarrollo destacan:

- Más del 85 % de la cuota de mercado de *Smartphones* en España llevan dicho Sistema Operativo (siendo la media Europea del 70 %) [27].
- Es la plataforma más accesible, tanto por herramientas de desarrollo gratuitas como por la documentación disponible.
- El lenguaje utilizado para la implementación es Java, en el cual se tiene una amplia experiencia.

El conjunto de estas características hacen de Android la plataforma de desarrollo más asequible para el desarrollo de este proyecto, consiguiendo un ecosistema libre de licencias o costos añadidos. Toda la información relativa al desarrollo en Android (SDK, IDE, API, tutoriales, código de ejemplo, etc.) se puede encontrar en la página de desarrolladores que Google ha creado a tal efecto: http://developer.android.com/. El IDE utilizado será el recomendado por la propia Google y al cual se puede acceder desde la mencionada página y que viene con el SDK integrado por defecto: **Eclipse**.

Una vez elegida la plataforma, es necesario buscar recursos que permitan ofrecer un sistema de cartografía y navegación. En este sentido, existen numerosas alternativas tanto gratuitas como de pago, con acceso al código fuente u obteniendo acceso a un API de funciones determinadas.

En concreto existe una alternativa de navegación *open source* llamada **OsmAnd** (http://osmand.net/), en la cual se basará el desarrollo. En base a su código fuente, accesible a través de GitHub (https://github.com/osmandapp/Osmand), se realizarán las modificaciones de la interfaz pertinentes y se añadirán los módulos necesarios para añadir

la funcionalidad deseada. El hecho de que sea *open source* y que su código fuente esté disponible de manera libre y gratuita permite modificar cualquier aspecto de la aplicación.

En los siguientes apartados se realizará un análisis más detallado del proyecto OsmAnd así como las modificaciones y/o añadidos necesarios para alcanzar la funcionalidad buscada.

5.6 Implementación

A continuación se detallarán los diferentes aspectos relacionados con la implementación del sistema, desde la aplicación del patrón de diseño para la plataforma elegida, pasando por el análisis de la aplicación en la que se basará el desarrollo, así como las modificaciones y añadidos realizados.

5.6.1 MVC en Android

Como se ha comentado anteriormente, se aplicará el patrón de diseño *Model-View-Controller*. En la plataforma elegida, Android, existe una peculiaridad y es que cada Vista, que se crea en *xml*, tiene asociada una *Activity* que la controla. Esto supone que si se tienen varias Vistas o pantallas diferentes, cada una de éstas tendrá asociada un *Activity*. El problema es que no es aconsejable acceder desde cada una de dichas actividades al Modelo, a pesar de que pertenezcan al nivel del Controlador. En lugar de eso, todas las actividades accederán al Modelo a través de un Controlador intermedio.

Dicho controlador intermedio se puede implementar en Android en un componente denominado *Application*, que es el que engloba la aplicación completa y dentro del cual están todas las actividades. Este elemento puede sobrescribirse para añadir los métodos necesarios para acceder al Modelo, de tal forma que cuando una *Activity* quiera acceder a los datos, lo hará a través de *Application* [28]. En la Figura 26 - MVC aplicación a Android se puede ver un esquemático del modelo propuesto.

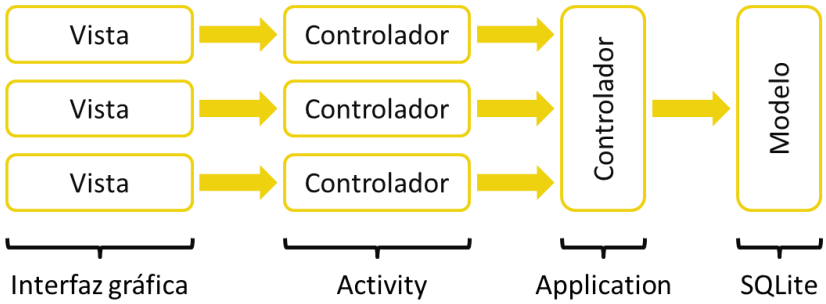


Figura 26 - MVC aplicación a Android

Una vez establecidas dichas bases para el desarrollo, se comenzará a realizar la implementación.

5.6.2 OsmAnd: estructura y modificaciones

A continuación se comentarán algunos detalles sobre la aplicación OsmAnd, así como las modificaciones pertinentes que se han tenido que realizar a su código. De igual forma, se detallarán las implementaciones añadidas a dicho proyecto y que se han realizado en forma de librería externas con el fin de preservar ambos desarrollos lo más separados

posibles (de cara a facilitar tanto el mantenimiento como futuros desarrollos de la aplicación).

5.6.2.1 Proyecto OsmAnd

Como se ha comentado al inicio de esta sección, se utilizará la aplicación OsmAnd como base para el desarrollo de este proyecto. OsmAnd (*Open Street Maps Automated Navigation Directions*) es una aplicación móvil de código libre que permite la visualización de mapas y provee un sistema de navegación (entre otros) de forma totalmente *offline* (sin necesidad de una conexión a Internet).

Esta aplicación hace uso de los datos cartográficos de OSM (*Open Street Maps*), un proyecto colaborativo para la creación de mapas libres y editables.

Utilizando esta aplicación se consigue obtener la base necesaria para obtener la información de ruta para el sistema de estimación a excepción de la altura, que no viene por defecto como dato en la cartografía de OSM. Más adelante en este documento se detallará la solución propuesta para obtener dicho dato de altitud.

Como todo proyecto Android, OsmAnd está estructurado de la siguiente forma:

- La carpeta **/src** contendrá todo el código fuente de la aplicación, código de la interfaz gráfica, clases auxiliares, etc;
- La carpeta **/res** contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, vídeos, cadenas de texto, etc. Los diferentes tipos de recursos se distribuyen entre las siguientes subcarpetas:
  - **/res/drawable/** contiene las imágenes (y otros elementos gráficos) usados en por la aplicación, pudiendo definir diferentes recursos dependiendo de la resolución y densidad de la pantalla del dispositivo, dividiéndose así en varias subcarpetas:
    - **/drawable-ldpi/** (densidad baja)
    - **/drawable-mdpi/** (densidad media)
    - **/drawable-hdpi/** (densidad alta)
    - **/drawable-xhdpi/** (densidad muy alta)
  - **/res/layout/** contiene los ficheros de definición XML de las diferentes pantallas de la interfaz gráfica, pudiendo definir distintos *layouts* dependiendo de la orientación del dispositivo se puede dividir en dos subcarpetas:
    - **/layout** (vertical)
    - **/layout-land** (horizontal)
  - **/res/anim/** contienen la definición de las animaciones utilizadas por la aplicación.
  - **/res/color/** contiene ficheros XML de definición de colores según estado.
  - **/res/menu/** contiene la definición XML de los menús de la aplicación.
  - **/res/xml/** contiene otros ficheros XML de datos utilizados por la aplicación.
  - **/res/raw/** contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos.

  - **/res/values/** contiene otros ficheros XML de recursos de la aplicación, como por ejemplo cadenas de texto (*strings.xml*), estilos (*styles.xml*), colores (*colors.xml*), *arrays* de valores (*arrays.xml*), etc.
- La carpeta **/gen** contiene una serie de elementos de código generados automáticamente al compilar el proyecto.
- La carpeta **/assets** contiene todos los demás ficheros auxiliares necesarios para la aplicación (y que se incluirán en su propio paquete), como por ejemplo ficheros de configuración, de datos, etc.
- La carpeta **/bin** contiene los elementos compilados de la aplicación y otros ficheros auxiliares. Cabe destacar el fichero con extensión *apk*, que es el ejecutable de la aplicación que se instalará en el dispositivo.
- La carpeta **/libs** contiene las librerías auxiliares, normalmente en formato *jar*, que se utilicen en la aplicación.
- El fichero **AndroidManifest.xml** contiene la definición en XML de los aspectos principales de la aplicación, como su identificación (nombre, versión, icono, etc.), sus componentes (pantallas, mensajes, etc.), las librerías auxiliares utilizadas, o los permisos necesarios para su ejecución (acceder a sistemas de geolocalización, enviar sms, etc.)

5.6.2.2 Integración en OsmAnd

Para empezar a trabajar con el código de OsmAnd, será necesario descargar la última versión de éste desde su repositorio oficial alojado en GitHub. Para ello se configurará como repositorio remoto en el repositorio donde se desarrollará el proyecto. Se tendrán por tanto dos repositorios configurados como remotos:

- *origin*: fuente original del repositorio del proyecto, alojado en Bitbucket como se comentó en el apartado 5.4. Es decir, donde se guardará la réplica exacta del repositorio local.
- *upstream*: repositorio oficial de OsmAnd, alojado en GitHub.

Con esta configuración se puede acceder a todos los *commit* (cada cambio realizado en el repositorio) del repositorio oficial, pudiendo seleccionar aquel que pertenece a la última versión estable haciendo un *merge* (mecanismo a través del cual se copia el repositorio en el estado en el que está en dicho *commit* al repositorio local de desarrollo).

Esta configuración permitirá a su vez actualizar a nuevas versiones estables de OsmAnd de manera más sencilla.

Una vez importado el proyecto en Eclipse, se puede ver la estructura general de la organización de su código, el cual ha tenido que modificarse en cierta medida para añadir las funcionalidades requeridas. Dichas modificaciones se detallan a continuación[6].

5.6.2.2.1 Menú principal

En la pantalla principal de OsmAnd existen cuatro botones con la siguiente funcionalidad:

- **Mapa**: acceso a la vista de mapa.

[6] El resultado de la interfaz gráfica así como una visión de la funcionalidad de la aplicación se podrá consultar en el Manual de Usuario, el cual se puede encontrar en el Anexo 9.3.

- **Buscar**: buscar destino (por PDI – Punto de Interés, dirección, coordenadas, favorito, historial o transporte).
- **Favoritos**: acceso directo a localizaciones guardadas como favoritas.
- **Opciones**: opciones de configuración de la aplicación.

En este menú se han añadido dos botones adicionales:

- **Gestión**: funcionalidades relacionadas con la motocicleta (gestión de rutas, histórico de avisos, control remoto y edición del perfil de usuario).
- **Navegación**: pantalla de visualización de telemetrías.

Para ello se ha modificado la clase

`net.osmand.plus.activities.MainMenuActivity.java`

y el *layout*

`menu.xml`

5.6.2.2.2 Widgets

En la pantalla de mapa OsmAnd ofrece la posibilidad de visualizar ciertas magnitudes como la velocidad instantánea, el número de satélites GPS que ha encontrado, la altitud, etc. Dichos objetos son denominados internamente como *widgets* y es posible añadir más creando la funcionalidad de éstos en

`net.osmand.plus.views.mapwidgets.MapInfoWidgetsFactory.java`

Típicamente son de tipo `TextInfoWidget` (muestran texto), con una función `updateInfo(…)` que se encarga de actualizar el valor a mostrar. No obstante, también puede ser del tipo `ImageViewWidget` el cual mostrará una imagen y realizará una tarea al pulsarlo.

Para agregar funcionalidad a dichos *widgets* es necesario registrarlos en el `MapWidgetRegistry mapInfoControls` en la clase

`net.osmand.plus.views.MapInfoLayer.java`

a través del método `registerAllControls()`.

Se agregarán por tanto los *widgets* de visualización de información especificados en el diseño en el apartado 4.4.4.1, además de dos *widget* de imagen correspondientes al botón de inicio/paro de grabación de ruta y de acceso a la pantalla de telemetrías.

5.6.2.2.3 Telemetrías

Como se ha comentado anteriormente, existe la posibilidad de acceder a una pantalla en la que se mostrará únicamente la información de telemetría en tiempo real. Dicha pantalla es accesible desde el menú principal desde el botón Navegación o desde el botón habilitado al uso en la pantalla de Mapa. Dicha pantalla se implementa con la clase

`net.osmand.ebike.Telemetry.java`

y su *layout*

`telemetry.xml`

Esta clase contiene las funciones que permiten cambiar la opción a mostrar, para actualizar los valores utilizando la clase de gestión de telemetrías y los iconos; y para cargar la configuración previa de dicha ventana.

5.6.2.2.4 Gestión

La pantalla de Gestión, accesible desde el botón habilitado a tal efecto en la pantalla principal. Dicha pantalla se implementa con la clase

`net.osmand.ebike.MenuAsistant.java`

y el *layout*

`menu_asistant.xml`

Tendrá cuatro posibles opciones como ya se ha comentado:

- **Gestión de rutas**: visualización del histórico de rutas realizadas, pudiendo consultar las diferentes características de éstas.
  - Clase: `net.osmand.ebike.GestionRutas.java`
  - *Layout*: `gestion_rutas.xml`

  Cuando se selecciona alguna ruta para visualizar la información de ésta, se lleva a otra pantalla:

  - Clase: `net.osmand.ebike.RouteInfo.java`
  - *Layout*: `old_route_info.xml`
- **Histórico de avisos/errores**: lista con el histórico de mensajes de aviso/error recibidos en alguna trama de telemetría.
  - Clase: `net.osmand.ebike.HistoricoEventos.java`
  - *Layout*: `historico_eventos.xml`
- **Control remoto**: acceso directo al botón de llamada de emergencia, al botón de obtención de la localización de la motocicleta y al botón de bloqueo de la motocicleta (estas dos últimas opciones están deshabilitadas por el momento).
  - Clase: `net.osmand.ebike.RemoteControl.java`
  - *Layout*: `remote_control.xml`
- **Perfil de usuario**: configuración del perfil de usuario (nombre, usuario, peso, peso extra, etc.). Esta información se guarda en un fichero de tipo *SharedPreferences* en Android, además de en la base de datos (en ésta no se guardan todos los datos introducidos), pero por el momento no se realiza ningún tipo de tarea de registro o identificación en los servidores de la compañía puesto que de momento no están activos.
  - Clase: `net.osmand.ebike.PerfilRegistro.java`
  - *Layout*: `perfil_registro.xml`

5.6.2.2.5 Aplicación

Como se ha comentado en el apartado 5.6.1, para implementar MVC en Android se utilizará como Controlador la clase *Application*. En el caso de OsmAnd, dicha clase ya es re-implementada y re-nombrada como *OsmandApplication*. Será por tanto en dicha clase donde se implementarán los métodos de acceso al Modelo.

En dicha clase se ha creado un atributo del tipo *CommunicationHandler* (que se explicará en el apartado 5.6.3) a través del cual se accederá al Modelo, sirviendo por tanto de canal de comunicación entre ambas capas, Controlador y Modelo.

Además de esto, se han añadido los siguientes métodos a dicha clase:

- Métodos *get* y *set* de dicho atributo.
- `private Profile getSelectedProfile()`: devuelve el perfil almacenado en la memoria del teléfono y lo inserta en la base de datos si es que no lo está.
- `private void updateSelectedProfile()`: actualiza el dato de odómetro en el perfil (leyéndolo de la base de datos a través del *CommunicationHandler*).
- `private ANN getSelectedANN()`: carga el modelo de la Red Neuronal guardado en memoria.

Será en el método *onCreate()* (método al que se llama al crear la instancia de dicho objeto, es decir al iniciar la aplicación) de dicha clase donde se inicializará el atributo añadido. Acto seguido a su inicialización se utilizará para abrir la comunicación con la base de datos y para establecer el perfil de usuario así como en modelo de Red Neuronal (lo lee de fichero y lo carga en memoria).

5.6.2.2.6 Tratamiento de ruta

Con el objetivo de obtener la información de ruta necesaria para realizar las estimaciones de viabilidad se ha incluido una clase adicional al código base de OsmAnd. Dicha clase es

```
net.osmand.plus.routing.RouteTreatment.java
```

Esta clase es la encargada del tratamiento de las rutas generadas por OsmAnd. Obtiene la ruta y realiza una consulta a un servidor externo para consultar la altitud de cada uno de los puntos GPS marcados en ésta (dicha implementación se detallará más adelante en este documento). Una vez se tiene la ruta con toda la información, se pasa a la Red Neuronal para que realice una estimación de la energía que se consumirá al realizar esa ruta. Posteriormente compara si dicha energía se puede asumir con la energía disponible en la motocicleta, informando al usuario del resultado.

Dicha clase implementa uno de los *listeners* internos de OsmAnd, de tal forma que cuando el sistema ha calculado un nueva ruta, esta clase sea llamada. Dicho *listener* es:

```
net.osmand.plus.routing.RoutingHelper.IRouteInformationListener
```

Destacar que la ruta obtenida de OsmAnd no dispone de información puntual de la velocidad estimada de la ruta. Dicha información, a pesar de que aparece como campo en la cartografía de OSM, en el caso de España la información es escasa. Debido a esto, lo que sí que se puede obtener de la ruta de OsmAnd es el tipo de vía al que pertenece un determinado punto GPS y en base a esto se le asigna una velocidad estándar. Destacar que

este sistema puede llevar a imprecisiones ya que la equivalencia de un tipo de vía con un límite de velocidad no es directa (no al menos en el caso de España, siento estas dependencias responsabilidad de OSM) [29]. La equivalencia que se ha utilizado es la representada en la Tabla 4 - Tipos de vías en OsmAnd y velocidad equivalente.

Tipo de vía OSM	Tipo de vía (equivalente)	Velocidad límite (km/h)
motorway	Autopista	120
trunk	General	100
primary	General	100
secondary	General	90
tertiary	General	90
unclassified	General	50
residential	Poblado	30
living_street	Residencial	20

Tabla 4 - Tipos de vías en OsmAnd y velocidad equivalente

El funcionamiento de esta clase es el siguiente:

- Cuando se calcula una ruta se llama a esta clase a través del *listener* que implementa.
- Se lanza la ejecución de una tarea asíncrona *EstimationTask* que desencadena los siguientes procesos:
  - Obtiene los puntos GPS de la ruta como una lista de objetos *Location* a partir de la ruta *RouteCalculationResult* que genera OsmAnd.
  - Añade a cada uno de esos puntos GPS información sobre la altitud de dicho punto.
  - Convierte dicha lista de puntos GPS en una lista de segmentos *Segment*.
  - Se pasa dicha lista de segmentos a la Red Neuronal *ANN* la cual estimará el consumo tanto para la ida como para la vuelta.
  - Se calculará la viabilidad tanto del trayecto de ida como el de ida y vuelta basándose en la estimación de la Red Neuronal y la energía restante en la batería y se hará saber dicho resultado al usuario a través de un mensaje en pantalla.

5.6.2.3 Compilación

Para poder realizar la compilación del proyecto OsmAnd sin problemas es necesario copiar dos archivos, *default.render.xml* y *routing.xml* (ambos en el directorio raíz del proyecto), al directorio */OsmAnd/bin/clases/net/osmand/* dentro de las carpetas */render* y */router* respectivamente.

Para que dicho trabajo se realice de forma automática cada vez que se compila el proyecto se ha añadido en Eclipse un fichero de ejecución por lotes que realice dicha copia en las propiedades del proyecto, en el apartado *Builders*.

5.6.3 Librería *Connector*

Además de las modificaciones comentadas en el apartado anterior, se han añadido una gran cantidad de funcionalidades a la aplicación para así conseguir el sistema deseado. Dichas implementaciones se han realizado como parte de una librería externa, de tal

forma que, tal como se ha comentado al introducir este apartado, el mantenimiento y futuras mejoras de la aplicación sean más fáciles de implementar.

El nombre elegido para esta librería es *Connector* y está compuesta por diferentes módulos que se explicarán a continuación.

5.6.3.1 Modelo

Se ha creado un paquete cuyo objetivo es definir los diferentes modelos de datos utilizados en el proyecto y que no pertenecen a ninguno de los otros paquetes, es decir las clases java que identifican unívocamente un elemento del sistema. El nombre del paquete es

```
es.uc3m.ebike.model
```

y en dicho paquete se implementan las siguientes clases:

- `Route.java`: identifica una ruta realizada por el usuario con el conjunto de características que se definieron en el diseño.
- `Segment.java`: identifica a un segmento de una ruta, es decir un trozo de ésta según se especificó en el apartado de diseño.
- `TelemetryFrame.java`: trama de telemetría con sus correspondientes datos de códigos de error, magnitudes físicas, etc.
- `GPSFrame.java`: trama GPS con las coordenadas geográficas y el *timestamp*.
- `Profile.java`: perfil de usuario.
- `ANNTrain.java`: objeto que identifica cada modelo de Red Neuronal generado de cara a guardarlo en la Base de Datos.

5.6.3.2 Base de datos

En Android se utiliza como sistema gestor de base de datos SQLite. SQLite no es una base de datos cliente-servidor, sino que se enlaza con la aplicación, integrándose como una parte más de ésta [30].

Para poder gestionar la Base de Datos en la aplicación, Android proporciona todas las herramientas que se necesitan, de manera que se pueden realizar todas las tareas de forma sencilla. Para la implementación de ésta, se tomará como referencia el tutorial que presenta Igor Ávila en su web [28].

Para este propósito se ha creado un paquete que englobe toda la lógica relativa al tratamiento con la Base de Datos denominado

```
es.uc3m.ebike.database
```

Dicho paquete está compuesto por las siguientes clases:

- `DBHelper.java`: clase que sirve para crear y abrir la Base de Datos (y en su caso actualizar el modelo).
- `TBPROFILE.java`: clase que define la tabla correspondiente al perfil de usuario, con sus correspondientes métodos de *insert*, *delete*, *update*, y *get*.

- `TBROUTE.java`: clase que define la tabla correspondiente a una ruta, con sus correspondientes métodos de *insert*, *delete*, *update*, y *get*.
- `TBANN_TRAINING.java`: clase que define la tabla correspondiente a un modelo de Red Neuronal, con sus correspondientes métodos de *insert*, *delete*, *update*, y *get*.
- `TBSEGMENT.java`: clase que define la tabla correspondiente a un segmento de ruta, con sus correspondientes métodos de *insert*, *delete*, *update*, y *get*.
- `TBGPS.java`: clase que define la tabla correspondiente a un dato de GPS, con sus correspondientes métodos de *insert*, *delete*, *update*, y *get*.
- `TBTELEMETRY.java`: clase que define la tabla correspondiente a una trama de telemetría, con sus correspondientes métodos de *insert*, *delete*, *update*, y *get*.
- `DBAdapter.java`: clase que define los métodos de acceso a la base de datos, actúa de pasarela a los diferentes métodos comentados anteriormente.

Únicamente será necesario trabajar con una instancia de la clase *DBAdapter* para poder trabajar con la base de datos.

5.6.3.3 Comunicación

En este módulo se desarrollan las tareas de comunicación del dispositivo, así como el planificador de peticiones y la gestión de dichas comunicaciones con el conjunto de la aplicación. Se ha creado un paquete denominado

```
es.uc3m.ebike.communication
```

Dicho paquete está compuesto por las siguientes clases:

- `Connection.java`: clase abstracta que encapsula los métodos de comunicación con la centralita.
- `BluetoothConnection.java`: implementación de la clase *Connection* para soportar conexiones mediante Bluetooth (en este caso, con la librería Bluetooth que nos proporciona el fabricante de ésta y que se comenta en el apartado 5.6.4).
- `USBConnection.java`: implementación de la clase *Connection* para soportar conexiones mediante USB (funcionalidad actualmente no disponible).
- `RequestsScheduler.java`: planificador de peticiones de telemetría y GPS. Se encarga de hacer peticiones de forma asíncrona y periódica a la centralita, actualizando un *buffer* de segmentos (que agrupará los datos de telemetría junto con dos puntos GPS cuyo *timestamp* se encuentre entre los de éstos) al cual se accederá a través de la instancia de *CommunicationHandler*, pudiendo así acceder a los datos de telemetría y GPS en tiempo real (sin que esto interfiera en la ejecución de la aplicación, ya que dicha tarea se realiza en segundo plano de ejecución). Se realizarán peticiones cada 10 segundos, agrupando típicamente 10 datos de telemetría por cada 2 posiciones GPS.
- `CommunicationHandler.java`: clase principal de la librería *Connector*. Funciona como puerta de enlace a todas las funciones externas a OsmAnd como acceder a la base de datos, acceder a los últimos datos de telemetría recibidos, iniciar y detener la ruta (y con ello que se inicie y pare la petición de tramas), etc.

5.6.3.4 Inteligencia Artificial

Todos los aspectos relacionados con la Inteligencia Artificial del sistema, es decir la implementación de la Red Neuronal, se han agrupado en un paquete llamado

```
es.uc3m.ebike.ai
```

En dicho paquete se pueden encontrar las siguientes clases:

- `ANN.java`: implementación de la Red Neuronal, con sus métodos de entrenamiento y estimación (tanto de la ruta de ida –*outgoing*– como de ida y vuelta –*round trip*–). Existirá una instancia de dicha clase en *CommunicationHandler* a través de la cual se accederán a todos los métodos de ésta. Para poder realizar el entrenamiento, esta clase recibirá la lista de segmentos que forman la ruta, calculando la variación de velocidad ΔV como la diferencia entre la velocidad media entre un segmento y el anterior. En el cálculo de la ruta de vuelta variará tanto esa magnitud como la variación de altura ΔA que será inversa a la de la ida. Se hace notar que el entrenamiento se realizará por rutas, cada cual se dividirá en 3 bloques (datos de entrenamiento, validación y test) como se ha comentado anteriormente.
- `ANNTrainTask.java`: tarea asíncrona (en segundo plano de ejecución) para realizar el entrenamiento de la Red Neuronal (ya que puede consumir muchos recursos de procesador y memoria). Si se llamara directamente al método *train* de *ANN* éste bloquearía el hilo principal, por lo que se emplea esta tarea asíncrona para realizar dicho entrenamiento.
- `MLP.java`: implementación básica de una Red Neuronal sobre la que se construye la implementación de más alto nivel *ANN*.
- `Synap.java`: implementación de lo que sería la sinapsis de la Red Neuronal. Empleada para construir la implementación de *MLP*.

5.6.3.5 Servicio Web: obtención de alturas

Como se ha comentado en anteriores apartados de este documento, no es posible obtener de OsmAnd información acerca de la altitud de los puntos GPS que conforman la ruta. Es por ello que se han buscado diferentes alternativas para así disponer de estos datos (ya que son de vital importancia para el funcionamiento del Modelo de Predicción de Consumo).

En concreto se ha utilizado una base de datos realizada por la NASA [31] con datos de elevación de aproximadamente el 80% de la superficie terrestre: **SRTM** (*Shuttle Radar Topography Mission*). Dicha información se puede descargar a través de la página web como una base de datos en crudo dividida por coordenadas geográficas como se puede observar en la Figura 27 - Fraccionado base de datos de alturas SRTM.

Cada fichero que puede ser descargado tiene información de un cuadrante de los mostrados en la figura, estando todos los datos de altura guardados de forma secuencial (de Oeste a Este y de Norte a Sur) con una precisión de 90 metros (1.201 x 1.201 muestras, cada una de las cuales corresponde con el centro geométrico de una celda de 90

metros). Cada uno de esos ficheros abarca un área de 1º de latitud x 1º de longitud y ocupa 2.75 MB.

Tomando los datos correspondientes a los cuadrantes de interés, es decir los correspondientes a España, se ha elaborado una Base de Datos con todos los datos ya organizados por coordenadas, de tal forma que se pueda realizar una consulta para una determinada posición GPS y ésta devuelva la elevación en dicho punto (con la precisión antes comentada). Se ha montado un servidor con un ISS y SQL Server el cual dispone de un servicio web para realizar peticiones de alturas.

La razón de utilizar este sistema para la consulta de alturas es que la cantidad de memoria necesaria para almacenar los datos de altitud es muy grande como para poder guardarla en un dispositivo móvil. Incluso descargando únicamente datos de una región concreta (por ejemplo, la Comunidad de Madrid) el tamaño es considerable.

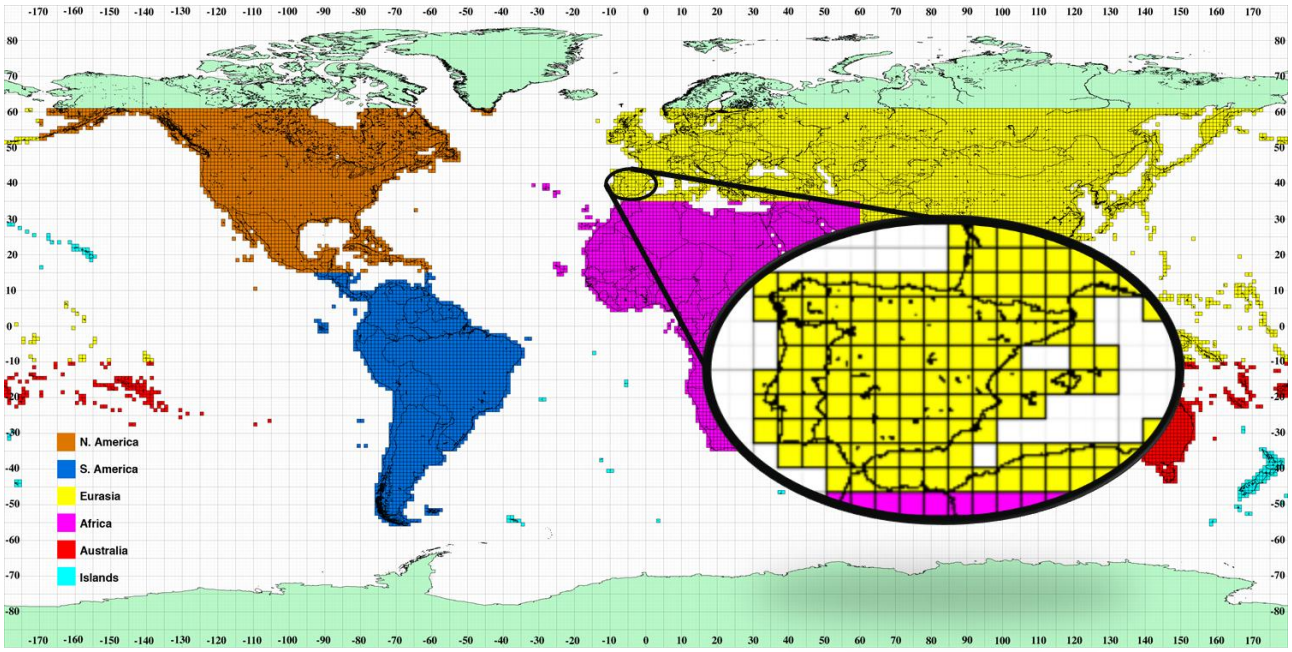


Figura 27 - Fraccionado base de datos de alturas SRTM

El servicio web que se utiliza recibe una lista de puntos GPS y devuelve esos mismos puntos pero con información de altitud. Para su integración con la aplicación se ha utilizado la librería *ksoap2-android* [32] y la herramienta Wsdl2Code [33] para generar el código Android a partir del fichero WSDL que define el servicio web publicado en un servidor propio.

Dicha herramienta genera una serie de clases *java* que se han incluido en el proyecto en el paquete

```
es.uc3m.ebike.webservices.elevations
```

Será a través de dichas clases como se puede acceder a los métodos del servicio web. En concreto, será a través de la clase *Elevations.java* como se obtendrá acceso a los diferentes métodos y será en esa misma clase donde se configure el espacio de nombre y la *url* del servicio web a través de las variables globales:

- `NAMESPACE ="http://ebike.services.es/Elevations"`
- `url="http://163.117.157.106:8080/services/Elevations.asmx"`

Además de dichas clases, se ha añadido una clase adicional que sirva de interfaz con la clase *Elevations* y que encapsule la información en el formato empleado en el resto de la implementación, es decir, que devuelva las posiciones GPS como objetos *Location* (clase propia de Android para el tratamiento de datos de geoposicionamiento). Dicha clase se denomina

`ElevationsServices.java`

y se encuentra en el paquete

`es.uc3m.ebike.webservices`

Esta clase se utilizará en la clase *RouteTreatment* cuando al obtener la ruta como un conjunto de puntos GPS de OsmAnd, cree una nueva añadiendo los datos de altura de cada uno de dichos puntos a través del método de *ElevationsServices*

```
List<Location> getLocationsElevations(List<Location> locations)
```

Posteriormente se tratarán dichos punto GPS como ya se ha explicado en el apartado 5.6.2.2.6 (conforme al diagrama de flujo del programa diseñado que se puede ver en el apartado 4.4.3.2).

5.6.3.6 Utilidades

Además del conjunto de funciones hasta ahora descritas, se han añadido algunas clases adicionales al programa a través del paquete denominado

`es.uc3m.ebike.utils`

En dicho paquete se pueden encontrar las siguientes clases:

- `ANNUtils.java`: dispone de dos métodos orientados al análisis del funcionamiento de la Red Neuronal y que pueden ser utilizados para guardar una copia de los errores de entrenamiento, validación y test en un fichero *csv* para su posterior análisis en una herramienta tipo Excel.
- `DateUtils.java`: dispone de un método estático que permite dar formato local a un objeto de fecha tipo *Date*. En el caso de España el formato es: *"31/10/2014 15:00"*.
- `DBUtils.java`: permite hacer una copia de la base de datos en otro directorio del dispositivo móvil y restaurar ésta si es necesario. La copia de seguridad se guardará en */storage/ebike/databases* y para que se pueda restaurar una base de datos tendrá que guardarse en ese mismo directorio con el nombre *eBike.db*. El objetivo de esta clase es principalmente utilizarse para tareas de depuración, pero se puede mantener o variar para dar la citada funcionalidad al usuario.
- `DialogUtils.java`: dispone de un método estático para crear y mostrar al usuario diálogos de confirmación con un título, un texto y un botón de aceptar.

- `GPSUtils.java`: dispone de un método estático para calcular la distancia en kilómetros entre dos puntos GPS a través de la fórmula del Haversine [34].
- `JSONUtils.java`: dispone de métodos para tratar los JSON de telemetría y GPS recibidos a través de la librería Bluetooth explicada en el apartado 5.6.4 y convertirlos según el tipo de datos manejado en el resto del programa, que en este caso son objetos del tipo *TelemetryFrame* y *GPSFrame*.
- `ObjectUtils.java`: dispone de métodos para pasar de una instancia de un *Object* a un *array* de *bytes*, así como de métodos para guardar el modelo de estimación de la *ANN* en disco y poder cargarlo de nuevo en memoria (por el momento se utiliza un único modelo, pero dado el dimensionamiento realizado en la base de datos, se podrían realizar varios modelos y guardar sus referencias según la tabla *Entrenamiento Red Neuronal* visto en el apartado 4.4.2).
- `RouteUtils.java`: dispone de un método a través del cual se obtiene la duración total de una ruta *Route*.
- `TestData.java`: dispone de métodos para simular rutas en base al modelo analizado en el apartado 3.3.2. Para ello, toma como entrada una lista de segmentos de una ruta *Segments* y simula la energía que consumiría el vehículo en una situación real. Esta clase se ha creado con el objetivo de evaluar las prestaciones del sistema, ya que no se han podido realizar pruebas con el vehículo real. El algoritmo de simulación empleado se detallará más adelante en el apartado 6 Resultados.

5.6.4 Librería Bluetooth

La empresa que diseña y fabrica la centralita de la motocicleta con la cual se ha de comunicar la aplicación es la encargada de facilitar una librería Bluetooth que proporcione la interfaz de comunicación necesaria para que se puedan tratar los datos recibidos.

Dicha librería está incluida en el proyecto y provee una única clase java *CommBT.java* con los siguientes métodos:

- Conectar.
- Desconectar.
- Leer telemetría.
- Leer GPS.

Será a través de dichos métodos como actuará la librería *Connector* para realizar las distintas tareas de comunicación que se han comentado en el apartado anterior.

Los datos de telemetría se obtendrán en formato JSON (estándar de intercambio de datos [35]), estando su estructura detallada en el Anexo 9.2).

Puesto que la comunicación Bluetooth es bloqueante, es necesario ejecutar dichos métodos de forma asíncrona, es decir, en un segundo hilo de ejecución. Esto se consigue a través de la ya comentada clase *RequestsScheduler* del paquete *communication*.

En teoría, cada petición de telemetría y/o GPS devolvería los últimos valores almacenados en el *buffer* de la centralita (con un límite máximo en función de la memoria disponible).

No obstante, en la versión con la que se ha trabajado es necesario especificar la cantidad de tramas que se quieren solicitar.

5.7 Conclusión

Para el desarrollo de este proyecto se ha tomado como referencia la metodología de desarrollo **Kanban**, la cual permite una gestión ágil del mismo así como una evolución continua y constante de las diferentes implementaciones.

Así mismo, se ha elegido trabajar con la plataforma **Android**. Más concretamente, se ha basado el desarrollo en una aplicación *open source* denominada **OsmAnd**, la cual se ha modificado para alcanzar las características deseadas.

La gestión del código se ha realizado utilizando el sistema de control de versiones distribuido **Git**. Dicho sistema, aunándolo con el flujo de trabajo **Gitflow**, permite mantener copias de seguridad, trabajar en nuevas características manteniendo copias estables, trabajar en equipo, etc.

En base a todo esto, se ha construido una aplicación para dispositivos Android que ofrece las características de una aplicación de navegación al uso (visionado de mapas y navegación guiada) pero ofreciendo además la posibilidad de conectarla con la motocicleta a través de Bluetooth y así poder ver datos de telemetría en tiempo real y grabar rutas que luego se podrán visualizar.

Además de ello, se ha implementado un sistema de Inteligencia Artificial basado en Redes Neuronales que aprende del comportamiento del piloto a través de las rutas grabadas, creando así un Modelo de Predicción de Consumo que permite obtener estimaciones de viabilidad de las rutas a realizar (tanto del trayecto de ida como del trayecto de ida y vuelta).

6 Resultados

6.1 Introducción

Tras haber diseñado e implementado el sistema completo, será necesario realizar algún tipo de prueba de verificación o de rendimiento del sistema.

Este apartado se centrará principalmente en el análisis de los resultados proporcionados por el Modelo de Predicción de Consumo generado con técnicas de *Machine Learning* como son las Redes Neuronales. No obstante, también se incluyen los resultados de una serie de pruebas realizadas para comprobar el correcto funcionamiento de la comunicación Bluetooth.

A continuación se detallarán el conjunto de pruebas a realizar, así como los resultados pertinentes y las modificaciones realizadas en caso de ser necesarias (o con objeto de evaluar diferentes aspectos en busca de un mejor rendimiento).

6.2 Modelo de Predicción de Consumo

Como se ha comentado anteriormente, el objetivo principal de este trabajo es diseñar, implementar y evaluar la implantación de un sistema de estimación de viabilidad basado en Redes Neuronales. Una vez presentada la teoría y diseñada e implementada una posible solución, es necesario realizar una serie de pruebas para cuantificar los resultados de este sistema.

6.2.1 Simulación de rutas

Dado que no ha sido posible probar el sistema en un entorno real (más allá de la comunicación Bluetooth), no se han podido obtener valores reales de consumo de la motocicleta. Debido a esto, se ha optado por **simular su comportamiento** para así disponer de una base de datos de rutas con la cual entrenar a la Red Neuronal.

Dicha simulación se basa en el modelo analizado en el apartado 3.3.2, realizando ciertas modificaciones para simular un entorno "más realista", es decir añadiendo cierta variabilidad aleatoria a los parámetros. Además de dicho modelo, se emplea la información de la ruta que obtiene OsmAnd junto con el servidor de alturas, de tal forma que dada la ruta con sus puntos GPS (incluida la altitud) se calculan los segmentos que la forman y se realiza una simulación de la energía que se consumiría. El modelo de simulación se detalla a continuación.

Para obtener la ruta con sus correspondientes segmentos se emplea la misma rutina utilizada para la estimación de la viabilidad explicada en el apartado 5.6.2.2.6. Sobre ese procedimiento se realizarán algunas modificaciones para que en lugar de proceder a la estimación mediante la Red Neuronal, se utilice dicha información de ruta para simular un comportamiento real.

Una vez obtenida dicha ruta se aplica el siguiente procedimiento para simular la energía que consumiría el sistema:

La ecuación (20) recoge todos los datos a la hora de simular el consumo; dicha ecuación está basada en el conocimiento que se tiene del sistema como se ha comentado anteriormente.

$$\Delta E = F_{aerodinámica} \cdot \Delta D + F_{rodadura} \cdot \Delta D + \Delta E_p + \Delta E_c + P_{Sist.Tx} \cdot \frac{\Delta D}{V} + P_{aux.} \cdot \frac{\Delta D}{V} \quad (20)$$

Se simula la **velocidad "real"** del vehículo a través de la ecuación (21). Donde la velocidad se modifica en base a la supuesta velocidad del segmento multiplicándola por una función de densidad de probabilidad uniforme entre 1 y 1.3, de tal forma que la velocidad estará entre la original y un 30% más con la misma probabilidad para todos los valores intermedios. Es decir, si dicho segmento tiene una velocidad media establecida por el tipo de vía de 120 km/h entonces la velocidad simulada estará entre dicho valor y 156 km/h, pudiendo tomar cualquiera de los valores intermedios con igual probabilidad.

$$V_{segmento\,(i)} = U[1{,}1.3] \cdot V_{segmento\,(i)} \cdot \beta + V_{segmento\,(i-1)} \cdot (\beta - 1) \quad (21)$$

Así mismo, se aplica un filtro de media móvil exponencial (*exponential moving average filter*) que suaviza la curva de velocidades medias, es decir evita cambios bruscos entre la velocidad de un segmento y otro. Esto es debido a que puede darse la situación de que en un segmento se tiene una velocidad de 120 km/h pero en el siguiente la velocidad es de 50km/h porque se ha cambiado el tipo de vía. Aplicando este tipo de filtro las variaciones bruscas se eliminan, generando así una curva más suave, en definitiva: más parecida a lo que haría un piloto real. En la Figura 28 - Variación de velocidad en función del filtrado realizado se puede ver cómo varía la "suavidad" de la curva en función del parámetro β que define el filtro en una ruta Leganés-Toledo.

A partir de estas observaciones, se ha seleccionado un valor de $\boldsymbol{\beta = 0.075}$ puesto que suaviza lo suficiente como para crear una curva "realista".

Además de ello, se calcula el **ángulo de inclinación** del segmento necesaria para calcular la resistencia a la rodadura a partir del incremento de distancia y el incremento de altura según la ecuación (22).

$$\alpha = arcsin\left(\frac{\Delta A}{\Delta D}\right) \rightarrow F_{rodadura} \quad (22)$$

Además hay que tener en cuenta que a pesar de que se utilice un frenado regenerativo, la energía recuperada no es ni de lejos similar a la equivalente gastada, por ejemplo, acelerando y decelerando a la misma velocidad en la misma distancia. Hay que tener en cuenta que si la eficiencia de transmisión de energía desde la batería hasta la rueda esté sobre el 80%, la energía recuperada tiene que realizar el mismo ciclo de vuelta, con lo que la eficiencia máxima no podrá sobrepasar el 80% * 80% = 64%.

En la simulación realizada se ha supuesto una **eficiencia en la regeneración** del 35%, por lo que el cálculo del incremento de energía se realizaría en base a la ecuación (23).

$$\Delta E_c = \begin{cases} \Delta E_c & si\ V_i > V_{i-1} \\ \Delta E_c \cdot 0.35 & si\ V_i < V_{i-1} \end{cases} \quad (23)$$

Es decir si se produce una deceleración, esto es, un frenado del vehículo, el incremento de energía tendrá signo negativo y estará escalado por ese 35% (en la realidad dicho valor no

será estático, sino que en función de la agresividad variará, disminuyendo para frenadas bruscas en las que tenga que intervenir el freno mecánico).

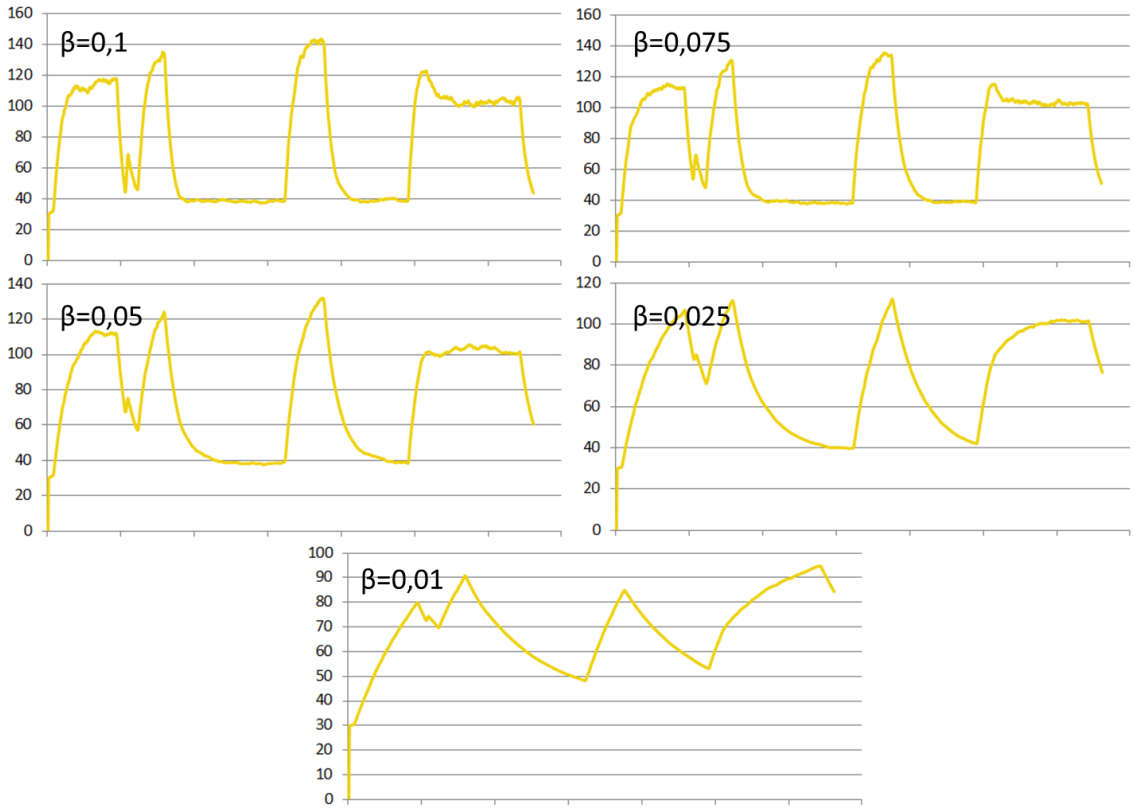


Figura 28 - Variación de velocidad en función del filtrado realizado

En base a dicho algoritmo de simulación, para el trayecto Leganés-Toledo mencionado anteriormente, se obtiene una **evolución de la energía** como la representada en la Figura 29 - Simulación de consumo energético Leganés-Toledo. Donde se han tomado como referencia los valores de consumo analizados en el apartado de diseño, obtenidos a partir de un vehículo Tesla Roadster, y que se pueden consultar en el apartado 3.3.2. Se ha establecido un nivel de energía inicial de 10 MJ (como referencia para ver el comportamiento de la simulación, no es un dato del fabricante).

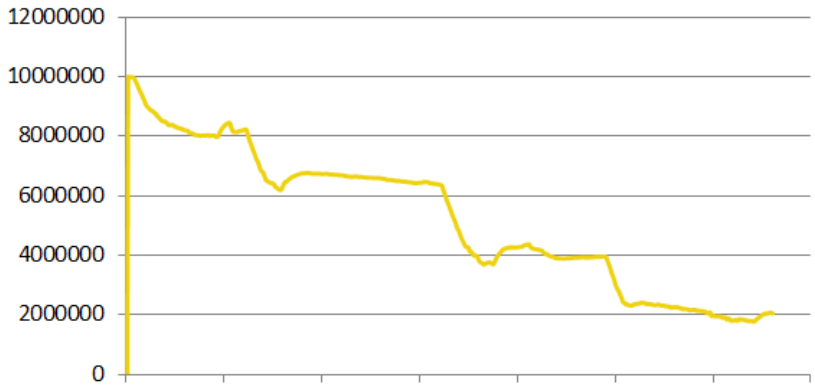


Figura 29 - Simulación de consumo energético Leganés-Toledo

Si se realiza dicha representación para los valores del filtro antes mostrados, se observa la evolución de la Figura 30 - Simulación de consumo energético Leganés-Toledo (varios filtros).

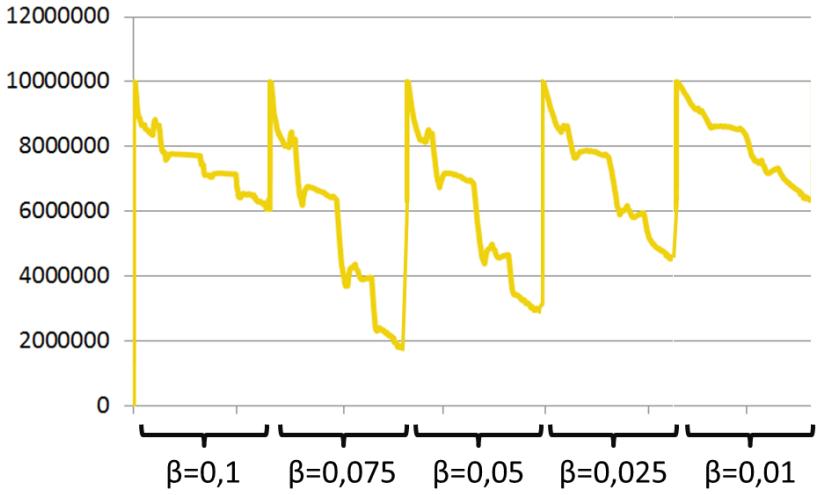


Figura 30 - Simulación de consumo energético Leganés-Toledo (varios filtros)

En la misma simulación se pueden superponer los datos de velocidad y altitud para comprobar que el sistema no tiene incongruencias. En la Figura 31 - Simulación de ruta Leganés-Toledo (energía, velocidad, altitud) se puede ver el resultado, estando en amarillo el nivel de **energía disponible**, en gris la **variación de velocidad** y en azul la **variación de altitud** (comenzando en Leganés a unos 665m y acabando en Toledo a unos 529m). En dicha imagen se puede observar como cuando el vehículo realiza una deceleración la energía en la batería se incrementa, así como en los tramos en los que se desciende.

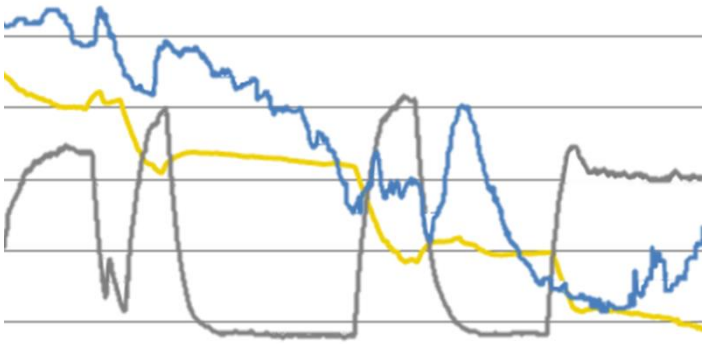


Figura 31 - Simulación de ruta Leganés-Toledo (energía, velocidad, altitud)

En base a estas pequeñas simulaciones se toma el modelo por bueno y se utilizará a lo largo de las pertinentes pruebas realizadas. En lo sucesivo de este documento se referirá a dichas simulaciones como el **consumo real** o como la **variable deseada** a estimar, dado que la Red Neuronal intentará igualar la forma de dicha función: la energía consumida en una ruta (línea amarilla en el gráfico anterior) a partir de la información disponible de ésta (velocidad, elevación, etc.).

6.2.2 Entrenamiento de la Red Neuronal

En primer lugar se harán algunas pruebas para comprobar que la Red Neuronal efectivamente puede llegar a entrenarse con los datos facilitados. Para dicha comprobación se harán diversos entrenamientos y se comprobará si el **error de entrenamiento y validación disminuyen** según aumentan las iteraciones (principalmente éste último, ya que el error de entrenamiento siempre disminuirá ya que ese es el objetivo del algoritmo).

Además de ello se comprobará el **coeficiente de correlación lineal** como bien se adelantó en el apartado 4.3.7.

En el algoritmo de entrenamiento se ha impuesto un número mínimo de iteraciones, el cual se cumplirá aunque el error de validación muestre signos de aumentar. Así mismo, se ha establecido un **límite superior de iteraciones** que el algoritmo no superará en ningún caso (es preferible computacionalmente realizar un nuevo entrenamiento si dado ese número de iteraciones no ha conseguido obtener un buen modelo).

A continuación se muestran las gráficas con los valores de error cuadrático de entrenamiento y validación, así como el coeficiente de correlación lineal (Figura 32 - Error de entrenamiento y validación RN). No se ha incluido el **tiempo de cómputo** de cada modelo puesto que el límite de iteraciones hace que éste **no llegue a 1 minuto**. El conjunto de datos lo forman 15 rutas de una distancia de entre 20 y 70 km, con una densidad de puntos GPS de entre 400 y 1400, es decir entre 200 y 700 segmentos de ruta (aproximadamente).

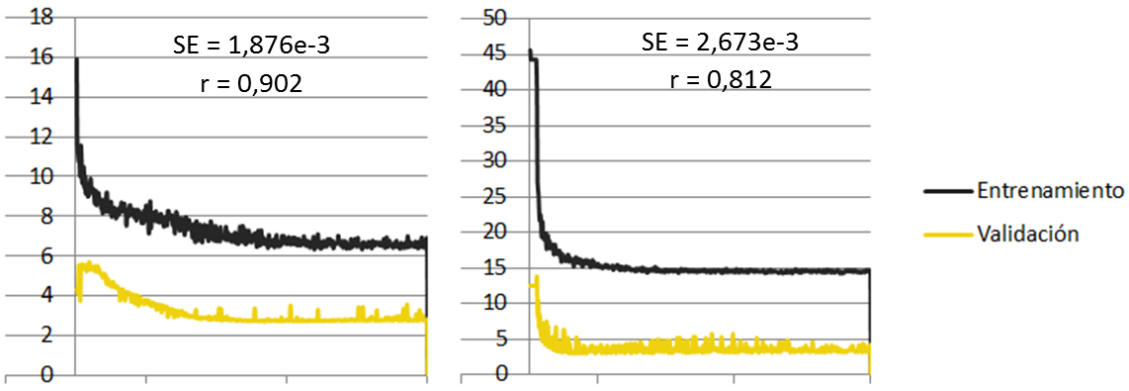


Figura 32 - Error de entrenamiento y validación RN

Como se puede observar, en estas primeras pruebas los resultados son satisfactorios, ya que se puede ver cómo el error de validación tiene una tendencia a disminuir y el factor de correlación tiene valores superiores a 0.7. En las pruebas realizadas en este aspecto, alrededor del **20% de los entrenamientos conseguían resultados muy pobres** ($r < 0.7$), por lo que sería una buena táctica **realizar tantos entrenamientos como sea necesario hasta alcanzar un coeficiente de correlación satisfactorio** (dado que cada entrenamiento no llega al minuto de tiempo, es más asequible este método que seguir iterando –además de que no siempre se llega a un mínimo de la función de coste–).

6.2.3 Pruebas

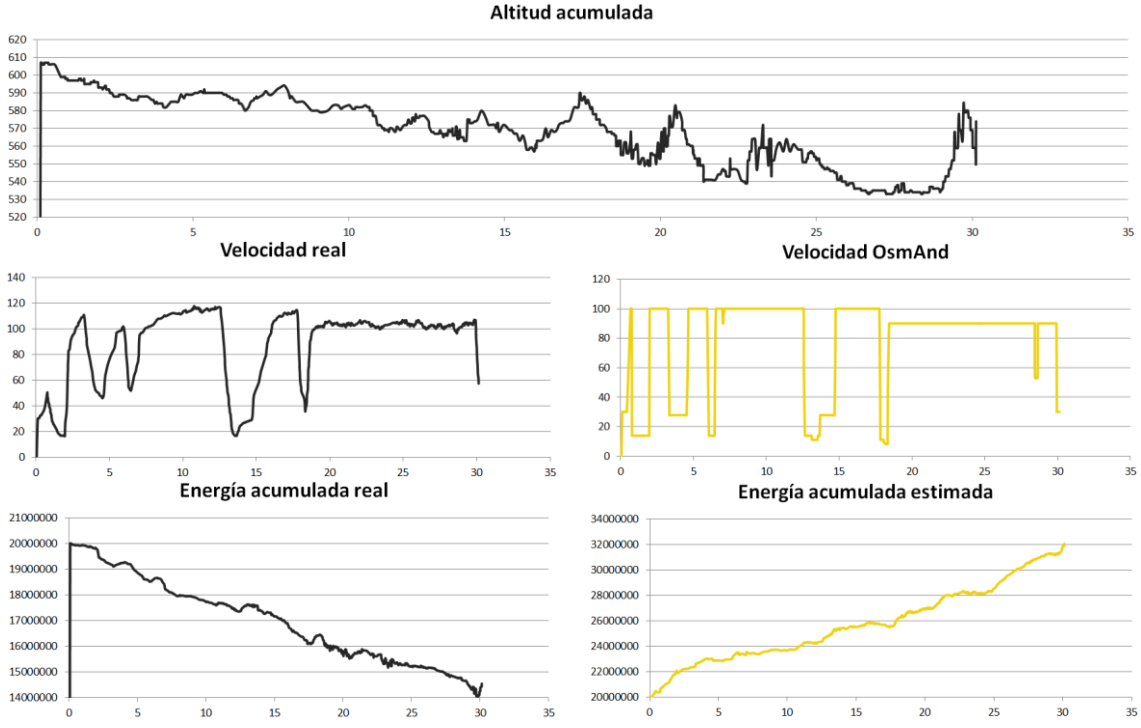
Con el objetivo de evaluar el comportamiento del modelo de predicción de consumo se han realizado una serie de pruebas.

6.2.3.1 Prueba 1

Se han simulado varias rutas, se ha entrenado la Red Neuronal y posteriormente se ha estimado el consumo de dichas rutas para comprobar con qué exactitud es capaz el sistema de modelar el comportamiento del vehículo. Para cada una de las rutas se mostrarán gráficos con la variación de altitud, la velocidad en ruta, la velocidad máxima de la vía según OsmAnd, la energía consumida, la energía consumida estimada y la representación del incremento de energía en cada momento (la simulada y la estimada). Todas estas gráficas se representarán en función de la distancia recorrida. Además, se añadirá una gráfica adicional con los incrementos de energía en función del número de segmento que sea.

6.2.3.1.1 Ruta 1

La ruta corresponde al trayecto Medina de Pomar – Frías, que tiene una distancia total de unos **30 km** y está formada por **941 segmentos** a una media de distancia de **32 m por segmento**. En la Figura 33 – Características Ruta 1 se pueden observar diferentes características de la ruta en cuestión.



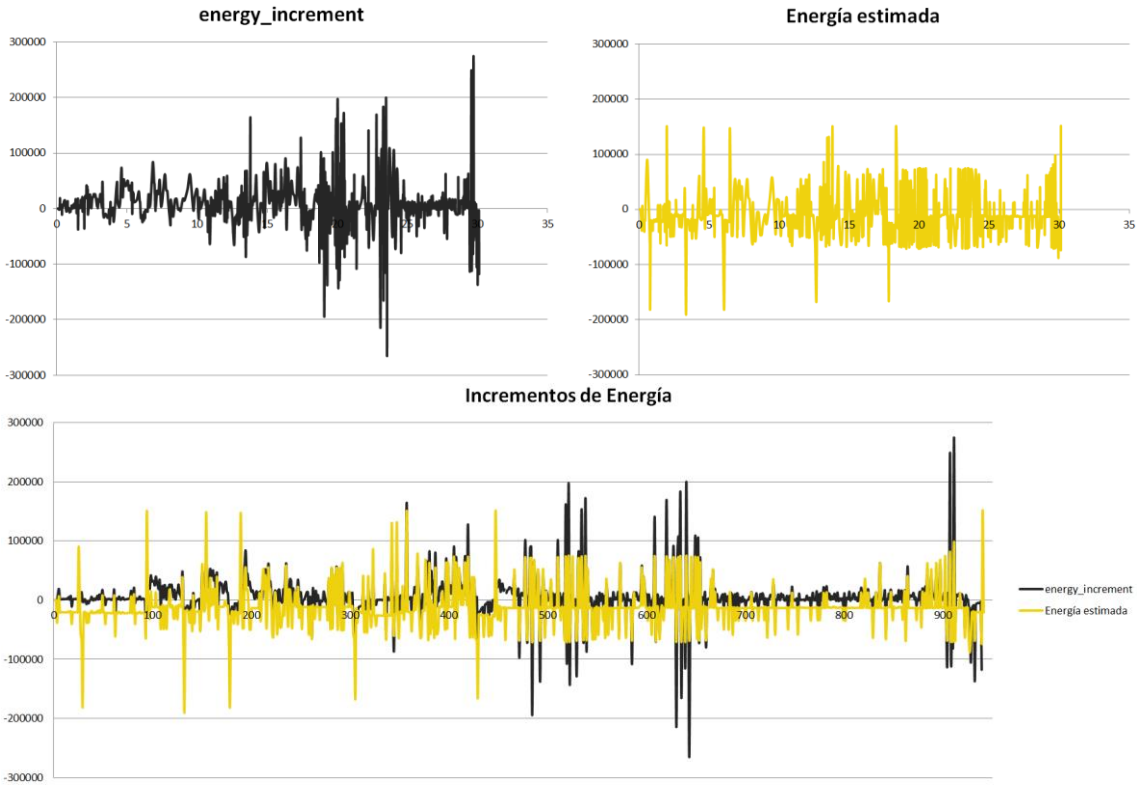
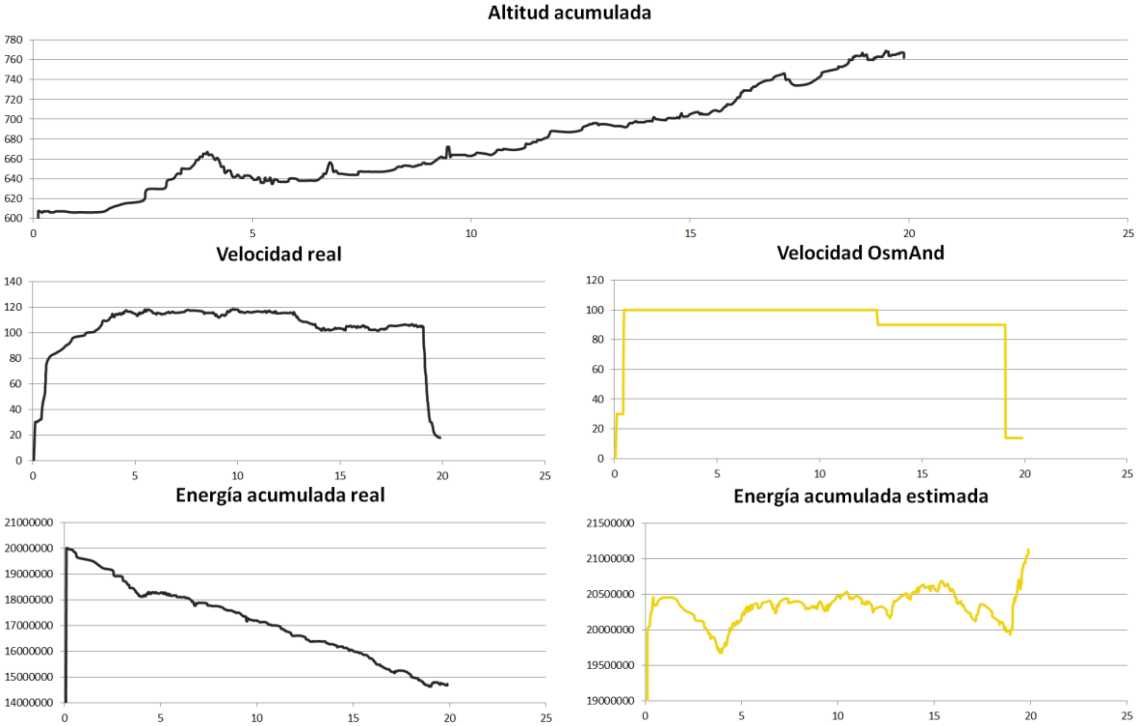


Figura 33 – Características Ruta 1

6.2.3.1.2 Ruta 2

La ruta corresponde al trayecto Medina de Pomar – Espinosa de los Monteros, que tiene una distancia total de unos **20 km** y está formada por **399 segmentos** a una media de distancia de **50 m por segmento**. En la Figura 34 - Características Ruta 2 se pueden observar diferentes características de la ruta en cuestión.



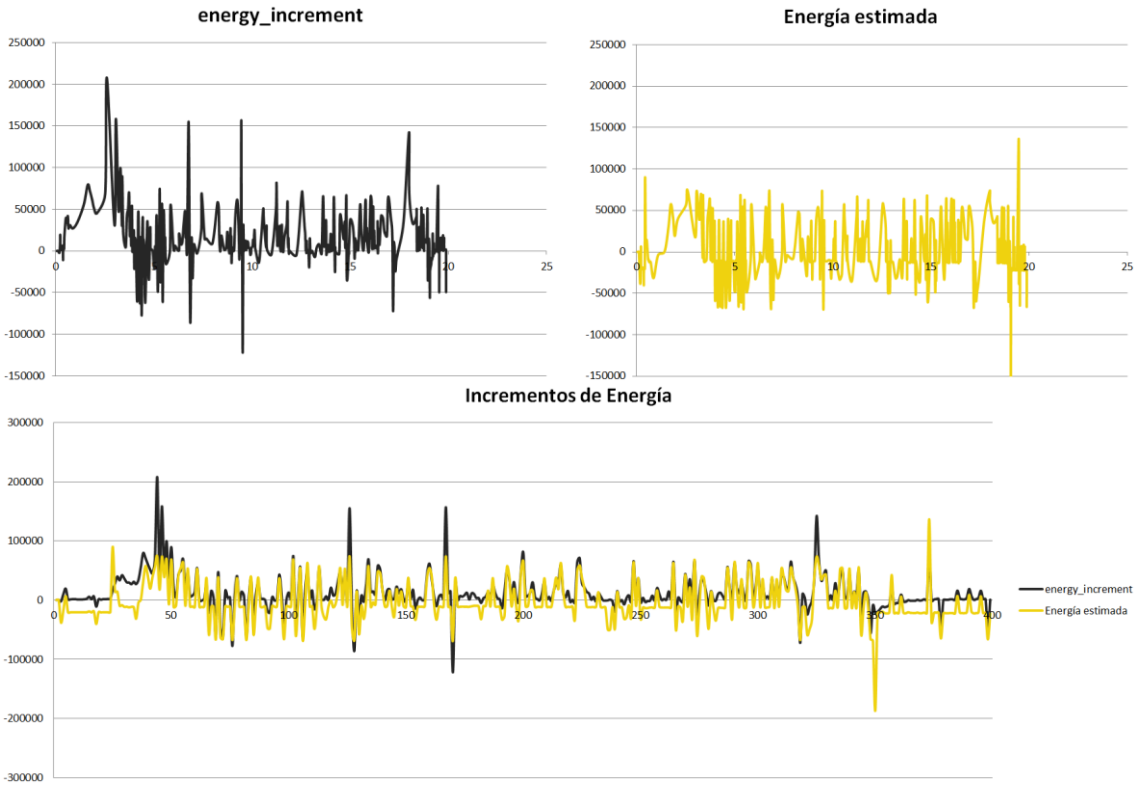
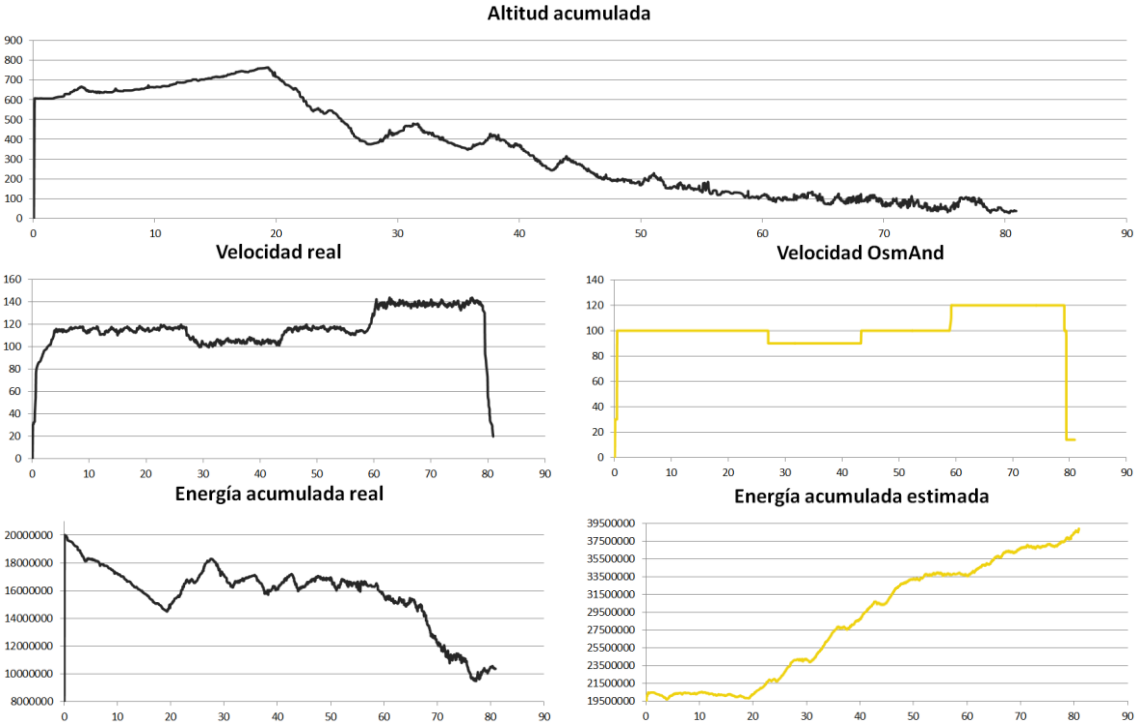


Figura 34 - Características Ruta 2

6.2.3.1.3 Ruta 3

La ruta corresponde al trayecto Medina de Pomar – Bilbao, que tiene una distancia total de unos **81 km** y está formada por **2.227 segmentos** a una media de distancia de **36 m por segmento**. En la Figura 35 - Características Ruta 3 se pueden observar diferentes características de la ruta en cuestión.



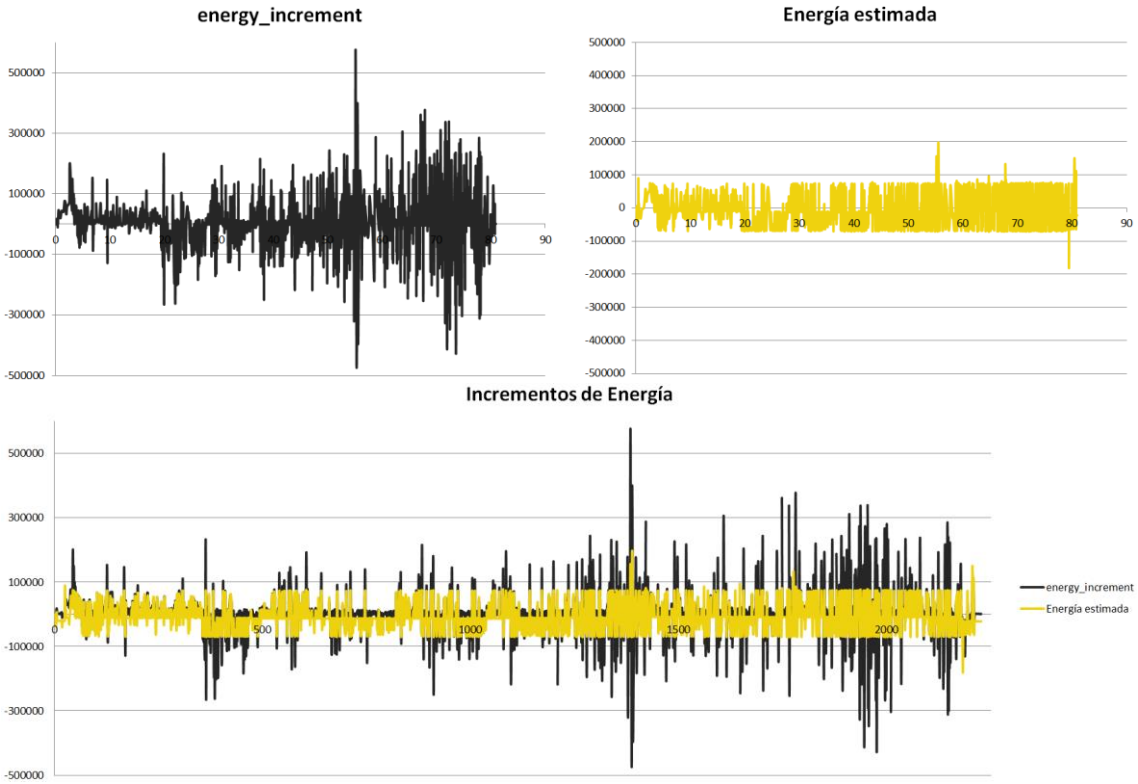
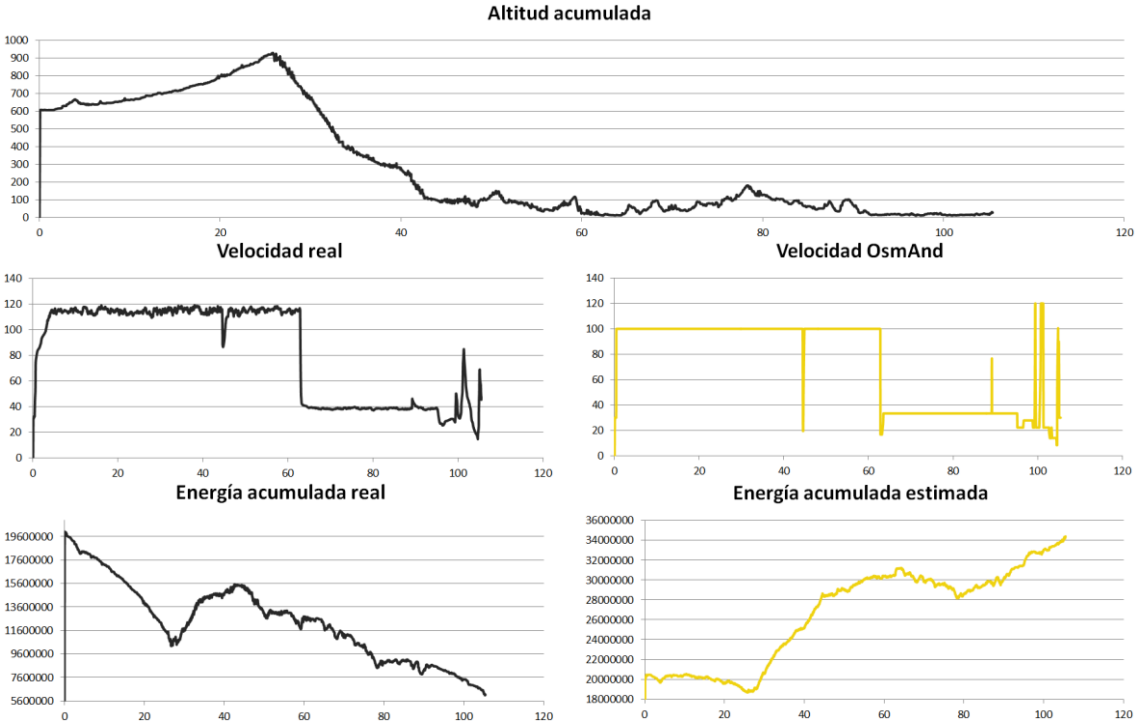


Figura 35 - Características Ruta 3

6.2.3.1.4 Ruta 4

La ruta corresponde al trayecto Medina de Pomar – Santander, que tiene una distancia total de unos **105 km** y está formada por **2.326 segmentos** a una media de distancia de **45 m por segmento**. En la Figura 36 - Características Ruta 4 se pueden observar diferentes características de la ruta en cuestión.



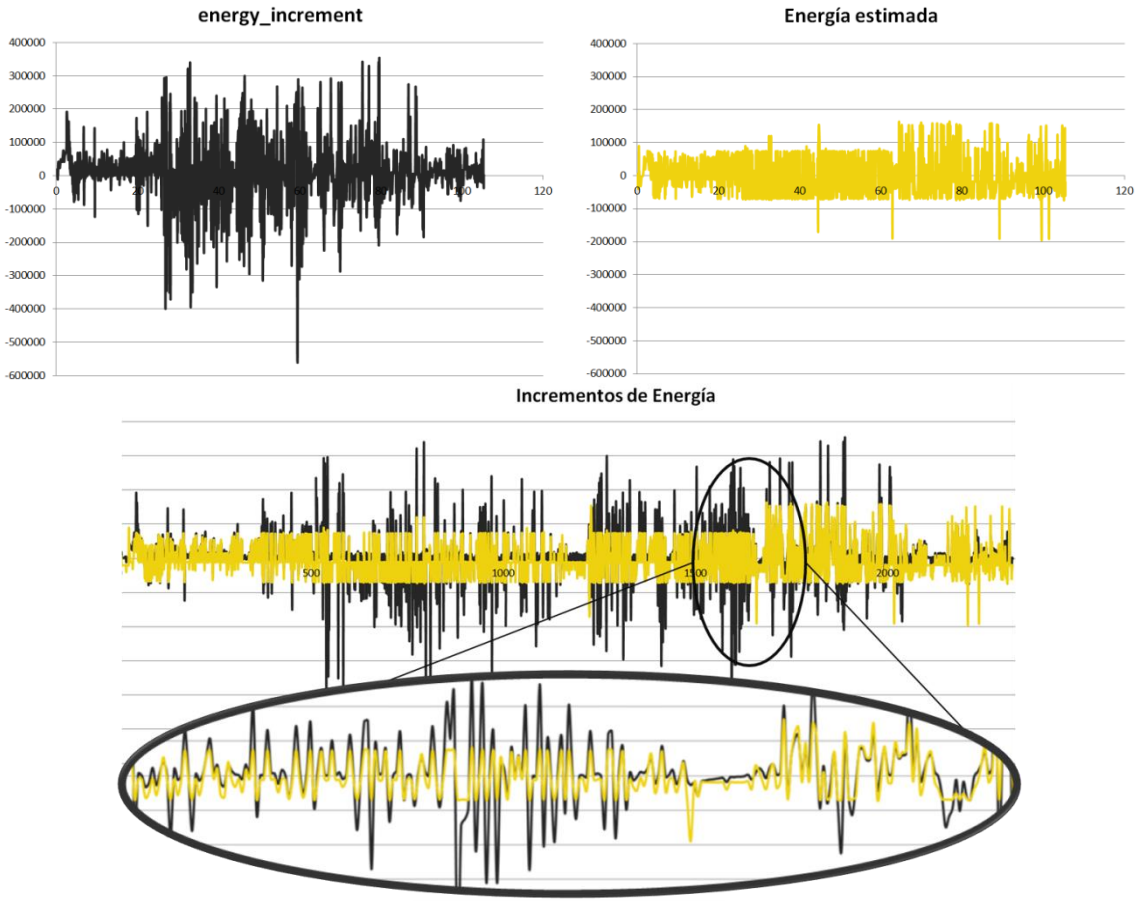
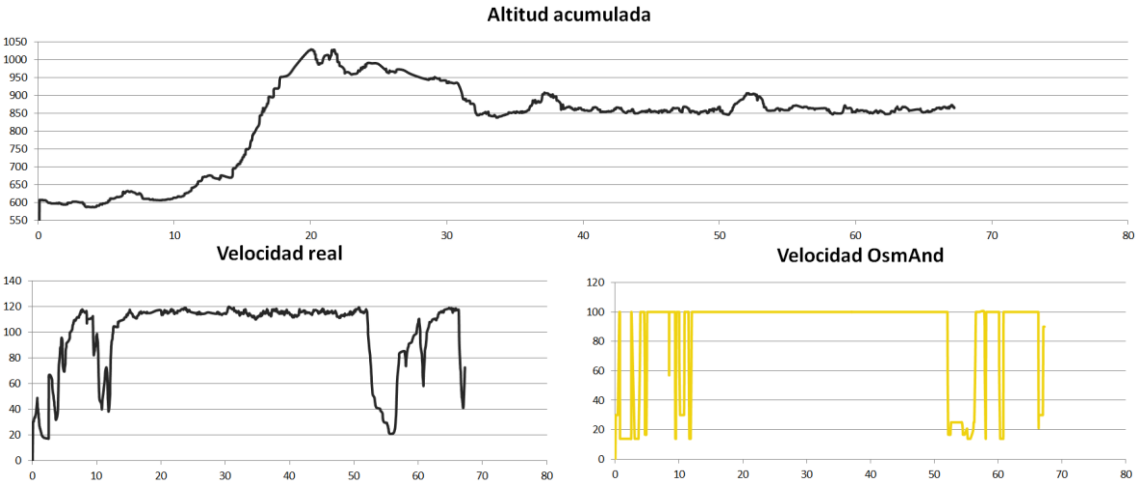


Figura 36 - Características Ruta 4

6.2.3.1.5 Ruta 5

La ruta corresponde al trayecto Medina de Pomar – Reinosa, que tiene una distancia total de unos **67 km** y está formada por **1.247 segmentos** a una media de distancia de **54 m por segmento**. En la Figura 37 - Características Ruta 5 se pueden observar diferentes características de la ruta en cuestión.



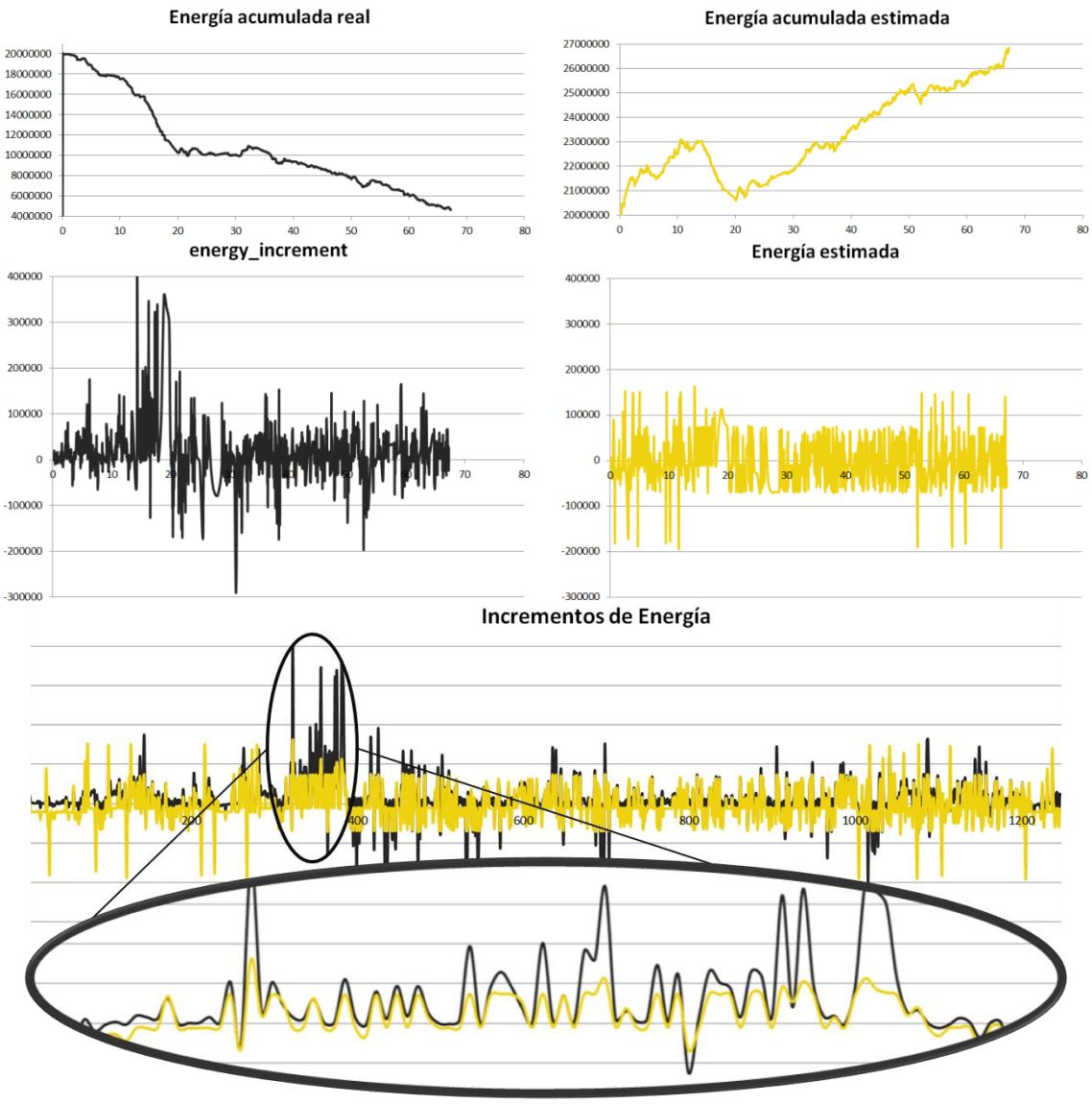


Figura 37 - Características Ruta 5

6.2.3.1.6 Ruta 6

La ruta corresponde al trayecto Medina de Pomar – Vitoria, que tiene una distancia total de unos **118 km** y está formada por **2.226 segmentos** a una media de distancia de **53 m por segmento**. En la Figura 38 - Características Ruta 6 se pueden observar diferentes características de la ruta en cuestión.

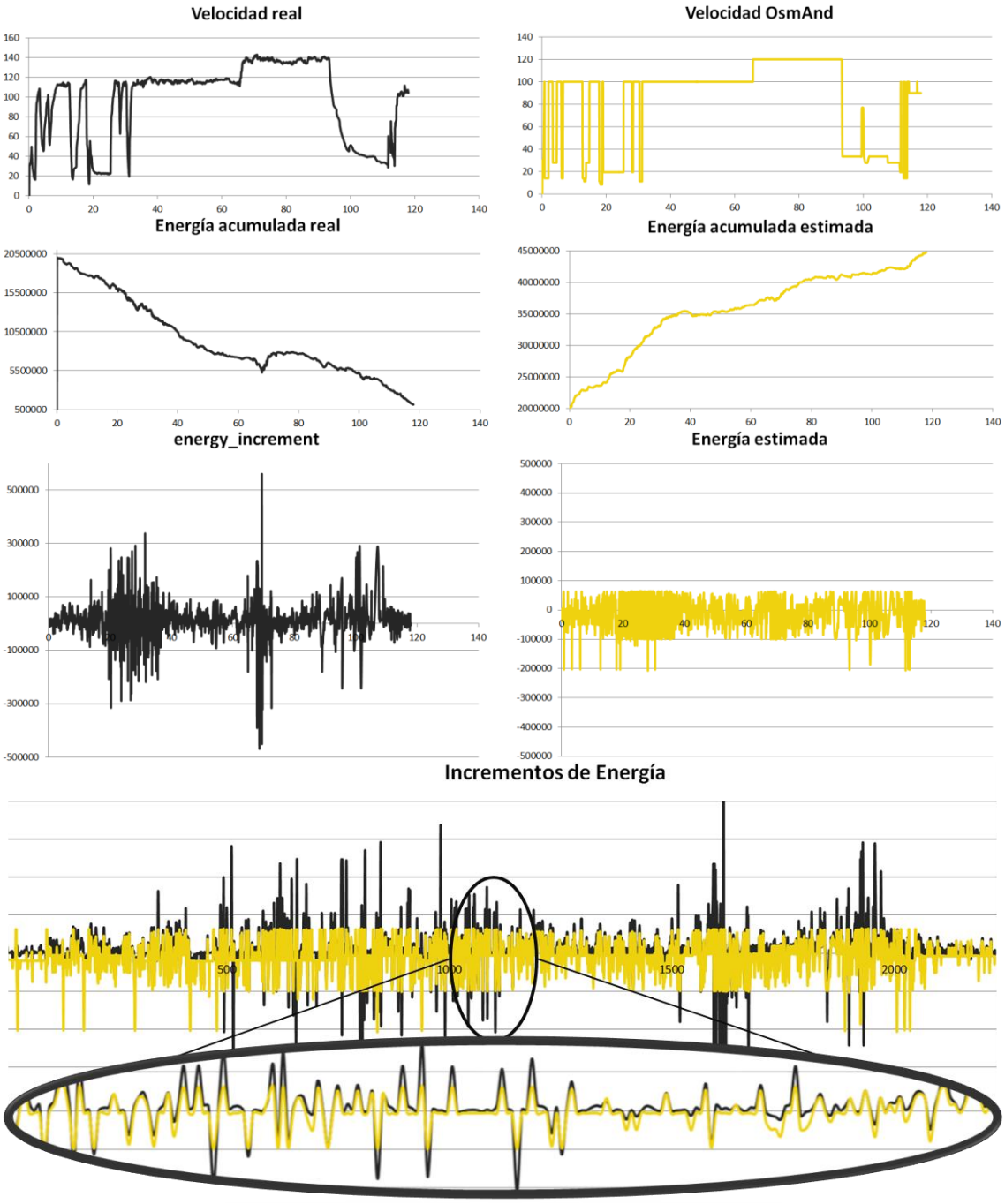


Figura 38 - Características Ruta 6

6.2.3.1.7 Ruta 7

La ruta corresponde al trayecto Medina de Pomar – Burgos, que tiene una distancia total de unos **95 km** y está formada por **2.760 segmentos** a una media de distancia de **34 m por segmento**. En la Figura 39 - Características Ruta 7 se pueden observar diferentes características de la ruta en cuestión.

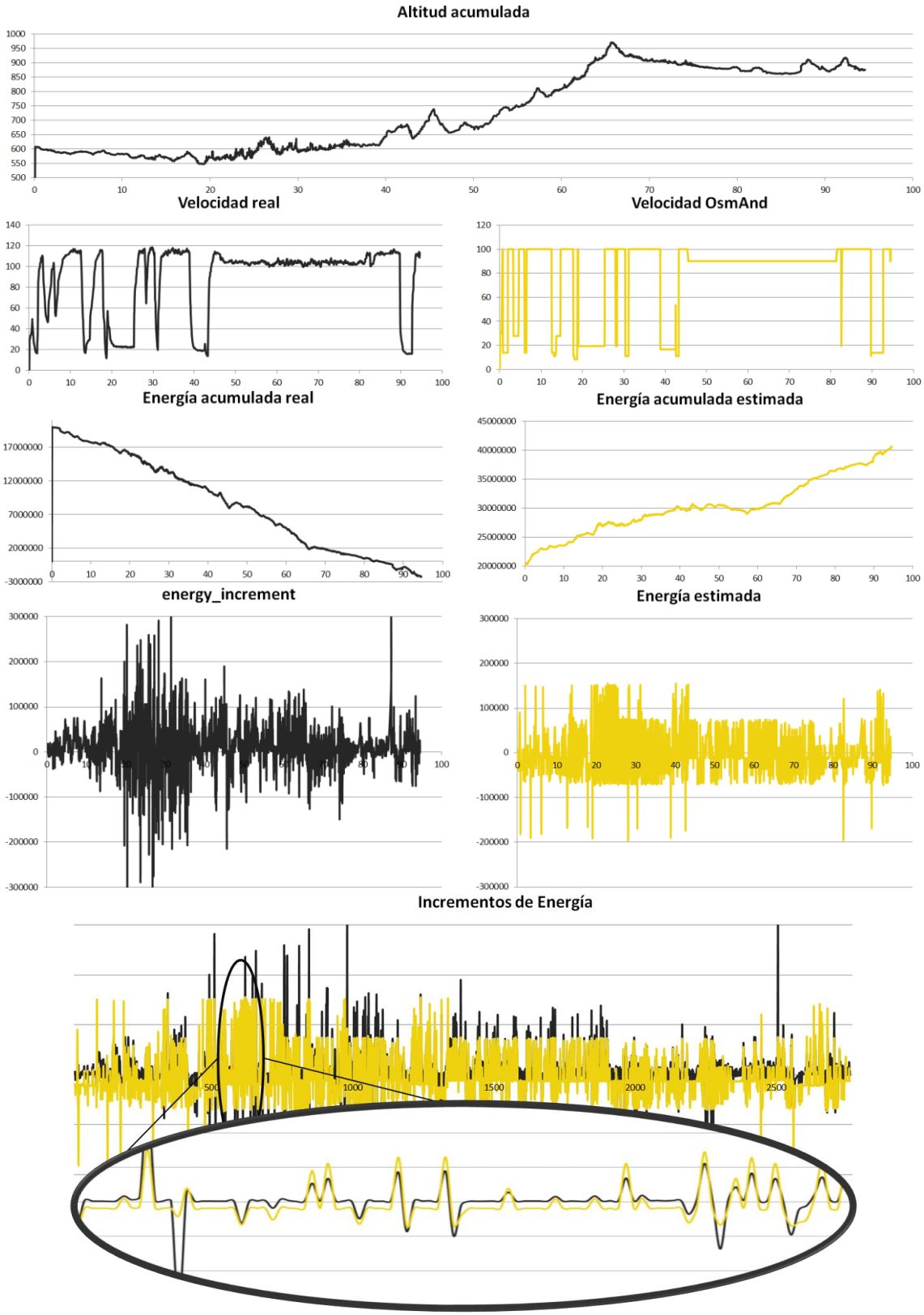


Figura 39 - Características Ruta 7

6.2.3.1.8 Análisis de los resultados

Como se puede ver en las diferentes rutas, **no se ha conseguido modelar un sistema que permita estimar la energía que se consumirá en una ruta** (a falta de más pruebas y modificaciones).

Sin embargo se puede comprobar que la Red Neuronal funciona bastante bien. Con dichas rutas en memoria, se han realizada hasta 10 entrenamientos, quedando el coeficiente de correlación lineal de todos ellos en torno a 0.75 (0.75 ± 0.002). En las pruebas anteriores el modelo de Red Neuronal generado tenía un coeficiente de $\boldsymbol{r = 0.759}$.

Además, como se puede comprobar en la última gráfica de cada apartado, donde se ve la función que intenta aproximar el modelo, éste lo hace muy bien. No aproxima del todo los picos de dicha función, pero eso es debido a que éstos son menos comunes y por tanto la Red no se entrena con mucha cantidad de éstos, adaptándose mejor a los más comunes.

Dicho comportamiento se ha conseguido alcanzar tras realizar algunas **modificaciones** en el diseño inicial de la Red Neuronal. En concreto, se ha añadido una quinta entrada cuyo valor es siempre 1 y cuya función es **introducir un *offset*** a la función (una *ordenada* en el origen). Además, se ha cambiado la **función de activación** por una **tangente hiperbólica**, esto se ha añadido para adaptarse mejor a las características del problema, ya que el incremento podía ser negativo cuando se regenerase energía en un segmento. Puesto que se ha añadido una neurona más de entrada, también se ha añadido una **neurona oculta más**, haciendo un total de cinco. Estas decisiones se han tomado teniendo en cuenta otras pruebas iniciales que se realizaron, y que ofrecían resultado como el de la Figura 40 - Estimación de energía RN (configuración inicial), donde se puede ver que la Red Neuronal no aproximaba los picos negativos.

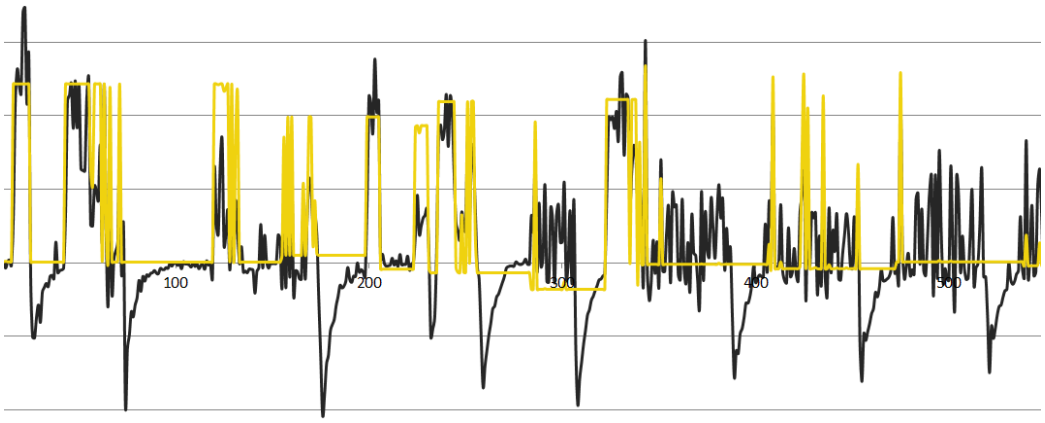


Figura 40 - Estimación de energía RN (configuración inicial)

No obstante, a pesar de que dicho comportamiento sea bueno, se puede ver que el resultado que interesa, la **energía total consumida**, no sólo es que no se aproxime a la función buscada, sino que ofrece en todos los casos una **tendencia ascendente** (como si el vehículo generase energía en lugar de consumirla) por lo que **el sistema no sería válido** para el propósito deseado (al menos en su estado actual a falta de realizar alguna prueba más).

6.2.3.2 Prueba 2

En esta ocasión, se modificará la velocidad que proporciona OsmAnd pasándolo por el mismo filtro que se emplea para simulación. De esta forma se suavizará la forma de la gráfica de variación de velocidad.

En la Figura 41 - Estimación de energía RN (sin filtrado previo de velocidad vs con él) se puede ver una comparación de los incrementos de energía del caso anterior y el actual para el caso de la primera ruta analizada.

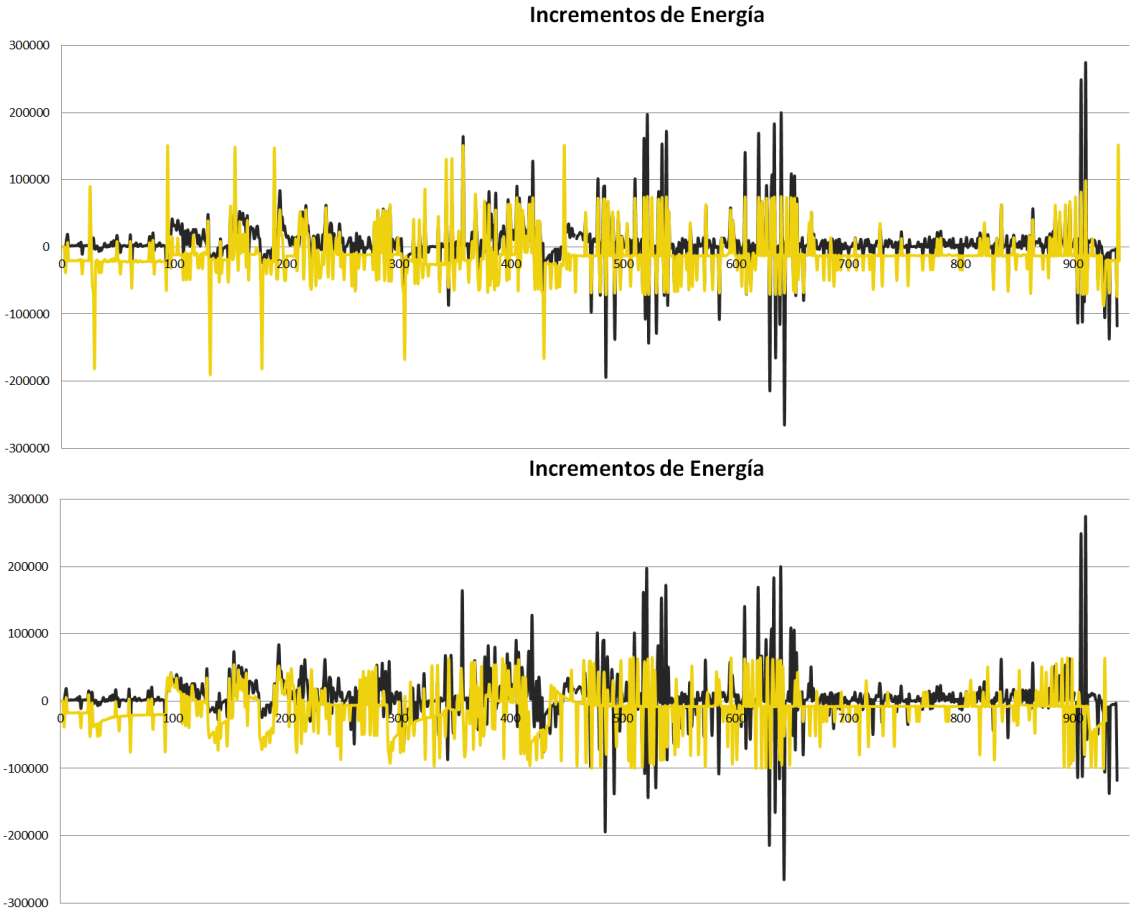


Figura 41 - Estimación de energía RN (sin filtrado previo de velocidad vs con él)

Como se puede observar, el comportamiento de la variable estimada es mucho mejor con este suavizado (se eliminan importantes picos en la gráfica). No obstante, lo importante es la energía acumulada (el gasto total), el cual se puede ver en la Figura 42 - Estimación de energía RN acumulada (sin filtrado previo de velocidad vs con él). Donde se puede distinguir la curva de antes en negro y la generada ahora en amarillo. Como se puede ver, los resultados no son mejores.

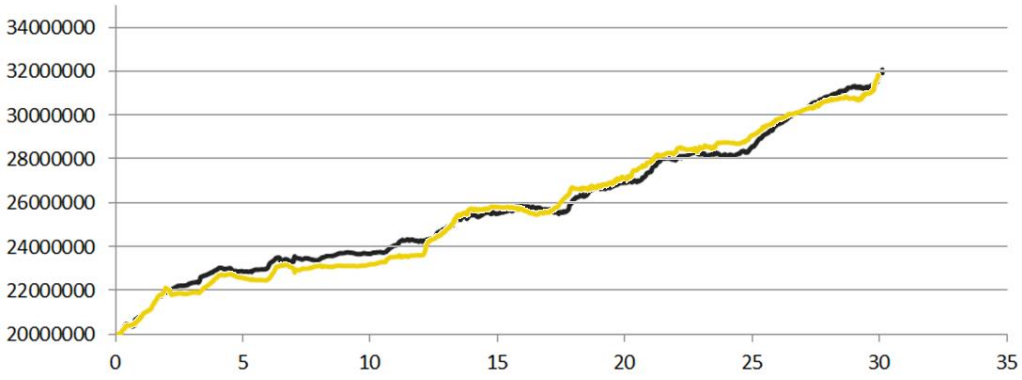


Figura 42 - Estimación de energía RN acumulada (sin filtrado previo de velocidad vs con él)

6.2.3.3 Prueba 3

Atendiendo a la Figura 41 - Estimación de energía RN (sin filtrado previo de velocidad vs con él) se puede observar que, si bien la forma de la gráfica se estima bastante bien, existe un *offset* que hace que la función estimada esté la mayoría de las veces por debajo de la original (sobre todo cuando las variaciones son pequeñas). Si tomamos un intervalo más pequeño de dicha gráfica se puede observar dicho efecto mejor, como se puede ver en la Figura 43 - Estimación energía RN (ampliado).

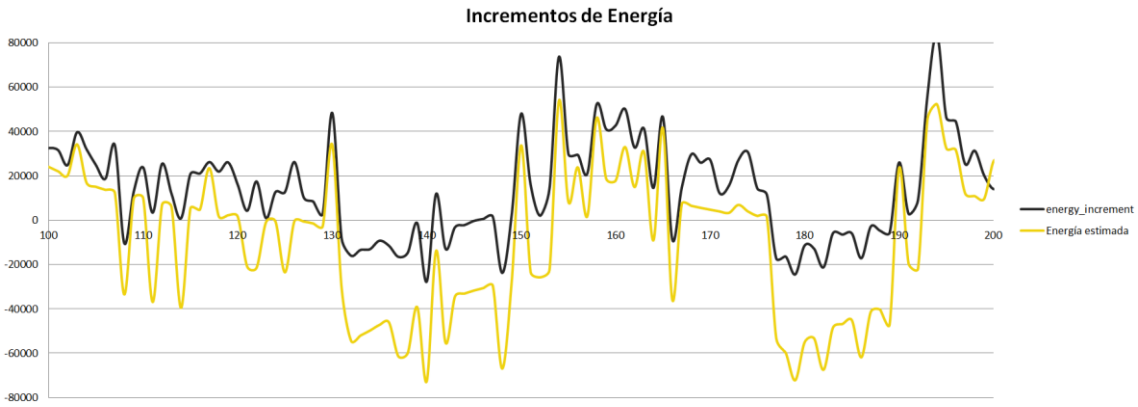


Figura 43 - Estimación energía RN (ampliado)

En este caso la Red Neuronal no ha sido capaz de extraer la información del *offset* como se esperaba. Para comprobar si es este hecho el que hace que la gráfica de energía acumulada presente este efecto, se realizará un análisis de los mismos parámetros pero añadiendo un *offset* de forma manual a los resultados analizados.

Para este caso, que corresponde con la Ruta 1: Medina de Pomar – Frías, se ha añadido un ***offset* de 19 kJ** (a partir de la observación de la figura anterior). La energía acumulada obtenida en este caso se puede ver en la Figura 44 - Energía estimada RN acumulada Ruta 1 (añadiendo offset), la cual muestra una variación mucho menor entre una y otra gráfica (además de que ahora la tendencia es descendente). En color negro se puede ver la gráfica original analizada en la Prueba 1, mientras que en color amarillo se puede ver la modificación de la gráfica anterior añadiéndole el *offset* mencionado.

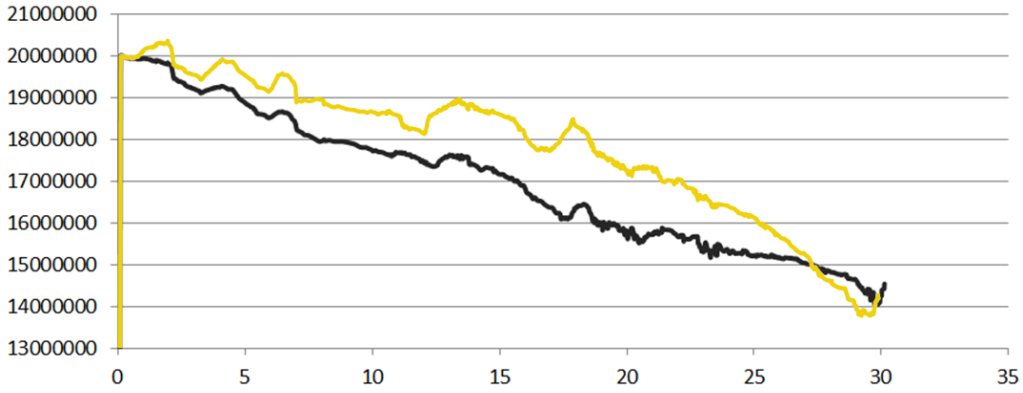


Figura 44 - Energía estimada RN acumulada Ruta 1 (añadiendo offset)

6.2.3.4 Prueba 4

En base a la prueba anterior, se realizará una comparación con todas las rutas añadiendo el mencionado *offset* a los valores de incremento de energía estimados. El *offset* añadido dependerá de cada ruta en particular y por ello este método no es susceptible de implementación (al no ser posible sistematizarlo). Sin embargo se realizará con el objetivo de entender mejor el problema abordado.

A continuación se representarán las gráficas correspondientes a la energía del vehículo durante el recorrido. En negro se representará la energía real consumida (simulada), mientras que en amarillo se representará la energía consumida estimada por la Red Neuronal, añadiendo un *offset* a las estimaciones de cada incremento de energía (el mismo a cada uno de ellos). Como la Ruta 1 ya se ha representado únicamente se pintarán las 6 restantes.

6.2.3.4.1 Ruta 2

En la Figura 45 - Energía estimada RN acumulada Ruta 2 (añadiendo offset) se puede ver el consumo de energía en la ruta Medina de Pomar – Espinosa de los Monteros (20 km) con un ***offset*** **de 16kJ**.

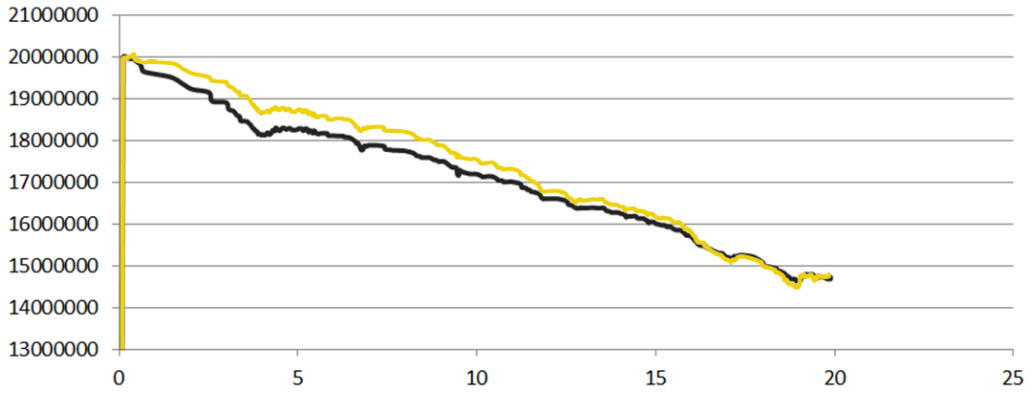


Figura 45 - Energía estimada RN acumulada Ruta 2 (añadiendo offset)

6.2.3.4.2 Ruta 3

En la Figura 46 - Energía estimada RN acumulada Ruta 3 (añadiendo offset) se puede ver el consumo de energía en la ruta Medina de Pomar – Bilbao (81 km) con un ***offset*** **de 13 kJ**.

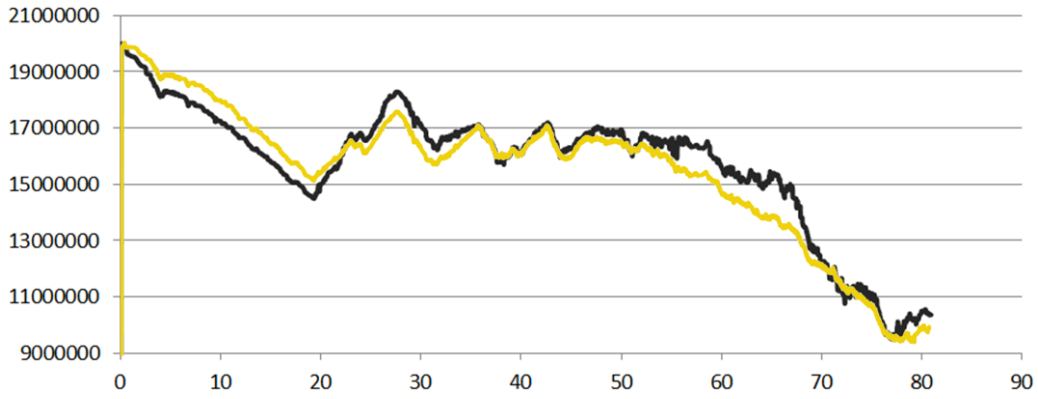


Figura 46 - Energía estimada RN acumulada Ruta 3 (añadiendo offset)

6.2.3.4.3 Ruta 4

En la Figura 47 - Energía estimada RN acumulada Ruta 4 (añadiendo offset) se puede ver el consumo de energía en la ruta Medina de Pomar – Santander (105 km) con un ***offset*** **de 12 kJ**.

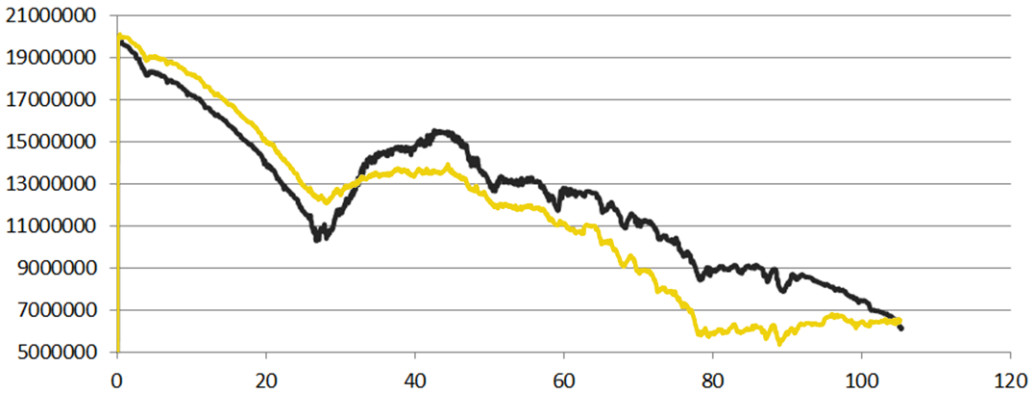


Figura 47 - Energía estimada RN acumulada Ruta 4 (añadiendo offset)

6.2.3.4.4 Ruta 5

En la Figura 48 - Energía estimada RN acumulada Ruta 5 (añadiendo offset) se puede ver el consumo de energía en la ruta Medina de Pomar – Reinosa (67 km) con un ***offset*** **de 18 kJ**.

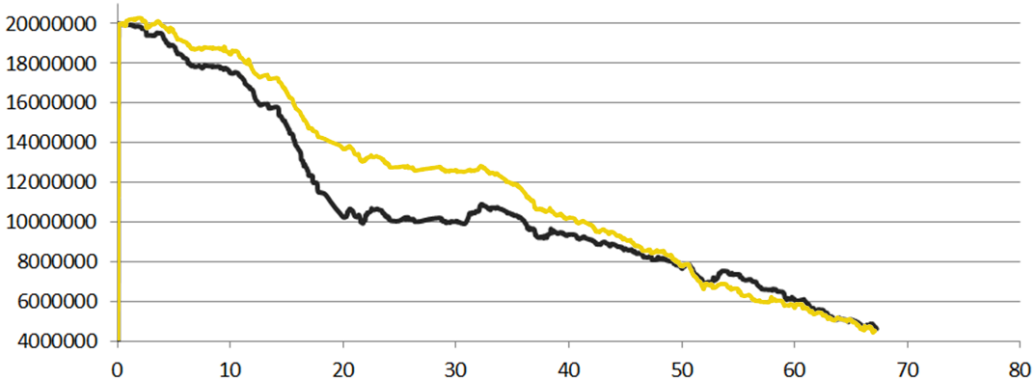


Figura 48 - Energía estimada RN acumulada Ruta 5 (añadiendo offset)

6.2.3.4.5 Ruta 6

En la Figura 49 - Energía estimada RN acumulada Ruta 6 (añadiendo offset) se puede ver el consumo de energía en la ruta Medina de Pomar – Vitoria (118 km) con un ***offset*** **de 20 kJ**.

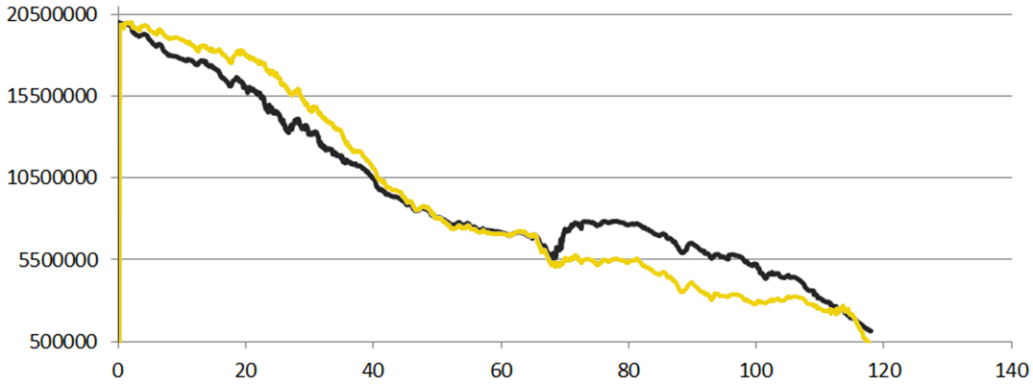


Figura 49 - Energía estimada RN acumulada Ruta 6 (añadiendo offset)

6.2.3.4.6 Ruta 7

En la Figura 50 - Energía estimada RN acumulada Ruta 7 (añadiendo offset) se puede ver el consumo de energía en la ruta Medina de Pomar – Burgos (95 km) con un ***offset*** **de 15.5 kJ**.

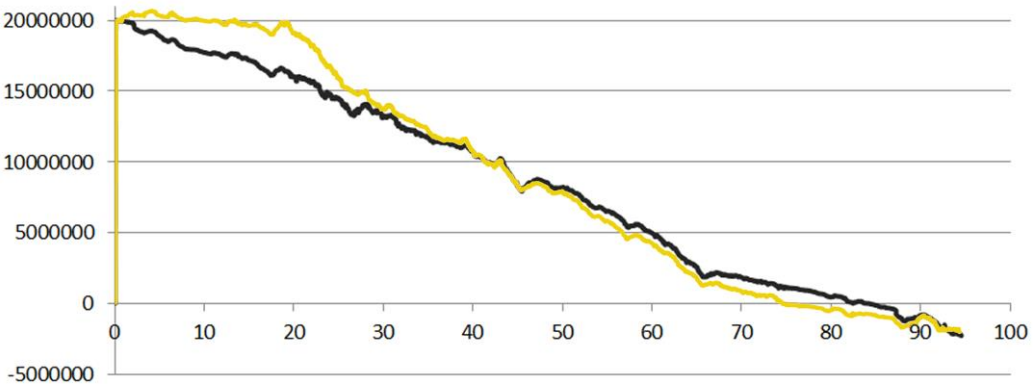


Figura 50 - Energía estimada RN acumulada Ruta 7 (añadiendo offset)

6.2.3.4.7 Análisis de los resultados

Como se puede intuir de la vista de las 7 gráficas anteriores, el problema evidentemente radica en mejorar el modelo de la Red Neuronal para que el comentado *offset* no genere tales incongruencias en la energía final consumida en una ruta. En dichas gráficas se puede ver cómo añadiendo una cantidad fija a cada predicción de consumo de energía de cada segmento de la ruta, se consigue un resultado muy bueno en cuanto a la estimación del consumo total de dicha ruta.

La clave reside entonces en encontrar la manera de que el sistema no dependa de dicha imprecisión, puesto que lo hace totalmente inútil para el propósito requerido. Es decir, ¿por qué la Red Neuronal da siempre una estimación de consumo de energía menor?

Para resolver dicho problema se podrían tomar **dos caminos diferentes**: por un lado intentar **mejorar el entrenamiento de la Red Neuronal** para que el modelo que genere sea por sí mismo suficiente para calcular el consumo en una ruta; por el otro lado se podría buscar alguna forma de poder **caracterizar sistemáticamente el *offset*** que le falta a las estimaciones realizadas por la Red Neuronal y sumarlo al resultado final, obteniendo un resultado similar al que se ha visto en este apartado, pero sin tener que probar diferentes

valores de dicho *offset* y ver cuál es el mejor (puesto que en un escenario real no se dispondrá de la gráfica de consumo real para comparar).

En las siguientes tablas se puede ver un resumen de las características de los incrementos de energía, tanto simulados como estimados. En la Tabla 5 - Suma total incrementos de energía se muestra la suma total de éstos según la categoría marcada. Así mismo, en la Tabla 6 - Media del total de incrementos de energía se muestra la media de dichos segmentos.

Suma	Offset óptimo (J)	ΔE	ΔE+	ΔE-	ΔE (simulado)	ΔE+ (simulado)	ΔE- (simulado)
Ruta 1	19.000	5.463.937	11.712.035	-6.248.098	-11.928.409	6.876.093	-18.804.502
Ruta 2	16.000	5.276.486	6.738.988	-1.462.502	-1.153.860	4.719.821	-5.873.682
Ruta 3	13.000	9.648.105	44.498.206	-34.850.101	-18.938.194	23.997.406	-42.935.601
Ruta 4	12.000	13.872.174	52.486.888	-38.614.714	-14.406.675	33.680.702	-48.087.377
Ruta 5	18.000	15.413.796	25.478.740	-10.064.943	-6.868.141	16.978.760	-23.846.902
Ruta 6	20.000	18.867.113	40.999.915	-22.132.801	-24.790.006	21.054.269	-45.844.276
Ruta 7	15.500	22.306.705	41.758.556	-19.451.850	-20.663.529	30.390.645,61	-51.054.175

Tabla 5 - Suma total incrementos de energía

Media	Offset óptimo	ΔE	ΔE+	ΔE-	ΔE (simulado)	ΔE+ (simulado)	ΔE- (simulado)
Ruta 1	19.000	5.839	17.820	-22.555	-12.689	46.148	-23.803
Ruta 2	16.000	13.249	22.229	-15.558	-2.899	39.662	-21.128
Ruta 3	13.000	4.409	30.944	-46.529	-8.507	55.293	-23.972
Ruta 4	12.000	6.033	32.762	-55.479	-6.196	60.036	-27.275
Ruta 5	18.000	12.370	29.083	-27.272	-5.512	50.084	-26.321
Ruta 6	20.000	8.563	25.882	-35.812	-11.141	37.529	-27.567
Ruta 7	15.500	8.143	20.499	-27.751	-7.489	55.356	-23.111

Tabla 6 - Media del total de incrementos de energía

En base a dichas tablas se puede hacer un análisis para comprobar la correlación de los datos que se disponen, es decir los datos de simulación, con el *offset* óptimo. De esta forma se podrá ver si se puede calcular dicho *offset* en base a la estimación realizada y a los parámetros mostrados en las tablas. En la Figura 51 - Correlación valores medios vs offset y en la Figura 52 - Correlación valores totales vs offset se puede ver dicho análisis.

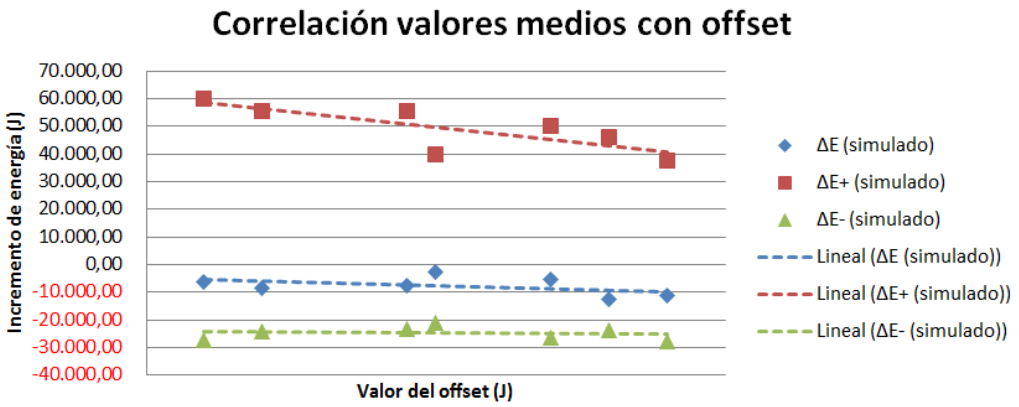


Figura 51 - Correlación valores medios vs offset

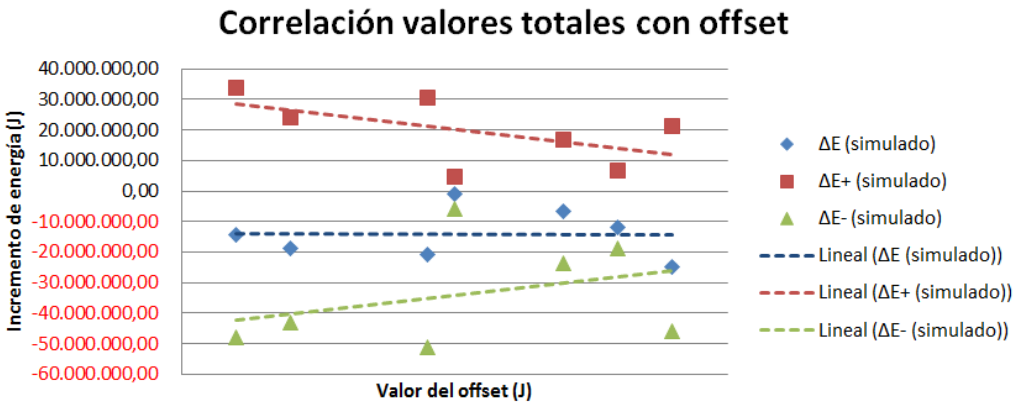


Figura 52 - Correlación valores totales vs offset

En dichos gráficos se puede comprobar que ciertamente sí que **existe cierta correlación** entre los valores medios y el *offset*, sobre todo en los valores medios de los incrementos de energía negativos. Se podría establecer por tanto una relación lineal que modele el *offset* en función del valor medio de los incrementos negativos. En la Figura 53 - Correlación incrementos de energía negativos vs offset se puede ver la representación de la correlación entre el *offset* y el valor medio de los incrementos negativos a una escala mayor.

Además de la aproximación por regresión lineal, se ha añadido una aproximación polinómica de orden 2. Al lado de cada línea se puede ver el coeficiente de correlación de *Pearson* al cuadrado. Es decir, una aproximación lineal de dichos datos tendría un coeficiente de correlación $r = 0.108$, mientras que una aproximación mediante un polinomio de orden 2 tendría una correlación $r = 0.808$. Es decir, la aproximación lineal no sirve para definir la correlación entre ambas variables. Por el contrario, la aproximación mediante el polinomio de orden 2 sí que aproximaría lo suficiente la relación entre dichas variables. No obstante, hay que tener en cuenta que el número de rutas es muy pequeño como para poder determinar una relación fija como la representada en el gráfico.

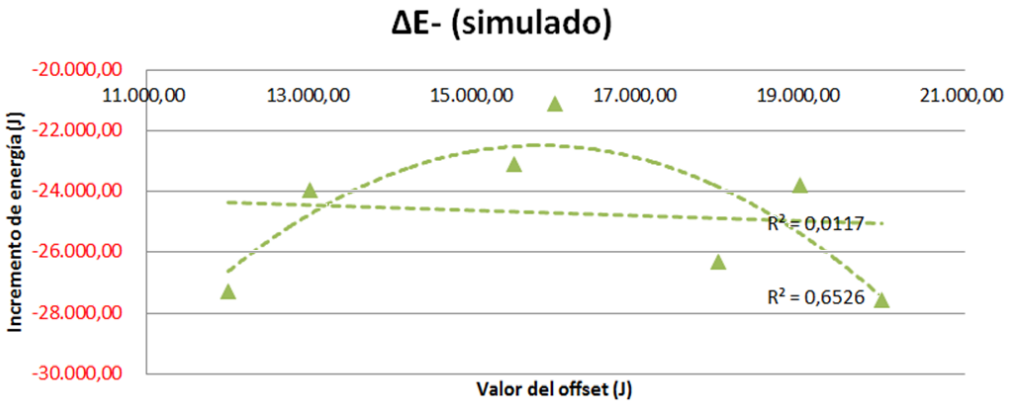


Figura 53 - Correlación incrementos de energía negativos vs offset

A partir de los datos analizados, se concluye que **no existe forma alguna de determinar el *offset*** necesario de cada ruta en base a los parámetros estimados por la Red Neuronal, y por tanto **será necesario evaluar otras soluciones** para resolver el problema.

6.2.3.5 Prueba 5

Como se ha visto hasta ahora, las imprecisiones de las estimaciones de cada incremento de energía no permiten realizar una estimación óptima de la energía total. Así mismo, cada una de esas estimaciones puntuales proporciona un valor sistemáticamente menor (el *offset* comentado en el apartado anterior).

Sin embargo, como se ha comentado anteriormente, la curva que representa los incrementos de energía sí que se aproxima bastante a la original (exceptuando el comentado *offset*). Teniendo esto en cuenta, se podrían utilizar dichas estimaciones de incrementos de energía conforme el usuario avanza sobre la ruta, realizando estimaciones del consumo total en base a la energía que realmente tiene el vehículo y no en base a la estimación del segmento predecesor.

Para dicha tarea habría que tomar un valor de *offset* fijo (al menos para el modelo que la Red Neuronal ha creado en este caso). Se podría tomar el valor medio de los *offset* aplicados anteriormente para hacer una pequeña prueba del funcionamiento con dicha configuración. Dicho valor medio es de 16 kJ.

En la Figura 54 - Energía estimada RN con cálculo periódico (final) se puede observar el comportamiento que tendría este tipo de sistema para la Ruta 1, realizando estimaciones de consumo cada 5 km. En color negro se puede observar el consumo real (simulado), mientras que el color amarillo representa la estimación realizada por el sistema, la cual se actualizará cada 5 km (de ahí los saltos que se observan).

Si no se realizan dichas estimaciones periódicas y se mantiene la estimación en el origen, la diferencia de energía sería de 2.8 MJ menos. Es decir, estimaría una energía consumida para esta ruta de 3MJ mientras que según la gráfica original la energía consumida sería de 5.8 MJ. En la Figura 55 - Energía estimada RN con cálculo periódico se pueden ver las estimaciones realizadas en cada uno de los puntos donde se re-calcula.

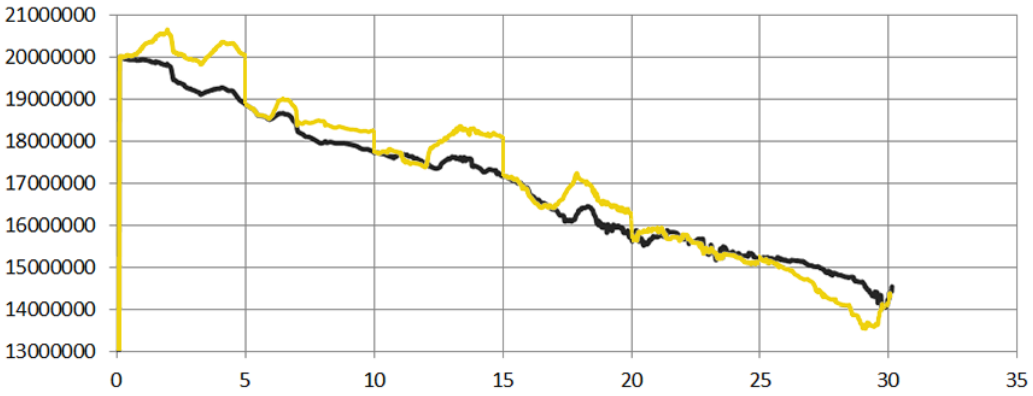


Figura 54 - Energía estimada RN con cálculo periódico (final)

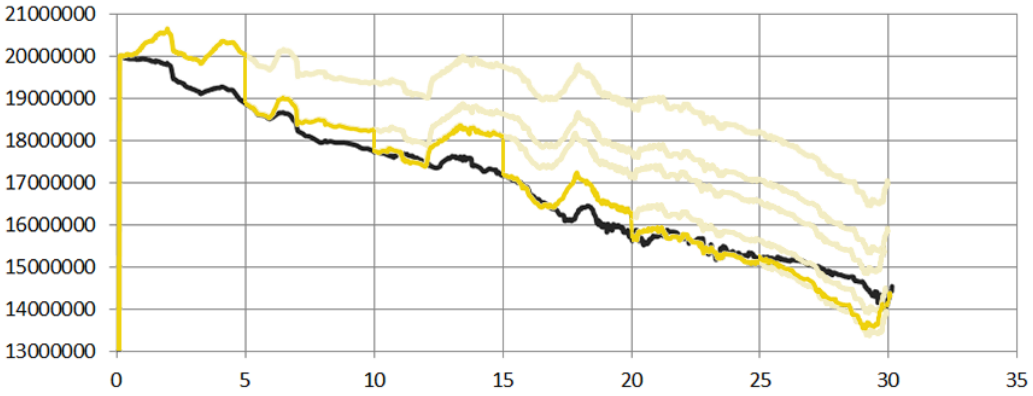


Figura 55 - Energía estimada RN con cálculo periódico

Como se puede observar, el resultado mejora considerablemente, sobre todo si las estimaciones periódicas se realizan con mayor frecuencia (cada 1 km por ejemplo). Aunque la imprecisión de la predicción al comienzo de la ruta puede no ser algo aceptable por el usuario.

Llegados a este punto queda analizar si esta implementación será suficiente o siquiera escalable (ya que se ha generalizado el *offset* de 16 MJ en base a la media de unas pocas rutas). La pregunta entonces sería ¿qué tipo de estimación es más precisa, ésta o una basada en el histórico de consumo como ha venido haciendo tradicionalmente la industria de la automoción?

6.2.3.6 Prueba 6

Con el objetivo de evaluar si el sistema comentado en el apartado anterior ofrece algún tipo de ventaja con respecto a las estimaciones que usualmente se realizan en la industria será necesario realizar alguna prueba adicional a las anteriores.

Tras un análisis de la energía total consumida analizada en la Prueba 1, y obteniendo su aproximación lineal equivalente se obtienen los resultados mostrados en la Tabla 7 - Aproximación lineal del consumo de energía. En dicha tabla se representa también la energía estimada en el inicio de la ruta por el sistema presentado en el apartado anterior, es decir con un *offset* impuesto de 16 kJ.

	Consumo medio por ruta	Consumo estimado (aprox. lineal)	Consumo estimado (RN)	Consumo real
Ruta 1	5.5 MJ / 30 km = 183 kJ/km	176 kJ/km · 30 km = 5.28 MJ	3 MJ	5.5 MJ
Ruta 2	5.3 MJ / 20 km = 265 kJ/km	3.52 MJ	5.2 MJ	5.3 MJ
Ruta 3	9.6 MJ / 81 km = 119 kJ/km	14.26 MJ	16.66 MJ	9.6 MJ
Ruta 4	13.9 MJ / 105 km = 132 kJ/km	18.48 MJ	22.78 MJ	13.9 MJ
Ruta 5	15.4 MJ / 67 km = 230 kJ/km	11.79 MJ	13.05 MJ	15.4 MJ
Ruta 6	18.9 MJ / 118 km = 160 kJ/km	20.77 MJ	10.8 MJ	18.9 MJ
Ruta 7	22.3 MJ / 95 km = 235 kJ/km	16.72 MJ	23.46 MJ	22.3 MJ
Media	**176 kJ/km**			

Tabla 7 - Aproximación lineal del consumo de energía

En la Figura 56 - Correlación de estimación lineal y estimación RN vs consumo real se representa un gráfico de dispersión para comprobar la correlación entre las dos magnitudes estimadas y la energía consumida real. En dicha representación se puede comprobar que, al menos con los datos disponibles, tiene un coeficiente de correlación mucho mejor la **estimación lineal**, con un valor de $\boldsymbol{r = 0.817}$; que la **estimación mediante la Red Neuronal** (ajustando el *offset* a un valor determinado), con un valor de $\boldsymbol{r = 0.692}$.

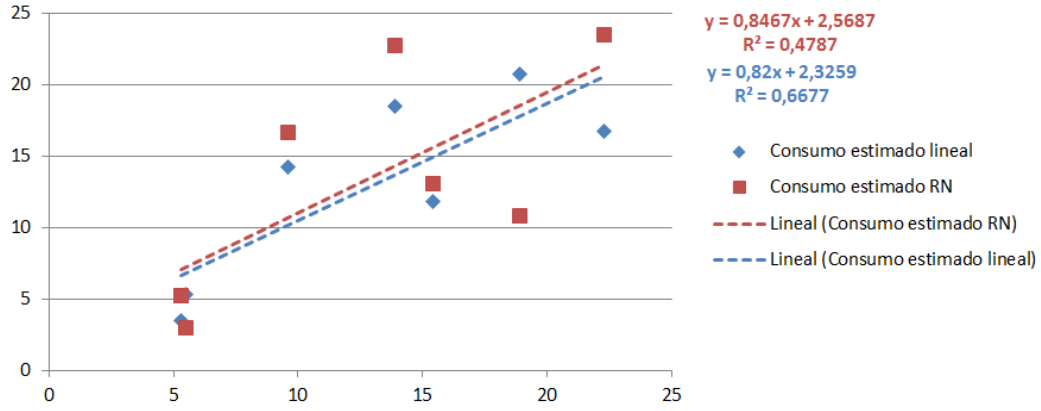


Figura 56 - Correlación de estimación lineal y estimación RN vs consumo real

Tras este pequeño análisis, se determina que una estimación lineal del consumo basado en el histórico de consumo del vehículo será mucho más precisa que la estimación realizada por el modelo creado con Redes Neuronales (a menos que se consiga generar un modelo en el cual el comentado *offset* no sea necesario, en cuyo caso la estimación sería prácticamente perfecta).

6.3 Comunicación Bluetooth

Además de las pruebas relativas al sistema de inteligencia artificial, también se ha realizado alguna **prueba de la comunicación Bluetooth**. Dichas pruebas se realizaron con la centralita que supuestamente llevaría instalada la motocicleta y a través de la cual se recibirán las tramas de telemetría y GPS al dispositivo móvil.

Se ha realizado en primer lugar una prueba para comprobar que el dispositivo móvil se podía conectar a dicha centralita. Para ello, cada vez que se realizaba la conexión el móvil enviaba un comando para encender un LED en la centralita. De igual modo, se enviaba un comando para apagar dicho LED de forma previa a la desconexión.

Una vez comprobado que el comportamiento en la conexión y desconexión es correcto, se ha comprobado que el sistema de petición de telemetría funciona, asegurando que los datos se recibían correctamente conforme a las especificaciones (ver Anexo 9.2).

El conjunto de pruebas realizadas ha tenido un resultado satisfactorio en todos los casos analizados, por lo que queda comprobado el **correcto funcionamiento de la comunicación Bluetooth**.

6.4 Conclusión

Los resultados derivados de las diferentes pruebas realizadas en el sistema de estimación basado en Redes Neuronales, esto es, el Modelo de Predicción de consumo, aportan como resultado la **inviabilidad de dicha técnica para modelar el comportamiento del vehículo**. Dicha conclusión está basada en las pruebas realizadas, lo que no implica que mediante otro tipo de pruebas o técnicas de análisis se pueda llegar a depurar el sistema de tal forma que sirva para el modelado del consumo del vehículo.

Como se ha comentado anteriormente, el trabajo se debería focalizar en buscar formas de entrenar la Red Neuronal que permitan ajustar mucho más la estimación que ésta realice, de tal forma que al estimar mejor los incrementos de energía, la energía total acumulada (la energía total que consumirá el vehículo para una determinada ruta) se estime con mucha mayor precisión. **El objetivo por tanto será que la Red Neuronal sea capaz de crear un modelo en el cual no exista el comentado *offset*** que hace que las estimaciones sean erróneas. Si se consiguiera entrenar el sistema de tal forma que no exista dicha desviación, se conseguiría un sistema de estimación prácticamente perfecto como se ha visto en las pruebas de concepto realizadas en la Prueba 4.

7 Conclusiones

7.1 Introducción

Llegado a este punto es interesante realizar una cierta reflexión acerca de lo conseguido con este proyecto. En concreto se analizarán los **objetivos logrados** con respecto a la planificación original, así como el conjunto de **conocimientos y/o técnicas aprendidas** a lo largo del desarrollo del proyecto. Se dará una visión cuantitativa del **trabajo realizado** (horas invertidas, líneas de código del programa, etc.) así como un análisis de las posibles **líneas de trabajo a seguir** para mejorar el sistema implementado.

7.2 Objetivos logrados

A lo largo de la realización de este proyecto, desde las fases iniciales de análisis y diseño, hasta la conclusión del mismo mediante la realización de diferentes pruebas, se han alcanzado ciertos hitos que es necesario remarcar.

En primer lugar, se ha conseguido construir una **aplicación para dispositivos Android** integrada perfectamente con un sistema de GPS (visionado de mapas y navegación paso a paso) a través del proyecto *open-source* OsmAnd. Esto ha sido posible gracias al uso del sistema de control de versiones *Git* así como herramientas web como *GitHub* (donde se aloja el código fuente de OsmAnd) o *BitBucket* (donde se aloja este proyecto). No obstante, a pesar de utilizar dicho sistema, la dinámica de desarrollo no hubiera sido posible sin adoptar la metodología de gestión de código *GitFlow*.

Además de ello, a dicha estructura básica se han añadido diferentes elementos tanto a nivel de interfaz como de lógica del sistema y modelo de datos. Se ha conseguido así un sistema altamente escalable y fácil de mantener a través de la utilización de la **metodología MVC** (Modelo–Vista–Controlador). Dichos elementos se han construido en base a una librería externa al proyecto original de OsmAnd, de tal forma que la evolución y mantenimiento del software sea una tarea menos tediosa.

Como parte de dichas implementaciones adicionales se ha construido un **sistema de comunicación** a través de la tecnología inalámbrica **Bluetooth**. Se ha puesto especial interés en el rendimiento de esta parte del código para que la transmisión de datos entre el dispositivo y el vehículo no interfiera en la ejecución normal de la aplicación, utilizando hilos de ejecución en segundo plano para la gestión de la conexión y el envío de datos.

Este conjunto de características hacen del sistema un completo *display* de la dinámica de conducción, mostrando tanto características de la ruta como telemetrías del vehículo, todo en tiempo real.

No obstante, el tratamiento de la telemetría no termina ahí, sino que el sistema almacena las rutas realizadas por el usuario con sus correspondientes datos. De esta forma permite realizar consultas de dichas rutas y ver características como la velocidad media, la energía consumida, la energía regenerada, gráficos con la altimetría o la velocidad a lo largo de la ruta, etc.

Además de todo esto, se ha construido un **sistema de Inteligencia Artificial** denominado **Modelo de Predicción de Consumo** cuya función es precisamente esa, ser capaz de estimar/predecir el consumo de energía en una determinada ruta en base a la orografía de ésta, el tipo de carretera (velocidad permitida) y el tipo de conducción de cada piloto. Para ello se ha implementado un sistema de **Redes Neuronales** que es capaz de aprender de las rutas que el usuario ya ha realizado y crear así un modelo de consumo que permita realizar estimaciones en rutas que aún no ha realizado.

De esta forma, el usuario obtiene una información más precisa del consumo que se producirá en una determinada ruta y de si es viable que emprenda el viaje (se realiza una estimación tanto de la energía que se consumiría a la ida como de la energía conjunta en el viaje de ida y vuelta). Este punto es muy importante debido a la limitación de autonomía de los vehículos eléctricos, al menos hasta que el número de puntos de recarga sea suficientemente grande como para evitar que el piloto se preocupe por quedarse sin energía.

Para la realización de las pruebas del Modelo de Predicción de Consumo no se ha podido disponer de datos reales de la motocicleta de Bultaco, por lo que se ha creado un **sistema de simulación** que modela el comportamiento de un vehículo eléctrico (los datos utilizados son los correspondientes a un Tesla Roadster).

A pesar de que las pruebas realizadas no son 100% concluyentes ya que este tipo de tecnología depende mucho del entrenamiento realizado (datos etiquetados disponibles para realizar dicho entrenamiento, pre-selección en función de los parámetros característicos de éstas, etc), en las pruebas realizadas no se ha conseguido el objetivo buscado. Es decir, **no se ha conseguido modelar el comportamiento del vehículo** a través del entrenamiento de una Red Neuronal.

A pesar de que las **estimaciones de consumo de cada tramo o segmento** que conforma la ruta estaban muy correladas con las reales, éstas estaban **desplazadas cierto *offset* con respecto a las reales**. Dicho *offset* en cada una de las rutas era diferente y por lo tanto no se podía generalizar ni elaborar un algoritmo que minimizase su impacto.

Se ha intentado modelar de alguna forma dicho *offset* para así poder eliminarlo en cada estimación de energía por segmento, pero no se ha podido encontrar solución alguna.

En conclusión, **no se descarta el uso de Redes Neuronales** para poder estimar el consumo total en una ruta pero es **necesario realizar más pruebas** que determinen la manera óptima de entrenar el sistema para que las estimaciones individuales carezcan del comentado *offset* y se pueda estimar el consumo total de una ruta con garantías.

7.3 Aprendizaje personal

En el desarrollo del proyecto se han ido adquiriendo diferentes conocimientos, técnicas o habilidades que merece la pena comentar.

En primer lugar, se ha profundizado en gran medida en las técnicas de *Machine Learning* o aprendizaje máquina. En concreto las técnicas de aprendizaje supervisado como son las Redes Neuronales. No sólo se ha entendido su comportamiento, sino que se ha aprendido

cómo diseñarlas en base al propósito buscado. Además de ello, se ha experimentado cómo implementarlas y utilizarlas en un entorno real. Se ha asimilado de igual modo diferentes técnicas de evaluación de éstas y/o métodos para mejorar el tiempo de entrenamiento, evitar el sobreajuste o evaluar su comportamiento, entre otras.

Además de esto, se han ampliado los conocimientos sobre el desarrollo en la plataforma Android, incluidos aspectos como la implementación de conexiones Bluetooth o de información de geolocalización.

Como complemento a dicho trabajo de programación, se ha aprendido también a manejar herramientas de gestión de código como el sistema de control de versiones *Git* (junto con *Bitbucket*, *SourceTree* y *GitHub*), así como el flujo de trabajo *GitFlow*. El conjunto de estos conocimientos ha permitido realizar una gestión del proyecto eficiente tanto para el trabajo individual como en equipo.

Además de todo lo comentado, se han adquirido gran cantidad de conocimientos sobre el comportamiento del vehículo eléctrico, así como del modelado de la no-linealidad de las baterías a través de diferentes artículos de investigación que se han ido estudiando.

7.4 Trabajo realizado

En este apartado se pretende dar una visión más amplia del trabajo desarrollado a lo largo de la ejecución del proyecto, tratando de complementar el apartado 5 en el que se detallaba el desarrollo realizado.

Al inicio de este proyecto se tenían ciertos conocimientos básicos sobre las técnicas de aprendizaje máquina, adquiridos a través de alguna asignatura de la carrera. No obstante, dada las características del proyecto, era necesario profundizar en dicho conocimiento, por lo que se han utilizado diferentes fuentes bibliográficas para ello. Durante los meses de Octubre y Noviembre de 2013 se estuvo profundizando en dicha materia, llegando a implementar un Perceptrón Multicapa en Java para poder entender mejor este sistema (dicha implementación está disponible en *https://github.com/dsaiztc/MultilayerPerceptron*).

Además de ello, a pesar de que se disponía de una base teórica al respecto, ha sido necesario ampliar los conocimientos del protocolo Bluetooth, así como de su implementación en la plataforma Android. Para entender bien el funcionamiento del mismo, se implementó un sistema cliente-servidor con un dispositivo Android (cliente) y un PC (servidor), el cual se utilizó para realizar diferentes pruebas de rendimiento (además de para familiarizarse con el sistema). El código correspondiente a dicho desarrollo está disponible en *https://github.com/dsaiztc/AndroidBluetoothConnection*.

A la hora de abordar el diseño de la aplicación, se han estudiado diferentes metodologías, adoptando finalmente MVC. Así mismo, se ha puesto un especial interés por aprender a utilizar el sistema de control de versiones *Git* junto con las herramientas asociadas antes comentadas, ya que es un imprescindible en el desarrollo profesional de hoy en día.

Como resultado del trabajo, se han implementado unas 51 clases *java*, con un total aproximado de 7.000 líneas de código.

7.5 Futuras líneas de trabajo

A lo largo del ciclo de vida de este proyecto han ido surgiendo nuevas ideas que pudieran dar lugar a mejoras en el sistema o incluso a crear otros nuevos. En este apartado se tratará de resumir dichas ideas, por si pudieran servir de inspiración para otros proyectos o trabajos. Se tratará de dar una visión de las posibles líneas a seguir en cuanto al sistema de navegación, para posteriormente analizar posibles implementaciones a realizar en el sistema y así como otro tipo de ideas o alternativas.

7.5.1 Sistema de estimación

Como se ha comentado en apartados anteriores, es necesario **continuar con el trabajo de investigación de la Red Neuronal** para así poder identificar por qué se produce un *offset* en las estimaciones de energía de cada segmento de la ruta. Será necesario realizar más entrenamientos de la red (con datos reales) y así poder evaluar con más certeza si el sistema es válido para el objetivo propuesto.

Así mismo, también se podrían **explorar otros sistemas para realizar la estimación**. Se podrían probar diferentes configuraciones del Perceptrón Multi-capa, diferentes algoritmos de aprendizaje (como técnicas como *deep-learning*) u otro tipo de métodos máquina, como por ejemplo los Procesos Gaussianos.

Así mismo, se podrían probar otras técnicas como los llamados **métodos generativos** (o semianalíticos), tratando de modelar la función de densidad de probabilidad de la energía consumida en un tramo en base a la distancia de éste, su incremento de altitud y su velocidad media.

Se podría intentar utilizar un **modelo semilineal de regresión**, utilizando la fórmula del consumo de energía del vehículo eléctrico que se ha utilizado en este trabajo para realizar las simulaciones. Para esto obviamente sería necesario disponer de datos reales de la motocicleta para construir dicho modelo.

Como complemento a esto, se podría utilizar algún tipo de **sistema inteligente para adaptar los parámetros utilizados para realizar la estimación**. Se plantea este escenario puesto que dependiendo del tipo de usuario, circulará a la velocidad que marca la vía o a mayor velocidad, haciendo la predicción inútil.

Así mismo, se podrían **añadir otros parámetros a la estimación** que influyen en gran medida en el consumo como son las **condiciones climáticas** (la temperatura afecta al rendimiento de la batería, así como al coeficiente de rodadura si la calzada está mojada) o las **condiciones de tráfico**.

7.5.2 Implementación

En cuanto a la implementación, se podrían mejorar ciertos aspectos como añadir la **consulta de altura de forma *offline***. Para ello sería necesario buscar formas de comprimir la información de alturas (como disminuyendo la precisión o que le usuario únicamente tenga que descargarse una región).

Así mismo, se podrían **añadir los puntos de recarga** disponibles actualmente como POI (*Point of Interest* – Puntos de interés), de tal forma que se pudieran integrar con las estimaciones y que el usuario pueda contar con dichos puntos a la hora de realizar una ruta.

Se podría también analizar la viabilidad de realizar otro tipo de implementaciones como por ejemplo implementar el **sistema de estimación en el servidor** (evitando la carga computacional al dispositivo móvil). Así mismo, se podrían utilizar **implementaciones de la Red Neuronal en lenguajes nativos** para así ganar en rendimiento.

7.5.3 Otros

Además del sistema implementado y las posibles alternativas o evoluciones comentadas, se podría dar el caso de otras aplicaciones para los sistemas estudiados. O incluso otras alternativas de implementación.

Se podría, por ejemplo, aplicar sistemas inteligentes como los estudiados para el **mantenimiento predictivo** del vehículo, de tal forma que en base a ciertas medidas que realice el sistema, pueda dar una estimación de cuándo hay que cambiar un componente o realizar una revisión (o incluso predecir el fallo de algún elemento y así evitar un accidente).

Así mismo, se plantea la posibilidad de **acoplar al vehículo un dispositivo Android** de manera fija (sin que tenga que ser el terminal que utiliza el usuario en su día a día). De esta forma se evitaría tener que estar pendiente del consumo de batería del móvil, de anclar el dispositivo al vehículo cada vez que se va a emprender un viaje, de tener que conectarlo por Bluetooth, etc. En este supuesto se podría desarrollar una aplicación más sencilla para el terminal del usuario con las características de control remoto que se han comentado, dejando el sistema de navegación y estimación inteligente en el dispositivo integrado en el vehículo. Además, a pesar de que supondría un coste adicional de cada unidad de vehículo, se evitaría tener que desarrollar un sistema tan completo en el resto de plataformas móviles que existen (principalmente iOS y Windows Phone).

Otra posible opción sería implementar una **herramienta online** que permitiese al usuario consultar todos los datos acerca de las rutas realizadas con su vehículo, pudiendo compartir dichas rutas con la comunidad de usuarios, organizar quedadas, etc.

7.6 Conclusión final

El objetivo de este sistema era proveer de una aplicación para dispositivos móviles que fuera capaz de conectarse a la motocicleta de propulsión eléctrica construida por Bultaco Motors y que proveyera de capacidades de navegación y consulta de mapas, así como visualización y guardado de datos de telemetría y GPS asociados a rutas realizadas.

En base a dicha información se construiría un sistema de Inteligencia Artificial que permitiese generar un Modelo de Predicción de Consumo adaptado al comportamiento de cada usuario en particular. De esta forma el usuario obtendría una estimación mucho más precisa de la energía que consumirá en una determinada ruta (en función de la orografía y

velocidad máxima, así como de sus características como piloto) y podría así decidir si realizar una ruta o no.

Dicho sistema se ha construido basándose en técnicas de *Machine Learning* como son las Redes Neuronales (*ANN – Artificial Neural Networks*) para la estimación de consumo, en la plataforma *open-source* de navegación y mapas OsmAnd como base del sistema, en la tecnología Bluetooth para la comunicación *Smartphone*–moto y en la plataforma Android para el conjunto del desarrollo (programación en lenguaje *Java*).

Además se han utilizado diferentes herramientas y metodologías de trabajo tanto para coordinar el proyecto como para la gestión del software desarrollado.

Se ha adquirido una gran cantidad de conocimiento tanto de la parte correspondiente a la inteligencia artificial como al desarrollo en la plataforma Android, así como sobre la coordinación de proyectos y la gestión eficiente del código. Todo ello además se ha llevado a cabo de la mano de un proyecto de innovación en ingeniería como es la apuesta de Bultaco por las motocicletas 100% eléctricas, tratándose así de un proyecto que no sólo se centra en el *software*, sino que ha de tener en cuenta las características de este tipo de vehículos para la implementación.

A pesar de que no se ha conseguido que las estimaciones sean lo suficientemente precisas con la citada técnica de aprendizaje máquina, el sistema se ha construido y funciona en su conjunto general (mapas, navegación, telemetría en tiempo real, grabación de rutas, etc.), resultando así un Proyecto Fin de Carrera muy completo para Ingeniería de Telecomunicación.

8 Bibliografía

[1] U. C. I. d. Madrid, «Las motos Bultaco vuelven con el apoyo del Parque Científico UC3M,» Mayo 2014. [En línea]. Available: http://portal.uc3m.es/portal/page/portal/investigacion/parque_cientifico/actualidad_agenda/Bultaco-mayo14. [Último acceso: Mayo 2014].

[2] Testla Motors Inc, «Using Energy Efficiently,» 2014. [En línea]. Available: http://www.teslamotors.com/goelectric/efficiency. [Último acceso: Junio 2014].

[3] A. Hughes y B. Drury, Electric Motors and Drives: Fundamentals, Types and Applications, Elsevier, 1990.

[4] J. Romm, «The Last Car You Would Ever Buy--Literally,» MIT Technology Review, 18 Junio 2008. [En línea]. Available: http://www.technologyreview.com/view/410290/the-last-car-you-would-ever-buy-literally/. [Último acceso: Abril 2014].

[5] Wikipedia, «Battery (electricity),» Mayo 2014. [En línea]. Available: http://en.wikipedia.org/wiki/Electrical_battery. [Último acceso: Junio 2014].

[6] Battery University, «Advantages and limitations of the Different Types of Batteries,» Octubre 2010. [En línea]. Available: http://batteryuniversity.com/learn/article/whats_the_best_battery. [Último acceso: Mayo 2014].

[7] P. Reed, «Your Fuel Economy Gauge Is Fibbing,» edmunds.com, Diciembre 2011. [En línea]. Available: http://www.edmunds.com/fuel-economy/your-fuel-economy-gauge-is-fibbing.html. [Último acceso: Marzo 2014].

[8] M. Ceraolo y G. Pede, «Techniques for Estimating the Residual Range of an Electric Vehicle,» *IEEE Transactions on Vehicular Technology,* vol. 50, nº 1, 2001.

[9] J. A. Oliva, C. Wiehrauch y T. Bertran, «A Model-Based Approach for Predicting the Remaining Driving Range in Electric Vehicles,» de *Annual Conference of the Prognostics and Health Management Society*, 2013.

[10] O. Zirn, M. Ahlborn, R. Heyne y C. Vetter, «Range Estimation for Electric Vehicles,» de *Proceedings of the 2012 Winter Simulation Conference*, 2012.

[11] M. Neaimeh, G. A. Hill, Y. Hübner y P. T. Blythe, «Routing Systems to Extend the Driving Range of Electric Vehicles».

[12] R. Hurlbrink, L. Winslow y R. Prins, «Electric Vehicle Energy Usage Modeling, Simulationg and Testing for Range Estimation».

[13] W. Wang, Y. Zhang, Y. Kobayashi y K. Shirai, «Remaining Diving Range Estimation of

Electric Vehicle».

[14] Wikipedia, «Inteligencia artificial,» 2014. [En línea]. Available: http://es.wikipedia.org/wiki/Inteligencia_artificial. [Último acceso: Marzo 2014].

[15] Solar Journey USA, «EV Energy Consumption,» 2008. [En línea]. Available: http://www.solarjourneyusa.com/EVdistanceAnalysis5.php#_ftn1. [Último acceso: Mayo 2014].

[16] K. S. M. a. W. H. Hornik, «Multilayer feedforward networks are universal approximators,» *Neural Networks,* nº 2, pp. 359-366, 1989.

[17] K. Hornik, «Some new results on neural network approximation,» *Neural Networks,* nº 6, pp. 1069-1072, 1993.

[18] W. S. Sarle, «Neural Nets,» Usenet newsgroup, 2002. [En línea]. Available: http://www.faqs.org/faqs/ai-faq/neural-nets/. [Último acceso: Junio 2014].

[19] L. Berke y P. Hajela, «Applications of artificial neural nets in structural mechanics,» *Structural and Multidisciplinary Optimization,* vol. 2, nº 4, p. 90–98, 1992.

[20] N. Tedeschi, «¿Qué es un Patrón de Diseño?,» Microsoft MSDN, [En línea]. Available: http://msdn.microsoft.com/es-es/library/bb972240.aspx. [Último acceso: Julio 2014].

[21] Wikipedia, «Software development process,» Septiembre 2014. [En línea]. Available: http://en.wikipedia.org/wiki/Software_development_process. [Último acceso: Septiembre 2014].

[22] R. Daría, «Metodologías de Desarrollo Ágiles Vs. Metodologías Tradicionales,» Marzo 2014. [En línea]. Available: http://rdsoporteymantenimientodepc.blogspot.com.es/2014/03/metodologias-de-desarrollo-agiles-vs.html. [Último acceso: Septiembre 2014].

[23] Wikipedia, «Agile software development,» [En línea]. Available: http://en.wikipedia.org/wiki/Agile_software_development. [Último acceso: Septiembre 2014].

[24] J. Garzás, «¿Qué es el método Kanban para la gestión de proyectos?,» 22 Noviembre 2011. [En línea]. Available: http://www.javiergarzas.com/2011/11/kanban.html. [Último acceso: Septiembre 2014].

[25] V. Driessen, «A successful Git branching model,» 5 Enero 2010. [En línea]. Available: http://nvie.com/posts/a-successful-git-branching-model/. [Último acceso: Septiembre 2014].

[26] S. Streeting, «Smart branching with SourceTree and Git-flow,» 1 Agosto 2012. [En línea]. Available: http://blog.sourcetreeapp.com/2012/08/01/smart-branching-

with-sourcetree-and-git-flow/. [Último acceso: 2 Marzo 2014].

[27] Kantar World Panel, «Smartphone fragmentation in Europe challenges status quo,» 26 Mayo 2014. [En línea]. Available: http://www.kantarworldpanel.com/global/News/European-smartphone-fragmentation-challenges-status-quo. [Último acceso: 7 Junio 2014].

[28] I. Ávila, «Programación Android, Base de Datos I,» 14 Septiembre 2013. [En línea]. Available: http://www.proyectosimio.com/programacion-android-base-de-datos-i-modelo-vista-controlador/. [Último acceso: Diciembre 2013].

[29] Open Street Maps, «OSM tags for routing/Maxspeed,» Agosto 2014. [En línea]. Available: http://wiki.openstreetmap.org/wiki/OSM_tags_for_routing/Maxspeed. [Último acceso: Septiembre 2014].

[30] SQLite, «SQLite,» [En línea]. Available: http://www.sqlite.org/about.html. [Último acceso: Febrero 2014].

[31] NASA, «Shuttle Radar Topography Mission (SRTM),» Junio 2009. [En línea]. Available: http://www2.jpl.nasa.gov/srtm/index.html. [Último acceso: Febrero 2014].

[32] «ksoap2-android,» [En línea]. Available: https://code.google.com/p/ksoap2-android/. [Último acceso: Febrero 2014].

[33] Bubble Software Solutions ltd, «Wsdl2Code,» [En línea]. Available: http://www.wsdl2code.com/pages/Home.aspx. [Último acceso: Abril 2014].

[34] Wikipedia, «Haversine formula,» Septiembre 2014. [En línea]. Available: http://en.wikipedia.org/wiki/Haversine_formula. [Último acceso: Septiembre 2014].

[35] Wikipedia, «JSON,» Agosto 2014. [En línea]. Available: http://es.wikipedia.org/wiki/JSON. [Último acceso: Septiembre 2014].

[36] K. Nice y J. Strickland, «How Fuel Cells Work,» How Stuff Works, 2011. [En línea]. Available: http://auto.howstuffworks.com/fuel-efficiency/alternative-fuels/fuel-cell.htm. [Último acceso: Febrero 2014].

[37] C. E. (Sandy) Thomas, President H2GEN Innovations Inc., «Fuel Cell and Battery Electric Vehicles Compared,» U.S. Department of Energy, Marzo 2014. [En línea]. Available: http://energy.gov/sites/prod/files/2014/03/f9/thomas_fcev_vs_battery_evs.pdf. [Último acceso: Junio 2014].

9 Anexos

9.1 Especificaciones técnicas: telemetría

9.1.1 Trama de telemetría

La trama de datos de telemetría está compuesta por cuatro grupos de datos:

CONTROL	ESTADO	DATOS	TIMESTAMP

- CONTROL: 1 byte de datos con indicadores de control de la moto.
- ESTADO: 5 bytes de datos con los valores de todos los sensores de la moto.
- DATOS: 18 bytes de datos con los valores de los distintas magnitudes a medir.
- TIMESTAMP: 2 bytes de datos con la fecha y hora en formato UTC.

En la siguiente tabla se detallan todos los datos a incluir

BIT	DATO	NUMERO DE BITS	Byte	Mapeo	COMENTARIOS
1	Start	1	1	byte	Bits de Control de la moto. Solo se envía este bloque una por trama y se corresponde con el último evento registrado antes del envío
2	Stop – Inmovilizador	1			
3	Stop – Bloqueo	1			
4	Stop – Caída	1			
5	Modo Normal	1			
6	Modo Eco	1			
7	Reservado	1			Para posible uso futuro
8	Reservado	1			Para posible uso futuro
9	Parada permanente	1	2	byte	Bit de Estado
10	Parada temporal.	1			Bit de Estado
11	Giro habilitado.	1			Bit de Estado
12	Modo Eco.	1			Bit de Estado
13	Reservado	1			Bit de Estado
14	Reservado	1			Bit de Estado
15	Reservado	1			Bit de Estado
16	Batería como cargador	1			Bit de Estado
17	Sobre-temperatura Controlador.	1	3	byte	Bit de Estado
18	Sobre-temperatura Motor.	1			Bit de Estado
19	Sobre-temperatura Disipador.	1			Bit de Estado
20	Sobre-temperatura BMS o baterías.	1			Bit de Estado
21	Sonda temperatura motor mal.	1			Bit de Estado
22	Sonda temperatura baterías abierta.	1			Bit de Estado
23	Sobre-temperatura / Desequilibrio ultra-condensadores.	1			Bit de Estado
24	Anomalía en corriente de fases.	1			Bit de Estado
25	Sobre-corriente de ultra-condensadores.	1	4	byte	Bit de Estado
26	Sobre-corriente de entrada.	1			Bit de Estado
27	Fallo de drivers de IGBT's.	1			Bit de Estado
28	Sobre-corriente de fases.	1			Bit de Estado
29	Error uControlador.	1			Bit de Estado
30	Fallo acelerador.	1			Bit de Estado
31	Fallo tacómetro.	1			Bit de Estado
32	Fallo freno.	1			Bit de Estado

33	Línea BMS abierta.	1	5	byte	Bit de Estado
34	V célula de batería fuera de margen.	1			Bit de Estado
35	V batería baja.	1			Bit de Estado
36	V batería alta.	1			Bit de Estado
37	Reservado	1			Bit de Estado
38	Reservado	1			Bit de Estado
39	Reservado	1			Bit de Estado
40	Reservado	1			Bit de Estado
41	Limitación de Par por temperatura de disipador.	1	6	byte	Bit de Estado
42	Limitación de Par por temperatura de motor.	1			Bit de Estado
43	Limitación de Par por V e I de batería.	1			Bit de Estado
44	Limitación de Par por sonda de temperatura del motor mal.	1			Bit de Estado
45	Limitación de Par por temperatura de batería.	1			Bit de Estado
46	Reservado	1			Bit de Estado
47	Frenado sobre ultra-condensadores deshabilitado.	1			Bit de Estado
48	Frenado eléctrico deshabilitado.	1			Bit de Estado
49-64	Voltaje de batería.	16 (2Bytes)	7-8	float	Real con precisión de 4 decimales
65-80	Corriente de batería	16 (2 Bytes)	9-10	float	Real con precisión de 4 decimales
81-96	Voltaje de ultra-condensadores.	16 (2 Bytes)	11-12	float	Real con precisión de 4 decimales
97-112	Corriente de ultra-condensadores	16 (2 Bytes)	13-14	float	Real con precisión de 4 decimales
113-128	Temperatura de controlador	16 (2 Bytes)	15-16	float	Real con precisión de 4 decimales
129-144	Temperatura de motor	16 (2 Bytes)	17-18	float	Real con precisión de 4 decimales
145-160	Temperatura de baterías	16 (2 Bytes)	19-20	float	Real con precisión de 4 decimales
161-176	Revoluciones / s del motor	16 (2 Bytes)	21-22	float	Real con precisión de 4 decimales
177-192	SOC, Amp.hr descargados.	16 (2 Bytes)	23-24	float	Real con precisión de 4 decimales
193-208	Fecha y Hora (timestamp)	16 (2 Bytes)	25-26	long	Real con precisión de 4 decimales

Se generará una trama completa de telemetría cada 200 ms, por lo que si las peticiones se realizan cada 10 segundos, se recibirá un total de 50 tramas en cada petición.

9.1.2 Trama de GPS

BIT	DATO	NUMERO DE BITS	Byte	Mapeo	COMENTARIOS
	Latitud	???		float	Real con precisión de 7 decimales
	Longitud	???		float	Real con precisión de 7 decimales
	Altura	???		float	Real con precisión de 1 decimales
	Timestamp	???		long	Real con precisión de 4 decimales

Se generará una trama completa de GPS cada 1 segundo, por lo que si las peticiones se realizan cada 10 segundos, se recibirán un total de 10 tramas en cada petición.

9.2 JSON Librería Bluetooth

La centralita actúa como servidor, de forma que responderá a las llamadas que se realicen desde la aplicación, no inicia la comunicación.

- *ConectarApp_BT()*: conexión con la centralita de la moto.
- *DesconectarApp_BT()*: desconexión con la centralita de la moto.
- *LeerTelemetría_BT()*: envolverá un JSON con los valores de la telemetría agrupados desde la última llamada realizada.
- *LeerGPS_BT()*: devolverá un JSON con las coordenadas GPS agrupadas desde la última llamada realizada.
- *LeerInforme_BT(nombre_fichero)*: obtiene el último informe de la moto y lo almacena en [nombre_fichero].

Nombre fichero debe ser la ruta completa y se le pasa por parámetro.

La librería de comunicaciones se encargará de encapsular todo el protocolo y mensajes de bajo nivel para las comunicaciones entre la centralita y el *smartphone*, y devolverá como resultado una estructura estándar en formato JSON.

A continuación se puede ver esta estructura para cada una de las llamadas.

9.2.1 JSON Telemetría

```
{“BA_telemetria”: [
    {“ba_t_item”:
          {
             “control”: “127”,
             “estado”: {“e1”: “65”, “e2”: “165”, “e3”: “54”, “e4”: “123”, “e5”: “5”},
              “datos”: {“d1”: “65.1203”, “d2”: “165768.0293”, “d3”: “56234.4235”, “d4”: “12345.0000” ,
                          “d5”: “5.01234”, “d6”: “57.1234” , “d7”: “9523.6540” , “d8”: “59.0000” , “d9”: “58.0951”},
              “time”:”260414:004534.234”
          }
    },
    {“ba_t_item”:
          {
             “control”: “127”,
             “estado”: {“e1”: “65”, “e2”: “165”, “e3”: “54”, “e4”: “123”, “e5”: “5”},
              “datos”: {“d1”: “65.1203”, “d2”: “165768.0293”, “d3”: “56234.4235”, “d4”: “12345.0000” ,
                          “d5”: “5.01234”, “d6”: “57.1234” , “d7”: “9523.6540” , “d8”: “59.0000” , “d9”: “58.0951”},
              “time”:” 260414:004534.434”
          }
    }
  ]
}
```

NOTAS:

- El valor de control es de un byte correspondiente a los ocho bits de control de la trama.
- Cada uno de los valores e1-e5 es de un byte de cada uno de los bits de estado: e1 corresponde a los 8 primeros bits y así sucesivamente.
- Cada uno de los valores de datos corresponde con el valor de una magnitud de la trama (2 bytes): d1 corresponde con “Voltaje de batería” y asi sucesivamente.
- EL campo TIME corresponde con la fecha/hora de la toma de los datos de la trama.

- El JSON tendrá tantos elementos BA_t_item como sea necesario, típicamente 50 si se piden datos cada 10 segundos.

9.2.2 JSON Geolocalización

```
{“BA_geoloc”: [
    {“ba_g_pos”:
          {
             “lat”: “4322.0289N”,
             “lon”: “00824.5210W”,
              “alt”: “39.9M”,
              “time”:”260414:004523.345”
          }
    },
    {“ba_g_pos”:
          {
             “lat”: “4322.0399N”,
             “lon”: “00824.5210W”,
              “alt”: “39.9M”,
             “time”:”260414:004524.345”
          }
    }
  ]
}
```

NOTAS:

- Cada campo lat, lon y alt corresponde con las magnitudes latitud, longitud y altitud
- EL campo TIME corresponde con la fecha/hora de la toma de los datos de la trama.
- El JSON tendrá tantos elementos BA_g_pos como sea necesario, típicamente 10 si se piden datos cada 10 segundos.

9.3 Manual de Usuario

Se adjunta a continuación el documento correspondiente al **Manual de Usuario** de la aplicación. Dicho manual se desarrolló en base a la imagen de marca de LGN Tech Design, nombre que tenía la empresa antes de adoptar el nombre de Bultaco Motors.

Contenido

Introducción al Manual de Usuario

Este manual pretende servir de introducción al funcionamiento básico de la aplicación *Smart eBike* que la empresa *LGN Tech Design SL* pone a disposición de sus usuario de motocicletas.

Dicha aplicación provee de funcionalidades de navegación guiada *offline* (sin conexión a *Internet*), gestión de rutas, gestión de la motocicleta, visionado de telemetría en tiempo real, etc.

A continuación se detallarán algunas de las características antes nombradas así como el funcionamiento general de la aplicación.

Smart eBike

La aplicación *Smart eBike* tiene dos categorías de contenido según esté asociado a la navegación guiada o a la motocicleta.

Imagen 1 - Pantalla principal

Como se puede ver, ambas categorías engloban a su vez diferentes funciones, representadas en la Imagen 1 - Pantalla principal como accesos directos a diferentes características:

- ***OsmAnd* (Navegación)**
  - Mapa
  - Buscar
  - Favoritos
  - Opciones
- ***eBike* (Telemetría y gestión)**
  - eBike gestión
  - eBike navegación

OsmAnd (Navegación)

Para la gestión de la navegación se emplea el sistema de navegación y visionado de mapas de *OsmAnd*[1], el cual funciona con datos de cartografía de *OpenStreetMaps*[2].

Este sistema proporciona al usuario un sistema de navegación guiado con mapas *offline* (se descargan en el dispositivo para que el usuario pueda consultarlos sin necesidad de conexión a internet). Además, se puede incorporar un sistema de voz para las indicaciones de dicha navegación.

Mapa

Como se puede ver en la Imagen 2 – Mapa, se tiene una vista completa del mapa con la situación actual proporcionada por el *GPS* (punto azul). En esta pantalla existen varios elementos principales que se han marcado con diferente numeración:

(1) Selección de datos a visualizar en (3)
(2) Ir a pantalla **eBike navegación** (telemetrías)
(3) Visualización de magnitudes
(4) Menú

[1] *OsmAnd: OSM Automated Navigation Directions* - http://www.osmand.net/
[2] *OpenStreetMaps (OSM)*: http://www.openstreetmap.org/

El usuario podrá grabar una ruta pulsando el botón rojo. Al pulsarlo de nuevo dicha ruta se guardará para posibles consultas a través de la pantalla de Gestión.

Imagen 2 – Mapa

Seleccionando el botón (1) se abrirá una lista con las diferentes magnitudes de interés que el usuario pordrá mostrar en el mapa como puede ser velocidad o consumo (ver Imagen 3 - Magnitudes mapa).

Para poder grabar una ruta el usuario tendrá que pulsar el botón de *REC* (2) y así la aplicación comenzará la adquisición de datos. Al finalizar la ruta, el usuario volverá a pulsar dicho botón, pudiendo entonces dar un nombre a la ruta que acaba de realizar (si no introduce ningún nombre, aparecerá como *desconocido*).

Si el botón (3) es seleccionado se accederá a la pantalla de visionado de telemetrías **eBike navegación**, pantalla que se verá más en detalle en la sección Navegación.

Como se puede observar, en la sección (4) se muestran una serie de parámetros asociados con la moto y que el usuario podrá ver en todo momento a la vez que sigue una determinada ruta.

Si se selecciona el botón (5) se mostrarán diferentes opciones como obtener indicaciones o buscar una dirección determinada.

El usuario puede ver varios parámetros en su pantalla mientras sigue la guía de navegación (incluídos valores de telemetría)

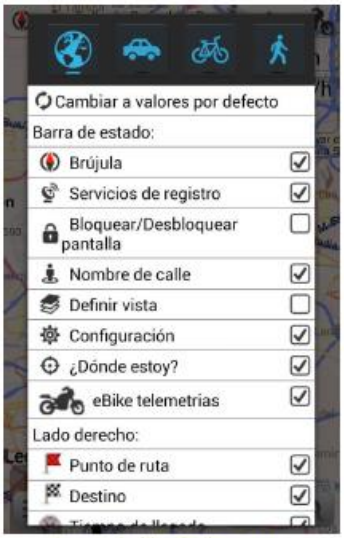


Imagen 3 - Magnitudes mapa

Buscar

Como se puede ver en la Imagen 4 - Buscar, es posible buscar el destino de navegación de diferentes formas:

- Buscar PDI (Punto de Interés)
- Buscar dirección
- Buscar por coordenadas
- Favoritos
- Historial de búsqueda
- Buscar transporte

Además, se pueden ver una serie de controles en la parte inferior (que puede que varíen de una selección a otra) cuya función es la siguiente:

- Obtener indicaciones
- Establecer como destino
- Mostrar en el mapa
- Añadir a favoritos

Es posible elegir el destino como en base a un PDI, una dirección y coodenadas GPS

Las búsquedas realizadas se guardan para futuras consultas

Imagen 4 - Buscar

Favoritos

En la pantalla de favoritos se guardarán los destinos guardados como favoritos. Por de defecto tiene las siguientes categorías:

- Amigos
- Casa
- Lugares
- Otros

Esta pantalla pretende ser un atajo directo para seleccionar aquellas direcciones que el usuario haga habitualmente o que haya guardado con anterioridad. En la Imagen 5 - Favoritos se puede ver el aspecto de dicha pantalla y las diferentes categorías (el usuario puede añadir cuantas quiera según su conveniencia).

Se pueden guardar tantos destinos como el usuario quiera (y en las categorías que quiera)

Imagen 5 - Favoritos

Opciones

En la pantalla de opciones se podrá acceder a todos los parámetros de configuración de la aplicación. Como se puede ver en la Imagen 6 - Opciones, existe una clasificación según sean opciones de:

- Gestión de mapas descargados
- Opciones generales de aplicación
- Navegación
- Extras

En concreto, en el primero de ellos es donde el usuario podrá seleccionar los mapas que desea descargar en el dispositivo (será necesario para que en la vista Mapa se generen las imágenes del mapa).

Es necesario descargar los mapas de algún país para poder emplear las funciones de navegación

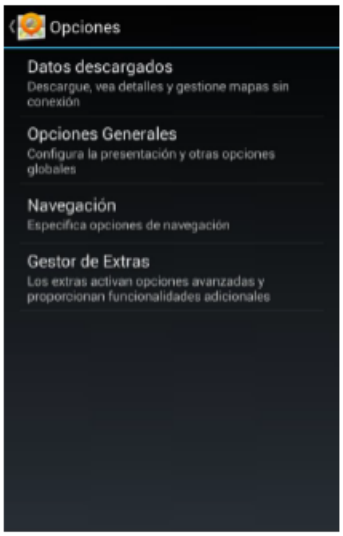


Imagen 6 - Opciones

eBike

Entre las diferentes funcionalidades que proporciona *eBike* se puede encontrar todas las relacionadas con la gestión de la motocicleta: telemetrías asociadas a rutas, telemetría en tiempo real, avisos de la motocicleta, gestión de perfil de usuario, control remoto (en caso de robo), etc.

En esta sección se detallarán las diferentes opciones accesibles al usuario.

Gestión

En el apartado de gestión, se puede acceder a diferentes opciones de configuración y gestión (ver Imagen 7 - Gestión):

- Gestión de rutas
- Avisos
- Control remoto
- Perfil de usuario

Es posible consultar diferentes parámetros de telemetría asociados a una ruta (velocidad media, cambios de altitud, etc.)

Imagen 7 - Gestión

La aplicación permite acceder a un menú de **Gestión de rutas** en el cual el usuario podrá visualizar las diferentes rutas que ha realizado así como los diferentes parámetros asociados a éstas (tiempo, distancia, variación de altura, energía consumida, etc.), algunos de ellos en forma de gráficas. Desde esa misma pantalla, podrá seleccionar una de las rutas para volver a realizarla (manteniendo pulsada la ruta en cuestión se desplegará el menú correspondiente).

Como en todo momento el *Smartphone* estará conectado a la motocicleta (bien sea a través de *Bluetooth* o de *USB*), se irán recolectando datos de telemetría. Algunos de esos datos se podrán mostrar tanto en la visualización de ruta comentada anteriormente (a modo de resumen) como en la pantalla de navegación (ver Mapa).

Así mismo, se podrán recibir **Avisos** de la moto, como puede ser el sobrecalentamiento del motor, y que podrán ser consultados en el correspondiente apartado. Dichos avisos estarán asociados a una determinada ruta (manteniendo pulsado el aviso se puede acceder a la ruta correspondiente).

Además, es posible acceder a algunas funciones de seguridad en la pantalla de **Control remoto** y que permiten acceder a las siguientes funciones:

- Llamada de emergencia: llamará a un número de teléfono de emergencias (por defecto 112, pero puede modificarse)
- Posición de la moto: enviará un SMS para pedir la localización de la moto (es necesario configurar el número en la pantalla de **Perfil de Usuario: Tlfn. Moto**)
- Bloquear: enviará un SMS para el solicitar el bloqueo de la moto (es necesario configurar el número en la pantalla de **Perfil de Usuario: Tlfn. Moto**)

Existen algunas configuraciones de **Perfil de Usuario**, las cuales se pueden ver en la Imagen 8 - Gestión: Perfil de Usuario, y que podemos clasificar según sus características:

- Datos de usuario: nombre y apellidos, usuario y contraseña (para la conexión con el servidor de LGN)
- Datos de la moto
- Características del piloto
- Teléfonos de emergencias y de moto (en caso de robo)

Es posible enviar varios comandos a la moto vía SMS (localización y bloqueo), para lo cual habrá que configurar el teléfono asociado

Imagen 8 - Gestión: Perfil de Usuario

Navegación

La pantalla de navegación ofrecerá una vista al usuario de diferentes parámetros dinámicos de la motocicleta y está pensada para actuar como complemento del panel de mandos de ésta.

Dicha pantalla se puede configurar para que muestre los 5 parámetros que el usuario estime oportuno (basta con mantener pulsado uno de ellos para que se muestre un menú con las diferentes opciones). De este modo, cuando el la moto esté en funcionamiento, podrá verse en tiempo real la variación de diferentes magnitudes. En la Imagen 9 – Navegación podemos ver la configuración de dicha pantalla.

Se pueden elegir hasta 5 magnitudes para mostrar en la pantalla de Navegación

Imagen 9 – Navegación

Además de las magnitudes a visualizar se muestra un acceso directo a la realización de llamadas de emergencia. Nada más pulsar dicho botón, se comenzará a realizar una llamada a los servicios de emergencia (teléfono configurado por el usuario en Gestión).

Las magnitudes que el usuario podrá seleccionar para mostrar son las siguientes (algunas dispondrán únicamente de referencia numérica y no visual debido a sus características):

- Velocidad media: promedio de la velocidad en la ruta que se está realizando.
- Velocidad máxima: máxima velocidad alcanzada en el transcurso de una ruta.

- Distancia recorrida: distancia total recorrida desde el inicio de la ruta.
- Autonomía en distancia: distancia estimada que se podrá recorrer según la energía disponible en las baterías.
- Energía consumida: energía cosumida desde el inicio de la ruta.
- Energía regenerada: energía recuperada a través del sistema de frenado regenerativo desde el inicio de la ruta.
- Consumo neto medio: consumo medio de energía.
- Factor de extensión de autonomía: ratio de energía obtenida a través del frenado regenerativo en función de la energía total consumida.
- Temperatura de motor.
- Temperatura en electrónica de tracción.
- Temperatura en batería.