Trabajo Fin de Grado

Modelo, diseño y presupuesto de una bicicleta eléctrica

Model, design and Budget of an e-bike

Autor/es

Adrián Novella Domingo

Director/es

José Francisco Sanz Osorio

Facultad / Escuela

EINA

Año

2019

ÍNDICE

ÍNDICE DE ILUSTRACIONES

ÍNDICE DE TABLAS

OBJETIVO

El presente trabajo pretende abordar el proceso de diseño de un vehículo eléctrico diseñado para la movilidad personal. Una breve puesta en escena sobre el estado actual de la movilidad urbana pondrá sobre la mesa el beneficio que supone sustituir parte del parque automovilístico de las ciudades por una forma de transportarse más barata, más sencilla y más respetuosa con el medio ambiente.

Esto contesta al por qué de este proyecto, sin embargo, la pregunta más difícil, o más larga de contestar, es como hacer esto posible. El alto precio de las baterías y motores de imanes permanentes, o una legislación a la que el repentino auge de este tipo de vehículos ha pillado de improviso serán solo algunos de los problemas a los que se enfrenta esta tecnología.

Dentro del entorno de la movilidad personal, este proyecto se enfoca en abordar la parte técnica del vehículo, mediante la realización de un modelo matemático, escrito en el lenguaje de programación Python, se ha hecho un esfuerzo por determinar, de la manera más precisa posible, las necesidades materiales del proyecto, que finalmente desembocaran en un presupuesto de fabricación.

Inicialmente la fabricación de un prototipo funcional iba a formar parte del proyecto, añadiendo así la posibilidad de comparar el desempeño real del sistema con el calculado previamente, sin embargo, el tiempo disponible ha obligado a dejar esta parte fuera del proyecto, quedando relegada a un proyecto personal posterior.

1. ANTECEDENTES

Según los resultados de la encuesta realizada por Comisiones Obreras de Aragón, junto con otras asociaciones en el año 2015 en Zaragoza [1], esta es la situación actual de la movilidad en la ciudad, que, en cierto modo, es extrapolable a otras ciudades del país.

- El 41,2% de los desplazamientos se hacen dentro del casco urbano, el 35,3% involucran el área periurbana mientras que solo el 23,4% salen fuera de la ciudad, estos últimos no son un objetivo práctico para los Vehículos de Movilidad Personal, bicicletas y bicicletas eléctricas, por ello no profundizaremos más en su estudio.
- De los desplazamientos en casco urbano, el 42,4% tiene menos de 29 años y el 32,5% tiene entre 29 y 44 años, para el caso de los desplazamientos por la zona periurbana el 36% son menores de 29 y el 41% tienen entre 29 y 44 años. Una vez más, suponemos que en edades superiores a 44 años este sistema de transporte pierde atractivo, y no sería realista tenerlos en cuenta como usuarios potenciales.
- Si centramos el foco en la movilidad laboral (ida y vuelta al trabajo), tenemos que, de media, un 55,3% de los trabajadores va en coche, cabe destacar que, tan solo un 8,1% de estos lo hace como acompañante.
- Los vehículos no motorizados solo se consideran una elección válida para viajes de menos de 20 minutos.
- Tan solo un 18,3% de los encuestados disponen de un medio de transporte sostenible proporcionado por la empresa, para un 74,3% de ellos esta medida consiste en un autobús.
- En cuanto a las razones para no usar el transporte público lideran la oferta inadecuada de este y su lentitud relativa a un vehículo privado.
- Tan solo un 4,4% de los encuestados va al trabajo en bicicleta.

La situación del tráfico desarrollada anteriormente se resume en los siguientes gráficos:

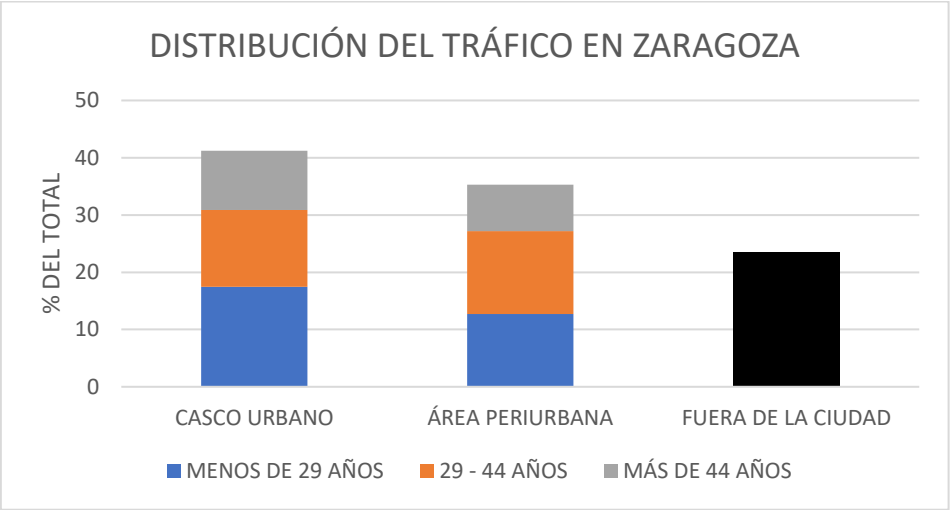


Ilustración 1 Distribución del tráfico en Zaragoza

Como podemos ver, tras una primera criba, en la que hemos tenido en cuenta la naturaleza de los desplazamientos y la edad de los encuestados, las franjas de población a las que podría interesarles un vehículo tal como es una bicicleta eléctrica son las azules y naranjas, o lo que es lo mismo, el 58% de la población.

Esta es la distribución actual de los vehículos que utilizan los encuestados:

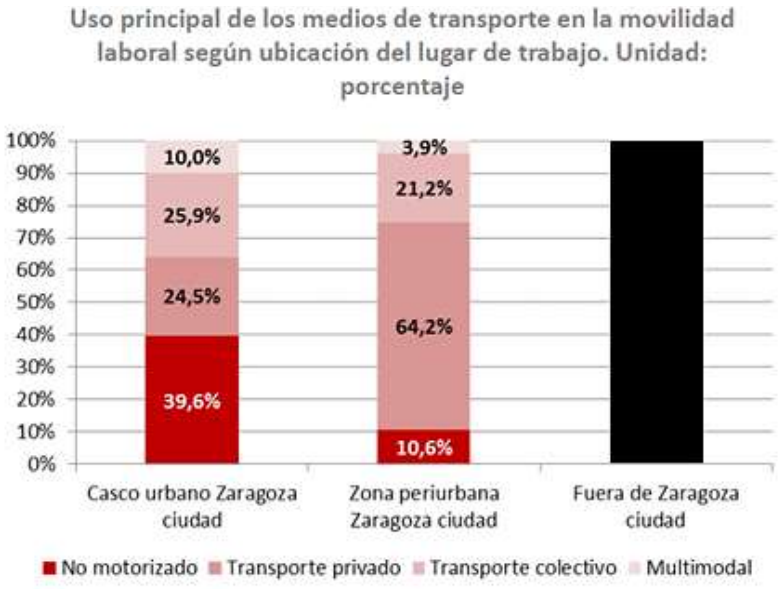


Ilustración 2 Distribución de los medios de transporte clasificados por zonas de Zaragoza

Ahora bien, hay que tener en cuenta que algunas personas estarán demasiado cerca como para necesitar un vehículo a motor, estas forman parte de la franja 'No motorizado' del gráfico anterior, y otras recorren demasiada distancia como para querer hacerlo en bicicleta, estas principalmente se encuentran en la franja perteneciente al transporte privado y colectivo de la zona periurbana.

2. NORMATIVA

La Ley sobre Tráfico, Circulación de Vehículos a Motor y Seguridad Vial, aprobada en el Real Decreto Legislativo 6/2015 define el ciclo de la siguiente manera:

"Ciclo. Vehículo provisto de, al menos, dos ruedas y propulsado exclusiva o principalmente por la energía muscular de la persona o personas que están sobre el vehículo, en particular por medio de pedales. Se incluyen en esta definición los ciclos de pedaleo asistido" [2]

Es en esta categoría donde entran las bicicletas comunes, así como, en concordancia con el Reglamento UE 168/2013 [3], se incluyen las bicicletas de pedaleo asistido en este reglamento se las excluye de la homologación requerida por los ciclomotores de la siguiente manera:

"El presente Reglamento no se aplicará a los vehículos siguientes:

[…]

Las bicicletas de pedales con pedaleo asistido, equipadas con un motor eléctrico auxiliar, de potencia nominal continua máxima inferior o igual a 250 W, cuya potencia disminuya progresivamente y que finalmente se interrumpa antes de que la velocidad del vehículo alcance los 25 km/h o si el ciclista deja de pedalear"

Lo cual significa que, las bicicletas de pedaleo asistido disfrutan de las ventajas de cualquier otra bicicleta, es decir, pueden circular por el carril bici, no requieren ningún permiso para conducirlas y, salvo que la normativa municipal dicte otra cosa, pueden llevarse sin necesidad de casco. Sin embargo, esto no es así para una bicicleta capaz de funcionar con un acelerador, ya que, al no ser excluidas explícitamente pasan a ser tratadas como un ciclomotor, uno de los objetivos de este proyecto es poner de manifiesto este hecho, y proponer su revisión. Es importante destacar que, en la fecha de redacción de este trabajo, en España los 250W mencionados en la cita anterior son 500W, aunque es de esperar que acabe igualándose a lo que marca la normativa europea.

Esta es la descripción para un ciclomotor de 2 ruedas que nos proporciona [2]:

"Vehículo de dos ruedas, con una velocidad máxima por construcción no superior a 45 km/h y con un motor de cilindrada inferior o igual a 50 cm^3, si es de combustión interna, o bien con una potencia continua nominal máxima inferior o igual a 4 kW si es de motor eléctrico."

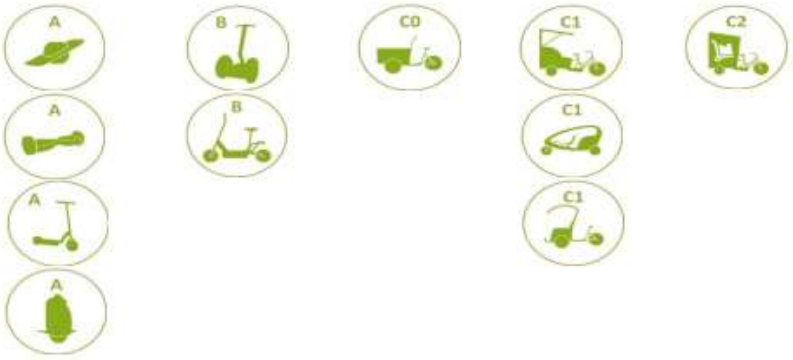
En materia de movilidad eléctrica, entran en juego otro tipo de vehículos, legislativamente distintos, llamados Vehículos de Movilidad Personal.

En este caso, la legislación corre a cargo de las autoridades municipales, a pesar de ello, para ayudar a regularlos y homogeneizar las normas lo máximo posible, la DGT provee la Instrucción 16/V-124 [4] con intención de que sirva como base para estas.

Merece la pena mencionar que, debido al rápido impacto de este tipo de vehículos en la circulación urbana, los organismos públicos se han visto obligados a improvisar su regulación, y que, está previsto legislarlos correctamente en un futuro próximo, incluyendo, se intuye, las bicicletas eléctricas en esta clasificación de VMP en una categoría propia. De momento esta es la única información que se ofrece acerca de la clasificación de estos tipos de vehículos en la instrucción mencionada arriba:

Características	A	B	C0	C1	C2
Velocidad máx.	20 km/h	30 km/h	45 km/h	45 km/h	
Masa	≤ 25 kg	≤ 50 kg	≤ 300 kg	≤ 300 kg	
Capacidad máx. (pers.)	1	1	1	3	
Ancho máx.	0,6 m	0,8 m	1,5 m	1,5 m	
Radio giro máx.	1 m	2 m	2 m	2 m	
Peligrosidad superficie frontal	1	3	3	3	
Altura máx.	2,1 m	2,1 m	2,1 m	2,1 m	
Longitud máx.	1 m	1,9 m	1,9 m	1,9 m	
Timbre	NO	SÍ	SÍ	SÍ	
Frenada	NO	SÍ	SÍ	SÍ	
DUM (distribución urbana mercancías)	NO	NO	NO	NO	SÍ
Transporte viajeros mediante pago de un precio	NO	NO	NO	SI	NO

Los VMP se clasifican en función de la altura y de los ángulos peligrosos que puedan provocar daños a una persona en un atropello. Se definen como ángulos peligrosos aquellos inferiores a 110° orientados en sentido de avance del VMP, o verso el conductor o pasajeros.



4 niveles de peligrosidad:

- Altura frontal inferior a 0.5 m sin ángulos peligrosos
- Altura frontal superior a 0.5 m sin ángulos peligrosos
- Altura frontal inferior a 0.5 m con ángulos peligrosos
- Altura frontal superior a 0.5 m con ángulos peligrosos

Ilustración 3 Extracto de la directiva de la DGT que clasifica los VMP

En el caso de querer comercializar vehículos de estas características, es importante mencionar que, en [5] se responsabiliza al fabricante de la homologación, liberando al usuario de esta tarea, y que, aunque aparentemente no hay nada que impida homologar el vehículo como VMP y permitir que funcione con un acelerador, parece más sensato esperar que, en la práctica, solo se permita homologarla como bicicleta de pedaleo asistido, bloqueando la función de vehículo autopropulsado, y relegar esta función a una futura actualización de firmware.

3. ESTADO DEL ARTE

Estado del arte es como se conoce habitualmente al panorama actual en un determinado entorno, en el momento en el que se ha realizado este trabajo las mencionadas en este apartado son las alternativas más populares que se pueden encontrar en el mercado, para esta búsqueda nos hemos basado en el catálogo que ofrece [6], la legislación actual influye significativamente en las opciones disponibles.

3.1. BICICLETAS DE PEDALEO ASISTIDO

Tal y como marca la normativa española las bicicletas de pedaleo asistido de hasta 500W de potencia son tratadas legalmente como bicicletas comunes. Es por eso por lo que la mayor parte del mercado se centra en estas características.

3.1.1. BICICLETAS DE MONTAÑA

Un nicho altamente popular para el mercado del pedaleo asistido es el deporte de montaña, principalmente debido a que facilita rutas difíciles a personas sin el nivel físico requerido para realizarlas, siendo también de ayuda al sector turístico de montaña. Suelen incorporar motores de 350-500W y baterías de 500Wh en adelante debido a que necesitan mayores prestaciones que las destinadas a ambientes urbanos.

En esta categoría encontramos precios muy variables que oscilan entre los 1600 y 6000€.

Ilustración 4 BH Atom - Precio: 1999€

3.1.2. BICICLETAS DE BARRA BAJA

Es así como se conocen a las clásicas bicicletas urbanas equipadas con una pequeña zona de carga, también existen versiones de pedaleo asistido de este tipo de vehículos cuyo fin es transportar objetos o niños con una silla dedicada a este fin.

Entre este tipo de vehículos son típicos los motores de 250W y baterías por debajo de los 500Wh. Sus precios oscilan entre los 1000 y 3200€

Ilustración 5 Legnano L260D - Precio: 999€

3.2. BICICLETAS PLEGABLES CON ACELERADOR

Se han vuelto especialmente populares en los últimos tiempos un tipo especial de bicicleta de menores prestaciones más pequeña y económica. Un hecho diferencial de estas es que algunos modelos cuentan con acelerador, esto se consigue aprovechando que debido a su pequeño tamaño cumplen sobradamente los requisitos para ser catalogado como VMP, aunque intuitivamente parezca un ciclo.

Son característicos los modelos de bajo coste importados desde China, los precios de las más populares oscilan entre los 500 y los 1000€.

Ilustración 6 FIIDO D2- Precio: 460€ (Importada)

3.3. KITS DE ELECTRIFICACIÓN

También están disponibles en el mercado una serie de kits para electrificar cualquier bicicleta de forma sencilla, suelen consistir en una rueda con el motor integrado para sustituir cualquiera de las que lleva, aunque también puede tratarse de un motor colocado sustituyendo el conjunto platos-pedales. La batería va directamente en la rueda en los kits más básicos o bien se coloca con anclajes en el lugar de la bolsa portaherramientas cuando esta es más grande, el precio en [6] oscila los 700-800€ para motores de 250W o bien puede importarse desde China por alrededor de 300€.

Ilustración 7 Uno de estos kits con todo incluido en la rueda - Precio: 290€ (importado)

3.4. CONCLUSIONES SOBRE EL MERCADO ACTUAL

Pese a que existen gran variedad de opciones la mayoría están sujetas a las restricciones respecto a los aceleradores, lo cual no hace el mercado menos interesante para los que buscan una forma de deporte más suave, pero quizás si afecte al tipo de público que busca un medio de transporte para ir y volver del trabajo.

Por otra parte, los precios comprando directamente en España son prohibitivos para una buena parte de la población, teniendo en cuenta además la probabilidad de que el vehículo sea robado, que ya de por si es alta en bicicletas comunes. Por otra parte, vemos como el precio baja drásticamente cuando el vehículo se importa directamente saltando los intermediarios, a pesar de que, en general, no disfrutan de la garantía de 2 años europea y que, en caso de ser parados en la aduana, habría que pagar un sobrecoste del 21% de IVA.

En un futuro cercano, es probable que, si la legislación comienza a regular correctamente este tipo de vehículos, al eliminarse la incertidumbre acerca de si podrán o no circular libremente su demanda y producción aumenten y se produzca una bajada de precio debido a la economía de escala.

4. DEFINICIÓN DEL PROTOTIPO

En este proyecto se va a proponer un vehículo con las siguientes características:

- El chasis del vehículo se corresponde con el de una bicicleta convencional.
- Dispondrá de un motor DC brushless integrado en la rueda delantera.
- Dispondrá de un microcontrolador que se encargará de manejar el motor.
- El sistema será alimentado con una batería compuesta de celdas de litio.
- Dispondrá de un selector con 4 modos de actuación:
    - *Apagado*: Se comporta como una bicicleta convencional.
    - *Recarga*: El sistema recarga las baterías a costa de un esfuerzo extra del conductor.
    - *Asistencia*: El conductor debe pedalear, pero realiza menos esfuerzo.
    - *Acelerador*: El motor actúa mediante el uso de un acelerador manual sin necesidad de pedalear.

Se presenta un diagrama del prototipo:

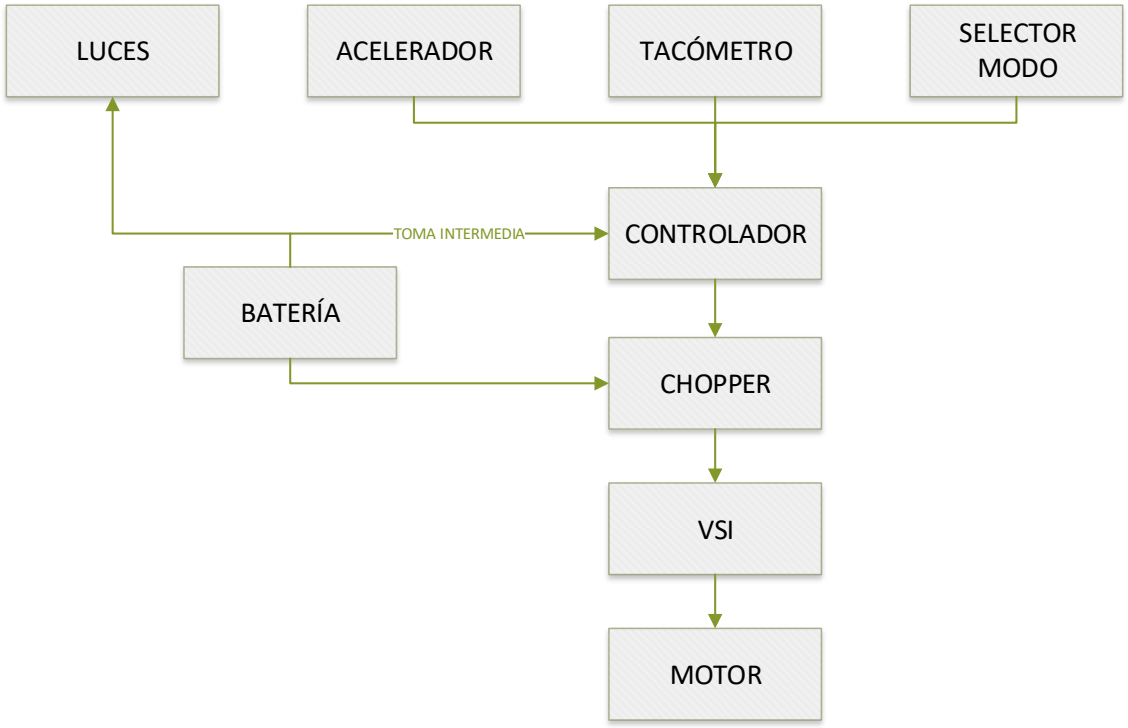


Ilustración 8 Diagrama del prototipo

5. MODELO ENERGÉTICO

Para diseñar el prototipo, es fundamental realizar un análisis a priori de las necesidades del vehículo, este proporcionará información vital para el proyecto, ya que, si está correctamente definido ayudará a elegir correctamente el motor, las baterías y la electrónica de potencia que debería montar el prototipo para cumplir con las características que se le piden, alejándonos lo más posible del, económicamente deficiente, sistema de prueba y error.

El modelo inicialmente iba a ser desarrollado utilizando MATLAB SIMULINK®, sin embargo, la necesidad de recursos informáticos que este necesita para funcionar de forma correcta superaba con creces los disponibles, por lo tanto, se ha optado por una solución basada en Python, que además de funcionar correctamente en ordenadores con pocos recursos, ha hecho del modelo un sistema modular más flexible y adaptable a otros tipos de vehículos, adicionalmente todo el software utilizado es libre y no requiere de subscripción, además se trata de un lenguaje con una buena curva de aprendizaje para personas que no se dedican profesionalmente a la programación.

El proceso de creación ha partido de una selección previa de los componentes, para posteriormente comprobar su validez, es decir los datos introducidos en el modelo son de componentes reales, la simulación se ha realizado para una rueda de 26", o lo que es lo mismo, un radio (r) de 0.33m y una relación de velocidad rueda-motor (G) = 1.

5.1. NOTAS ACLARATORIAS

Se mencionará a menudo a lo largo de la explicación del modelo el término *clase*, las clases en Python pueden entenderse como una definición de un objeto, al que se le asignan *propiedades* (variables) y *métodos* (funciones), son útiles en la programación orientada a objetos y tienen la ventaja de hacer el código más modular y menos confuso.

Otro concepto recurrente es el de módulo, al hablar de un módulo nos referimos a un archivo *".py",* que contiene texto plano, tal y como puede encontrarse en el Anexo 4, un paquete es una carpeta que contiene varios de estos módulos. La razón para crear estos módulos no es otra que dividir un problema complejo en otros más pequeños, lo que no solo hace el programa más fácil de hacer, sino también más fácil de mantener.

Las *listas* son objetos de Python, contienen un conjunto de *n* elementos de cualquier otra clase, pueden tener varias dimensiones (lista que contiene listas), estos estarán situados entre corchetes, y separados por comas.

Como norma general las variables constantes tienen su nombre escrito en mayúsculas, las clases solo la primera letra, y las funciones se escriben en minúsculas.

5.2. FUNCIONAMIENTO DEL MODELO

Los distintos módulos que componen el sistema son coordinados por la *Clase Modelo*, la cual contiene una instancia de todas las demás clases que serán explicadas con más detalle en sucesivos apartados, para explicar el funcionamiento del sistema completo se utilizará el siguiente diagrama de bloques a modo de apoyo:

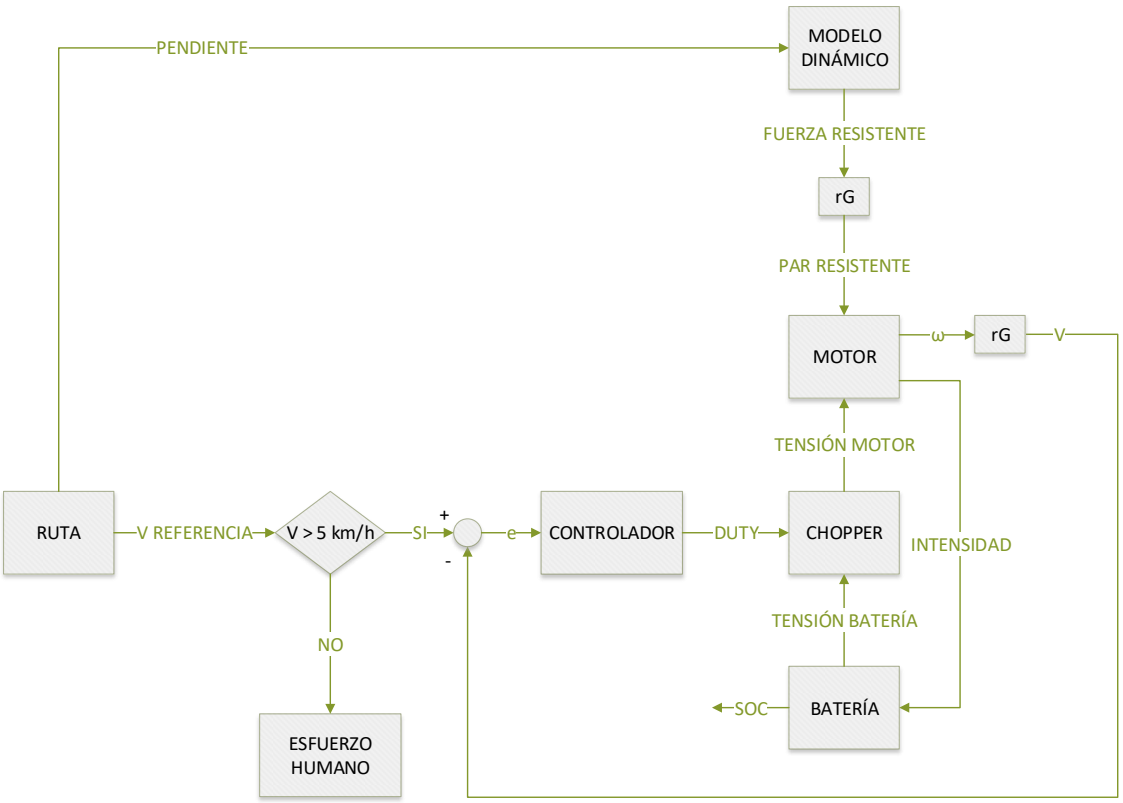


Ilustración 9 Diagrama representativo del funcionamiento del modelo

La *Clase Modelo* tan solo contiene un método, llamado *simular*, el cual consiste en un bucle que recorre los valores de la consigna.

La consigna, es decir los valores de velocidad que se le piden al sistema en cada momento, así como la pendiente que le afecta son designados por la ruta, el objeto encargado de manejar estos datos, lo siguiente es realizar una comparación, si la velocidad actual es menor a 5 km/h el vehículo no arrancará, es decir se necesita un pequeño impulso para que empiece a funcionar, esto se hace para evitar que se produzcan picos bruscos de corriente en el arranque.

Con la velocidad de referencia y la actual obtenemos el error, que será analizado por el controlador, que tal y como se verá más adelante se tratará de un PI.

El controlador arroja como resultado el valor del *duty cycle*, el cual en un sistema real sería la señal de entrada de un generador PWM que, a su vez, controlaría el *chopper*, sin embargo, a efectos de cálculo tan solo tendremos en cuenta lo siguiente:

$$V_{motor} = V_{batería} * D$$

$$I_{batería} = I_{motor} * D$$

Es decir, a partir del *duty* junto con la tensión de la batería obtenemos directamente la tensión de entrada al motor.

La otra entrada del motor es el par resistente que debe vencer, que será calculado a partir de la fuerza que calcula el *modelo dinámico*, el proceso se detallará en su correspondiente apartado.

A partir de estos datos, el motor calcula la velocidad angular, que será convertida a velocidad lineal y devuelta al controlador para la siguiente iteración, también arroja como resultado el valor de intensidad, siendo este positivo cuando es consumida y negativo cuando es generada, este valor es enviado al módulo de la batería, que lo utiliza para calcular el *State Of Charge (SOC)*, que es como se denomina a la capacidad restante de la batería en tanto por ciento.

En cada iteración se añaden los resultados de los cálculos al objeto encargado de guardarlos, los valores que guarda son:

- Velocidad real y de consigna
- Fuerzas total y resistente
- Fuerza de rozamiento, aerodinámica, gravitatoria, de aceleración lineal y angular
- Par total
- SOC
- Tensiones e intensidades de la batería y el motor
- Potencias eléctrica y mecánica
- Distancia recorrida

Estos resultados se almacenan en vectores con un valor por iteración.

El modelo ha sido diseñado para simular el comportamiento funcionando con el *Modo Acelerador*, por ser este el energéticamente más exigente, en la práctica el conductor puede pedalear en cualquiera de los modos de operación, lo que implicaría disponer de mayor potencia y una mayor autonomía.

Por defecto la frecuencia de muestreo es de 100Hz, es decir se realiza un cálculo cada 10ms, se recomienda usar esta frecuencia ya que aumentarla aumenta también el tiempo de cálculo y no produce cambios importantes en los resultados.

Se ha despreciado para la simulación el consumo del controlador y las luces.

5.3. RUTAS

Las rutas son consignas de velocidad y pendiente, en este caso, para adecuarnos lo máximo posible a la realidad se van a utilizar datos grabados mediante la aplicación de Android *GPS Logger*.

En el estudio se trabaja con un total de 4 rutas, las 4 han sido obtenidos circulando dentro de la zona urbana de Zaragoza, las rutas 1, 2 y 3, se corresponden a un tráfico normal, en horas diurnas (junto a otras bicicletas y peatones por el carril bici y con frecuentes paradas en semáforos) y respetando las normas de circulación, por este motivo son especialmente útiles en las pruebas de autonomía, ya que arrojarán resultados realistas. En la ruta número 4 se ha buscado forzar al sistema mediante aceleraciones y deceleraciones bruscas, supone una buena medida de las cualidades del aparato. El vehículo utilizado para estas rutas ha sido un patinete eléctrico con motor de 250W, por lo que su desempeño debería ser similar al que buscamos en nuestro vehículo.

Se ha tenido en cuenta la altura en cada punto para considerar el efecto de la pendiente en los cálculos, pese a todo, se comprueba que, debido a que Zaragoza es una ciudad principalmente llana, el efecto de las pendientes no afecta demasiado a las medidas.

En caso de necesitar datos más precisos podría resultar interesante medir la velocidad del viento en contra, así como la temperatura, valor que afecta a la capacidad de las baterías.

En el modelo, las rutas están contenidas en un objeto de la *Clase Ruta*, la cual requiere como parámetro el nombre de dicha ruta, que debe de estar alojada por defecto en la carpeta *datos/rutas* en formato *csv*, los archivos alojados en esta carpeta no son directamente los que arroja la aplicación *GPS Logger*, si no una versión limpia de estos en la que solo se almacenan la velocidad, en m/s y la altura respecto al nivel del mar, en metros, cada fila de estos datos representa un tiempo configurable con la variable *PASO_CONSIGNA*, en el fichero de configuración. Esta limpieza de datos se realiza con el módulo *arreglar_rutas* explicado más tarde.

También se ha añadido la posibilidad de simular una entrada escalón, útil para determinar la respuesta dinámica del sistema, así como, mediante la simulación de un escalón largo, el consumo correspondiente a una velocidad constante, se define escribiendo el nombre de la ruta en formato *"escalon-t1-v1-t2-v2",* donde t1 y t2 son el tiempo que se circula a la velocidad 1 y 2 respectivamente, ambas velocidades en km/h.

Para facilitar el uso de este modelo se han añadido a la *Clase Ruta* algunas de las propiedades del objeto lista de Python, como son la posibilidad de indexarla, ej. al escribir ruta[10:50] nos devolverá los segundos 10 a 50 de la ruta, también tenemos la posibilidad de escoger un solo elemento con ruta[i], o de escoger datos salteados con la forma ruta[inicio:fin:salto]. Otras propiedades añadidas son la de obtener la longitud de la ruta con el método len(ruta) o la de repetir la ruta n veces escribiendo ruta * n, esto

último es útil cuando se quiere estudiar el comportamiento del sistema a medida que se agota la batería.

En el Anexo 1 se puede encontrar una representación de los datos de las 4 rutas utilizadas.

5.4. MODELO DINAMICO

El modelo dinámico se ha diseñado como un módulo dentro del simulador, su función es la de realizar los cálculos correspondientes a las necesidades energéticas del modelo. La *clase ModeloDinamico* recibe como entradas la pendiente (m), la velocidad del intervalo anterior (v_ant), y la velocidad que se acaba de alcanzar (v). Como salidas tiene una propiedad llamada fuerzas, que calcula y devuelve un diccionario con todas las fuerzas explicadas más abajo, y una función llamada calcular_J, que debe de ser llamada una única vez después de haber escogido la batería y el motor, para así tener en cuenta las masas de estos en el cálculo.

5.4.1. CÁLCULO DEL MOMENTO DE INERCIA DEL SISTEMA

Para calcular este valor se ha sumado la inercia provocada por la masa del vehículo transportada al motor, así como la masa giratoria de las ruedas y de sí mismo, resultando en:

$$J_{total} = J_{vehiculo} + J_{ruedas} + J_{motor}$$

A partir de la energía cinética obtenemos:

$$J_{vehiculo} = m_{total} * r_{rueda}^2 * G^2$$

Que, para una persona de 75 kg, estimando el peso del vehículo en 15 queda:

$$J_{vehiculo} = 90 * 0.33^2 = 9.801\ kg \cdot m^2$$

Las ruedas las consideramos como cilindros huecos de 2kg de masa

$$J_{ruedas} = 2 * \frac{1}{2} m_{ruedas} * r_{rueda}^2 * G^2 = 2 * 0.33^2 = 0.2178\ kg \cdot m^2$$

Para el motor se ha aproximado utilizando toda su masa y tratándolo como un cilindro macizo.

$$J_{motor} = m_{motor} * r_{motor}^2 = 2.8 * \frac{0.135^2}{4} = 0.0128\ kg \cdot m^2$$

Luego nos queda:

$$J_{total} = 9.801 + 0.2178 + 0.0128 = 10.03\ kg \cdot m^2$$

5.4.2. CÁLCULO DE LAS FUERZAS

Este modelo tiene en cuenta 5 tipos de fuerzas, 3 de las cuales, rozamiento, aerodinámica y gravitatoria, constituyen la fuerza resistente, a la que sumadas las fuerzas de aceleración lineal y angular constituyen la fuerza total.

1. FUERZA DE ACELERACIÓN LINEAL

    - Es la mayor con diferencia, ya que, debido a las características de la circulación urbana, con múltiples aceleraciones rápidas, constituye un factor clave en las necesidades de potencia, ya que una fuerte aceleración implica un alto par y a su vez una alta corriente, esta última si no es debidamente controlada puede ocasionar daños irreversibles en los componentes del vehículo.

    - Si bien esta fuerza es en cierta medida controlable por medio del regulador, una aceleración demasiado baja haría perder utilidad al vehículo, por lo tanto, se necesita alcanzar un equilibrio entre prestaciones y potencia, la cual, además de afectar directamente al presupuesto, también se ve afectada por la normativa, tema que se abordará en su correspondiente apartado.

    - Cabe mencionar que cuando la aceleración es negativa hablamos de deceleración, esta energía es aprovechada mediante la característica de los motores eléctricos de ejercer un frenado regenerativo, de la cual se hablará más adelante.

    - Se calcula mediante la siguiente fórmula: $Fal = m * a$

Es de especial importancia el apartado de la masa, que dado el poco peso del vehículo depende en gran medida del peso del conductor.

2. FUERZA DE ROZAMIENTO

    - Hace referencia a la resistencia a la rodadura del vehículo con el suelo, se modela de la siguiente forma: $Froz = c_{roz} * m * g$

    - Donde c_{roz} es el coeficiente de rozamiento con el suelo, este ha sido obtenido mediante una utilidad disponible en [7], dándosele un valor de 0.004.

3. FUERZA AERODINÁMICA

- Esta fuerza se modela de la siguiente forma:
- $Fae = \frac{1}{2} * \rho * A * c_{arrastre} * (v + v_{aire})^2$
- Como podemos comprobar depende de diversos factores:
  - ρ: Densidad del aire, es un valor fijo, 1.25 kg/m^2
  - A: Área frontal, en nuestro caso depende casi enteramente del conductor, en el artículo que se puede encontrar en [8] la estiman entre 0.3 y 0.6 m^2 dependiendo de la posición que adopte el ciclista, como en nuestro caso la posición es de paseo y por lo tanto bastante erguida, lo hemos estimado en 0.5 m^2.
  - $c_{arrastre}$: Cuantifica la resistencia que opone el cuerpo contra un medio fluido como es el aire, al igual que con el área obtenemos su valor de [8], que estimamos en 0.88.
  - v: Velocidad del vehículo, el hecho de que el vehículo está diseñado para bajas velocidades hace que esta fuerza tenga menos importancia de la que tiene en coches o camiones.
  - v_{aire}: Indica la velocidad del aire directamente en contra del ciclista, no ha sido cuantificada por lo que se pasa como parámetro variable.

4. FUERZA GRAVITATORIA

- Es la que depende directamente de la pendiente a la que se esté enfrentando el vehículo, se modela de la siguiente forma: $Fg = m * g * \sin(\alpha)$.
- Donde α es el ángulo de la pendiente, siendo positivo cuando es hacia arriba y negativo cuando es hacia abajo.
- En lugares con pendientes pronunciadas puede ser un factor importante, sin embargo, en lugares llanos pasa a segundo plano.

5. FUERZA DE ACELERACIÓN ANGULAR

- Representa la fuerza de las masas inerciales en movimiento, se modela de la siguiente forma: $Faa = I * a * \frac{G^2}{r^2}$.
- Aquí I se corresponde con el momento de inercia que hemos estimado a partir de las llantas (peso de en torno a los 2kg) tratándolas como cilindros huecos, y añadiéndole la inercia del motor.
- G es la relación de transmisión entre el motor y la rueda, que en nuestro caso es 1, y r es el radio de la rueda, que para una de 26” son 0.33m.

5.5. MOTOR

Este módulo ha sido diseñado pensando en la posibilidad de adaptarlo a diferentes motores, para ello, en el caso de querer estudiar un motor diferente habría que añadir un fichero con la denominación que queramos darle al motor, utilizando como plantilla el que ya existe para el motor MXUS-XF08 en la carpeta *modelo/datos/motores*.

5.5.1. CÁLCULO DE LOS PARÁMETROS

Uno de los problemas a enfrentar en la realización de este modelo ha sido la falta de datos precisos acerca de estos motores debido a los escuetos datasheet proporcionados por los fabricantes, por ello se han tenido que tomar algunas suposiciones como ciertas que ahora se detallarán.

Los datos sobre el motor escogido se encuentran en el anexo.

CÁLCULO DE LA K

La constante del motor ha sido relativamente fácil de calcular, ya que el datasheet incluía los valores de par (M) y corriente (I) nominales, recordemos que en un motor brushless, debido a que el flujo es generado por imanes y por lo tanto constante, la constante de par y la de velocidad son iguales.

$$K = \frac{M}{I} = \frac{12.69}{9.448} = 1.3431$$

Como puede comprobarse, esta K tiene unidades de N·m/A o V·s.

CÁLCULO DE LA R

La resistencia ha sido calculada utilizando el rendimiento, como el que ofrece el datasheet es el total, se ha asumido que este se divide a partes iguales entre el rendimiento eléctrico y el rendimiento mecánico, es decir:

$$\eta_{elctrico} = \eta_{mecanico} = \sqrt{\eta_{total}} = \sqrt{0.78} = 0.8831$$

A partir de estos datos se ha calculado de la siguiente manera:

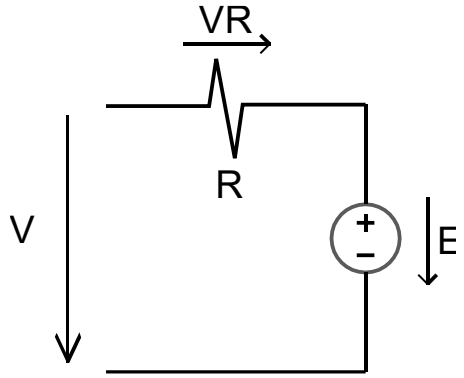
$$R = (1 - \eta_{electrico}) * \frac{P}{\eta_{total}} * \frac{1}{I^2} = (1 - 0.8831) * \frac{265.7}{0.78} * \frac{1}{9.448^2} = 0.4458\ \Omega$$

Donde P e I son la potencia útil nominal y corriente nominal respectivamente.

5.5.2. CÁLCULO DE LA VELOCIDAD

El motor posee un método utilizado para calcular la velocidad angular (ω) llamado *calcular_w*, este se basa en lo siguiente:

A la tensión que le llega al motor se le resta la E del motor, calculada a partir de la velocidad angular y la constante del motor, de esta manera obtenemos la tensión en la resistencia tal y como se puede apreciar en el siguiente circuito equivalente:



Como se puede apreciar, en este esquema se ha omitido la parte inductiva del motor, esta suposición ha sido tomada debido a la falta de datos reales sobre esta u otros que nos permitieran sacarla indirectamente, de todas formas, se trata de un error asumible en un motor de estas características.

Del circuito de arriba obtenemos la fórmula de la intensidad:

$$I = \frac{V - E}{R}$$

Recordando que:

$$E = K * \omega$$

En el programa la I se calcula como propiedad para mejorar la limpieza del mismo.

Una vez obtenida la corriente, calculamos el par eléctrico a partir de la siguiente ecuación de los motores de corriente continua con excitación permanente:

$$Me = K * I$$

Aplicando la segunda ley de Newton a nuestro caso tenemos:

$$Mac = Me - Mr$$

$$J\frac{d\omega}{dt} = Me - Mr$$

$$\omega = \int_0^t \left(\frac{(Me - Mr)}{J}\right)$$

Donde Mr es el par resistente y J es el momento de inercia de todo el sistema, ambos datos obtenidos del modelo dinámico.

Esta será la velocidad angular de salida, que al multiplicarla por el radio y por la relacción de transmisión G nos dará la velocidad lineal.

Lo explicado en este apartado puede resumirse mediante el siguiente diagrama:

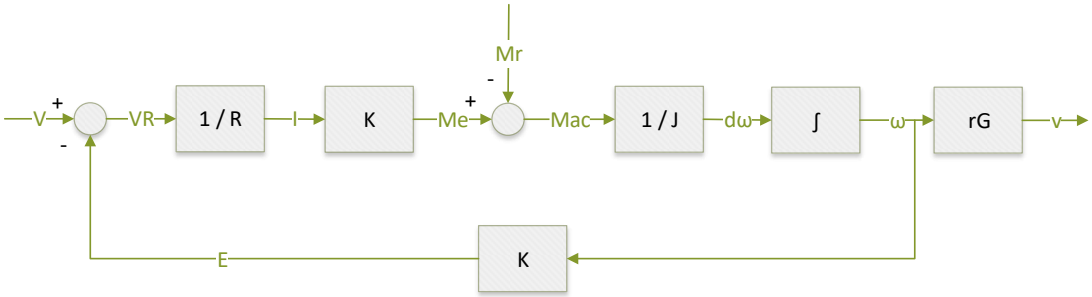


Ilustración 10 Planta del motor

Como medida a adoptar en un futuro para mejorar la precisión del modelo, se podrían utilizar los valores del datasheet del motor para sustituir la constante del motor, que según se puede deducir a partir de los datos, deja de ser constante a partir de los 20 Nm de par.

5.6. CONTROLADOR

El controlador escogido para el proyecto ha sido un controlador PI en paralelo, que en la implementación del código se basa en la *Clase PID*, a la que se le asigna una Kd = 0, ya que esta no aporta ningún beneficio claro al sistema.

De esta forma la ecuación del controlador en el dominio complejo queda de la siguiente forma:

$$C(s) = Kp + \frac{Ki}{s}$$

Es decir, el controlador calcula la salida a partir de la señal de error de la siguiente manera:

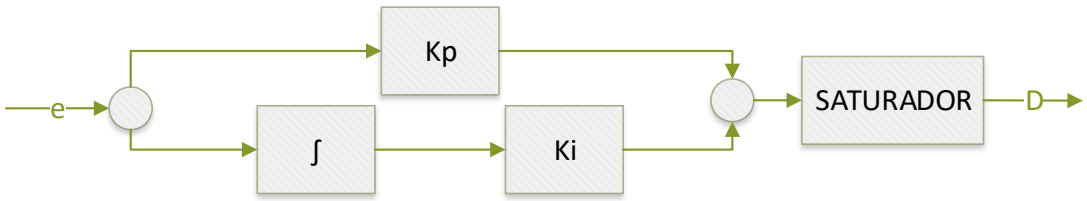


Ilustración 11 Controlador

Para calcular los valores de Kp y Ki se ha seguido el método analítico conocido como asignación de polos, la idea es ubicar los polos en el lugar deseado por medio de los parámetros del controlador, el primer paso será calcular la planta del motor en lazo abierto, para lo cual nos basaremos en el esquema presentado en el apartado del motor, pero esta vez el análisis será en el dominio complejo.

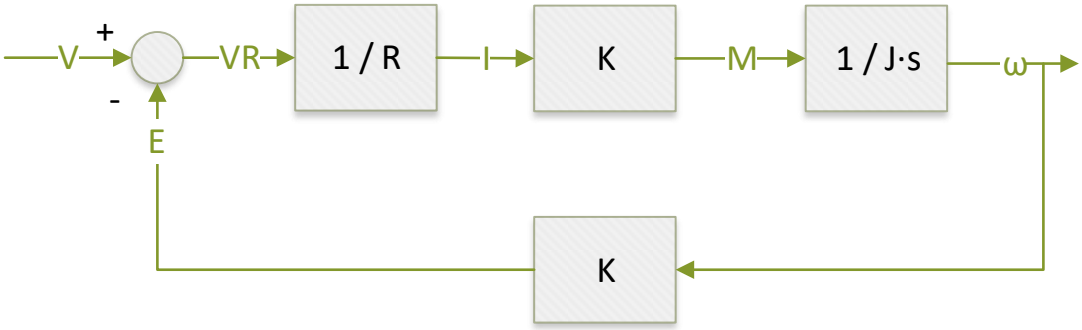


Ilustración 12 Planta del motor en el dominio complejo

De aquí obtenemos la planta en lazo abierto como:

$$G_{LA}(s) = \frac{\omega(s)}{V(s)} = \frac{\frac{K}{RJs}}{1 + \frac{K}{RJs}} = \frac{\frac{1}{K}}{\frac{RJ}{K^2}s + 1}$$

El segundo paso será calcular la función de transferencia del sistema en lazo cerrado, representado por el siguiente diagrama, donde *Vbat* es la tensión de la batería:

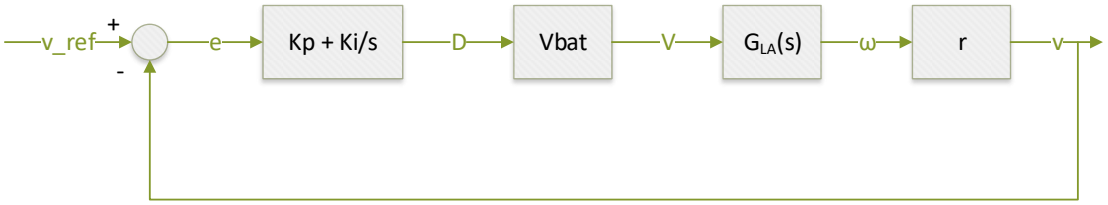


Ilustración 13 Lazo de control

Luego la ganancia en lazo cerrado queda:

$$G_{LC}(s) = \frac{\left(Kp + \frac{Ki}{s}\right) * Vbat * G_{LA}(s) * r}{1 + \left(Kp + \frac{Ki}{s}\right) * Vbat * G_{LA}(s) * r}$$

De esta ecuación nos interesa el denominador, conocido como ecuación característica (EC).

$$EC = 1 + \left(Kp + \frac{Ki}{s}\right) * Vbat * G_{LA}(s) * r = \frac{1 + \left(Kp + \frac{Ki}{s}\right) * \frac{Vbat * r}{K}}{\frac{RJ}{K^2}s + 1} = 0 \rightarrow$$

$$EC = \frac{RJ}{K^2}s + 1 + \left(Kp + \frac{Ki}{s}\right) * \frac{Vbat * r}{K} = 0 \rightarrow$$

$$EC = s^2 + \frac{K^2 + K * Kp * Vbat * r}{RJ}s + \frac{K * Ki * Vbat * r}{RJ} = 0$$

Esta ultima ecuación tiene la forma:

$$s^2 + 2\zeta\omega_n s + \omega_n^2 = 0$$

De donde despejamos Kp y Ki:

$$Kp = \frac{2 * \zeta * \omega_{n*} R * J - K^2}{K * Vbat * r}$$

$$Ki = \frac{\omega_n^2 * R * J}{K * Vbat * r}$$

El factor ζ puede obtenerse a partir del factor de amortiguamiento (Mp) deseado, despues de simular el modelo con diferentes valores un 5% parece un valor razonable.

$$\zeta = \sqrt{\frac{\ln(Mp)^2}{\pi^2 + \ln(Mp)^2}} = \sqrt{\frac{\ln(0.05)^2}{\pi^2 + \ln(0.05)^2}} = 0.6901$$

La frecuencia natural del sistema, para una banda de tolerancia de un 5% se define como:

$$\omega_n = \frac{3}{\zeta Tss}$$

Donde Tss es el tiempo de establecimiento, una vez escogido el sistema intentará estabilizarse antes de que pase dicho tiempo.

En general, bajar este tiempo, resultará en un controlador con mejor respuesta dinámica, a cambio de un mayor consumo de corriente, por lo que es necesario encontrar una situación de equilibrio.

Tras simular varios valores un Tss de 8 parece adecuado para nuestro sistema, en el apartado de resultados veremos lo que eso supone.

Calculando la frecuencia natural de esta manera obtenemos:

$$\omega_n = \frac{3}{\zeta Tss} = \frac{3}{0.69 * 8} = 0.5434$$

El resto de valores necesarios son:

- R = 0.4458 Ω
- K = 1.3431
- J = 10.03 kg·m^2
- Vbat, supuesta en 36V, la tensión nominal de la batería
- r = 0.33m

Y con todo lo necesario para calcular las constantes Kp y Ki procedemos a su cálculo:

$$Kp = \frac{2 * 0.6904 * 0.5434 * 0.4458 * 10.03 - 1.3431^2}{1.3431 * 36 * 0.33} = 0.0971$$

$$Ki = \frac{0.5434^2 * 0.4458 * 10.03}{1.3431 * 36 * 0.33} = 0.0827$$

5.7. BATERÍA

De forma similar al módulo del motor, el modelo de la batería también está diseñado para ser capaz de trabajar con distintos tipos de celdas, aunque esta optimizado para ajustarse a las celdas de litio. El modelo cuenta con los datos para la celda NCR18650B de Panasonic, para añadir otros tipos de celdas bastaría con utilizar como plantilla la capeta que se encuentra en datos/celdas y modificar los archivos datos.txt y coeficientes.txt con los datos de la nueva celda, así como el nombre de la carpeta para poder así acceder a la celda desde el modelo.

La batería estará compuesta por un numero definido por el usuario de celdas en serie y ramas en paralelo. Se creará una nueva mediante la *Clase Bateria.*

El módulo, que recibe como entrada la intensidad requerida por el sistema, calcula internamente el State Of Charge (SOC), a partir de la carga que ha sido consumida y la capacidad nominal. Previamente se aplican correctamente los rendimientos en función de si se está consumiendo energía o recuperándola.

Por otra parte, se calcula la tensión, a partir de dicha carga consumida, la función que relaciona estas dos propiedades ha sido obtenida a partir de una regresión polinómica de cuarto orden, ajustada con los parámetros que el fabricante presenta en una gráfica. Esta función también es dependiente del coeficiente de descarga de la batería, es decir, la intensidad media en relación con la intensidad que descargaría la batería en una hora, el fabricante proporciona datos para 0.2C, 0.5C, 1C y 2C.

Es importante tener en cuenta que, por seguridad, las baterías de litio tienen una tensión máxima y mínima que debe respetarse, así como corrientes máximas de carga y descarga, que de incumplirse podrían comprometer el funcionamiento de la batería e incluso la seguridad de esta. El funcionamiento se resume en el siguiente diagrama:

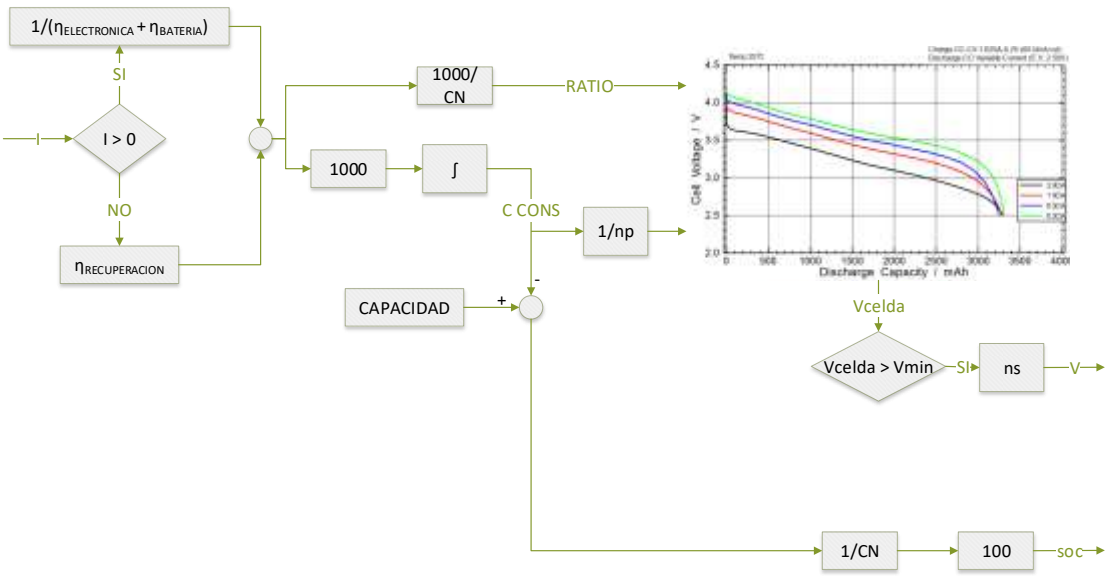


Ilustración 14 Diagrama de funcionamiento del módulo batería

Queda pendiente añadir al módulo los efectos de la temperatura.

5.8. OTROS MODULOS

El programa incluye otros módulos que realizan funciones auxiliares para el modelo, estos son:

5.8.1. resultados

Este módulo consiste en una sola clase que contiene listas con los resultados de interés, también contiene un método llamado trazar para imprimir cualquiera de estos resultados en forma de gráfica, así como el método *guardar_excel* para exportar todos estos datos en formato Excel.

5.8.2. arreglar_rutas:

Su función es modificar el formato de los datos arrojados por la aplicación utilizada para grabar las rutas. Consiste en una serie de órdenes que extraen los datos de velocidad, los de altitud son calculados utilizando la latitud y longitud mediante la API de Bing Maps [9] que es posible utilizar con una cuenta de Microsoft.

5.8.3. regresión_celdas

Utiliza un fichero llamado datos_descarga.csv que debe encontrarse en la carpeta de datos de la celda, donde deben introducirse los datos de tensión en función de la capacidad consumida, para añadir soporte a otras celdas debe utilizarse la existente para la celda NCR18650B como plantilla, el resultado es un archivo *txt* con los coeficientes correspondientes a una regresión polinómica de cuarto grado, que es la que mejor se adapta a la curva típica de descarga de una celda de litio. Los coeficientes de la regresión han sido calculados utilizando la utilidad *polyfit*, disponible en el módulo *numpy*.

5.8.4. funciones_utiles

Este módulo se creó con la intención de alojar clases y funciones que utilizan varios de los módulos del proyecto, contiene la *Clase Cronometro*, que se utiliza como medida del tiempo restante de simulación, útil si se utiliza una frecuencia de muestreo muy alta, la función datos_txt, que se utiliza para obtener datos guardados en un archivo *txt* con un formato específico del proyecto, por último, la función saturador limita la señal de entrada acorde con los valores *min* y *max* definidos.

5.8.5. Manejar_resultados

Se trata de una utilidad preparada para ser llamada por el módulo resultados, es capaz de imprimir un solo valor, o realizar gráficas en 2d y 3d, en el caso de las primeras se pueden añadir varios valores a imprimir, un ejemplo de la sintaxis a emplear puede encontrarse en el archivo *run.py* añadido en el Anexo 4. También puede guardar los datos en formato xlsx.

6. RESULTADOS

6.1. SEÑAL DE ESCALÓN

6.1.1. RESPUESTA DINÁMICA

Un baremo útil para valorar al controlador es ver la respuesta al escalón, en este caso se ha provocado un escalón de 0 a 25 km/h, teniendo en cuenta que es condición del modelo que el conductor hace todo el trabajo hasta alcanzar los 5 km/h.

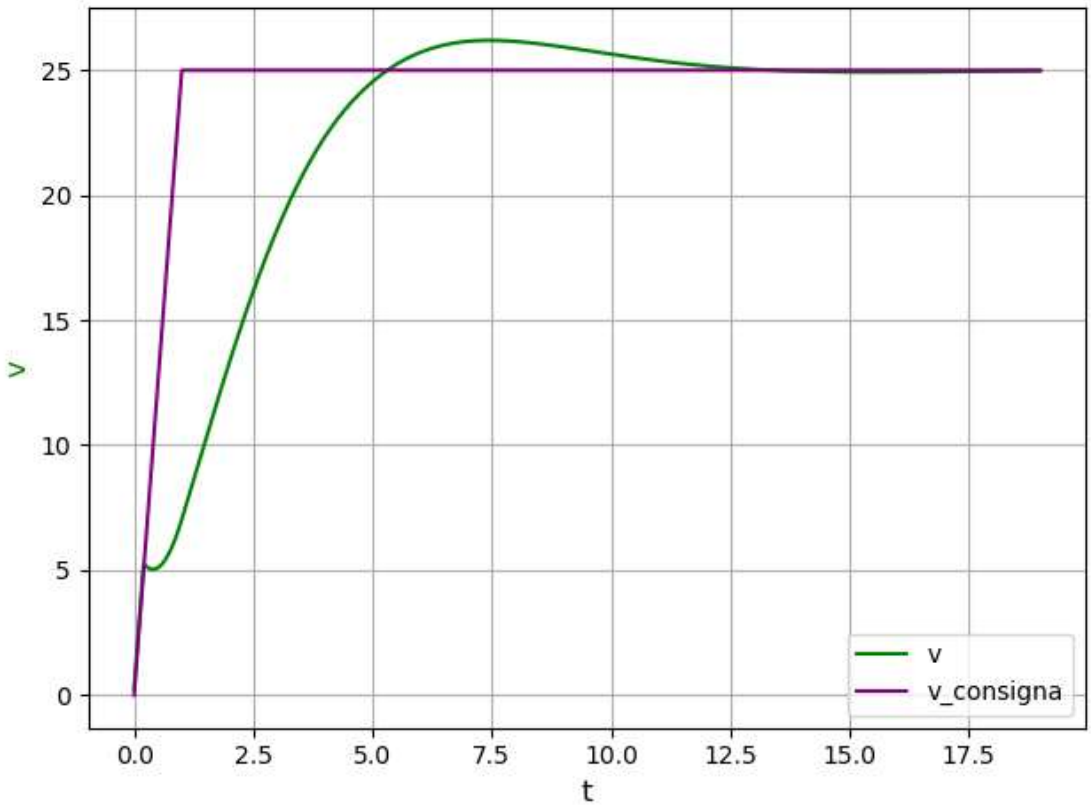


Ilustración 15 Respuesta al escalón

Como se puede observar la oscilación no sobrepasa el 5% (26.25 km/h) y alcanza la estabilidad aproximadamente a los 8 segundos, estos eran los valores requeridos cuando se ajustó el controlador PI.

6.1.2. AUTONOMÍA A VELOCIDAD CONSTANTE

Para realizar este test se ha metido como entrada un escalón de 10 minutos de duración, el modelo calcula automáticamente cual es el consumo por kilómetro y lo arroja como resultado.

La energía por kilómetro resultante es de 8.8 Wh/km lo que con la batería escogida de 360 Wh nos da una autonomía de 41 km.

6.2. RESULTADOS PARA LA RUTA 4

La ruta 4 se realizó forzando al sistema a acelerar y frenar bruscamente en cortos periodos de tiempo, por lo que es especialmente útil para ver la respuesta del controlador.

6.2.1. VELOCIDAD

Se observa que los resultados son bastante aproximados a la consigna, diferencia imperceptible para un usuario del vehículo.

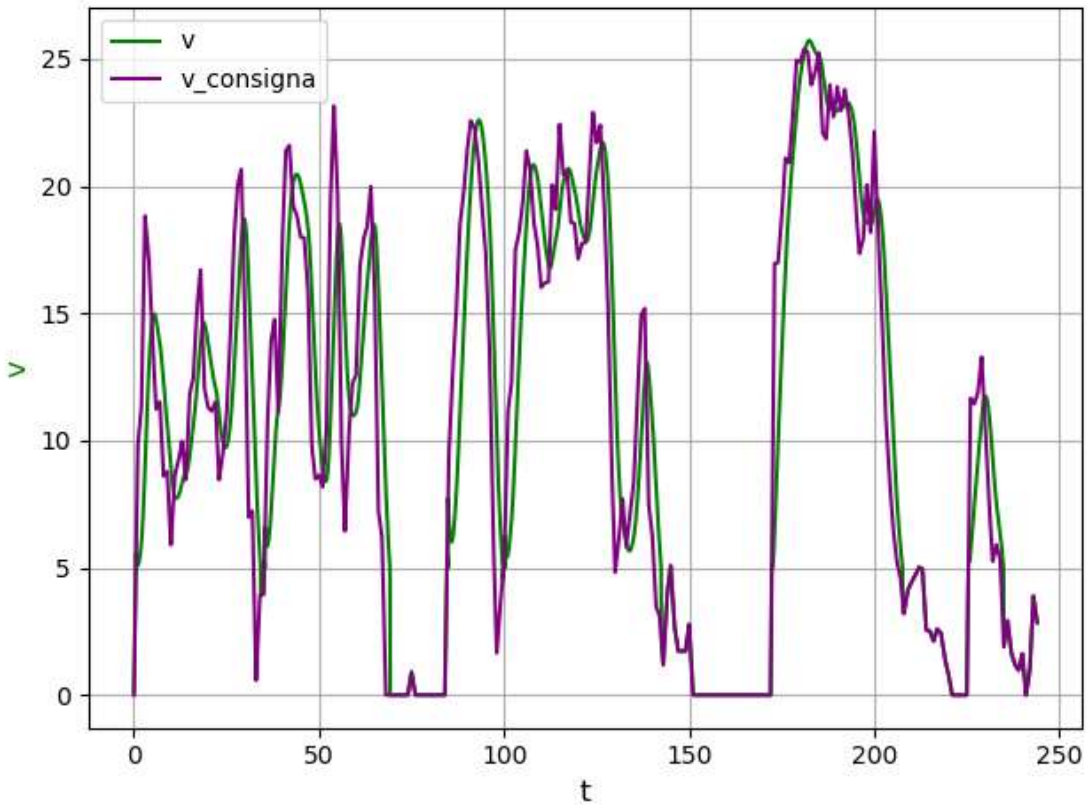


Ilustración 16 Velocidades real y de consigna

6.2.2. INTENSIDAD

Se han registrado la corriente en el motor y en la batería, son las siguientes:

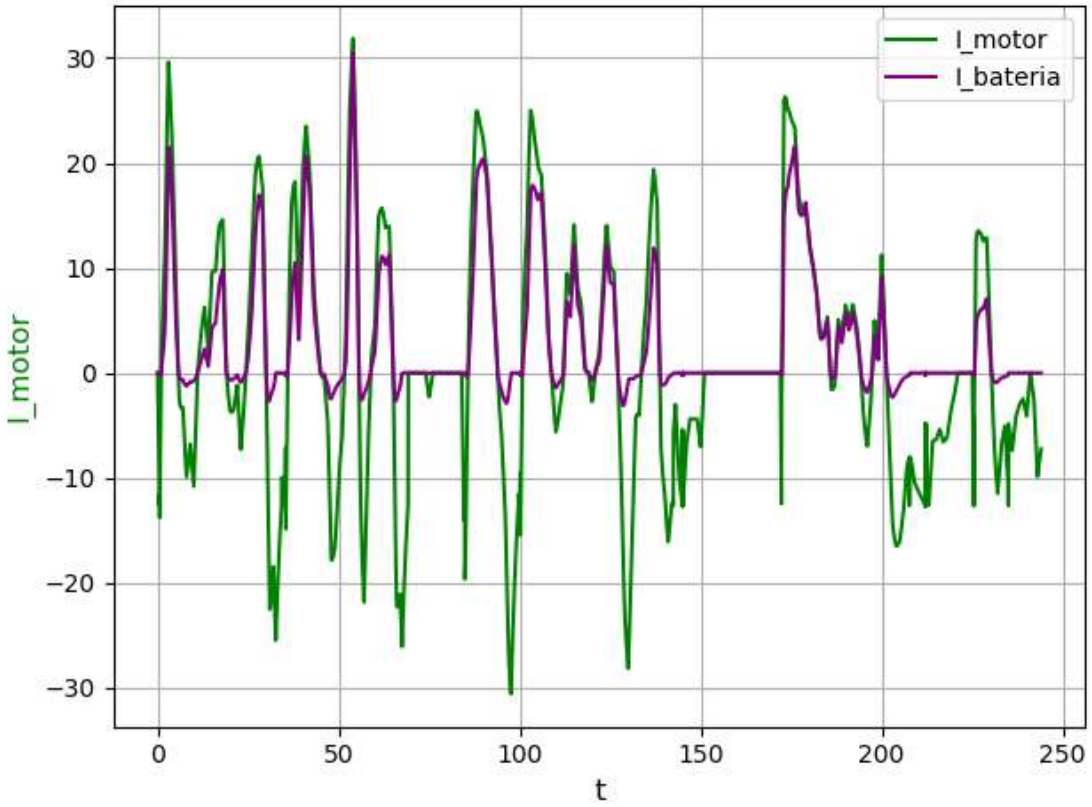


Ilustración 17 Corrientes en el motor y la batería

A partir de la intensidad en la batería se dedujo que colocar dos ramas en paralelo no era suficiente, ya que cada rama no debería de suministrar más de 6.7A por un tiempo prolongado, por ese motivo se pusieron 3 ramas a pesar de no estar buscando una mayor autonomía de la que se conseguía con 2.

De la relación entre la intensidad de la batería y la del motor se dedujo que se aprovecha una parte muy pequeña de la corriente negativa para cargar la batería, la manera de aumentar este valor se explica en el Anexo 6, lo que se explica en este anexo es aplicable al prototipo, aunque no al modelo, ya que no existen datos del uso del freno mecánico, necesario si quiere conseguirse un buen aprovechamiento de la batería.

Los valores de intensidad en el bus no llegan a superar los 30A, valor a tener en cuenta al seleccionar la electrónica de potencia.

6.2.3. SOC Y AUTONOMÍA

Al tratarse de una ruta de apenas 712 metros la variación en el SOC es de poco más del 2%, sin embargo, al poner la curva junto a la velocidad se aprecia como esta desciende más rápidamente cuanto mayor es la velocidad, y permanece constante en los momentos en los que está parado.

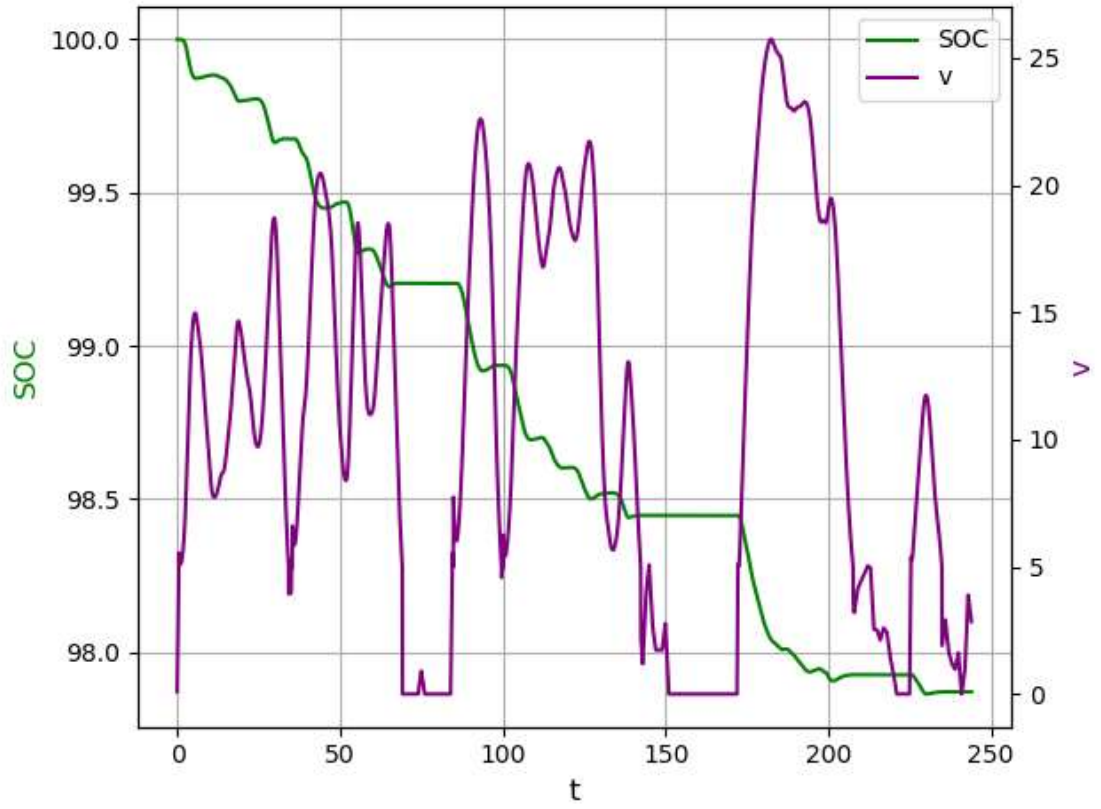


Ilustración 18 SOC y velocidad

El consumo por kilómetro en esta ruta es de 12 Wh/km, lo que nos dota de una autonomía en estas condiciones de 30 km.

6.2.4. DISTRIBUCIÓN DE LAS FUERZAS

Se analizan las fuerzas de aceleración lineal (f_al), de aceleración angular (f_aa), rozamiento (f_roz), aerodinámica (f_ae) y gravitatoria (f_g), para determinar cuáles son las dominantes.

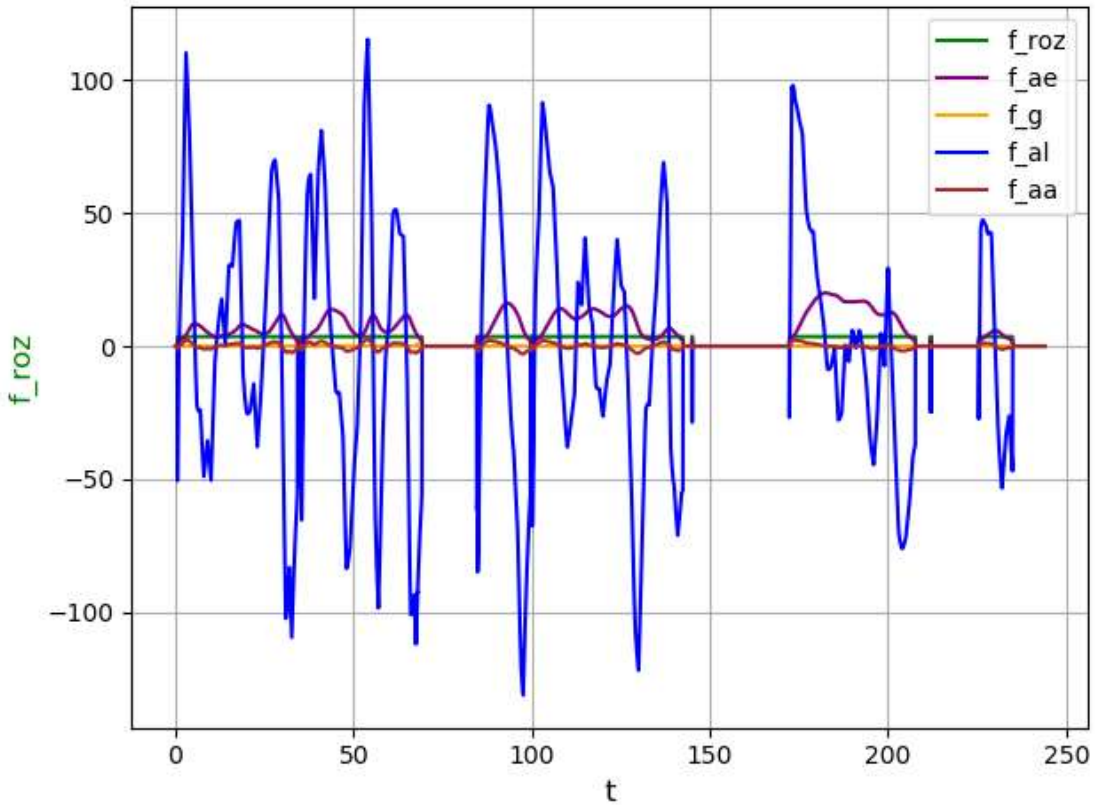


Ilustración 19 Fuerzas

Como puede observarse la fuerza dominante con diferencia es la de aceleración lineal, seguida por la resistencia aerodinámica seguida de la fuerza de rozamiento y la aceleración angular, como la altura se mantuvo constante a lo largo de toda la ruta (la API de la cual se obtuvieron estos datos redondea a la unidad), la fuerza gravitatoria es nula en este caso.

6.2.5. PAR Y TENSIÓN DE LA BATERÍA

Se ha querido colocar estas dos gráficas juntas verticalmente para que se pueda apreciar la relación inversa que existe entre ellos, ya que el par provoca un aumento directo del C-rate, relacionado con la tensión de la batería.

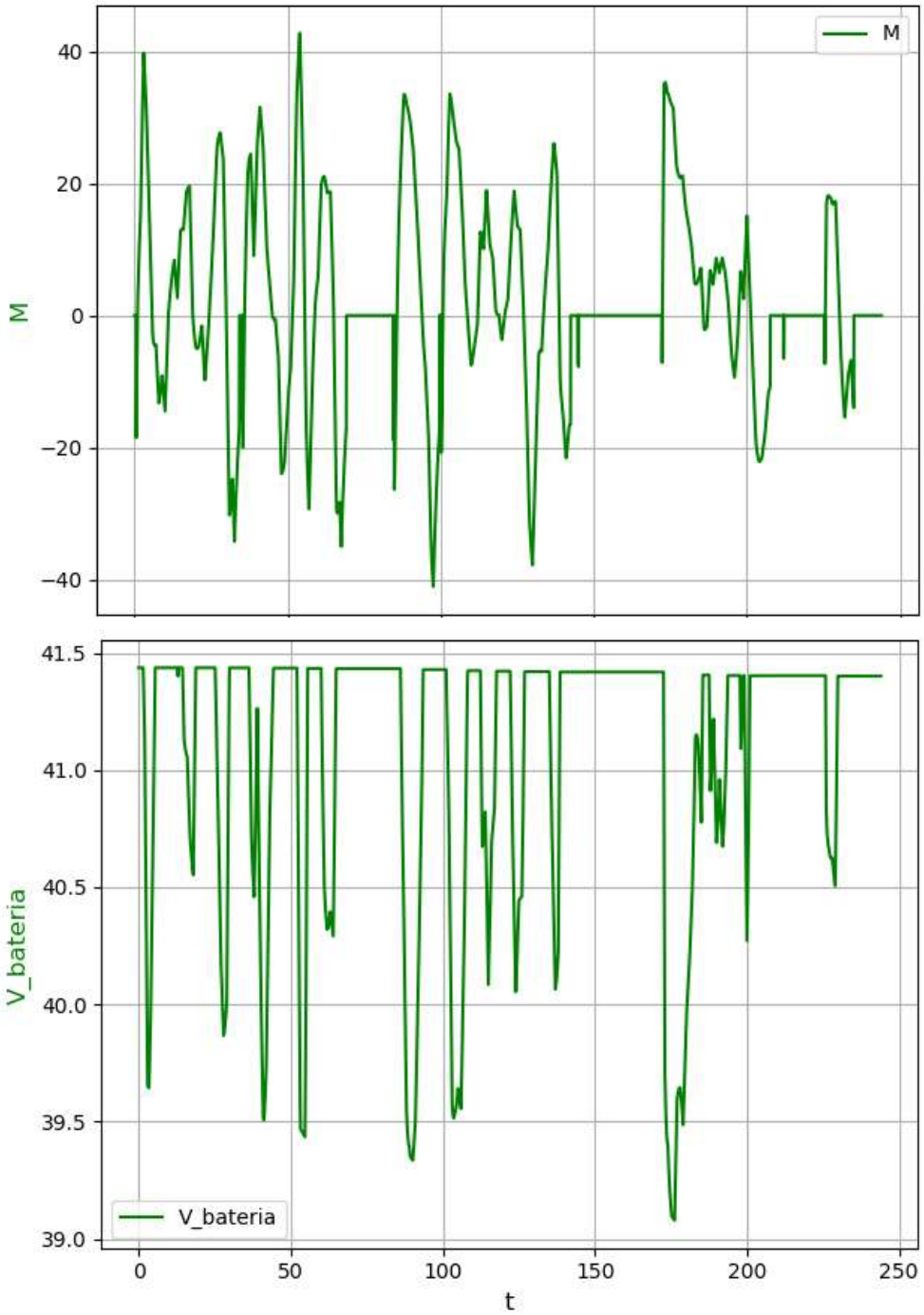


Ilustración 20 Par y tensión en la batería

6.2.6. POTENCIAS MECÁNICA Y ELÉCTRICA

Esta gráfica deja al descubierto una característica propia de este tipo de aparatos, y es el poco margen con el que trabajan respecto a sus valores nominales, se aprecia como se sobrepasan continuamente, por cortos periodos de tiempo, los valores de potencia del vehículo con el que fueron grabadas las rutas, aunque, la aparición de estos vehículos queda demasiado reciente como para asegurar que este hecho afecte a la vida útil del producto.

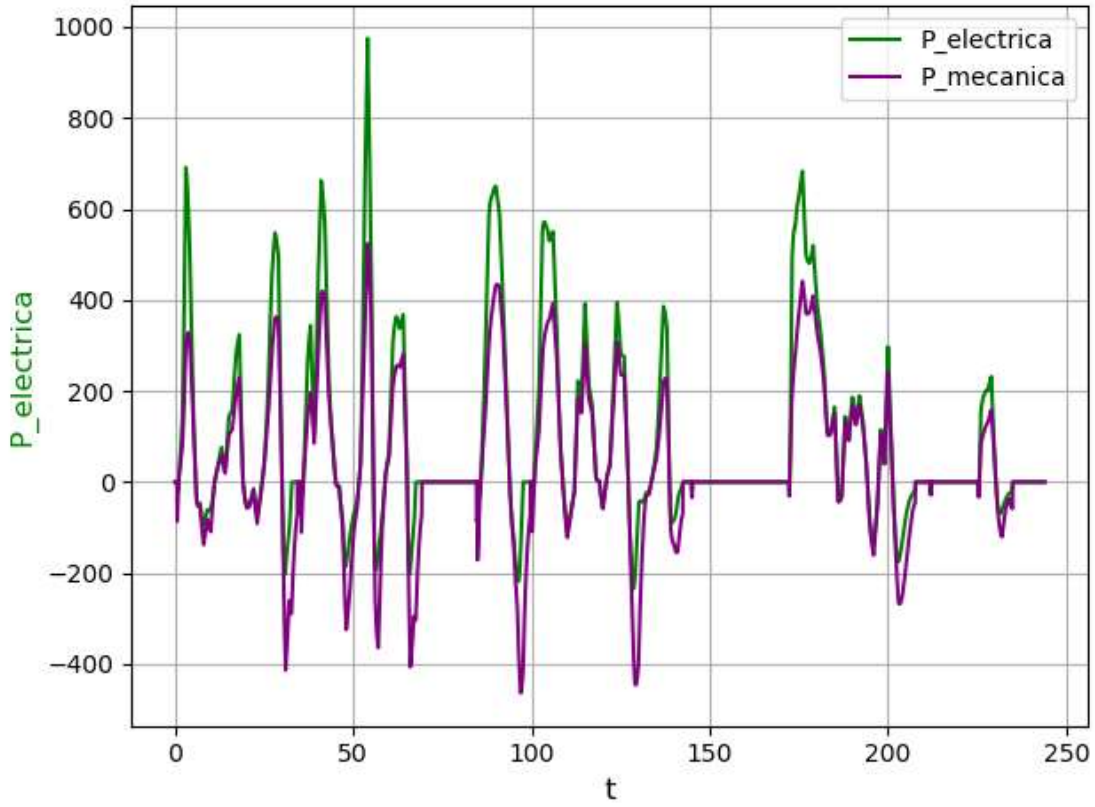


Ilustración 21 Potencias eléctrica y mecánica

Estos son los resultados que se consideran más interesantes acerca de este proyecto, aquí se han explicado para la ruta numero 4 por ser la más exigente, en el Anexo 5 pueden encontrarse estos mismos resultados para el resto de rutas.

7. SELECCIÓN DE MATERIALES

Para este proyecto se necesitarán una serie de elementos ya manufacturados para poder montar un prototipo funcional, todos los materiales han sido escogidos con una perspectiva al por menor, lo que implica lo siguiente:

- El precio es mayor que comprando al por mayor
- Solo se pueden utilizar elementos que ya se encuentren en fabricación, no se tienen recursos para fabricar elementos diseñados específicamente para este producto.
- Se ha preferido la importación directa por la diferencia de precio
- Algunos elementos como el tacómetro, tendrán que hacerse a mano

La lista de materiales utilizados es la siguiente:

7.1. MOTOR

Teniendo en cuenta las restricciones legales, el motor no debería superar los 250W, dentro de estos, son especialmente populares para este tipo de vehículos los motores instalados en rueda (hub motor), estos motores son de tipo DC-Brushless, debido a que su construcción permite colocar imanes en el rotor, constituido por la parte externa del motor, de esta forma conseguimos tener un eje fijo a través del cual podemos meter los cables, sin que estos sufran torsión alguna, condición necesaria para utilizar el motor en rueda.

7.1.1. OPCIÓN ESCOGIDA

El motor escogido para este proyecto es el MXUS XF08, que cumple con las especificaciones requeridas manteniendo un precio comedido para su sector.

Ilustración 22 Imagen del motor colocado en la rueda

Las características nominales de este motor son:

Tabla 1 Características del motor DC Brushless

Tensión	36V
Potencia	250W
Velocidad angular	235 rpm
Velocidad (rueda de 26”)	25-30km/h
Rendimiento	80%
Par	10.5Nm
Peso	3kg
Diámetro	13.5cm
Precio	92€

7.1.2. ALTERNATIVAS

Ilustración 23 MY1025 - Precio:31€ (importado)

Debido al alto precio de los motores con imanes de neodimio, se ha planteado como alternativa el uso de un motor DC convencional, el montaje tendría que realizarse manualmente anclándolo a la horquilla, fabricando una rueda dentada a medida anclada a esta para conseguir la velocidad correcta, se adjunta una tabla de lo que se consideran como pros y contras de cada alternativa:

Tabla 2 Comparativa entre motores

MOTOR	DC	DC BRUSHLESS
VENTAJA	PRECIO	FACIL MONTAJE
DESVENTAJA	CORRIENTES MAYORES	CONTROL MAS DIFICIL

Los motivos para desechar la opción del motor DC convencional han sido la dificultad para conseguir una rueda dentada de las características requeridas así como el hecho de que menor tensión (24V) implica mayor corriente y habría que dimensionar el resto de componentes acorde a esto, lo que haría la diferencia de precio menos interesante.

7.2. BATERÍA

7.2.1. OPCIÓN ESCOGIDA

La batería que se ha escogido se compondrá de 10 celdas en serie y 3 ramas en paralelo, el fundamento de esta decisión es que, para obtener los 36V nominales que necesita el motor se necesitan 10 celdas de 3.6V nominales, las 3 ramas en paralelo provienen de establecer una intensidad máxima nominal de 2C para cada celda, es decir 6.7A, con esta premisa y a la vista de los resultados del apartado anterior, se ha tomado la decisión de utilizar 3 ramas. la celda escogida es la NCR18650B de Panasonic, con esta configuración se obtendrían las siguientes características nominales:

Tabla 3 Características de la batería

Tensión	36V
Energía	360Wh
Corriente máxima	20.1A
Corriente de carga	4.875A
Precio	10uds 19.1€

Pueden encontrarse más detalles sobre las características de las celdas utilizadas en el datasheet, que puede encontrarse en el Anexo 2.

Ilustración 24 Aspecto de la celda NCR18650B

Después de esto el proceso consistiría en utilizar packs de separadores para unir las celdas correctamente, y después, soldar las celdas en la configuración correcta utilizando cinta de níquel y estaño, o si es posible, soldadura por puntos.

Se le añadirá una toma intermedia con 3 de los 10 grupos de celdas en serie para así obtener la tensión necesaria para alimentar la parte de control de la bicicleta. Esta decisión se ha podido tomar gracias a que la batería incorporará un BMS que corregirá el desequilibrio que se producirá en estas, que por otra parte es pequeño debido al bajo consumo del controlador en comparación con el motor.

7.2.2. ALTERNATIVAS

Se ha barajado la posibilidad de usar celdas de Ni-Cd, sin embargo, tienen peor relación capacidad-peso, capacidad-precio y menor tensión, por lo que no nos aportaban ninguna ventaja.

También se ha investigado la viabilidad de utilizar celdas tipo pouch, cuya peor relación capacidad-precio se compensa en cierto modo por su volumen más compacto, sin embargo, su baja tensión en comparación con su capacidad la hace inconcebible para este proyecto.

Estas es una comparación de las celdas estudiadas:

Tabla 4 Características de las celdas estudiadas

	TENSION (V)	ENERGIA (Wh)	PRECIO (€)
NCR18650 (PANASONIC)	3.6	12	1.9
Ni-Cd (VINIWOX)	1.2	1.8	0.84
POUCH (SIN MARCA)	3.2	128	40.5

Ilustración 25 Celdas Ni-Cd

Ilustración 26 Celda pouch

7.3. BMS

Debido a que el vehículo se alimenta a partir de una batería formada por celdas de litio, resulta imprescindible la utilización de un Battery Management System para asegurar el correcto funcionamiento de esta.

7.3.1. OPCIÓN ESCOGIDA

Este es el controlador escogido:

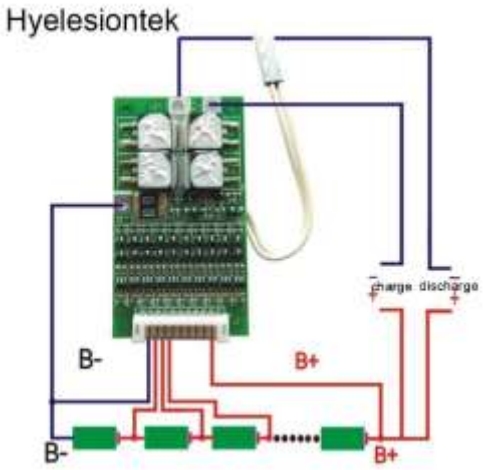


Ilustración 27 Circuito BMS - Precio: 9.3€ (importado)

Sus características son:

Tabla 5 Características del BMS

Tensión nominal	36V (Equilibrado de 10 celdas en serie)
Corriente nominal	40A
Precio	9.3€

Debido a que la batería está formada por 3 ramas de 10 celdas en serie, a cada entrada del BMS se le asignarán 3 celdas en paralelo.

7.3.2. ALTERNATIVAS

Las opciones barajadas han sido, o bien, comprar uno ya fabricado que cumpla con los requisitos, o bien fabricar el circuito base y añadir el control de este a la lista de tareas a realizar por el controlador principal. Debido a la importancia de este sistema, se ha optado por la primera opción, de todas formas, puede encontrarse el circuito planteado en el Anexo 3, su construcción se basa en una serie de resistencias de descarga, conectadas entre cada grupo ce celdas en paralelo y el negativo de la batería, el transistor que provoca que pase corriente a través de cada una de estas resistencias se activará cuando el controlador detecte una cierta diferencia entre la tensión de los distintos grupos de celdas, e intentará descargar ligeramente las celdas con más tensión para así equilibrar el sistema, también dispone de un interruptor general que la desconectará en caso de que la tensión total de la batería supere los límites superior e inferior establecidos.

7.4. CONTROLADOR

7.4.1. OPCIÓN ESCOGIDA

Debido al carácter de este proyecto, cuya intención es más el aprendizaje acerca del vehículo que la efectividad en costes, desde el comienzo se pensó en un controlador basado en una placa Arduino. Su bajo coste junto con su, relativamente sencilla programación, con multitud de información acerca de esta en la red, la hacen idónea para este proyecto. Su principal desventaja es la carencia de robustez, que podría afectar a su funcionamiento en entornos industriales, pero en principio no debería afectar en las condiciones en las que trabaja este vehículo. El modelo de Arduino que parece más correcto para esta aplicación es la placa Mega 2560, con mayor capacidad de cálculo que otras más básicas y un número mayor de entradas y salidas.

Ilustración 28 Placa Arduino Mega 2560 - Precio: 12€

El Arduino debe de programarse en lenguaje C++, las funciones que deben resolverse en el bucle son:

- Medir la tensión del acelerador (potenciómetro)
- Comprobar en qué modo de operación se está funcionando
- Conocer la velocidad a partir de un tacómetro instalado en la rueda trasera
- Aplicar la tensión analógica al puente H en función de la tensión deseada
- Estimar el SOC restante a partir de los datos de intensidad en la batería

7.4.2. ALTERNATIVAS

Si lo que se busca es obtener el resultado de la forma más sencilla y barata posible, dejando a un lado aprender cómo funciona el sistema, la reciente expansión de este tipo de vehículos ha traído al mercado controladores que ya incluyen el convertidor necesario para un motor DC Brushless, además estos ya vienen correctamente protegidos e incluyen radiadores para la disipación del calor, todo ello por un precio menor del que cuestan los componentes por separado.

Ilustración 29 Presentación del controlador - Precio: 9.4€ (importado)

Ilustración 30 Detalle de las conexiones

7.5. CHOPPER (CASO MOTOR DC)

7.5.1. OPCIÓN ESCOGIDA

A pesar de que, este tipo de vehículos, al no estar diseñados para funcionar marcha atrás, no necesiten más que un convertidor puente en dos cuadrantes, al tener las opciones limitadas a elementos que ya estén en fabricación se ha escogido un controlador en cuatro cuadrantes, el cual se utilizará para girar en un único sentido desaprovechando parte de los componentes.

En el caso de un motor DC, la conexión solo requiere de un chopper para controlar la velocidad.

Las condiciones de este controlador es que nos permita obtener la tensión nominal del motor sin problemas, y que soporte corrientes mayores de las que arrojan los resultados de los test. También debe de ser controlable con la tensión de salida del controlador.

Ilustración 31 Puente H escogido - Precio: 17.2€ (importado)

Sus características son, tras instalar un correcto sistema de erradicación del calor:

Tabla 6 Características del convertidor chopper

Tensión de salida	3.5-36V
Corriente máxima continua	20A
Corriente máxima de pico	110A
Tensión de entrada	0-0.8V (BAJO) 2-5.5V (ALTO)
Precio	17.2€

7.5.2. ALTERNATIVAS

Por un momento se planteó la posibilidad de construir un simple convertidor puente en dos cuadrantes mediante el uso de transistores MOSFET, utilizando la propia salida PWM del Arduino, sin embargo, no fue posible encontrar ningún transistor capaz de operar en el rango de intensidades requerido con tan solo 5V como tensión de puerta, con lo cual habría que añadir un driver, lo que ocasionaría problemas de montaje y encarecería la placa, por lo que esta opción perdía sentido.

7.6. CHOPPER + VSI (CASO MOTOR DC BRUSHLESS)

Si el motor a controlar es un DC Brushless, el manejo de la velocidad requiere además de un puente Voltage Source Inverter, que mediante sensores Hall, determine la posición del rotor y active la fase correcta del motor, en este caso la variedad encontrada ha sido escasa y solo se ha encontrado una opción comercial viable.

7.6.1. OPCIÓN ESCOGIDA

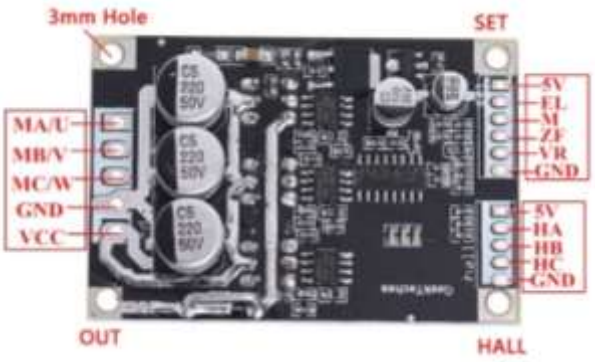


Ilustración 32 Puente para DC Brushless - Precio: 9.9€ (importado)

Características:

Tabla 7 Características del convertidor chopper + VSI

Tension nominal	36V
Corriente nominal	15A
Señal de control	0.1-5V

7.7. CARGADOR

Otra parte importante del vehículo, aunque se encuentre fuera de el es el cargador, una vez más la opción de fabricarlo a partir de los componentes se ve desplazada por la fabricación masiva de este tipo de componentes ya formados.

7.7.1. OPCIÓN ESCOGIDA

La opción escogida ha sido un cargador de 36V y 2A, que debería de cargar la batería al completo en aproximadamente 5 horas.

Ilustración 33 Cargador escogido - Precio: 7.2€ (importado)

7.7.2. ALTERNATIVAS

En un comienzo se pensó en integrar el circuito dentro del vehículo, y que, a efectos del usuario, el cargador fuera un cable directamente conectado a la corriente doméstica, la baja competitividad de esta opción frente a las ya fabricadas supuso que fuese desechada.

7.8. OTROS MATERIALES

Para la realización de este proyecto son necesarios otros elementos de apoyo que, al no constituir un impacto económico tan grande como los anteriores no se han dotado de categoría propia, estos son:

7.8.1. CINTA DE NÍQUEL

Se utiliza para soldar los terminales de las celdas de la batería.

Ilustración 34 Cinta de níquel - Precio: 1€ (importada)

7.8.2. SEPARADORES CELDAS 18650

Es recomendable usarlas para la fabricación de la batería.

Ilustración 35 Separadores - Precio: 4.6€ pack de 100 (importados)

7.8.3. FOCO LED

Para poder circular en horas nocturnas.

Ilustración 36 Foco led - Precio: 3.8€ (importado)

7.8.4. ACELERADOR

Se ha optado por un puño acelerador, en lugar de un acelerador de pulgar.

Ilustración 37 Acelerador - Precio: 3.1€ (importado)

7.9. CONCLUSIONES ACERCA DE LOS MATERIALES

La economía de escala es un factor clave en el diseño del prototipo, ya que, como se puede comprobar, cualquier intención de diseñar alguno de los componentes de este proyecto se ve rápidamente descartada a causa de una opción, no solo más sencilla, al delegar su fabricación a otra empresa, sino también más barata de obtener el mismo resultado.

Incluso cuando se estudian diversas opciones para comprar, podemos ver que la complejidad del elemento apenas afecta al precio del producto en comparación con la cantidad de unidades fabricadas, como podemos comprobar el precio de un chopper aislado (17.2€) es mucho mayor de uno de potencia similar que además trae un puente VSI incorporado (9.9€), y este sigue siendo más caro que un circuito que tiene estas dos cosas y además incluye el controlador ya programado específicamente para este tipo de aplicaciones y la carcasa con refrigeración(9.4€).

Dejando a un lado el ámbito económico, en el Anexo 3 puede encontrarse el circuito eléctrico del proyecto, que sigue teniendo interés académico.

En la parte superior de este encontramos el *Arduino*, encargado de realizar el control sobre el sistema, abajo a la izquierda se encuentra la batería, que además de alimentar al controlador a través de una toma intermedia también alimenta a un semipuente, que controla la tensión que llega al bus de continua del inversor y que, por

último, llegará al motor, cuya posición es devuelta al *Arduino* para que tenga la información necesaria para activar las salidas del antes mencionado inversor.

8. PRESUPUESTO

En el presupuesto se han tenido en cuenta todos los componentes antes mencionados, este presupuesto se ha realizado para la versión escogida, compuesta por el motor DC Brushless y realizando el proceso de control manualmente.

Se ha introducido un 10% extra a modo de aproximación para los gastos derivados de otros pequeños componentes necesarios para su construcción, (cables, cinta, estaño, compartimentos…).

Otro aspecto que se ha tenido en cuenta, a pesar de su baja probabilidad, es que el motor, el único aparato de esta lista que cumple los requisitos, sea parado en la aduana, en cuyo caso habría que abonar el IVA correspondiente.

Este presupuesto solo incluye la electrificación, y a este, habría que sumarle el precio de la bicicleta a electrificar, para obtener así el precio total del vehículo.

Tabla 8 Presupuesto

COMPONENTE	PRECIO	% Total
MOTOR	92.00 €	46.0%
BATERÍA	57.30 €	28.6%
BMS	9.30 €	4.6%
CONTROLADOR	12.00 €	6.0%
CHOPPER+VSI	9.90 €	4.9%
CARGADOR	7.20 €	3.6%
CINTA NÍQUEL	1.00 €	0.5%
SEPARADORES	4.60 €	2.3%
FOCO	3.80 €	1.9%
ACELERADOR	3.10 €	1.5%
Total	200.20 €	
+10% OTROS GASTOS	220.22 €	
+21% IVA MOTOR	239.54 €	

9. CONCLUSIONES

Este medio de transporte propone una alternativa barata rápida, segura y ecológica al vehículo privado para moverse en territorio urbano, en lo relativo al negocio, constituye además un mercado altamente emergente.

La realización del modelo ha arrojado algunos resultados interesantes, como su bajo consumo de energía o el bajo margen con el que operan sus componentes, sirviendo además al propósito de selección de los materiales.

En lo relativo a dicha selección de materiales, China lidera el mercado, ofreciendo los componentes a bajo precio, incluso tratándose de compra al por menor, lo cual puede constituir un factor clave en la expansión de este tipo de vehículos.

En definitiva, hablamos de un medio de transporte muy a tener en cuenta por todas las ventajas que aporta al individuo, medio ambiente y circulación.

10. BIBLIOGRAFÍA

[1] «Encuesta sobre Movilidad al trabajo de trabajadores residentes en Zaragoza, 2015».

[2] *Real Decreto Legislativo 6/2015.*

[3] *Reglamento UE 168/2013.*

[4] *Instrucción 16/V-124.*

[5] *Real Decreto 339/2014.*

[6] «Biobike,» [En línea]. Available: https://www.biobike.es.

[7] «HP Wizard,» [En línea]. Available: https://hpwizard.com/tire-friction-coefficient.html.

[8] «Cyling Power Lab,» [En línea]. Available: https://www.cyclingpowerlab.com/cyclingaerodynamics.aspx.

[9] «API Bing Maps,» [En línea]. Available: http://dev.virtualearth.net/.

ANEXO 1 – RUTAS

A continuación, se van a presentar los valores de velocidad y altitud sobre el nivel del mar respecto del tiempo.

RUTA 1 – 3047m

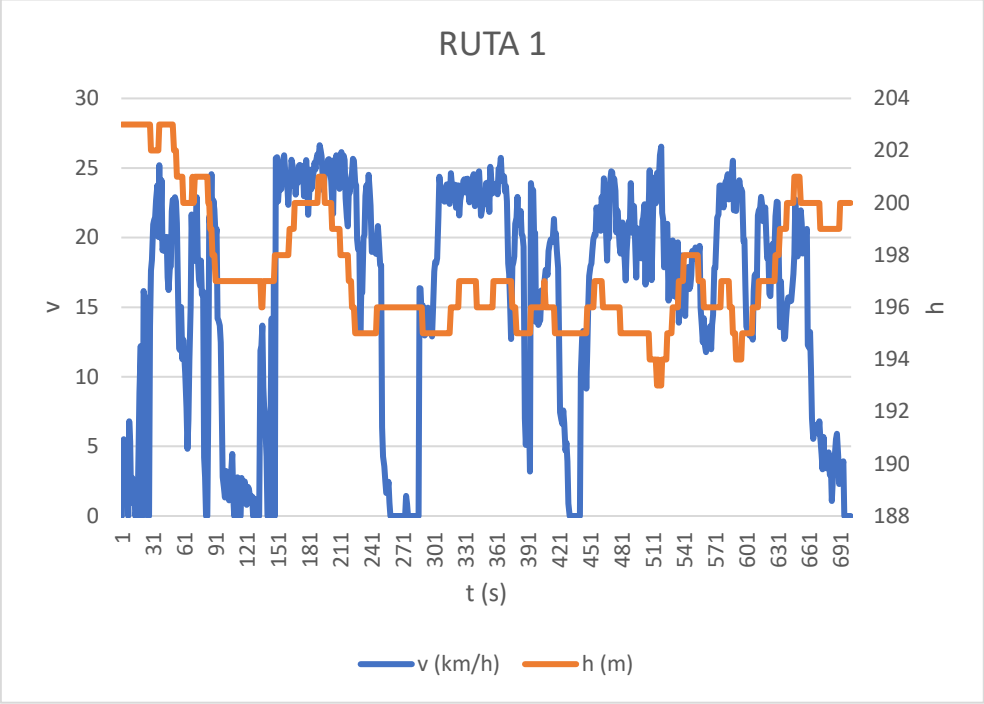


Ilustración 38 Consigna de velocidad y altitud Ruta1

RUTA 2 – 4116m

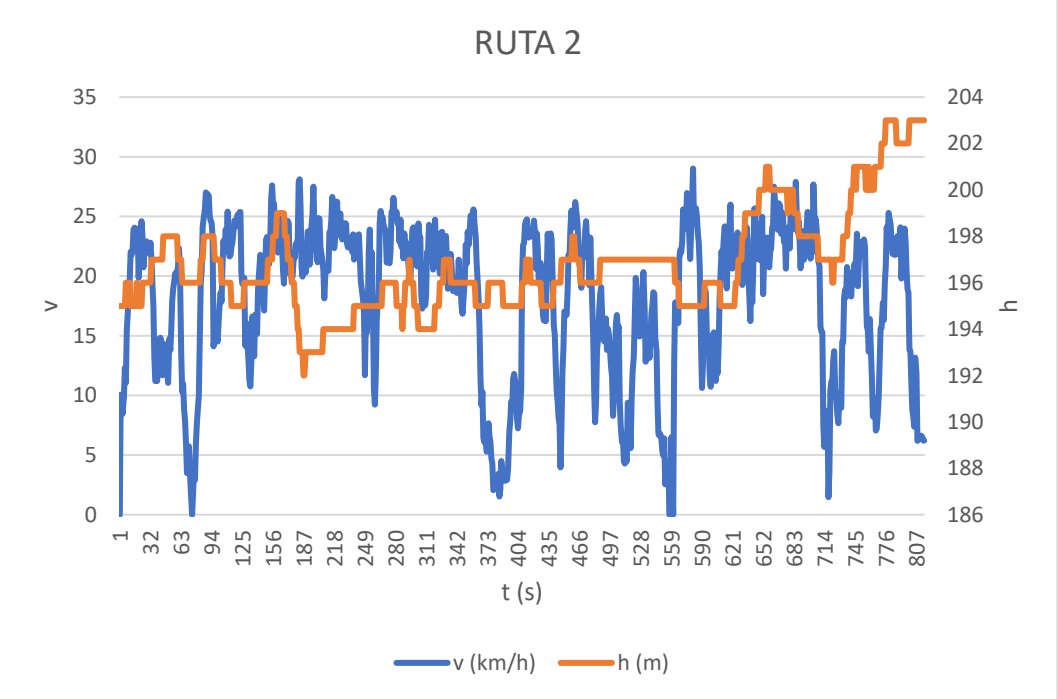


Ilustración 39 Consigna de velocidad y altitud Ruta2

RUTA 3 – 6259m

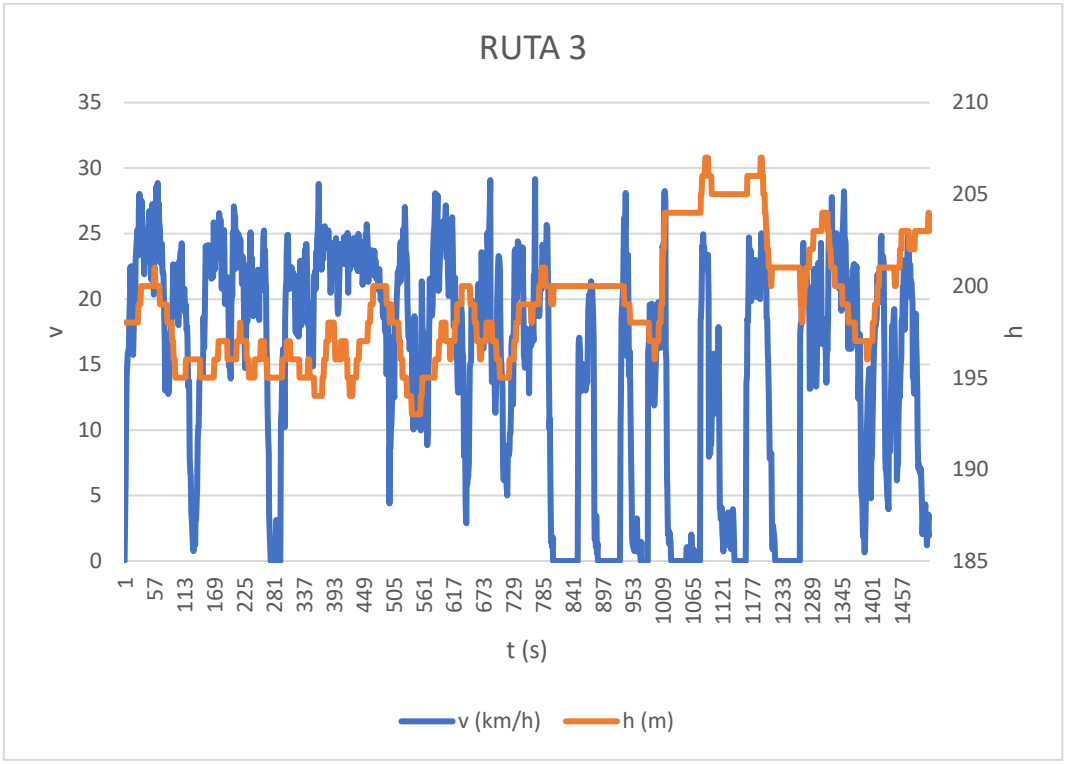


Ilustración 40 Consigna de velocidad y altitud Ruta3

RUTA 4 – 712m

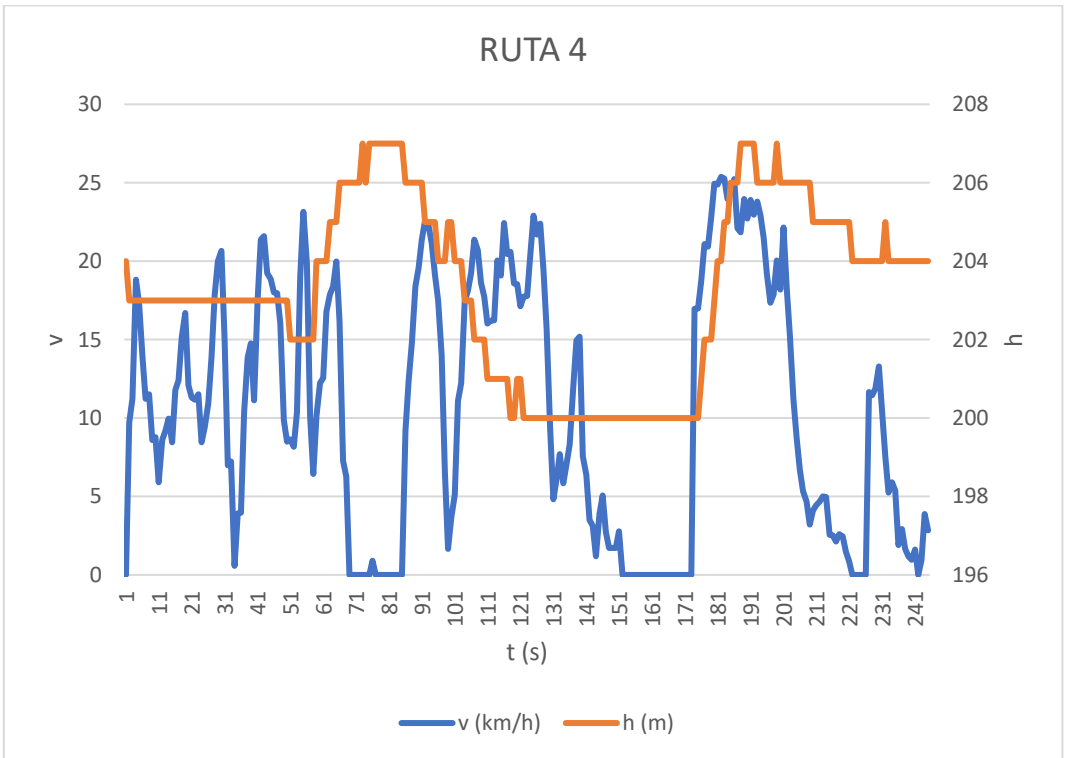


Ilustración 41 Consigna de velocidad y altitud Ruta4

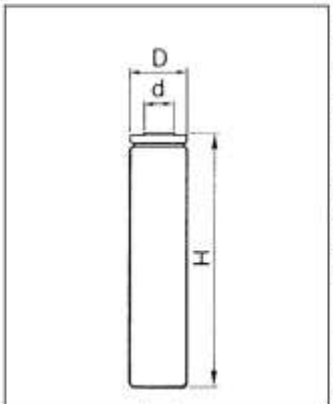
ANEXO 2 – DATASHEETS

NCR18650B

Lithium ion
Rechargeable battery

Cell Type NCR18650B

Specifications



Dimensions(Typ.) of Bare Cell		
	H	64.93mm
	D	18.2mm
	d	7.9mm

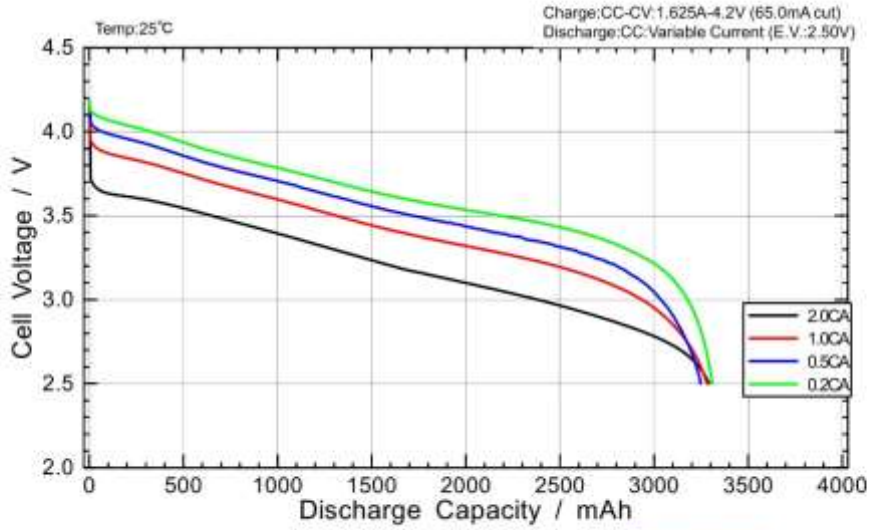
Discharged State after Assembling

Rated Capacity (at 20°C)		Min.3200mAh
Nominal Capacity (at 25°C)		Min.3250mAh Typ.3350mAh
Nominal Voltage		3.6V
Charging Method		Constant Current -Constant Voltage
Charging Voltage		4.2V
Charging Current		Std.1625mA
Charging Time		4.0hrs.
Ambient Temperature	Charge	+10～+45°C
	Discharge	-20～+60°C
	Storage	-20～+50°C
Weight (Max.)		47.5g
Dimensions (Max.) Maximum size without tube	(D)	18.25mm
	(H)	65.10mm
Volumetric Energy Density		676Wh/l
Gravimetric Energy Density		243Wh/kg

Panasonic ideas for life

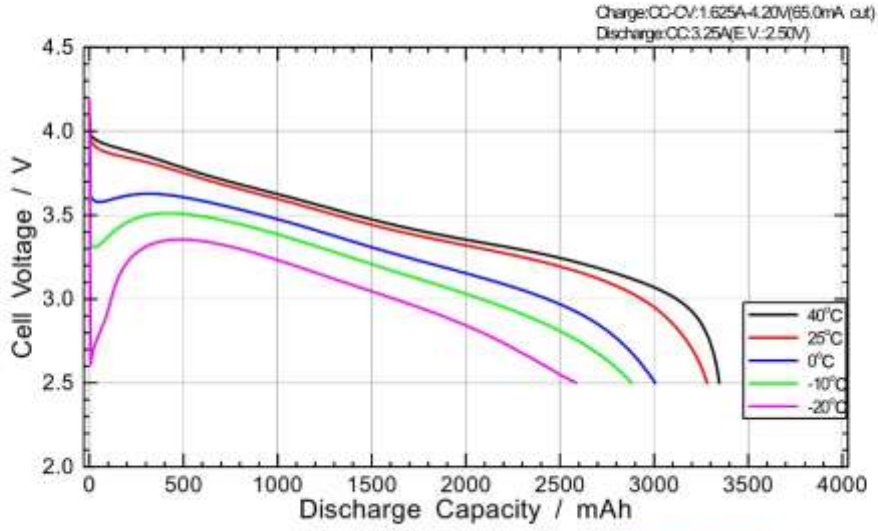
Lithium ion
Rechargeable battery

Discharge Rate Characteristics for NCR18650B

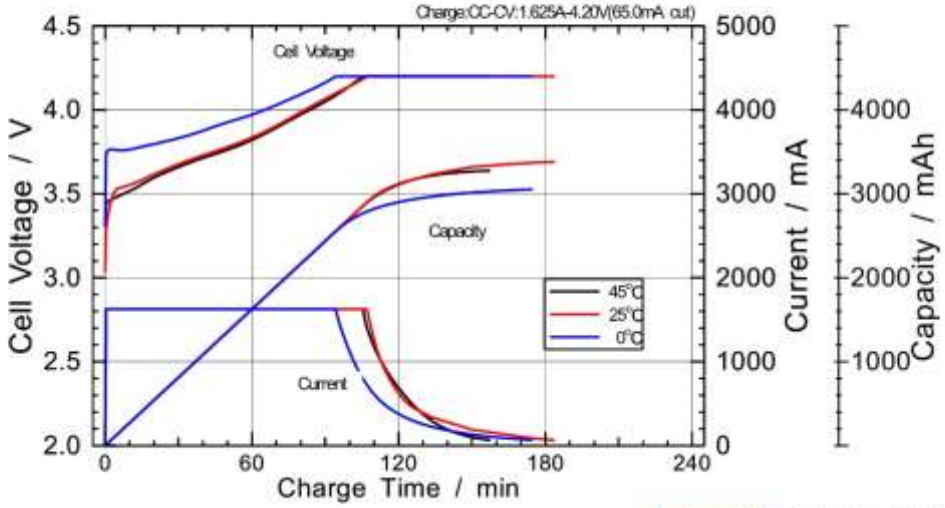


Panasonic ideas for life

Discharge Temperature Characteristics for NCR18650B



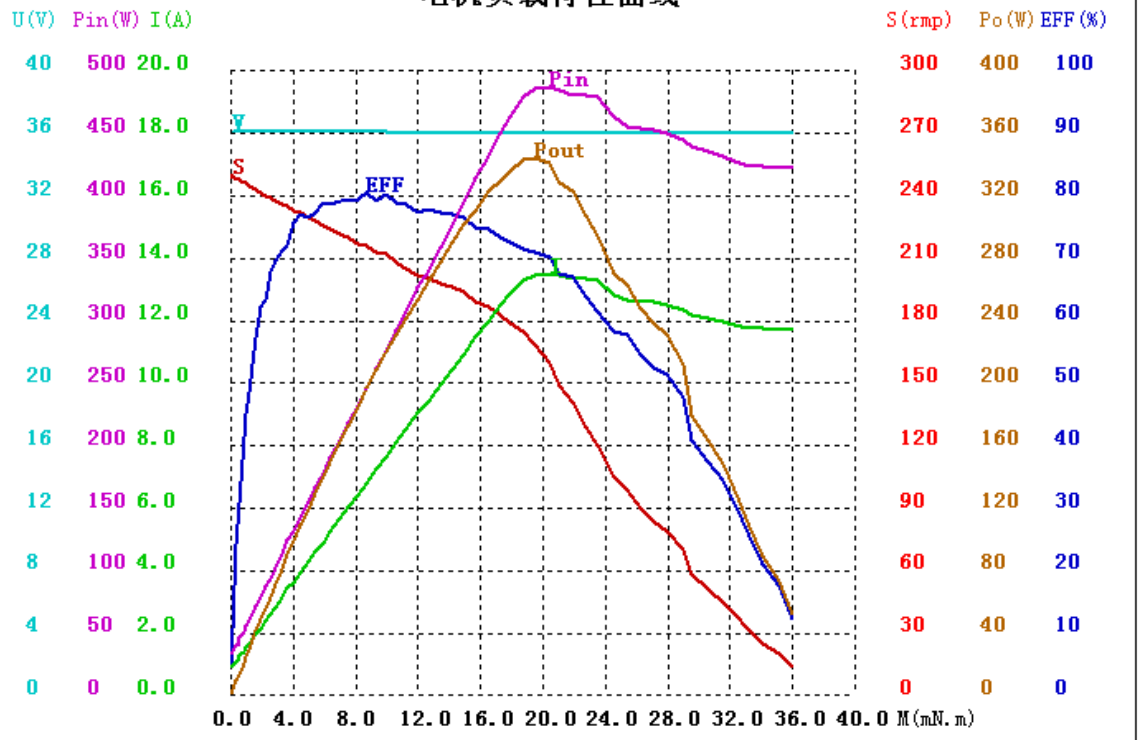
Charge Characteristics for NCR18650B



电机测试报告

厂家名称(Company)：	Text1	额定电压(Voltage)：	36(V)
电机型号(Type)：	XF08	额定功率(Power Rated)：	Text1(W)
备注(Remark)：	6	额定转速(Speed Rated)：	Text1(rmp)
测试人(Tester)：	Text1	测试日期(Test Date)：	2017-03-20

电机负载特性曲线



空载点(Unload):
- 转速 = Text1 (rpm)
- 电流 = Text1 (A)

最大转矩点(Max_Torque):
- 转矩 = 35.94 (mN.m)
- 转速 = 14 (rpm)
- 电流 = 11.77 (A)
- 输出功率 = 52.68 (W)
- 效率 = 12.4 (%)

最大输出功率点(Max_Pout):
- 转矩 = 19.44 (mN.m)
- 转速 = 169 (rpm)
- 电流 = 13.47 (A)
- 输出功率 = 344.1 (W)
- 效率 = 70.9 (%)

最高效率点(Max_Pout):
- 转矩 = 8.73 (mN.m)
- 转速 = 216 (rpm)
- 电流 = 6.802 (A)
- 输出功率 = 197.6 (W)
- 效率 = 80.5 (%)

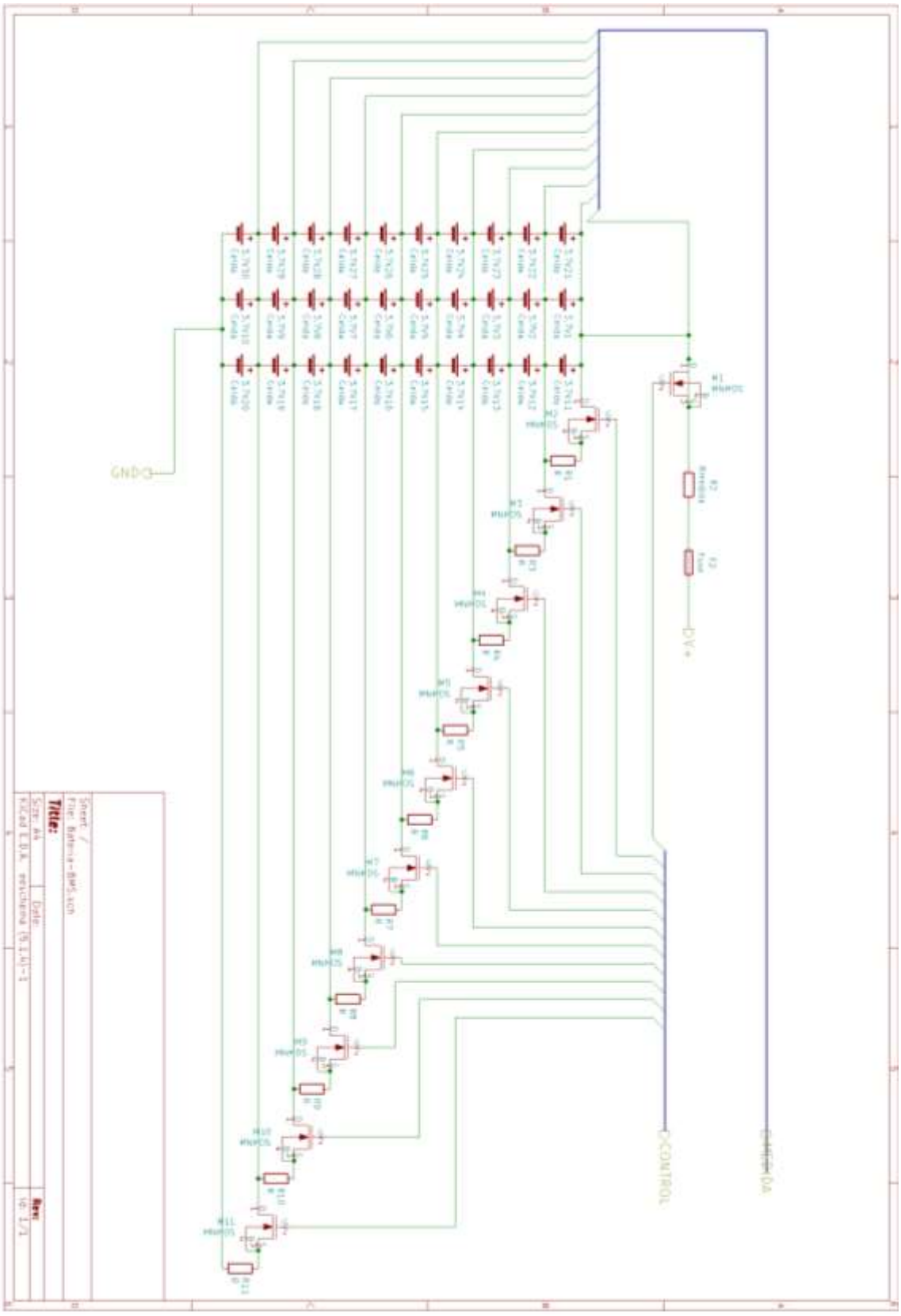
	MXUS		
motor type	XF08	test date	2017-03-20
remark	6	test person	Text1
rated voltage	36	rated rpm	200
rated power	265.7	no-load rpm	249
no-load voltage	36.17		

serial number	voltage	electric current	output power	torque	rpm	output power	efficiency
1	36.17	0.939	34.00	0.06	249	1.71	5.0
2	36.17	0.943	34.12	0.08	250	2.17	6.4
3	36.16	0.985	35.34	0.14	249	3.82	10.8
4	36.16	1.034	37.41	0.25	249	6.62	17.7
5	36.16	1.104	39.95	0.37	248	9.62	24.1
6	36.16	1.166	42.18	0.49	248	12.96	30.7
7	36.15	1.292	46.73	0.49	248	12.96	30.7
8	36.15	1.390	50.28	0.80	246	20.84	41.4
9	36.15	1.539	55.64	0.97	246	25.11	45.1
10	36.15	1.667	60.28	1.13	245	29.21	48.5
11	36.14	1.822	65.86	1.36	244	34.96	53.1
12	36.14	2.004	72.45	1.63	243	41.55	57.3
13	36.14	2.165	78.26	1.92	242	48.74	62.3
14	36.14	2.422	87.55	2.20	240	55.38	63.3
15	36.13	2.605	92.99	2.52	239	63.14	67.9
16	36.13	2.849	102.9	2.87	238	71.73	69.7
17	36.13	3.131	113.1	3.26	236	80.72	71.4
18	36.12	3.459	124.9	3.65	235	90.04	72.1
19	36.12	3.623	130.9	4.06	233	99.24	75.8
20	36.12	3.939	142.3	4.50	232	109.5	77.0
21	36.12	4.316	155.9	4.95	230	119.3	76.5
22	36.11	4.639	167.5	5.42	228	129.4	77.3
23	36.11	4.908	177.2	5.91	226	139.9	79.0
24	36.11	5.276	190.5	6.40	224	150.1	78.8
25	36.11	5.676	204.9	6.98	222	162.3	79.2
26	36.10	6.044	218.2	7.51	220	173.1	79.3
27	36.10	6.417	231.7	8.10	217	184.1	79.5
28	36.10	6.802	245.5	8.73	216	197.6	80.5
29	36.09	7.284	262.9	9.35	213	208.6	79.3
30	36.09	7.633	275.5	9.95	212	220.9	80.2
31	36.08	8.119	293.0	10.62	208	231.4	79.0
32	36.08	8.523	307.5	11.27	205	241.9	78.7
33	36.07	9.067	327.1	11.99	202	253.6	77.5
34	36.07	9.448	340.8	12.69	200	265.7	78.0
35	36.07	9.958	359.2	13.40	198	277.9	77.4
36	36.06	10.44	376.7	14.15	196	290.5	77.1

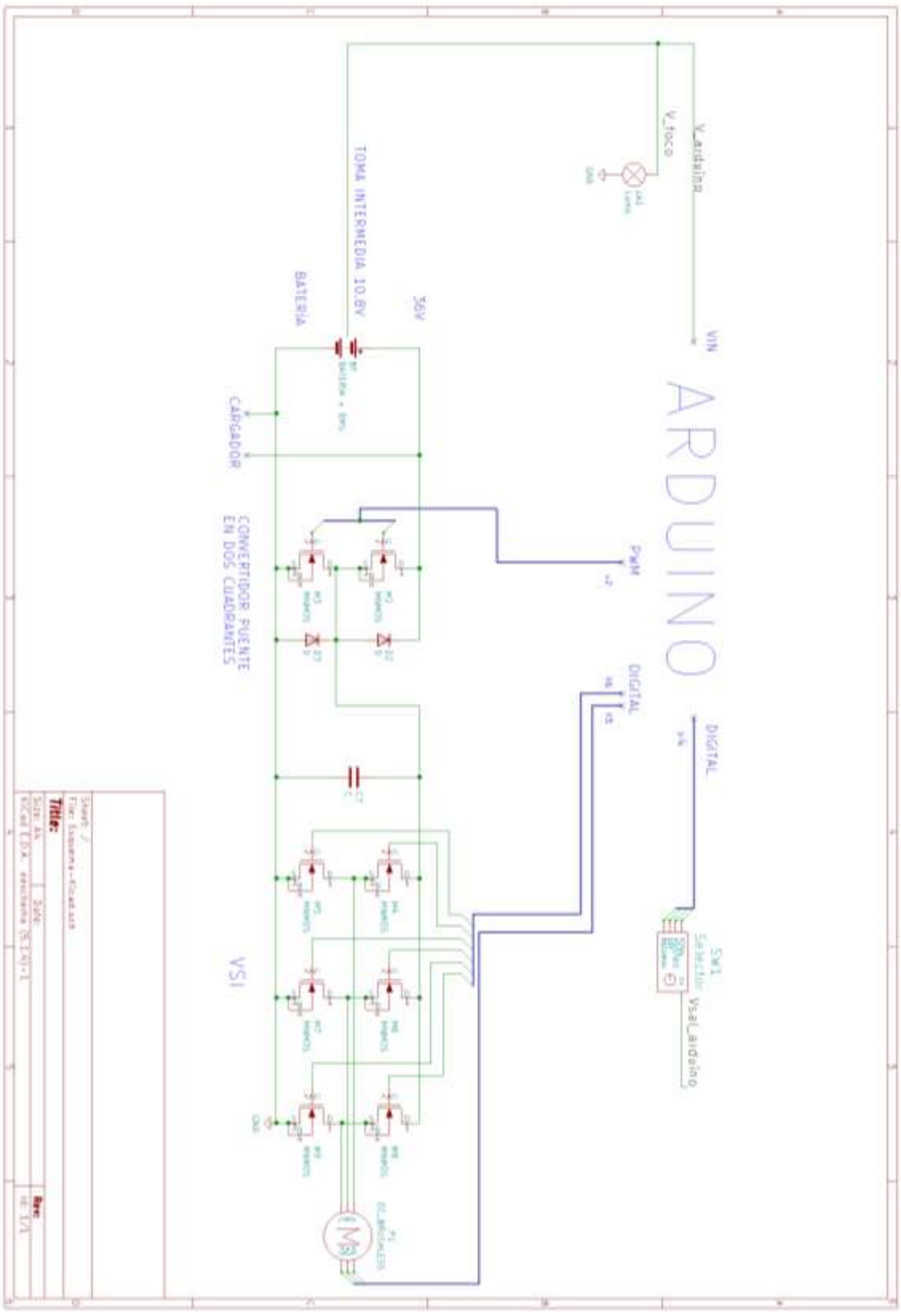
37	36.06	10.94	394.6	14.88	194	302.3	76.6
38	36.05	11.49	414.4	15.71	189	310.9	75.0
39	36.05	11.98	432.0	16.48	187	322.8	74.7
40	36.05	12.49	450.3	17.21	183	329.9	73.3
41	36.04	12.87	463.9	17.94	179	336.4	72.5
42	36.03	13.31	479.6	18.74	175	343.5	71.6
43	36.03	13.47	485.6	19.44	169	344.1	70.9
44	36.03	13.52	487.3	20.39	160	341.7	70.1
45	36.03	13.46	485.2	21.06	149	328.6	67.7
46	36.03	13.35	481.1	21.88	141	323.0	67.1
47	36.03	13.34	480.7	22.74	129	307.2	63.9
48	36.03	13.31	479.7	23.48	120	295.0	61.5
49	36.03	12.85	463.1	24.55	105	269.9	58.3
50	36.03	12.65	456.1	25.38	99	263.1	57.7
51	36.03	12.59	453.8	26.30	90	247.8	54.6
52	36.02	12.59	453.6	27.12	84	238.5	52.6
53	36.02	12.50	450.4	27.99	79	231.5	51.4
54	36.02	12.35	445.0	28.96	70	212.3	47.7
55	36.02	12.19	439.2	29.56	58	179.5	40.9
56	36.02	12.08	435.3	30.56	51	163.2	37.5
57	36.02	11.97	431.4	31.57	45	148.7	34.5
58	36.02	11.88	428.1	32.25	39	131.7	30.8
59	36.02	11.78	424.4	33.26	31	107.9	25.4
60	36.02	11.75	423.4	34.13	25	89.36	21.1
61	36.02	11.72	422.3	35.19	20	73.70	17.5
62	36.01	11.77	424.1	35.94	14	52.68	12.4

ANEXO 3 – CIRCUITOS

BMS (OPCIÓN DESCARTADA)



CIRCUITO COMPLETO



ANEXO 4 – MODELO

NOTAS ACLARATORIAS

El código está escrito para el lenguaje de programación Python3, disponible en: https://www.python.org/

Las librerías necesarias se encuentran en el archivo requirements.txt y son los siguientes:

- pandas
- matplotlib
- numpy
- openpyxl
- plotly

Se recomienda instalarlas mediante la herramienta pip.

Los archivos __init__.py están vacíos, son necesarios para que Python interprete la carpeta como un paquete.

ESTRUCTURA DEL MODELO

```
Modelo
|   requirements.txt
|   run.py
|
+---datos
|   +---celdas
|   |   \---NCR18650B
|   |         coeficientes.txt
|   |         datos.txt
|   |         datos_descarga.csv
|   |
|   +---motores
|   |      MXUS-XF08.txt
|   |
|   \---rutas
|         Ruta1.csv
|         Ruta2.csv
|         Ruta3.csv
|         Ruta4.csv
|
+---resultados
\---simulador
    |   configuracion.py
    |   modelo.py
    |   __init__.py
    |
    +---modulos
    |      bateria.py
    |      controlador.py
    |      modelo_dinamico.py
    |      motor.py
    |      resultados.py
    |      ruta.py
    |      __init__.py
    |
    \---utilidades
        |   funciones_utiles.py
        |   manejar_resultados.py
        |   __init__.py
        |
        \---formato_datos
              arreglar_rutas.py
              regresion_celdas.py
              __init__.py
```

run.py

```
from simulador.modelo import Modelo


# Seleccionar una ruta
nombre_ruta = 'Ruta1'

# Se inicia el modelo y se procede a su simulación
modelo = Modelo(nombre_ruta)
modelo.simular()


# Escoger que resultados se muestran, también se pueden guardar en un excel
modelo.resultados.trazar('t-v, v_consigna', 'energia_km')
modelo.resultados.guardar_excel()
```

configuracion.py

```
# Parámetros de los componentes
modelo_motor = 'MXUS-XF08'
datos_bateria = {'modelo_celda':'NCR18650B', 'ns':10, 'np':3}


# Parámetros de la ruta
PASO_CONSIGNA = 1 #s. tiempo entre las medidas de cosigna de v y m
FRECUENCIA_MUESTREO = 100
PASO = PASO_CONSIGNA / FRECUENCIA_MUESTREO #Esta linea se calcula automáticam
ente y no debe tocarse


# Datos del controlador
datos_controlador = {'kp':0.0971, 'ki':0.0827, 'kd':0}


# Parámetros técnicos
RENDIMIENTO_RECUPERACION = 0.55 #Para el frenado regenerativo
RENDIMIENTO_ELECTRONICA = 0.9
RENDIMIENTO_BATERIA = 0.9


# Parametros del vehículo
MASA_VEHICULO = 12 #kg
RADIO_RUEDA = 0.33 #m
C_ROZ = 0.004 #Coeficiente de rozamiento con el suelo
G = 1 #Relacción rueda-motor


# Datos conductor
MASA_CONDUCTOR = 75 #kg
AREA_FRONTAL = 0.5 #m^2
C_ARRASTRE = 0.88 #Coeficiente fricción con el aire


# Otros parámetros de la simulación
VEL_AIRE = 5 #km/h, aire en contra del avance


# Directorios internos
RUTA_MOTORES = './datos/motores'
RUTA_CELDAS = 'datos/celdas'
RUTA_RUTAS = 'datos/rutas'
RUTA_RESULTADOS = 'resultados'
```

modelo.py

```
# Impartamos algunos valores del archivo de configuracion
from .configuracion import RADIO_RUEDA, G, PASO
from .configuracion import modelo_motor, datos_bateria, datos_controlador

# Imortamos los módulos propios que componen el modelo
from simulador.modulos.ruta import Ruta
from .modulos.motor import Motor
from .modulos.bateria import Bateria
from .modulos.controlador import PID
from .modulos.modelo_dinamico import ModeloDinamico
from .modulos.resultados import Resultados
from .utilidades.funciones_utiles import Cronometro

# Importamos algunas librerías útiles
from math import pi


class Modelo:
    """Clase que coordina las diferentes partes del vehiculo"""
    def __init__(self, nombre_ruta: str):
        """Función que es llamada al definir el modelo, inicia los diferentes
 componentes"""
        # Definimos la batería
        self.bateria = Bateria(**datos_bateria)
        # Definimos el controlador
        self.controlador = PID(**datos_controlador, sat={'min':0, 'max':1})
        # Definimos la consigna
        self.ruta = Ruta(nombre_ruta)
        # Definimos el modelo dinámico
        self.modelo_dinamico = ModeloDinamico()
        # Definimos el motor
        self.motor = Motor(modelo_motor)
        # Definimos el objeto que alojará los resultados
        self.resultados = Resultados(self.ruta)
        # Añadimos la masa del motor y las baterías al modelo dinámico
        self.modelo_dinamico.masa += self.motor.datos['masa'] + self.bateria.
masa
        self.modelo_dinamico.J_motor = self.motor.datos['J']
        # Calculamos la J total del sistema
        self.modelo_dinamico.calcular_J()
        self.motor.J_total = self.modelo_dinamico.J
```

```
def simular(self, imprimir: bool=True):
        """Realiza la simulación con los datos del modelo"""
        v_ant = 0
        iteraciones_totales = (len(self.ruta) - 1) / 10 #Calculo para el cron
ometro de tiempo restante, cuenta cada 10 iteraciones
        crono = Cronometro(iteraciones_totales)
        for i in range(1, len(self.ruta)):
            # Sacamos los valores de v y m de la ruta
            v_ref, self.modelo_dinamico.m = self.ruta.v[i], self.ruta.m[i]
            # Tenemos en cuenta la condición de velocidad mínima de arranque
            if v_ant > 5 / 3.6: #Se supera la velocidad mínima
                # Pasamos la señal de error al controlador
                self.controlador.e = v_ref - v_ant
                # Y obtenemos el duty a su salida
                duty = self.controlador.salida
                # Multiplicandolo por la tensión de la batería obtenemos la t
ensión de bus que le llega al motor
                self.motor.V = self.bateria.V * duty
                # Calculamos el par resistente a partir de los datos del mode
lo dinámico
                Fr = self.modelo_dinamico.fuerzas['F_resistente']
                Mr = Fr * RADIO_RUEDA
                # Calculamos la velocidad angular y la lineal
                w = self.motor.calcular_w(Mr)
                v = w * RADIO_RUEDA * G
                # Calculamos las fuerzas para añadirlas a los resultados
                self.modelo_dinamico.v = v
                fuerzas = self.modelo_dinamico.fuerzas
            else:
                duty, self.motor.V = 0, 0
                # La velocidad de salida es la de referencia
                v = v_ref
                w = v / RADIO_RUEDA / G
                self.motor.w = w
                # La fuerza que realiza el motor es nula
                fuerzas = {'F_total':0, 'F_resistente':0, 'f_roz':0, 'f_ae':0
, 'f_g':0, 'f_al':0, 'f_aa':0}

            # Pasamos a la batería la intensidad que se consume
            self.bateria.I = self.motor.I * duty
            # Añadimos las variables interesantes a la lista para posteriorme
nte usarlas
            self.resultados.v.append(v * 3.6)
            self.resultados.a.append((v - v_ant) / PASO)
            self.resultados.F.append(fuerzas['F_total'])
            self.resultados.F_res.append(fuerzas['F_resistente'])
            self.resultados.f_roz.append(fuerzas['f_roz'])
            self.resultados.f_g.append(fuerzas['f_g'])
            self.resultados.f_ae.append(fuerzas['f_ae'])
```

```
            self.resultados.f_aa.append(fuerzas['f_aa'])
            self.resultados.f_al.append(fuerzas['f_al'])
            self.resultados.M.append(fuerzas['F_total'] * RADIO_RUEDA)
            self.resultados.SOC.append(self.bateria.SOC)
            self.resultados.V_bateria.append(self.bateria.V)
            self.resultados.I_bateria.append(self.bateria.I)
            self.resultados.V_motor.append(self.motor.V)
            self.resultados.I_motor.append(self.motor.I)
            self.resultados.P_electrica.append(self.motor.V * self.motor.I)
            self.resultados.P_mecanica.append(v * fuerzas['F_total'])
            self.resultados.distancia_recorrida.append(self.resultados.distan
cia_recorrida[-1] + v * PASO)

            if imprimir and not i % 10: #El tiempo restante se calcula cada 1
0 ciclos para hacerlo más estable
                crono.imprimir_restante()

            # Actualizamos la velocidad anterior del ciclo siguiente como la
actual de este
            v_ant = v
            self.motor.w_ant = w
            self.modelo_dinamico.v_ant = v_ant

        self.resultados.distancia_recorrida.pop(0) # Elimina el primer elemen
to por consistencia de los vectores
        self.resultados.energia_km = (sum(self.resultados.P_electrica) * PASO
 / 3600) / (self.resultados.distancia_recorrida[-1] / 1000) #Wh/km
```

motor.py

```
from ..utilidades.funciones_utiles import datos_txt
from ..configuracion import RUTA_MOTORES, PASO, G, RADIO_RUEDA
from math import sqrt



class Motor:
    def __init__(self, modelo_motor: str):
        self.J_total = 0 # Momento de inercia de todo el sistema
        self.datos = datos_txt(f'{RUTA_MOTORES}/{modelo_motor}') # Valores de
 la hoja de datos
        # Calculo de la constante del motor (Única al tratarse de un motor de
 imanes permanentes)
        Pn = self.datos['potencia_nominal']
        Mn = self.datos['par_nominal']
        In = self.datos['intensidad_nominal']
        self.K = Mn / In
        # Calculamos el rendimiento eléctrico y mecánico a partir del total
        # a falta de datos precisos partiremos de la suposición de que ambos
son iguales
        # es decir r_mec * r_elec = r_total
        r_total = self.datos['rendimiento']
        r_elec = sqrt(r_total)
        # Calculo de la resistencia
        self.R = (1 - r_elec) * (Pn / r_total) / In**2
        # Inicialización de variables
        self.w_ant = 0
        self.w = 0
        self.V = 0

    @property
    def I(self):
        """Calcula la corriente a partir del circuito equivalente del motor""
"
        return (self.V - self.K * self.w_ant) / self.R


    def calcular_w(self, Mr: float):
        """Devuelve la velocidad angular del motor"""
        # Calculamos el par eléctrico a partir de la constante del motor y la
 corriente que consume
        Me = self.K * self.I
        Mac = Me - Mr
        dw = Mac / self.J_total
        self.w += dw * PASO
        return self.w
```

modelo_dinamico.py

```
from ..configuracion import PASO, C_ROZ, AREA_FRONTAL, C_ARRASTRE, VEL_AIRE,
G, RADIO_RUEDA, MASA_CONDUCTOR, MASA_VEHICULO
from math import sin

# Variables inmutables
g = 9.81
DENSIDAD_AIRE = 1.25

class ModeloDinamico:
    """Clase encargada de calcular las fuerzas que afectan al vehículo"""
    def __init__(self):
        self.masa = MASA_CONDUCTOR + MASA_VEHICULO
        self.J_ruedas = 2 * (1 / 2) * (2 * RADIO_RUEDA**2 * G**2) #Estimamos
el peso de las llantas en 2kg y las tratamos como cilindros huecos
        self.J_motor = 0
        self.J = 0
        # Entradas del modelo
        self.v = 0
        self.v_ant = 0
        self.m = 0


    def calcular_J(self):
        """Esta función debe llamarse cuando se hayan incluido las masas del
motor
        y las baterías en el modelo, almacena el valor de forma interna, no r
etorna nada"""
        self.J = self.masa * RADIO_RUEDA**2 * G**2 + self.J_ruedas + self.J_m
otor


    @property
    def fuerzas(self):
        """Devuelve un diccionario con las fuerzas que afectan al vehículo
        F_total: Suma de todas las ferzas que afectan al vehículo
        F_resistente: Suma de las fuerzas de rozamiento, aerodinámica y gravi
tatoria
        f_roz: Fuerza de rozamiento
        f_ae: Fuerza aerodinámica
        f_g: Fuerza fravitatoria
        f_al: Fuerza de aceleración lineal
        f_aa: fuerza de aceleración angular"""
        # Calculo de la aceleracion
        a = (self.v - self.v_ant) / PASO
        # Fuerza de rozamiento con el suelo
        f_roz = C_ROZ * self.masa * g if self.v_ant > 0 else 0
        # Fuerza aerodinamica
```

```
        f_ae = (1/2) * DENSIDAD_AIRE * AREA_FRONTAL * C_ARRASTRE * (self.v_an
t + (VEL_AIRE / 3.6))**2 if self.v_ant > 0 else 0
        # Fuerza gravitatoria (pendientes)
        f_g = self.masa * g * sin(self.m) if self.v_ant > 0 else 0
        # Fuerza de aceleracion lineal
        f_al = self.masa * a
        # Fuerza de aceleracion angular
        f_aa = (self.J_ruedas + self.J_motor) * a  / (RADIO_RUEDA**2 * G**2)
        # Calculo de la fuerza total y la resistente
        F_total = f_roz + f_ae + f_g + f_al + f_aa
        F_resistente = f_roz + f_ae + f_g
        return {'F_total':F_total, 'F_resistente':F_resistente, 'f_roz':f_roz
, 'f_ae':f_ae, 'f_g':f_g, 'f_al':f_al, 'f_aa':f_aa}
```

controlador.py

```
from ..configuracion import PASO
from ..utilidades.funciones_utiles import saturador
from math import inf



class PID:
    """Regulador PID:
    kp:  constante proporcional
    ki: constante integral
    kd: constante derivativa
    sat: dict. {'min': valor, 'max': valor} límites de saturación"""
    def __init__(self, kp:float=0, ki:float=0, kd:float=0, sat:dict={'min':-
inf, 'max':inf}):
        self.e_ant = 0
        self.kp, self.ki, self.kd = kp, ki, kd
        self.sat = sat
        self.integral = 0
        self.e = 0


    @property
    def salida(self):
        """Calcula la salida del PID"""
        derivada = (self.e - self.e_ant) / PASO
        self.e_ant = self.e
        self.integral += self.e * PASO
        return saturador(self.kp * self.e + self.ki * self.integral + self.kd
 * derivada, **self.sat)
```

bateria.py

```
from ..utilidades.funciones_utiles import datos_txt
from ..configuracion import PASO, RUTA_CELDAS, RENDIMIENTO_RECUPERACION, REND
IMIENTO_ELECTRONICA, RENDIMIENTO_BATERIA
import numpy as np


class Bateria:
    """Crea una batería con la configuracion indicada de celdas
    modelo_celda: nombre de la celda escogida, debe de estar en el directorio
 datos
    ns: numero de celdas en serie
    np: numero de ramas en paralelo"""
    def __init__(self, modelo_celda: str, ns: int, np: int):
        self.datos = datos_txt(f'{RUTA_CELDAS}/{modelo_celda}/datos') #Datos
de cada celda
        self.ns, self.np = ns, np #Celdas en serie y paralelo
        self.capacidad = self.datos['capacidad_nominal'] * self.np
        self.capacidad_nominal = self.datos['capacidad_nominal'] * self.np
        self._coefs = datos_txt(f'{RUTA_CELDAS}/{modelo_celda}/coeficientes',
 recibir_listas=True)
        self.masa = self.datos['masa'] * ns * np
        self.I_max, self.I_min = self.datos['I_max'] * self.np, -
self.datos['I_max_carga'] * self.np
        self._t = PASO # Tiempo que lleva descargándose la batería, comienza
a partir de la primera iteracion
        self.ratio = 0


    @property
    def I(self):
        """Intensidad consumida en cada momento"""
        return self._I


    @I.setter
    def I(self, valor: float):
        """Utiliza la intensidad para calcular la energía consumida, tambien
calcula el self.ratio medio de descarga"""
        valor = valor * RENDIMIENTO_RECUPERACION if valor < 0 else valor / RE
NDIMIENTO_BATERIA / RENDIMIENTO_ELECTRONICA
        self.ratio = valor * 1000 / self.capacidad_nominal
        self.capacidad -
= valor * 1000 * PASO / 3600 # Se descarga por valor de la intensidad en mAh
        self._t += PASO
        self._I = valor
```

```
    @property
    def V(self):
        """Tensión de la batería, por las propiedades de la celda de litio es
ta depende de la
        capacidad consumida así como del C-rate"""
        cc = (self.capacidad_nominal - self.capacidad) / self.np #Capacidad c
onsumida por celda
        # Se definen las funciones para cada self.ratio con los ceficientes d
e la regresión polinómica
        f02, f05 = np.poly1d(self._coefs['0.2C']), np.poly1d(self._coefs['0.5
C'])
        f1, f2 = np.poly1d(self._coefs['1C']), np.poly1d(self._coefs['2C'])
        # Se revuelve una tensión en función del C-rate
        if self.ratio <= 0.2:
            V = f02(cc) * self.ns
        elif 0.2 < self.ratio <= 0.5:
            V = (f02(cc) * (0.5 - self.ratio)/(0.5 - 0.2) + f05(cc) * (self.r
atio - 0.2)/(0.5 - 0.2)) * self.ns
        elif 0.5 < self.ratio <= 1:
            V = (f05(cc) * (1 - self.ratio)/(1 - 0.5) + f1(cc) * (self.ratio
- 0.5)/(1 - 0.5)) * self.ns
        elif 1 < self.ratio <= 2:
            V = (f1(cc) * (2 - self.ratio)/(2 - 1) + f2(cc) * (self.ratio - 1
)/(2 - 1)) * self.ns
        else:
            V = f2(cc) * self.ns
        # Si la tensión baja de V_min corta
        return V if V > self.datos['V_min'] * self.ns else 0



    @property
    def SOC(self):
        """Porcentaje restante de batería"""
        return self.capacidad / self.capacidad_nominal * 100
```

resultados.py

```
from ..utilidades.manejar_resultados import imprimir1d, imprimir2d, imprimir3
d, guardar_excel


class Resultados:
    def __init__(self, ruta):
        """Todos los resultados están en unidades del SI, como aclaración, la
 pendiente <m> está en radianes
        y la energia_km en Wh/km"""
        self.t = ruta.t[:-1]
        self.F, self.F_res, self.f_roz, self.f_ae, self.f_g, self.f_al, self.
f_aa = [], [], [], [], [], [], []
        self.M = []
        self.P_mecanica, self.P_electrica = [], []
        self.SOC = []
        self.V_bateria = []
        self.I_bateria = []
        self.V_motor = []
        self.I_motor = []
        self.distancia_recorrida = [0]
        self.v = []
        self.v_consigna = [v * 3.6 for v in ruta.v[:-1]]
        self.a = []
        self.h = ruta.h[:-1]
        self.m = ruta.m[:-1]
        self.energia_km = 0
        self.nombre_ruta = ruta.nombre


    def trazar(self, *args: str, mostrar: bool=True, guardar: bool=False, lib
reria3d: str='matplotlib'):
        """La forma de entrada de los argumentos será "x-y1,y2"
        para que, en la impresión 2d, una variable tenga eje propio debe de e
star precedida por una barra '|'
        mostrar: muestra la gráfica
        guardar: guarda la gráfica
        libreria3d: será 'matplotlib' o 'pyplot'"""
        for arg in args:
            ejes = [eje.split(',') for eje in arg.split('-')]
            dimension = len(ejes)

            if dimension == 1:
                lx = ejes[0]
                imprimir1d(lx, self)

            elif dimension == 2:
                x, ly = ejes[0][0].replace(' ', ''), ejes[1]
```

```
                imprimir2d(x, ly, self, self.nombre_ruta, guardar, mostrar)

            elif dimension == 3:
                x, y, z = [ejes[i][0].replace(' ', '') for i in range(3)]
                imprimir3d(x, y, z, self, self.nombre_ruta, mostrar, libreria
3d)


    def guardar_excel(self):
        guardar_excel(self, self.nombre_ruta)
```

ruta.py

```
from pandas import read_csv
from ..configuracion import RUTA_RUTAS, FRECUENCIA_MUESTREO, PASO
from math import asin
import re
import numpy as np


def interpolar_h(h: list):
    """Devuelve la altura interpolando linealmente entre las variaciones
    de esa forma se evitan los escalones de altura que provocan pendientes ir
reales"""
    res = []
    d = 0
    for i in range(len(h) - 1):
        if h[i] == h[i+1]:
            d += 1
        else:
            res.extend(np.linspace(h[i], h[i+1], (d+1), endpoint=False))
            d = 0
    res.extend(np.linspace(h[i], h[i+1], d, endpoint=False))
    res.append(h[-1])
    return res



def interpolar_lista(lista: list, frecuencia_muestreo: int):
    """Adapta la lista dada a la frecuencia de muestreo dada mediante interpo
lación lineal"""
    res = []
    for i in range(len(lista) - 1):
        res.extend(np.linspace(lista[i], lista[i+1], frecuencia_muestreo, end
point=False))
    res.append(lista[-1])
    return res


class Ruta:
    """Clase que almacena los parámetros de interés de la ruta dada"""
    def __init__(self, nombre_ruta: str=None):
        # Para la ruta predefinida tipo escalón
        if nombre_ruta and re.match(r'^escalon(-
\d+\.*\d*){4}', nombre_ruta):
            t1, v1, t2, v2 = [float(n) for n in re.findall(r'\d+\.*\d*', nomb
re_ruta)]
            self.v = [v1 / 3.6 if i <= t1 else v2 / 3.6 for i in range(int(t1
+t2))]
            self.m = [0] * int(t1+t2)
```

```
            self.h = [0] * int(t1+t2)
        # Rutas grabadas
        elif nombre_ruta: #Crea una ruta a partir de los datos
            archivo = read_csv(f'{RUTA_RUTAS}/{nombre_ruta}.csv')
            self.v = list(archivo['v (m/s)'])
            self.h = interpolar_h(list(archivo['h (m)']))
        # Rutas vacías
        else:
            self.v, self.h, self.m, self.t = [], [], [], []
            self.largo = 0

        # Guardamos el nombre de la ruta en un parámetro
        self.nombre = nombre_ruta
        # Adaptamos la ruta a la frecuencia de muestreo mediante interpolació
n
        if nombre_ruta:
            self.t = interpolar_lista(range(len(self)), FRECUENCIA_MUESTREO)
            self.v = interpolar_lista(self.v, FRECUENCIA_MUESTREO)
            self.largo = sum(self.v) * PASO
            self.h = interpolar_lista(self.h, FRECUENCIA_MUESTREO)
            self.m = [0]
            # Calculamos la pendiente en cada momento
            for i in range(len(self.h) - 1):
                var_h, var_d = self.h[i+1] - self.h[i], self.v[i] * self.t[i]
                if abs(var_h) < abs(var_d):
                    self.m.append(asin(var_h / var_d))
                else:
                    self.m.append(0)



    def __getitem__(self, i):
        """Añadimos la funcionalidad de indexar la ruta"""
        start = i.start * FRECUENCIA_MUESTREO if i.start else 0
        stop = i.stop * FRECUENCIA_MUESTREO if i.stop else len(self)
        step = i.step * FRECUENCIA_MUESTREO if i.step else 1

        i = slice(start, stop, step)
        nueva_ruta = Ruta()
        nueva_ruta.v, nueva_ruta.h, nueva_ruta.m, nueva_ruta.t = self.v[i], s
elf.h[i], self.m[i], self.t[i]
        nueva_ruta.nombre = self.nombre + f' [{start / FRECUENCIA_MUESTREO}-
{stop / FRECUENCIA_MUESTREO}]'
        return nueva_ruta


    def __len__(self):
        """Añadimos el atributo de longitud (cantidad de elementos) a la ruta
"""
```

```
        return len(self.v)


    def __mul__(self, valor):
        """Añadimos la funcionalidad de multiplicar la ruta para alargarla de
 forma artificial"""
        nueva_ruta = Ruta()
        nueva_ruta.v, nueva_ruta.h, nueva_ruta.m, nueva_ruta.t = self.v * val
or, self.h * valor, self.m * valor, self.t * valor
        nueva_ruta.nombre = self.nombre + f' x {valor}'
        return nueva_ruta
```

funciones_utiles.py

```
"""Módulo que agrupa varias funciones de uso general en el proyecto"""


from math import inf
import time
from datetime import timedelta


class Cronometro:
    """Función útil para estimar el tiempo restante de simulación"""
    def __init__(self, iteraciones_totales: int):
        self.t0 = time.time()
        self.iteraciones_totales = iteraciones_totales
        self.iteraciones = 0
        self._restante = inf


    def imprimir_restante(self):
        """Debe colocarse al final del bucle medido por el número de iteracio
nes,
        imprime directamente el tiempo restante en formato h:mm:ss"""
        t = time.time()
        self.iteraciones += 1
        if self.iteraciones < self.iteraciones_totales:
            t_res = int((t-
self.t0) * (self.iteraciones_totales - self.iteraciones) / self.iteraciones)
        else:
            t_res = 0
        minutos_restantes = timedelta(seconds=t_res)
        if  minutos_restantes != self._restante:
            print(f'Tiempo restante: {minutos_restantes}.', end='\r')
            self._restante = minutos_restantes
        return self._restante


def datos_txt(ruta_fichero: str, recibir_listas: bool=False):
    """Funcion diseñado para leer los datos de los componentes en formato txt
 con la forma:
    parámetro1,valor,unidad
    parámetro2,valor,unidad
    ...
    el parámetro unidad es opcional.
    Parámetros de la función:
        ruta_fichero: str. Ruta al fichero con los datos
        recibir_listas: bool. Será True si lo que se quiere obtener de cada p
arámetro es una lista (no habrá parámetro unidad)
    Devuelve un diccionario con los datos y su valor
```

```
    """
    with open(f'{ruta_fichero}.txt', 'r') as archivo:
        lista = archivo.readlines()
    dic_datos = {}
    for linea in lista:
        linea = linea.split(',')
        linea[-1] = linea[-
1].replace('\n', '') #Elimina el salto de lina del ultimo elemento
        if recibir_listas:
            dic_datos[linea[0]] = [float(e) for e in linea[1:]]
        else:
            dic_datos[linea[0]] = float(linea[1])

    return dic_datos


def saturador(entrada: float, min: float=-inf, max: float=inf):
    """Función que analiza el valor de entrada y lo satura a los valores míni
mos y máximos especificados"""
    if entrada < min:
        return min
    elif entrada > max:
        return max
    else:
        return entrada
```

manejar_resultados.py

```
"""Contiene funciones de ayuda para imprimir los resultados"""


import matplotlib.pyplot as plt
from mpl_toolkits import mplot3d
from mpl_toolkits.axes_grid1 import host_subplot
import plotly.graph_objects as go
import pandas as pd
import numpy as np
from ..configuracion import RUTA_RESULTADOS, PASO


def imprimir1d(lx, resultados):
    """Imprime los resultados en la forma variable = valor"""
    for x in lx:
        x = x.replace(' ', '') #Ignora los espacios
        print(f'{x} = {getattr(resultados, x)}')


def imprimir2d(x, ly, resultados, nombre_ruta, guardar, mostrar):
    """Imprime gráficas formateandolas adecuadamente"""
    # Obtenemos la variable x y la lista de variables y
    fig = plt.figure()
    host = fig.add_subplot()
    ax = {}
    etiquetas = []
    colores = 'green', 'purple', 'orange', 'blue', 'brown', 'cyan', 'gray', '
olive', 'pink'
    tamaño_fuente = 12
    modo_fuente = 'normal'
    i = 0
    a = 0 #Alejamiento de los ejes
    for y in ly:
        y = y.replace(' ', '') #Ignora los espacios
        if '|' in y: #Se requiere un eje propio
            y = y.replace('|', '')
            ax[y] = host.twinx()
            ax[y].set_ylabel(y, color=colores[i], fontsize=tamaño_fuente, wei
ght=modo_fuente)
            ax[y].spines['right'].set_position(('outward', 50 * a))
            a += 1
        e, = plt.plot(getattr(resultados, x), getattr(resultados,y), label=y,
 color=colores[i])
        etiquetas.append(e)
        i += 1

    host.legend(handles=etiquetas, loc='best')
```

```
    host.set_ylabel(ly[0], color=colores[0], fontsize=tamaño_fuente, weight=m
odo_fuente)
    host.set_xlabel(x, fontsize=tamaño_fuente, weight=modo_fuente)
    host.grid()
    plt.tight_layout()
    if guardar:
        plt.savefig(f'{RUTA_RESULTADOS}/{nombre_ruta}[{x}-
{", ".join(ly).replace("|", "")}].png')
    if mostrar:
        plt.show()


def imprimir3d(x, y, z, resultados, nombre_ruta, mostrar, libreria):
    """Imprime gráficas en 3D, si la fracuencia de muestreo es alta puede dar
 un fallo por sobrecarga de memoria"""
    X, Y = np.meshgrid(getattr(resultados, x), getattr(resultados, y))
    Z = np.tile(getattr(resultados, z), (len(getattr(resultados, z)), 1))
    if libreria == 'matplotlib':
        fig = plt.figure()
        ax = fig.add_subplot(1,1,1, projection='3d')
        ax.set_xlabel(x)
        ax.set_ylabel(y)
        ax.set_zlabel(z)
        ax.plot_surface(X, Y, Z)
        if mostrar:
            plt.show()
    if libreria == 'pyplot':
        datos = [go.Surface(x=X, y=Y, z=Z)]
        diseño = {
            'scene':{
                'xaxis':{'title':{'text':x, 'font':{'size':18}}},
                'yaxis':{'title':{'text':y, 'font':{'size':18}}},
                'zaxis':{'title':{'text':z, 'font':{'size':18}}}
                    }
                }
        fig = go.Figure(data=datos, layout=diseño)
        if mostrar:
            fig.show()


def _adaptar_excel(datos):
    """Función necesaria para igualar los arrays de la función resultados
    y evitar un error en pandas al guardarlo, además adapta los resultados pa
ra
    que se muestre un dato por segundo"""
    for clave in datos.keys():
        if not isinstance(datos[clave], list):
            datos[clave] = [datos[clave]]
```

```
    largo = max([len(datos[clave]) for clave in datos.keys()]) * PASO
    for clave in datos.keys():
        datos[clave] = datos[clave][::int(1/PASO)]
        while len(datos[clave]) < largo:
            datos[clave].append('')
    return datos


def guardar_excel(resultados, nombre_ruta):
    """Guarda un excel con todos los datos de la simulación"""
    df = pd.DataFrame(data=_adaptar_excel(resultados.__dict__))
    writer = pd.ExcelWriter(f'{RUTA_RESULTADOS}/{nombre_ruta}.xlsx')
    df.to_excel(writer, 'Resultados', index=False, freeze_panes=(1,0))
    writer.save()
```

arreglar_rutas.py

```
"""Da formato a las rutas obtenidas de la aplicación GPS Logger"""


import requests
import numpy as np


def arreglar(nombre):
    print(nombre.split('/')[-1])
    nombre_arreglado = nombre.split('/')[-1].replace('.csv', '')+'.csv'
    datos_arreglados = []
    archivo = [linea.split(',') for linea in open(nombre, 'r').readlines()[:-
1]]
    datos_arreglados.append(('v (m/s)', 'h (m)')) # Primera fila con la leyen
da
    datos = {'long':[], 'lat':[], 'h':[], 'v':[]}

    def obtener_elevacion(lat, long):
        """Obtiene la elevación a partir de Bing Maps, ya que la calculada po
r la
        aplicacion no sirve"""
        CLAVE_BING = 'INSERTAR AQUÍ LA CLAVE DE LA API'
        url = f'http://dev.virtualearth.net/REST/v1/Elevation/List?points={la
t},{long}&key={CLAVE_BING}'
        r = requests.get(url).json()  # json object, various ways you can ext
ract value
        elevacion = r['resourceSets'][0]['resources'][0]['elevations'][0]
        return elevacion


    datos['long'] = [float(fila[0]) for fila in archivo[1:]]
    datos['lat'] = [float(fila[1]) for fila in archivo[1:]]
    datos['v'] = [round(float(fila[4]),2) for fila in archivo[1:]]
    datos['h'] = [obtener_elevacion(lat, long) for lat, long in zip(datos['la
t'], datos['long'])]


    for i in range(len(datos['v'])):
        datos_arreglados.append([datos['v'][i], datos['h'][i]])

    np.savetxt(dir_rutas + nombre_arreglado, datos_arreglados, delimiter=',',
 fmt='%s')


if __name__ == '__main__':
    import os
    dir_rutas_bruto = '../../../datos/rutas/rutas_en_bruto/'
```

```
    dir_rutas = '../../../datos/rutas/'

    nombre = 'Ruta6.csv'
    arreglar(dir_rutas_bruto + nombre)

    # for nombre in os.listdir(dir_rutas_bruto):
    #     if '.csv' in nombre or '.CSV' in nombre:
    #         arreglar(dir_rutas_bruto + nombre)
    print('Listo')
```

regresion_celdas.py

```
"""Guarda un archivo con los coeficientes de la regresión polinómica de las c
eldas de batería"""

import numpy as np
from pandas import read_csv

# Variables
GRADO_POLINOMIOS = 4 # Aparentemente es el que mejor resultado da
MODELO_BATERIA = 'NCR18650B'
RUTA_DATOS_BATERIA = f'../../datos/celdas'
DATOS_BATERIA = f'{RUTA_DATOS_BATERIA}/{MODELO_BATERIA}/datos_descarga.csv'

# Se importan los datos sacados a partir de la gráfica proporcionada por el f
abricante
archivo = read_csv(DATOS_BATERIA)
x = list(archivo['mA-descarga']) # entrada - mA consumidos
# Salidas en funcion del ratio de descarga
V_02C = list(archivo['0.2C'])
V_05C = list(archivo['0.5C'])
V_1C = list(archivo['1C'])
V_2C = list(archivo['2C'])

# Obtención de los coeficientes
coefs_02C = np.polyfit(x, V_02C, GRADO_POLINOMIOS)
coefs_05C = np.polyfit(x, V_05C, GRADO_POLINOMIOS)
coefs_1C = np.polyfit(x, V_1C, GRADO_POLINOMIOS)
coefs_2C = np.polyfit(x, V_2C, GRADO_POLINOMIOS)

# Guardar los coeficientes
with open(f'{RUTA_DATOS_BATERIA}/{MODELO_BATERIA}/coeficientes.txt', 'w') as
archivo_coefs:
    archivo_coefs.write(f'0.2C,{",".join([str(c) for c in coefs_02C])}\n')
    archivo_coefs.write(f'0.5C,{",".join([str(c) for c in coefs_05C])}\n')
    archivo_coefs.write(f'1C,{",".join([str(c) for c in coefs_1C])}\n')
    archivo_coefs.write(f'2C,{",".join([str(c) for c in coefs_2C])}\n')
```

ANEXO 5 – RESULTADOS RUTAS 1, 2 Y 3

RUTA 1

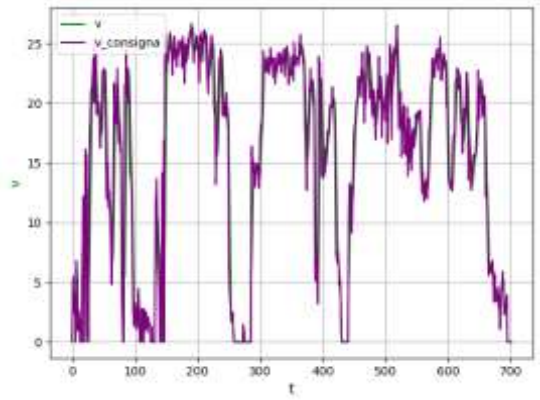


Ilustración 42 Velocidad en la Ruta 1

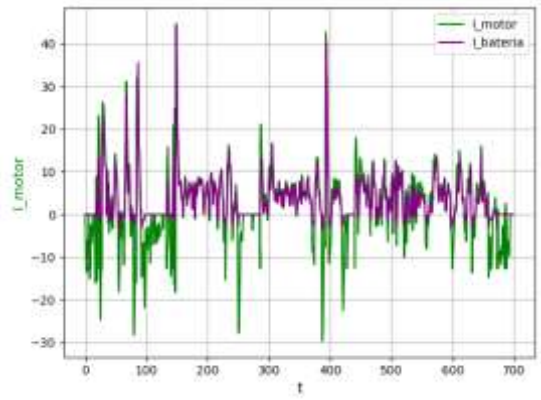


Ilustración 43 Intensidad en la Ruta 1

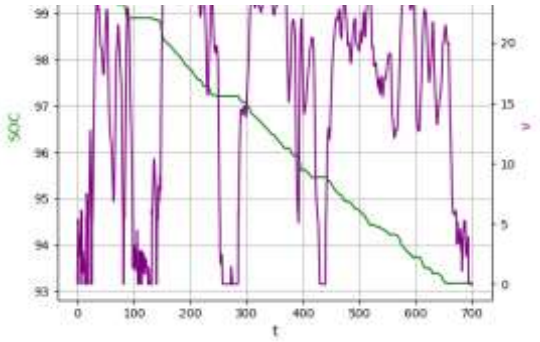


Ilustración 44 SOC en la Ruta 1

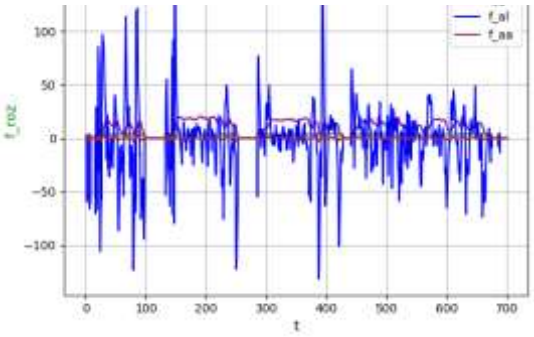


Ilustración 45 Fuerzas en la Ruta 1

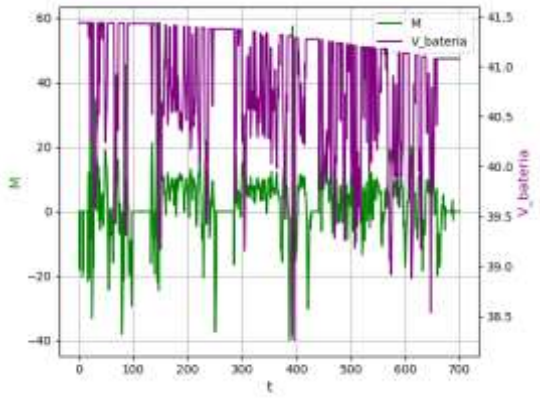


Ilustración 46 Par y tensión en la batería en la Ruta 1

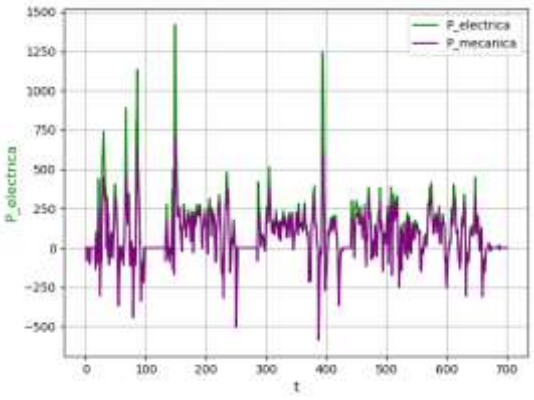


Ilustración 47 Potencia en la Ruta 1

RUTA 2

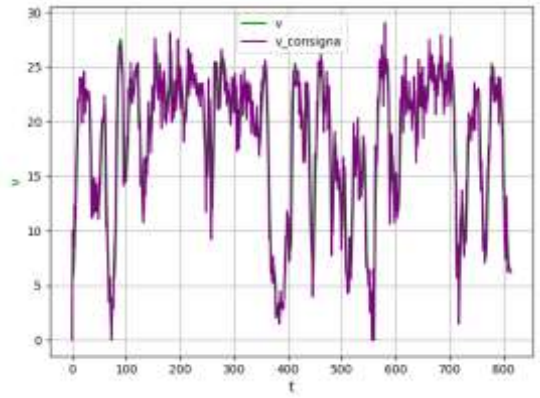


Ilustración 48 Velocidad en la Ruta 2

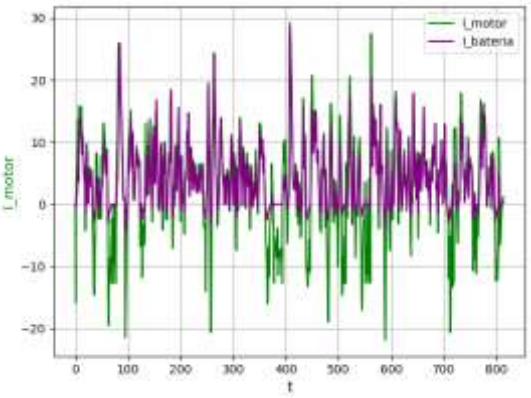


Ilustración 49 Intensidad en la Ruta 2

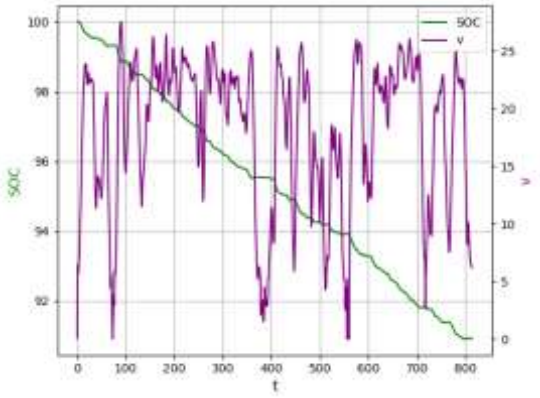


Ilustración 50 SOC en la Ruta 2

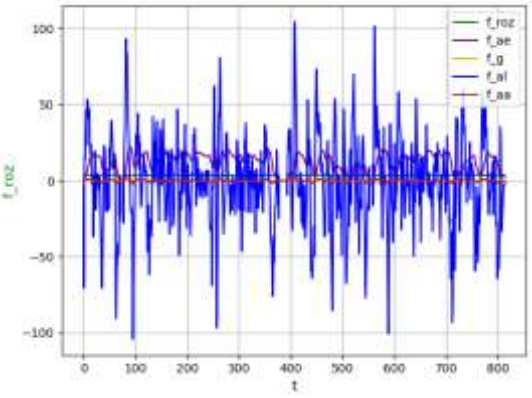


Ilustración 51 Fuerzas en la Ruta 2

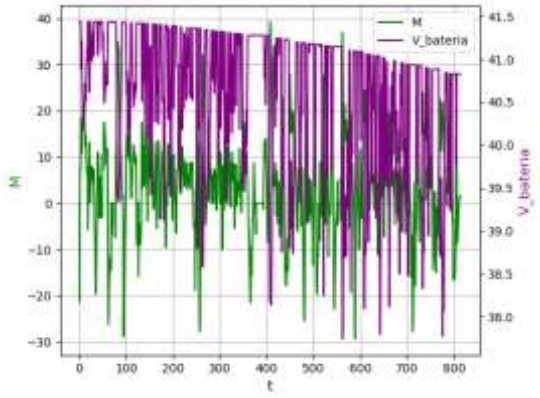


Ilustración 52 Par y tensión en la batería en la Ruta 2

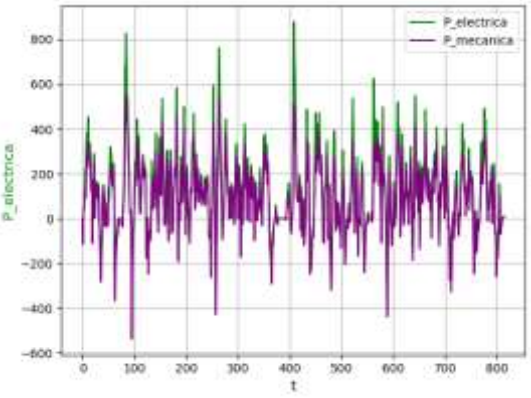


Ilustración 53 Potencia en la Ruta 2

RUTA 3

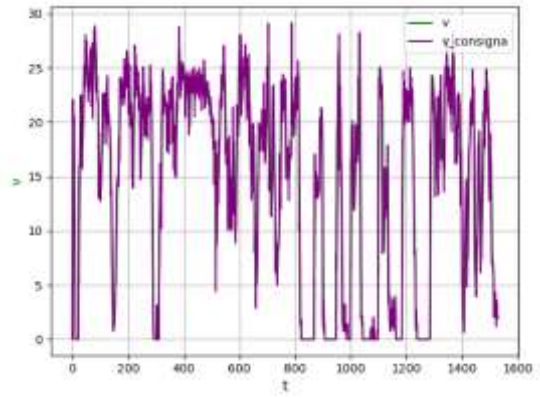


Ilustración 54 Velocidad en la Ruta 3

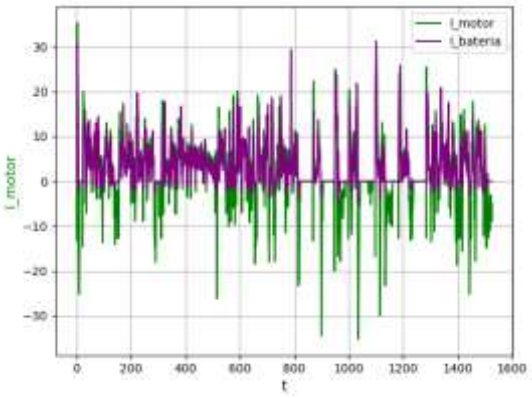


Ilustración 55 Intensidad en la Ruta 3

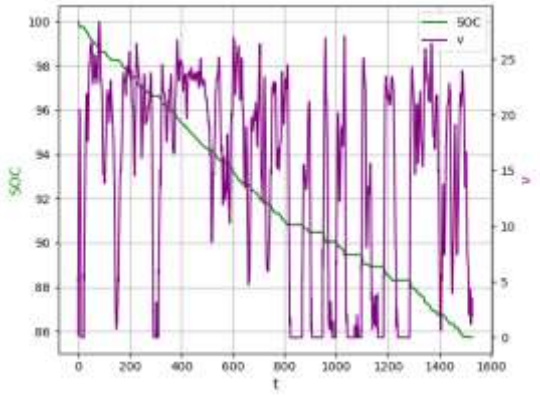


Ilustración 56 SOC en la Ruta 3

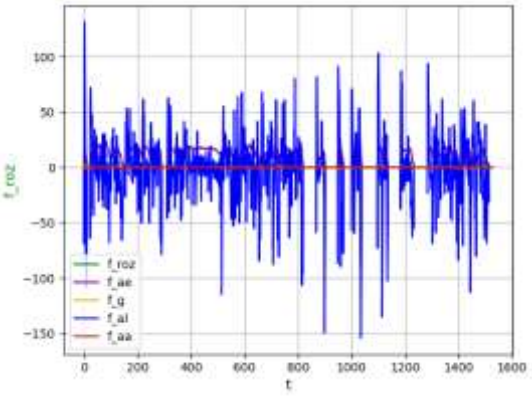


Ilustración 57 Fuerzas en la Ruta 3

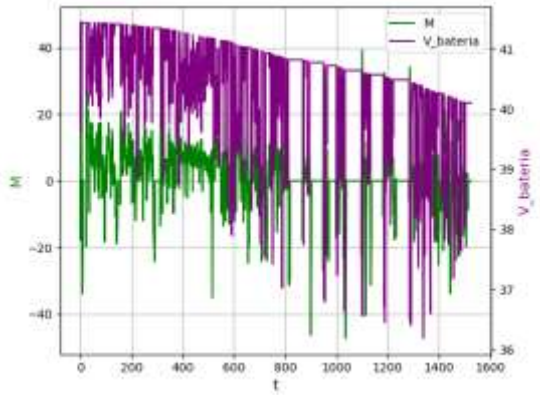


Ilustración 58 Par y tensión en la batería en la Ruta 3

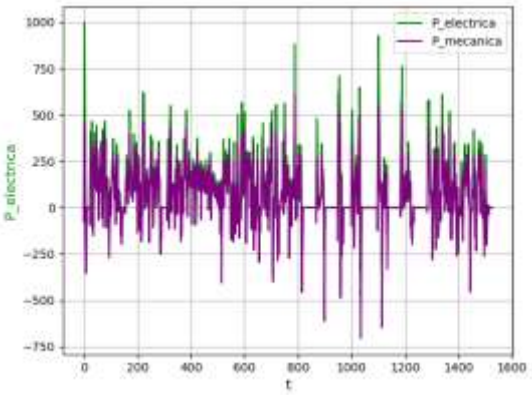


Ilustración 59 Potencia en la Ruta 3

ANEXO 6 – FRENADO REGENERATIVO

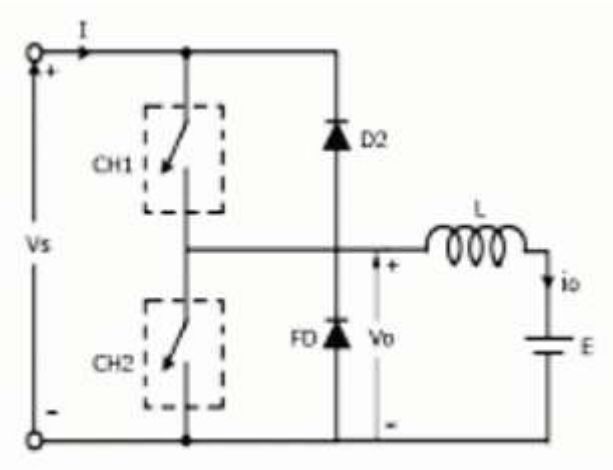


Ilustración 60 Chopper en 2 cuadrantes

El modelo en Python se ha escrito tomando como base el comportamiento de un chopper en dos cuadrantes como el de la figura, partimos de estas dos ecuaciones:

$$V_0 = V_S * D$$

$$I = I_0 * D$$

De las ecuaciones del motor deducimos que:

$$I_0 = \frac{V_0 - E}{R} = \frac{V_0 - K * \omega}{R} = \frac{V_S * D - K * \omega}{R} \rightarrow$$

$$I = \frac{V_S * D - K * \omega}{R} * D$$

Derivando esta función obtenemos su máximo y su mínimo:

- mínimo en $D = 0$, con $I = 0$
- máximo en $D = \frac{K}{2 * V_S} * \omega$, con $I = -\frac{K^2 \omega^2}{4R * V_S}$

Esto implica que, con la consigna de velocidad dada en el modelo, el duty estará en torno al 0, forzado por la entrada, lo cual explica la baja recuperación que aparece en el modelo, en una situación real, podría programarse el controlador para tener dos modos de frenado, el débil, en el que se busca la máxima recuperación y se deje el resto de la potencia de frenado al freno mecánico, y el fuerte en el que el duty se fuerce a 0 consiguiendo así una frenada regenerativo más potente pero asumiendo que esa energía será desaprovechada, sería recomendable realizar pruebas con el motor físicamente para determinar un límite para este frenado, debido a que la energía que no se recupera se está convirtiendo en calor dentro del motor, lo que podría dañarlo.