



FIB

Facultat d'Informàtica
de Barcelona

UNIVERSITAT POLITÈCNICA DE CATALUNYA

Títol: Sistema de control y monitorización de datos de una bicicleta eléctrica con sensor de par en el pedalier utilizando teléfonos inteligentes

Volum:1

Alumne: Carlos Morata Núñez

Director/Ponent: Antonio B. Martínez Velasco

Departament: ESAII

Data: 12 de Maig de 2012

DADES DEL PROJECTE

Títol del Projecte: Sistema de control y monitorización de datos de una bicicleta eléctrica con sensor de par en el pedalier utilizando teléfonos inteligentes

Nom de l'estudiant: Carlos Morata Núñez

Titulació: Enginyeria Informàtica

Crèdits: 37,5

Director/Ponent: Antonio B. Martínez Velasco

Departament: ESAII

MEMBRES DEL TRIBUNAL (nom i signatura)

President: Joan Climent Vilaró

Vocal: Francisco Javier Larrosa Bondia

Director: Antonio B. Martínez Velasco

QUALIFICACIÓ

Qualificació numèrica:

Qualificació descriptiva:

Data:

Índice

1. Introducción	9
2. Estado del Arte	11
2.1.Bicicleta eléctrica	11
2.2.Aplicaciones para smartPhones	16
2.3.Sensores de fuerza	17
2.4.Medidores del par realizado por el ciclista	19
2.4.1.SRM	19
2.4.2.Power Tap	21
2.4.3.Look keo power	22
2.4.4.Ergomo	23
2.4.5.Encoders ESAll	24
2.4.6.Galgas ESAll	26
3. Objetivos	29
4. Banco de Pruebas	33
4.1.Funcionamiento	33
4.2.Control del freno electromagnético	34
4.2.1.Modificación electrónica ecocontrol	35
4.2.2.Firmware	39
4.3.Electrónica 4PGA	44
4.3.1.Escalado de la ganancia	46
4.3.2.Ajuste del “offset”	47
4.3.3.Coeficientes de temperatura y EEPROM	47
4.3.4.Microcontrolador	48
4.4.Medición de fuerzas	49
4.4.1.Principios de la extensiometría	49
4.4.2.Calibración	49
4.4.3.Recalibración en laboratorio	50

4.5. Encoder	56
4.6. Modificaciones en la estructura del rodillo	60
4.7. Cálculo del par realizado por la bicicleta	65
5. Pedalier Thun	67
5.1. Especificaciones	67
5.2. Principios de funcionamiento	70
5.3. Desensamblaje del pedalier	72
5.4. Incorporación al diseño de la bicicleta	74
5.5. Simulación de la bicicleta	77
5.6. Modelo para la emulación del control de la bicicleta	86
5.6.1. Bloque CAN Interface	86
5.6.2. Bloques CanWrite y CanRead	88
5.6.3. Bloque SampleTimeSync	89
5.6.4. Bloque Ecocontrol	90
5.6.5. Bloque Rodillo	91
5.6.6. Bloque 4PGA	91
5.6.7. Bloque ecogate	91
5.7. Obtención de la curva motor	92
5.7.1. Entrega de par del freno eléctrico	92
5.7.2. Curva motor	95
6. Estrategias de Control: Diseño y Selección	97
6.1. Estrategias diseñadas	99
6.1.1. Estrategia 1	99
6.1.2. Estrategia 2	100
6.1.3. Estrategia 3	101
6.2. Estrategia seleccionada	101
6.3. Implementación en Simulink	101
6.4. Implementación de la estrategia en el controlador de la bicicleta	103
6.5. Evaluación de la estrategia seleccionada	104

6.5.1.Evaluación en el banco de pruebas	104
6.5.2.Evaluación en el exterior	105
7. Monitorización	107
7.1.Bluetooth	108
 7.1.1.Electronica ecoGate	109
 7.1.2.Modbus serie	110
 7.1.3.Modbus sobre java	115
7.2.Aplicación para teléfono Android	115
 7.2.1.Ciclo de vida de una aplicación Android	116
 7.2.2.Estructura de la aplicación	117
 7.2.3.Conexión de un dispositivo	118
 7.2.4.Pestaña Speed	119
 7.2.5.Pestaña Energy	120
 7.2.6.Pestaña Map	121
 7.2.7.Pestaña Summary	122
7.3.Aplicación para tablet Android	123
 7.3.1.Grabación de datos	125
 7.3.2.Visualización de experimentos	126
8. Evaluación de Resultados	129
 8.1.Metas conseguidas	129
 8.2.Ampliaciones/mejoras	129
9. Valoración Económica	131
 9.1.Recursos materiales	131
 9.2.Recursos humanos	132
 9.3.Coste Total	132
10. Planificación	133
11. Bibliografía	135
12. Agradecimientos	137
13. Anexos	139

Anexo 1: El puente de Wheatstone	139
Anexo 2: Fragmentos de código de las aplicaciones Android	141
Anexo 3: Bus CAN	159

Capítulo 1

Introducción

Este proyecto final de carrera (PFC) es parte de un proyecto donde colabora la UPC con la empresa Ecobike y que consiste en el desarrollo de una bicicleta eléctrica que requiere incorporar tecnologías de la información. Ecobike es una empresa pionera en la comercialización y desarrollo de bicicletas eléctricas en España. La fabricación de las bicicletas se produce en Cataluña y gracias a la colaboración con la UPC, Ecobike dispone de un sistema electrónico propio. Ambos puntos permiten tener un control directo sobre el diseño y la tecnología de las bicicletas, así como estar preparados para desarrollar nuevos productos en función de las necesidades del mercado.



Figura 1.1: Bicicleta Ecobike modelo elegance

Actualmente el uso de las bicicletas eléctricas está experimentando un crecimiento espectacular, sobretodo para desplazamientos urbanos, y muchos fabricantes, tanto los nuevos como los tradicionales productores de bicicletas, están comenzando a comercializar modelos de bicicletas eléctricas intentando ocupar una cuota del cada vez más importante mercado de las *ebikes*. Para hacernos una idea de la situación actual del mercado, en la feria Eurobike (la más importante de Europa) se dedicó un pabellón completo a las bicicletas eléctricas. Con toda esta competencia, desarrollar una bicicleta con un alto nivel de prestaciones, así como disponer de una extensa

1. Introducción

gama que se adapte a las necesidades de cada posible usuario, puede ser un buen comienzo para atraer la atención del público.



Figura 1.2: Feria Eurobike: Zona de test de bicicletas eléctricas

Dentro de este marco, mi Proyecto Final de Carrera se basa principalmente en la consecución de dos objetivos:

- a) Diseñar nuevas estrategias de control del motor de la bicicleta cuando un usuario-ciclista ejerce un Par sobre los pedales,
- b) Visualizar de forma gráfica el trabajo realizado por el ciclista en comparación al aportado por el motor, utilizando un Smartphone. Por otro lado, también se visualizarán los datos de la bicicleta, tanto los derivados de la odometría, como los particulares de una bicicleta eléctrica.

Intencionadamente, se ha omitido la mayor parte del código desarrollado, pues está sujeto a un contrato de confidencialidad entre la UPC y Ecobike.

Capítulo 2

Estado del Arte

En este apartado se presenta información de los avances hechos hasta el momento en bicicletas eléctricas. También se hace un repaso de las técnicas relacionadas con el presente PFC, tales como sensores detectores de fuerza y electrónicas que se utilizarán para la consecución de los objetivos.

2.1.Bicicleta eléctrica

Una bicicleta eléctrica es un tipo de vehículo compuesto por una bicicleta a la que se le ha acoplado un motor eléctrico y todos los componentes necesarios para su funcionamiento con la finalidad de ayudar en el avance de la bicicleta. En la Unión Europea, legalmente tienen la consideración de bicicletas a efectos de circulación, siempre que:

- Sólo proporcionen asistencia mientras se pedalea.
- El motor se desconecte a partir de 25 km/h.
- Su potencia no sea superior a 250 W.

Los motores que utilizan las bicicletas eléctricas son motores brushless. Un motor eléctrico sin escobillas o motor brushless es un motor eléctrico que no emplea escobillas para realizar el cambio de polaridad en el rotor. Los motores brushless, se muestran muy ventajosos con respecto a los motores con escobillas, ya que son más baratos de fabricar, pesan menos y requieren menos mantenimiento. En contrapartida, su control es más complejo, pero gracias a la electrónica, esta complejidad prácticamente se ha eliminado a día de hoy. El motor utilizado por las bicicletas Ecobike consta de tres bobinas, así como de tres sensores de efecto hall que nos indican en todo momento la posición del rotor del motor. Se sabe que existe una codificación de colores intencionada para relacionar los sensores de efecto Hall (en el interior del motor), con las tres bobinas. Para que gire el motor se deben polarizar las bobinas dependiendo del estado en que se encuentren los sensores en cada instante.



Figura 2.1: Estator de un motor Brushless utilizado por ecobike

En el mercado existen actualmente dos estrategias para controlar el motor: bicicletas ped-electric y bicicletas power on demand.

Las bicicletas que se decantan por la estrategia power on demand, permiten circular a la velocidad máxima (25km/h) con el único requisito que el ciclista esté pedaleando, sin necesidad de ejercer un gran par sobre los pedales. Las bicicletas fabricadas por Ecobike utilizan esta estrategia.

Las ped-electric son las bicicletas que llevan incorporado un sensor de par y proporcionan una ayuda proporcional al par realizado por el ciclista sobre los pedales. Si se quiere una gran ayuda se tiene que realizar un gran esfuerzo. La mayoría de sistemas utilizados por los diferentes fabricantes utilizan los mismos principios. Es por ello que para dar una idea más clara, se explicará con más detalle el sistema de Panasonic. Las bicicletas Panasonic utilizan motor central, esto significa que el motor está diseñado para estar en el centro de la bicicleta, permitiendo una fácil integración con el tren motriz. Esta configuración también permite el uso completo del cambio de marchas de la bicicleta. La energía del motor pasa a través de la transmisión de la bicicleta antes de llegar a la rueda. Las bicicletas Ecobike, en contrapartida, incorporan el motor en la rueda trasera. Para los objetivos de este PFC resulta indiferente la configuración que adapte la bicicleta.

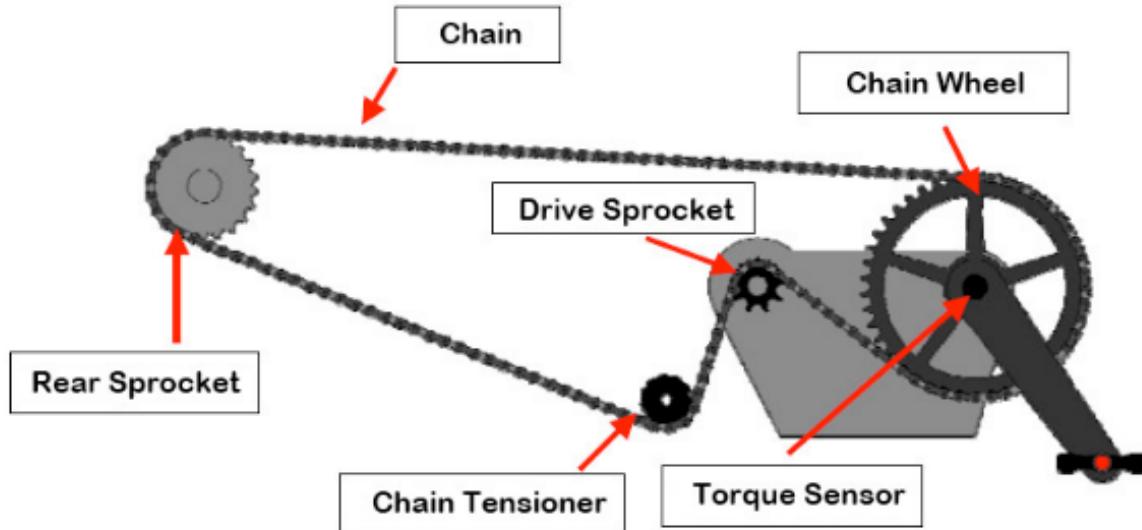


Figura 2.2: Elementos de la transmisión Panasonic

La llamada por Panasonic “drive unit” incorpora el motor, el controlador y el sensor de par. En la siguiente imagen se puede observar la carcasa donde se encuentran todos estos elementos.

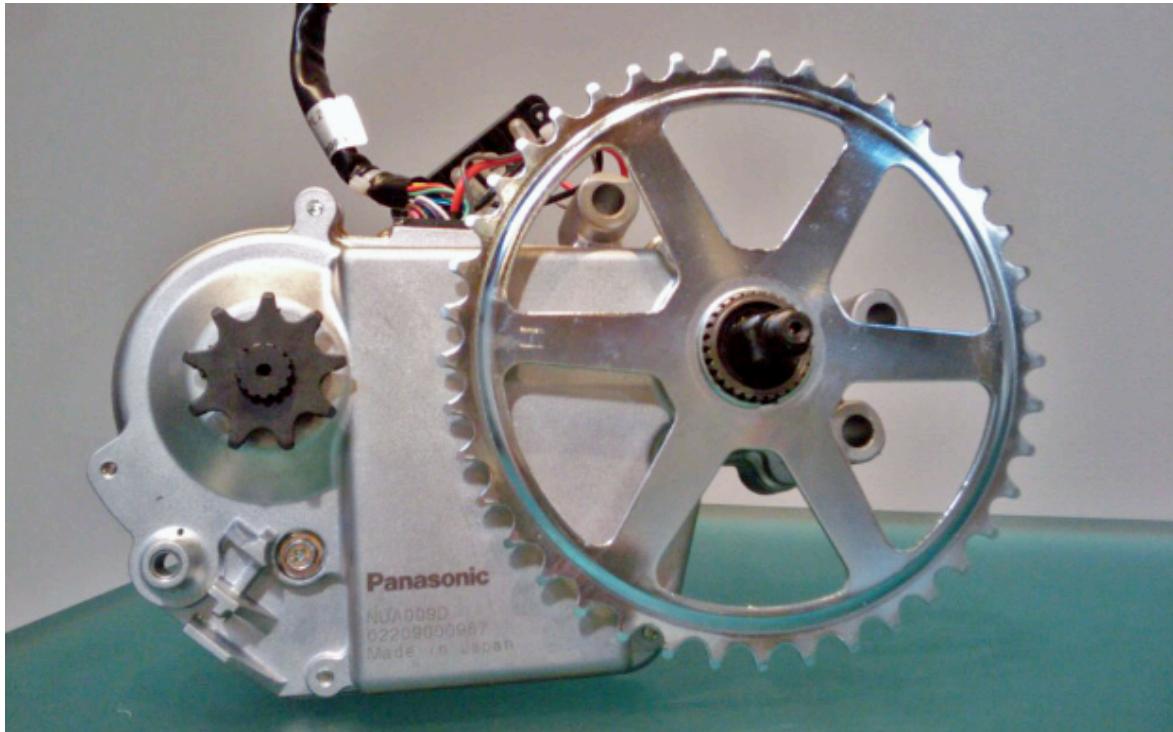


Figura 2.3: Drive Unit Panasonic

El controlador y el sensor de par detectan la fuerza que realiza el ciclista en su pedaleo y ajusta la cantidad de asistencia que el motor le da a través del piñón (que en la figura 2.3 podemos apreciar de color negro).

2. Estado del Arte

El sensor de torque está incorporado en el pedalier. En el siguiente diagrama (figura 2.4) se puede apreciar un esquema de corte transversal del sensor de par, tal y como aparecería si se mirase desde la parte trasera de la bicicleta.

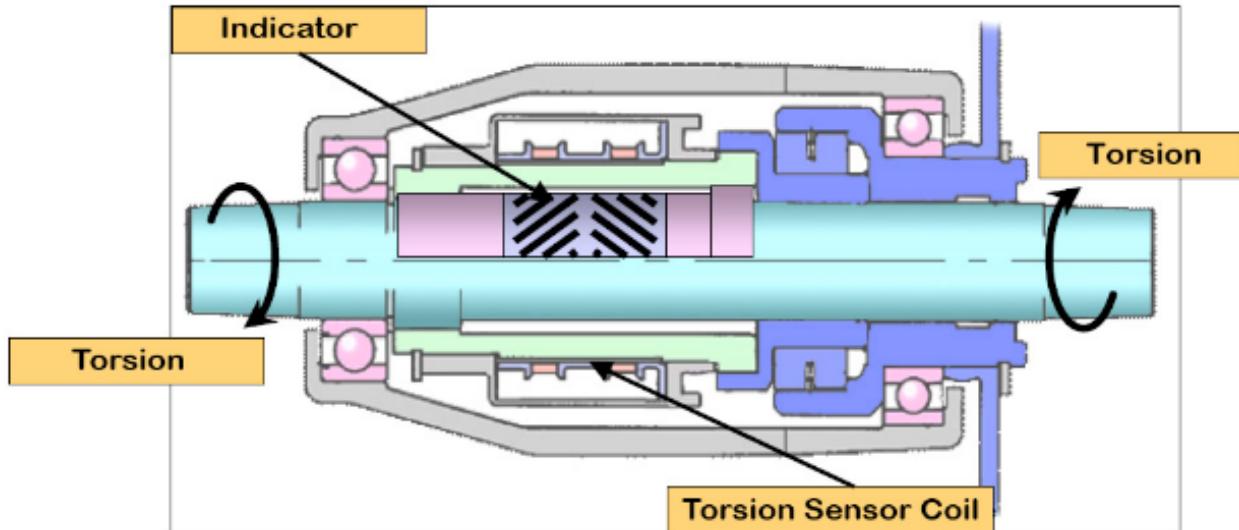


Figura 2.4: Sensor de par Panasonic

Se mide el par mediante la torsión del eje de pedalier. Se utiliza un anillo indicador (magnetizado) en el medio del eje y una bobina que detecta las deformaciones del anillo indicador. A más torsión, más par es ejercido por el ciclista.

Algunos fabricantes de bicicletas ped-electric son: Ave,BH, Flyer, Gazelle, Panasonic, Winora, Kalkhoff y Hercules.



Figura 2.5: Algunos modelos de bicicletas ped electric

La ayuda que brinda el motor al ciclista recibe el nombre de pedaleo asistido. Las bicicletas generalmente permiten al usuario elegir el nivel de asistencia mediante un mando regulador, como por ejemplo, un acelerador o un selector.



Figura 2.6: Ejemplo de selector (izquierda) y acelerador (derecha)

2.2. Aplicaciones para smartPhones

Los teléfonos móviles se han convertido en el gadget ideal de acompañamiento en cualquier momento, pues sus funciones son cada día más variadas. Sin embargo, las aplicaciones que se emplean en el móvil cumplen un rol importante, y cada vez su número es mayor. Dentro de las aplicaciones genéricas para la práctica deportiva, una de las más completas y utilizadas es Endomondo. Ésta es un aplicación para móviles con GPS que se puede usar en el iPhone, en un Blackberry, en teléfonos Nokia o con Android, es una herramienta muy válida para todos aquellas personas que practican deporte al aire libre.

Endomondo permite un seguimiento en tiempo real del recorrido realizado por el ciclista, midiendo distancia, velocidad, altitud y localización gracias al GPS y los mapas de Google Maps.

Además, en endomondo.com se encuentra un sitio donde ver la historia de los entrenamientos realizados, el progreso, etc. Endomondo cuenta con una fuerte función social que permite competir con amigos, compartir opiniones e invitar a concursos.

Además, funciona con Polar WearLink lo que permite monitorizar la frecuencia cardíaca mientras se realiza la ruta en bicicleta.



Figura 2.7: Pantalla principal de la aplicación Endomondo

Las diferentes alternativas que se encuentran en el mercado (como por ejemplo My Tracks de google, Cardio Trainer, Andando, Buddy Runner) ofrecen unas características muy similares a las ofrecidas por Endomondo. Todas ofrecen el seguimiento por GPS de la actividad física realizada.

2.3.Sensores de fuerza

Un sensor de fuerza es un dispositivo que permite medir las deformaciones o esfuerzos que se producen en un objeto cuando se le aplica una fuerza. Inventado por Edward E. Simmons i Arthur C. Rige, el tipo más común de sensor de fuerza está compuesto por una bobina plana de cable conductor introducida dentro de un aislante eléctrico.

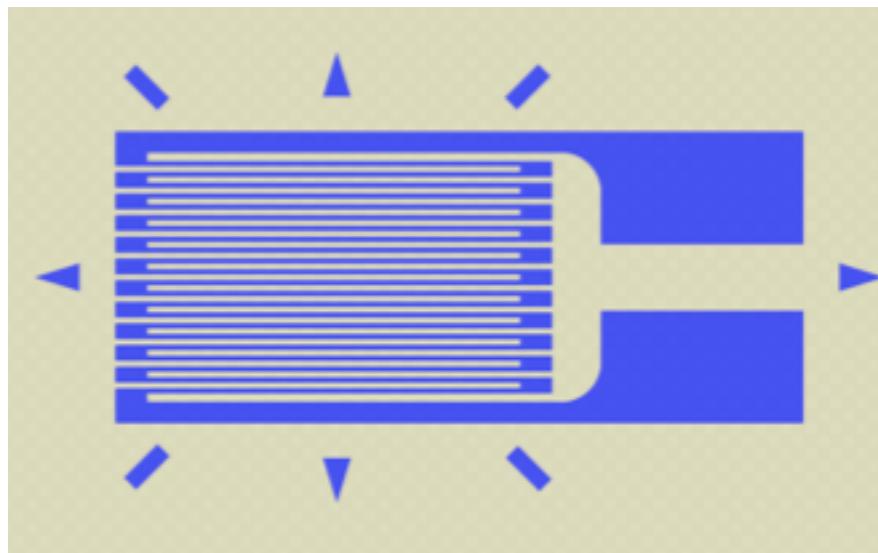


Figura 2.8: Sensor de fuerza de Simmons y Ruge

Este dispositivo se fija en el objeto mediante un adhesivo, como por ejemplo cyanocrilato. Cuando el objeto es deformado, la bobina plana (galga) también se deforma, provocando un cambio en su resistencia eléctrica. Este cambio en la resistencia, normalmente medido por un puente de Wheatstone (mirar anexo), está relacionado con el esfuerzo mediante una cantidad llamada “Factor de Galga” (GF) de la forma siguiente:

$$GF = \frac{\Delta R}{R_G \cdot \epsilon}$$

donde R_G es la resistencia de la galga sin deformar, ΔR es al cambio en la resistencia provocado por el esfuerzo y ϵ es el esfuerzo.

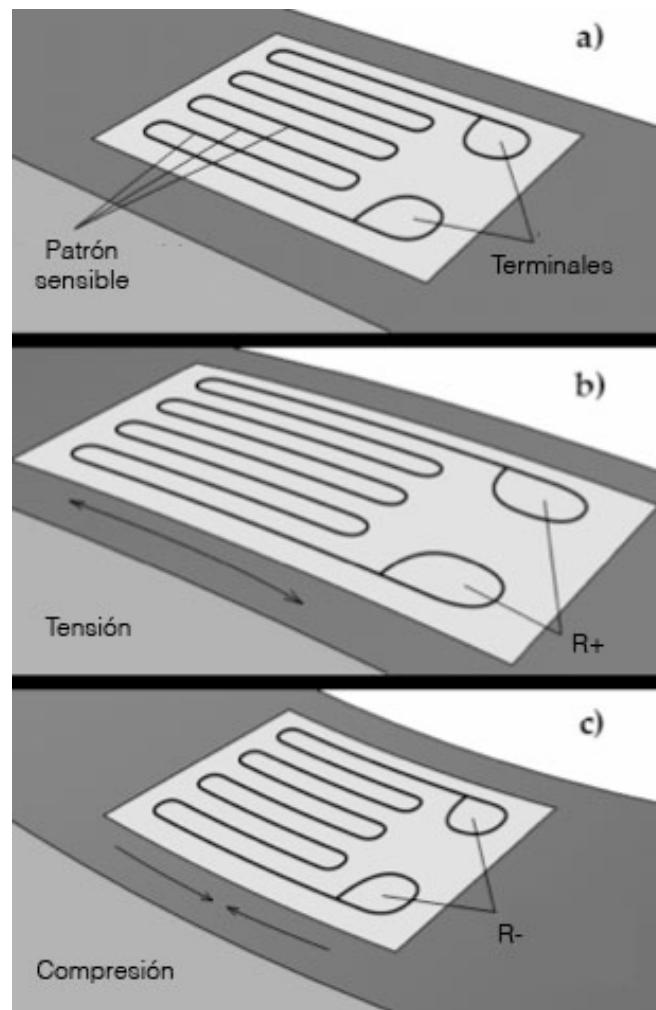


Figura 2.9: Variación de la resistencia debida a la deformación

En función del uso i las condiciones de funcionamiento del sensor de fuerza, se requieren diferentes configuraciones de galgas. Encontramos por ejemplo galgas que deben funcionar durante décadas y galgas que son usadas sólo durante unos días.

Las variaciones térmicas pueden causar multitud de efectos sobre la medida realizada. El objeto puede cambiar de dimensiones debido a la dilatación térmica, cosa que puede hacer variar la resistencia de la galga. Para evitar este fenómeno se fabrican galgas que presentan la misma resistencia eléctrica en un amplio rango de temperaturas, como por ejemplo el constantano. Por otro lado, algunas galgas son diseñadas para que la variación de resistencia debida a la temperatura que sufre la galga, cancele la variación de resistencia debida a la dilatación del material sobre el cual está montada. En el caso de galgas que no auto-compensan, es posible minimizar las medidas derivadas de variaciones de temperatura utilizando puentes de Wheatstone. Para conseguirlo es necesario montar un puente formado por cuatro galgas: dos colocadas en el objeto y dos colocadas de forma que no se vean deformadas al aplicar un esfuerzo al objeto. Las cuatro galgas deben estar a la misma temperatura.

2.4. Medidores del par realizado por el ciclista

En el apartado anterior se ha visto la teoría mediante la que funcionan los sensores de fuerza más comunes. En este se ahonda en los diferentes productos que se pueden encontrar en el mercado para medir el par realizado por un ciclista sobre los pedales. Se muestran también dos prototipos de pedales sensorizados realizados en el departamento ESAII. El sistema elegido para desarrollar este proyecto (pedalier Thun) dispone de su propio capítulo, pues el estudio realizado sobre él es más extenso.

2.4.1. SRM

El sistema de entrenamiento SRM permite conocer en el momento y de forma continua la potencia que está desarrollando el ciclista. El Powermeter se compone de bielas, platos y unidad de medición. La unidad de medición, situada entre la biela y los platos, está fabricada de una aleación de aluminio de alta resistencia y templado especial. En su interior se encuentran galgas extensiométricas que miden la deformación entre las bielas y el plato que transmite el par a la cadena. La deformación del material registra la fuerza resultante (par de giro) así como también se registra la frecuencia de pedaleo (velocidad angular). Estos parámetros se transforman en una señal digital y se transmiten por telemetría (inductivamente) a un sensor en el cuadro de la bicicleta. La medida de la energía se realiza sin efecto retroactivo, de forma que no se pierde energía. El Powermeter trabaja con compensación de temperatura, es 100% lineal y resistente al agua.



Figura 2.10: SRM, unidad de medición

2. Estado del Arte



Figura 2.11: Electrónica de la unidad de medición SRM

El principio de funcionamiento del SRM (la medición de la deformación entre biela y plato) queda bastante claro en la imagen 2.12



Figura 2.12: prototipo de bielas SRM (1986)

Srm permite medir el par realizado por ambas piernas, pues tanto con el pedal derecho, como con el izquierdo, se produce la deformación entre plato y biela.

Precio aproximado: 2000€

2.4.2.Power Tap

Los sensores de potencia PowerTap miden de forma precisa la potencia que se ejerce mientras se pedalea. El sensor mide la potencia desde el buje trasero de la rueda enviando la información a un computador de forma inalámbrica. Además de la potencia se obtienen datos de velocidad y cadencia de pedaleo desde el buje. Existen varios modelos de sensores, todos con las mismas funcionalidades, cuya diferencia principal está en el material y peso del buje, así como en el precio. El principio de funcionamiento se basa en medir, mediante galgas extensiométricas, la deformación entre el eje y la carcasa del buje.



Figura 2.13: buje powerTap

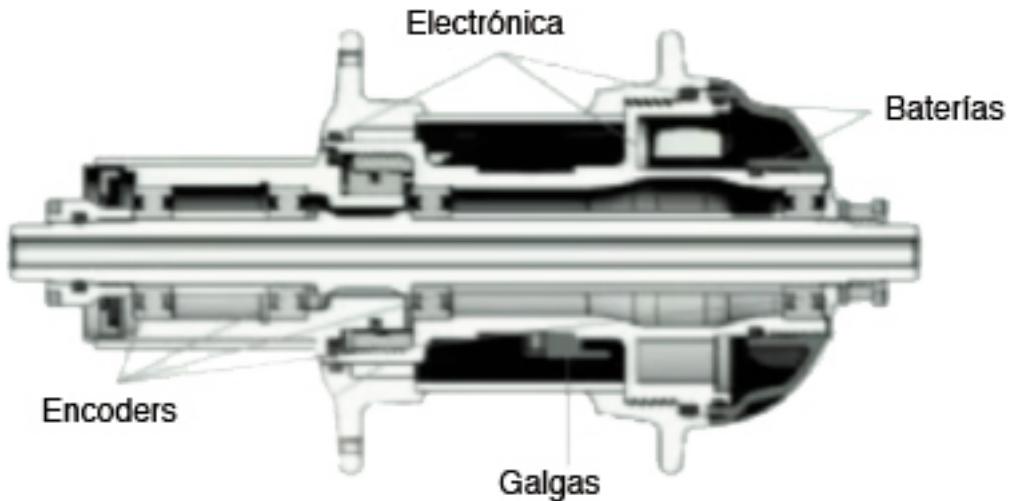


Figura 2.14: elementos internos del powerTap

Power Tap permite medir el par realizado por ambas piernas, pues tanto con el pedal derecho, como con el izquierdo, se produce la deformación en el buje trasero.

Precio aproximado: 2000€

2.4.3.Look keo power

A finales de 2011, se presentaron los pedales sensorizados look keo power. El transmisor de los datos de potencia, denominado Polar P5 power transmitter, se inserta en el eje de los pedales instrumentados Look Keo Power que utilizan 8 galgas para la medición de las fuerzas que operan sobre cada uno de los pedales permitiendo calcular la potencia mecánica aportada por cada pierna del ciclista de manera independiente.



Figura 2.15: Pedal look keo power



Figura 2.16: Situación de las galgas y la electrónica

Según el fabricante, el error de medición de potencia es menor al 2% y el precio de venta al público en Europa se estima entre 1500-1800 euros sin la ciclocomputadora.

2.4.4.Ergomo

En el sistema de medición ergomo, es en el pedalier donde se determina la potencia realizada por el ciclista sobre los pedales. Durante pedaleo, el eje gira ligeramente, esta torsión se mide mediante la comparación de dos señales. El sistema es suficientemente sensible para detectar una distorsión mínima de sólo 0,0025 grados. La medición de la torsión del eje se realiza mediante dos encoders colocados en cada extremo. Mediante el desfase de un encoder sobre el otro, podemos inferir la torsión realizada sobre el eje, y por lo tanto, el par realizado por el ciclista.

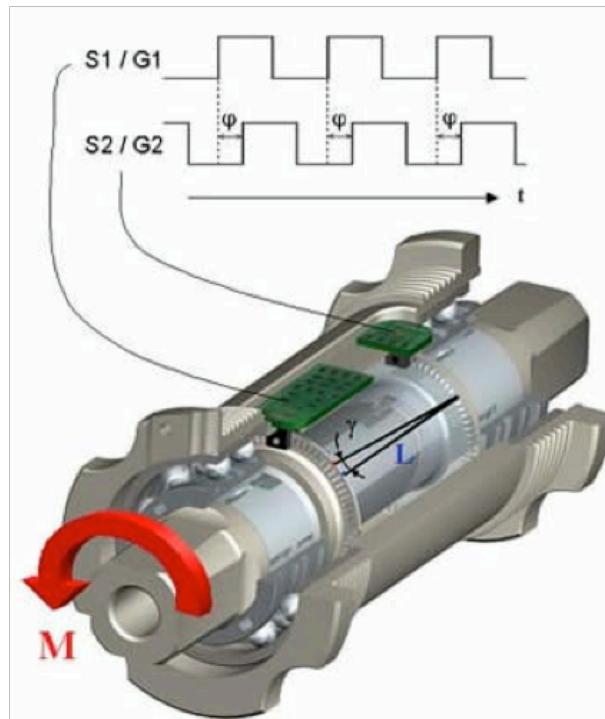


Figura 2.17: Esquema de funcionamiento del pedalier ergomo

El sistema ergomo solo permite saber la potencia realizada con la pierna izquierda, pues es la única que deforma el eje. El par realizado con la pierna derecha no deforma el eje ya que es transmitido por la biela directamente al sistema plato/cadena.

Precio aproximado: 800€

2.4.5. Encoders ESAII

En el año 2009, en el departamento de ESAII, se realizó un proyecto para conseguir medir el par realizado por un ciclista midiendo la deformación del eje al igual que hace el sistema Ergomo.

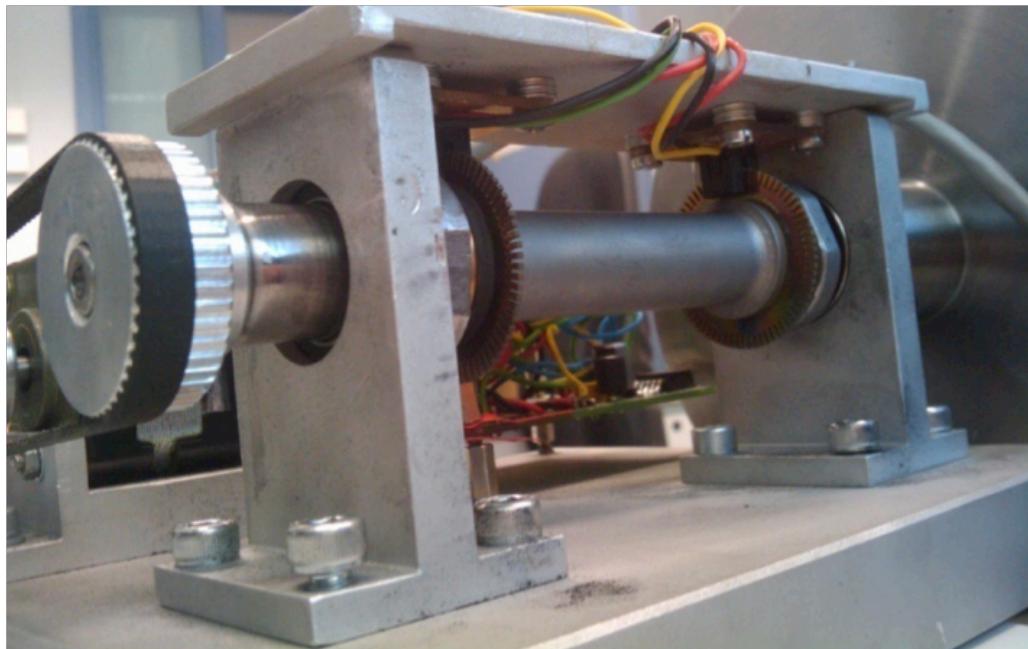


Figura 2.18: Banco de pruebas utilizado para la experimentación con los encoders

Después de experimentar con discos dentados fabricados por diferentes proveedores se desestimó continuar con el proyecto debido a la dificultad de conseguir discos dentados suficientemente precisos como para obtener una medida fiable. Los problemas encontrados en los discos fueron los siguientes:

- Los discos no eran exactamente iguales.
- Las ranuras y los dientes no eran perfectamente radiales ya que incorporaban irregularidades en la superficie de corte.
- El ángulo formado por las diferentes ranuras y dientes presentaba una oscilación considerable.

En la siguiente gráfica puede observarse como, girando a velocidad constante, el tiempo entre los flancos producidos por el disco no es constante.

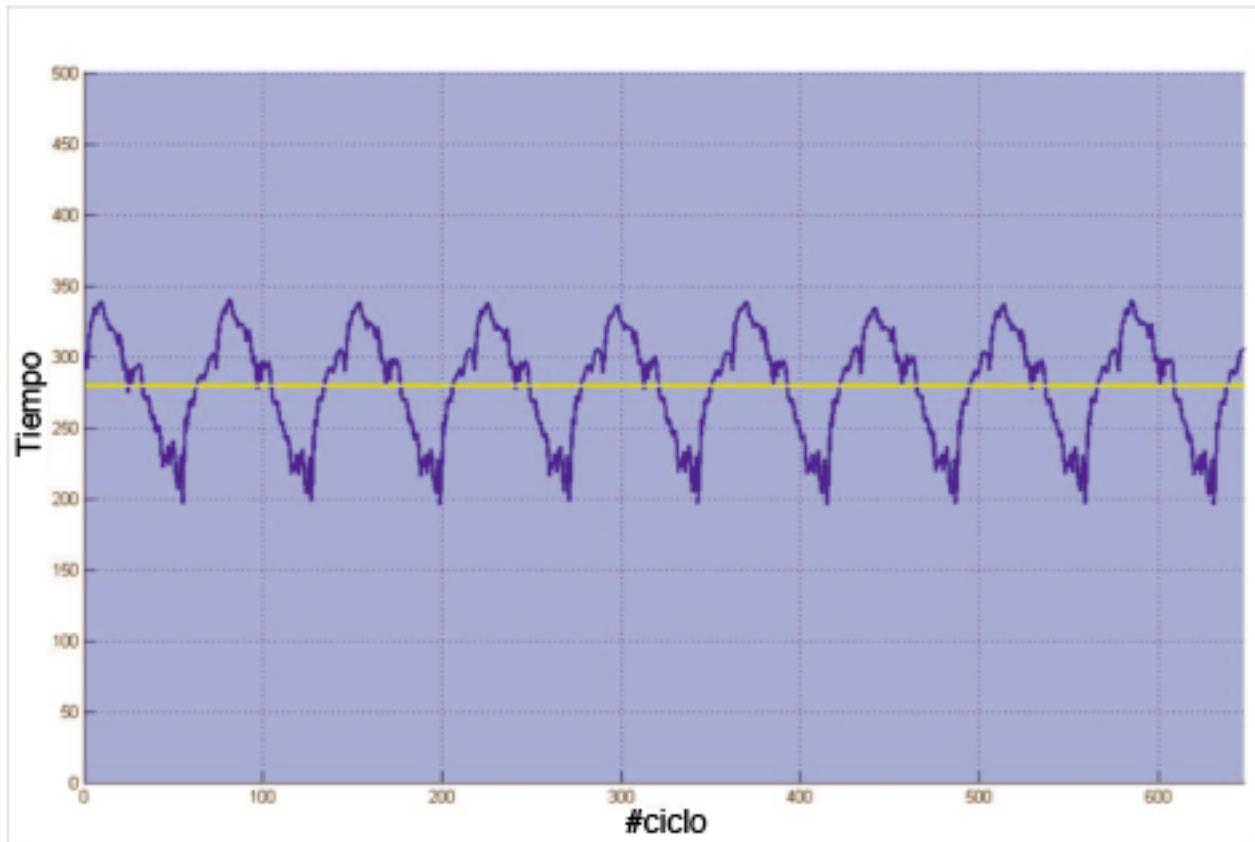


Figura 2.19: Tiempo de flanco producido por el giro de los discos

2.4.6. Galgas ESAII

En enero de 2011, se llevó a cabo otro proyecto en el departamento de ESAII para conocer el par realizado por un ciclista sobre los pedales. Para la lectura del esfuerzo realizado por el ciclista se ubicaron galgas extensiometrías en las bielas de unos pedales convenientemente modificados. Se consiguió una lectura precisa de la deformación producida en ambas bielas, pero no se llegó a un prototipo funcional que pudiese ser utilizado en un bicicleta, a falta de la transmisión inalámbrica de la medida del par desde el sensor hasta el controlador de la bicicleta.



Figura 2.20: Dirección de la fuerza medida (arriba) y posición de la galga en la biela (abajo)

Capítulo 3

Objetivos

Este proyecto tiene dos objetivos: a) crear nuevas estrategias de control sobre la consigna del motor de la bicicleta cuando un usuario-ciclista ejerce un Par sobre los pedales, b) visualizar, utilizando un Smartphone, los datos de la bicicleta, tanto los derivados de la odometría, como los particulares de una bicicleta eléctrica. Para llevar a cabo estos objetivos se precisa realizar las siguientes tareas:

- Realizar un estudio de mercado sobre bicicletas eléctricas con sistemas de ayuda al pedaleo que utilicen sensores de par.
- Integrar en el sistema de control de la bicicleta un sensor de Par (pedalier sensorizado). Este componente ofrece una señal analógica proporcional al par ejercido por el ciclista sobre el punto de apoyo que ofrece el pedal.
- Compartir con los sistemas presentes el dato obtenido del sensor de Par mediante el bus de comunicaciones CAN de la bicicleta.
- Construcción de un banco de pruebas para facilitar el trabajo de creación y test de las estrategias de control del motor. Este banco de pruebas medirá el Par realizado por el motor que aparece al aplicar un freno sobre la rotación de la rueda trasera de la bicicleta. Esta información también será accesible en el bus CAN.
- Diseño de estrategias de control mediante la emulación de modelos creados en Simulink/Matlab.
- Implementar y testear las estrategias de control en la electrónica de la bicicleta.
- Desarrollar la interfaz gráfica, para un Smartphone sobre una plataforma Android de Goolge, que represente en tiempo real al usuario de la bicicleta tanto los datos derivados de la odometría como los particulares de una bicicleta eléctrica.
- Documentar la memoria académico-técnica del proyecto final de carrera.

El diagrama de bloques que se adjunta muestra la interrelación de los elementos importantes para llevar a cabo las metas mencionadas.

3. Objetivos

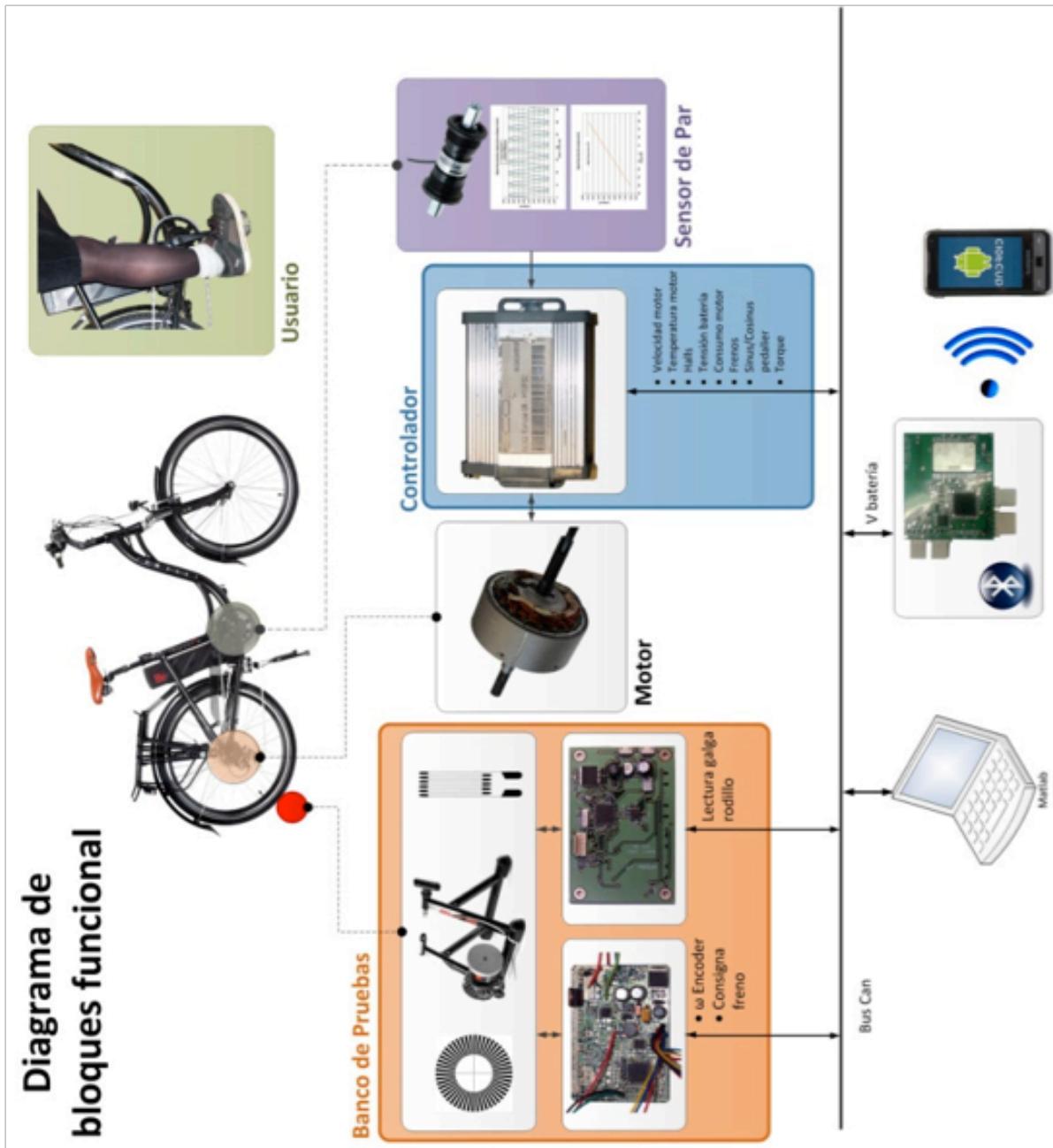


Figura 3.1: Diagrama de bloques funcional de los elementos presentes en el PFC

Para cumplir los objetivos se deberá contar con un entorno de trabajo que nos permita tener la información proveniente de un banco de pruebas, del controlador y del sensor de Par y por otro lado poder comunicarnos con un Smartphone a través de Bluetooth así como poder emular las estrategias de control en Simulink/Matlab.

Capítulo 4

Banco de Pruebas

Para la consecución de los diferentes objetivos marcados en este PFC se requiere la construcción de un banco de pruebas para facilitar el trabajo de creación y test de las estrategias de control del motor. Este banco de pruebas medirá el par realizado por la bicicleta (conjunto motor-ciclista) que aparece al aplicar un freno sobre la rotación de la rueda trasera de la bicicleta. Esta información también será accesible en el bus CAN. Tener el par resistente que genera el freno en todo momento nos permitirá calibrar otros elementos como por ejemplo el par realizado por el ciclista o obtener las gráficas reales del motor que lleva incorporada la bicicleta. Para la implementación del banco de pruebas, se utilizará un banco de entrenamiento (rodillo) de la marca italiana Elite, más concretamente el modelo RealPower CT, que permite generar hasta 1500W de resistencia.



Figura 4.1: Elite Real Power CT

4.1. Funcionamiento

Cuando montamos la bicicleta en el banco de entrenamiento, la rueda tractora de la bicicleta toca un rodillo y cuando esta gira hace girar el rodillo. El eje de este rodillo va unido al rotor del freno magnético. Por tanto, el par generado por la rueda tractora de la bicicleta se transmite, mediante el rodillo, hasta el rotor del freno magnético, tal y como se muestra en la figura 4.2.

El estator del freno electromagnético es solidario a la palanca incorporada en el montaje y los dos elementos pueden girar libremente respecto al eje del rotor, de tal manera que la palanca queda apoyada en una plancha metálica deformable

Cuando alimentamos el estator, el freno magnético comienza a actuar y el estator intenta girar como el rotor debido a que se genera una atracción electromagnética entre rotor y estator que frena al rotor. En esta situación se induce un par en el conjunto estator-palanca que se traduce en una fuerza aplicada en el punto donde la palanca se apoya en la plancha deformable. Esta fuerza deforma la plancha y, midiendo esta deformación mediante galgas extensiométricas y utilizando la electrónica 4PGA, se puede saber el valor de la fuerza, y por tanto, el par del freno.

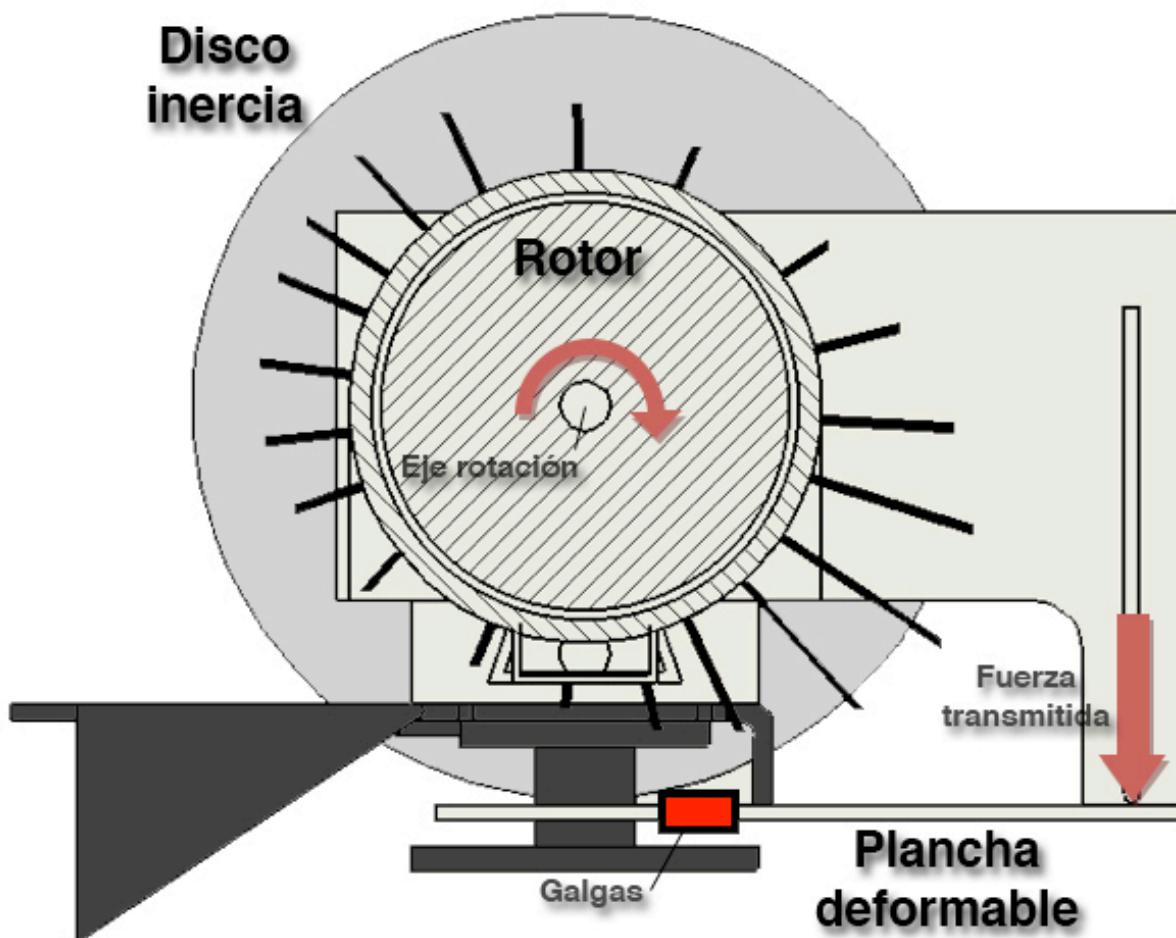


Figura 4.2: Esquema de funcionamiento del freno (vista en sección).

4.2. Control del freno electromagnético

De serie, el rodillo incorpora una electrónica que permite indicar, mediante el software de Elite o un selector, el nivel de frenado que queremos realizar. Este control, al ser un sistema propietario de Elite, no permite ser incorporado al diseño de nuestro banco de pruebas. Es por ello que para controlar el nivel de frenado del rodillo, adaptaremos una electrónica Econcontrol, cosa que permitirá además, enviar la consigna de par resistente al rodillo mediante el bus CAN.



Figura 4.3: Selector de potencia de frenado del rodillo.

4.2.1.Modificación electrónica ecocontrol

Para controlar el freno se utiliza una electrónica Econcontrol, desarrollada en el departamento de ESAII. Se utiliza esta electrónica, debido a la facilidad con las que nos permite alcanzar nuestra meta y a que dispone de todos los elementos requeridos para poder controlar el rodillo: un microcontrolador con salidas de PWM, transistores mosfets, modulo CAN, etapa de alimentación a 5V y 12V. Originalmente, una Ecocontrol, se utiliza para controlar un motor brushless de tres bobinas. Una electrónica ecocontrol

La modificación realizada en la electrónica consiste en:

- Soldar solo los componentes necesarios (ver esquemáticos)
- Conexión de una línea de 18V a la electrónica
- Adaptar el ancho de las vías necesarias para poner la línea de 18V
- La salida del PWM generado debe ser a 18V

4. Banco de Pruebas

Se pueden observar los esquemáticos de la Ecocontrol modificada a continuación:

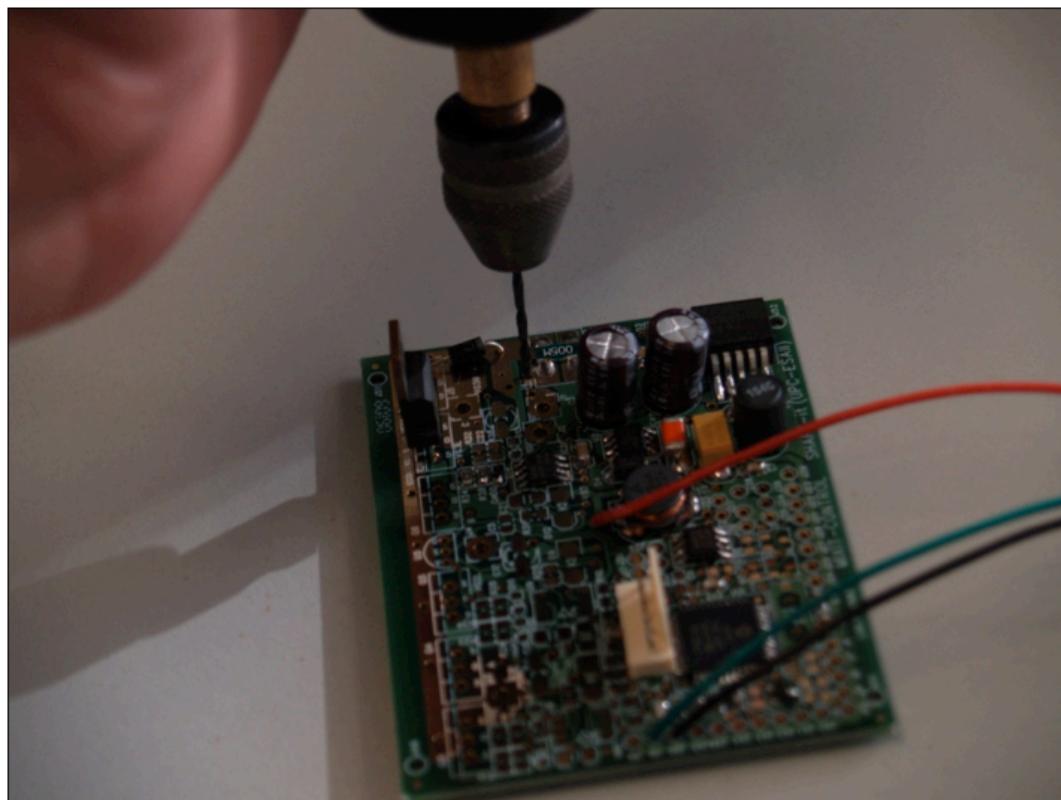


Figura 4.4: Modificación del ancho de las pistas para alimentar la salida de los mosfets a 18V

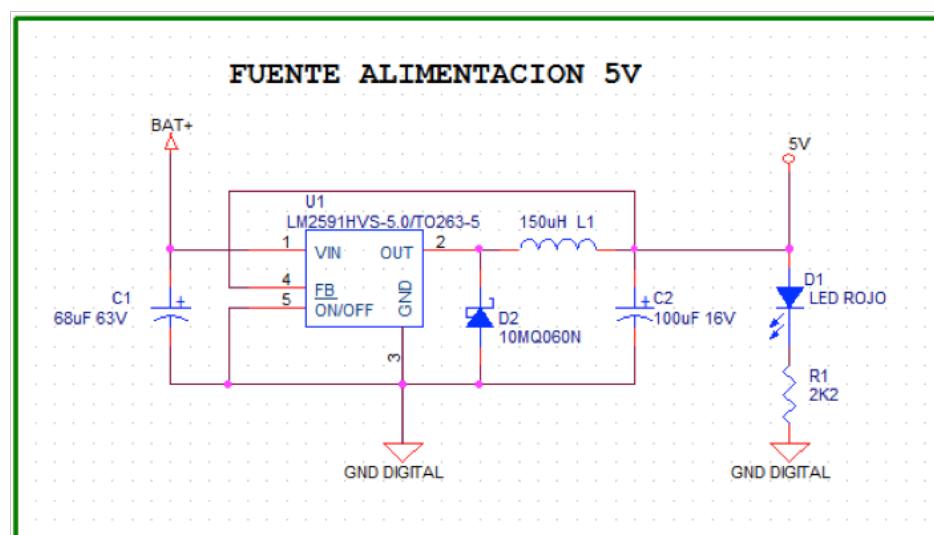


Figura 4.5: Esquemático de la fuente de alimentación de 5V

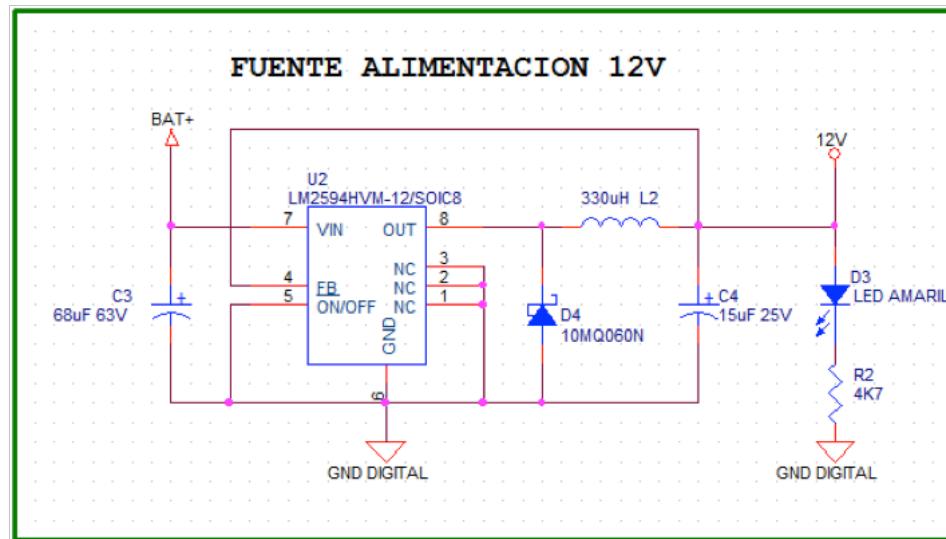


Figura 4.6: Esquemático de la fuente de alimentación de 5V

4. Banco de Pruebas

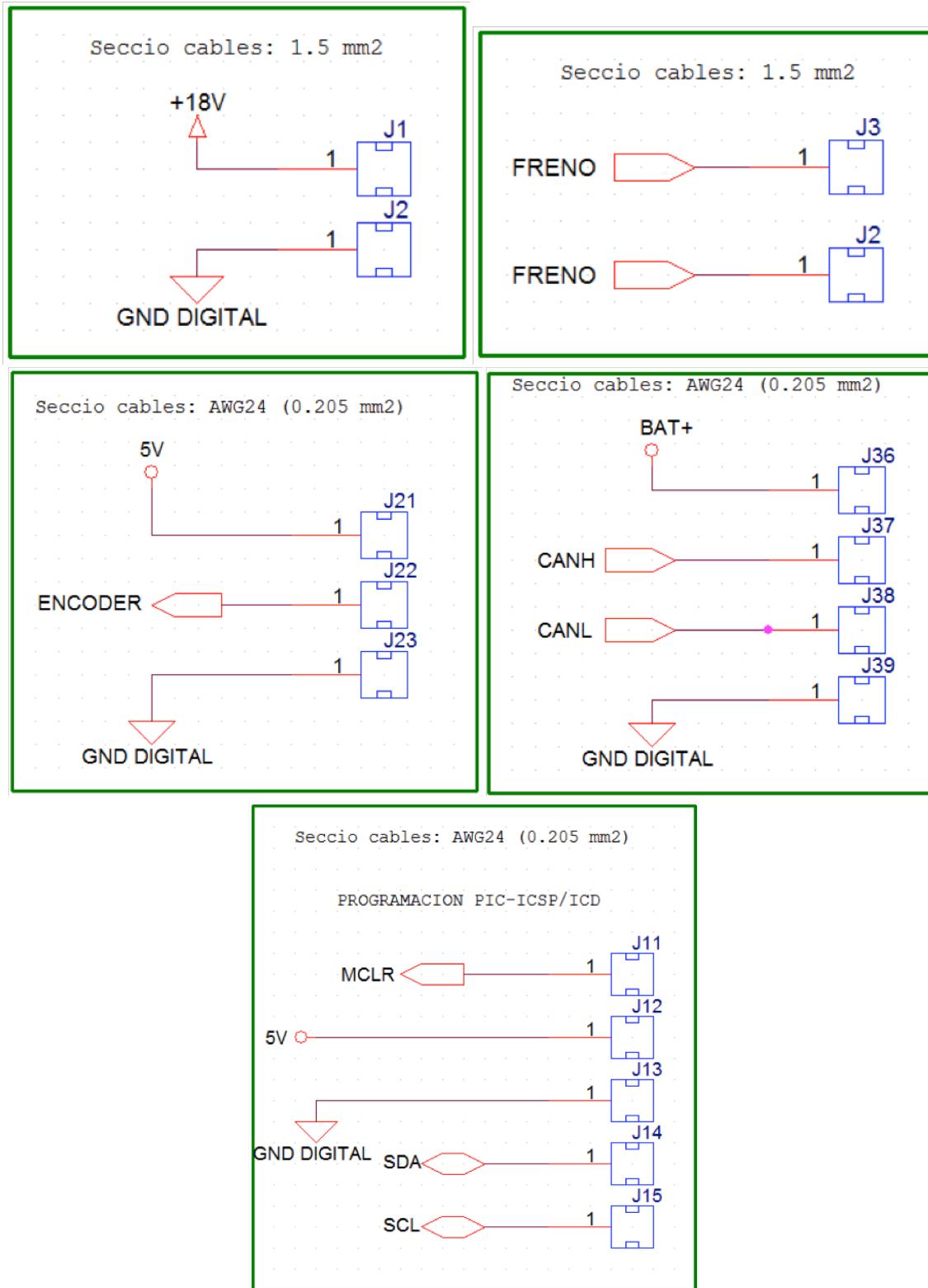


Figura 4.7: Sección y holes de montaje de los cables.

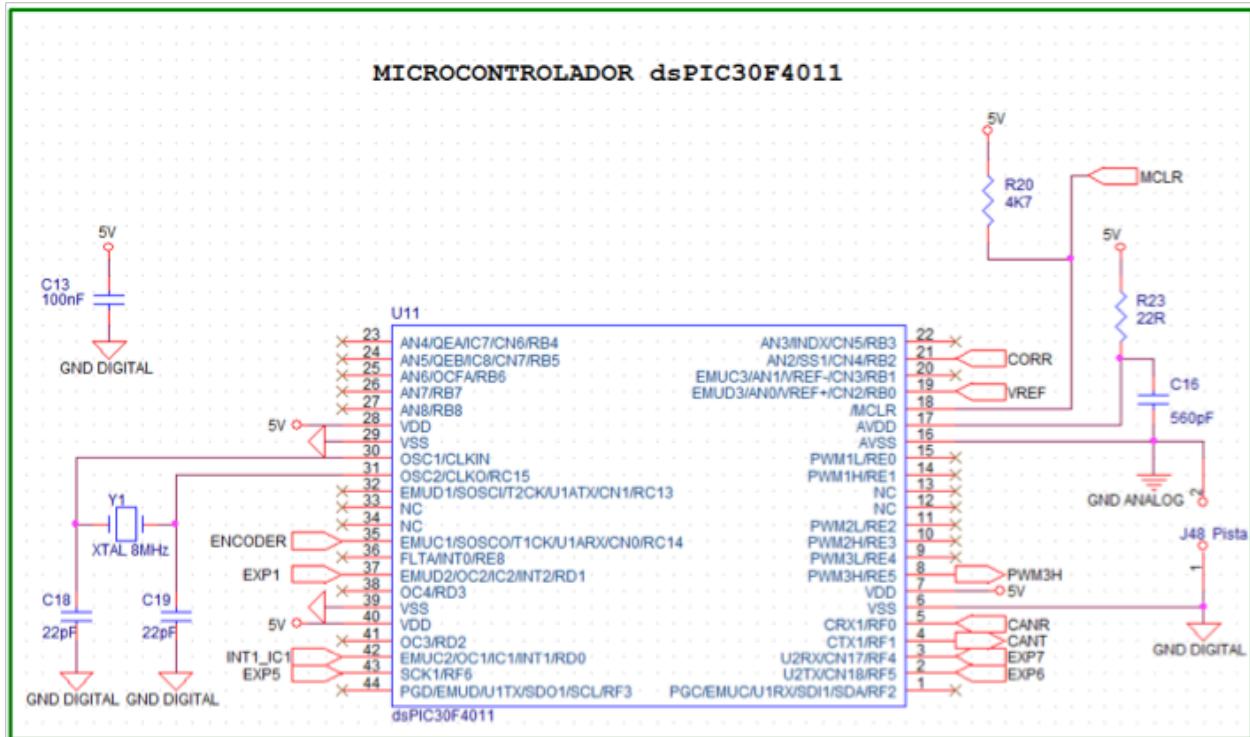


Figura 4.8: Esquemático del microcontrolador

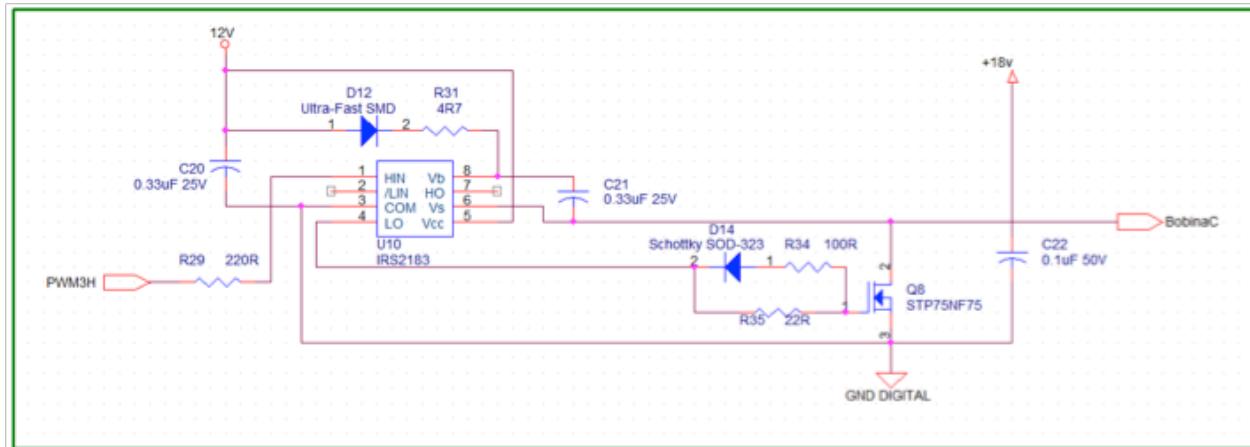


Figura 4.9: Esquemático de la rama mosfet high de salida del PWM

4.2.2.Firmware

La electrónica utilizada dispone de un microcontrolador dsPIC30F4011. como se ha visto en el apartado anterior, para controlar el nivel de frenado del rodillo tenemos que generar una señal eléctrica variable que permita controlar la corriente consumida por el freno. Esta señal será un PWM (señal modulada). Uno de los motivos por los que se ha escogido la alternativa de utilizar una señal PWM, es por tal de poder reutilizar una electrónica desarrollada en el departamento con todos los beneficios que ello conlleva. De esta forma, se puede diseñar el sistema de forma análoga al del control del motor de la bicicleta. Este control del motor, utiliza una señal PWM para

4. Banco de Pruebas

activar y desactivar el motor que hace girar la rueda. En el esquema anterior (figura 4.9), se puede ver que la señal PWM, no se conecta directamente al mosfet de control, esto es debido a que el puerto de salida del microcontrolador no da suficiente corriente como para activar un transistor de estas características a la frecuencia deseada. Por esta razón, la señal generada por el microcontrolador pasa por un driver que permite proporcionar la corriente necesaria para que el mosfet se active/desactive según el valor específico de la señal de control.

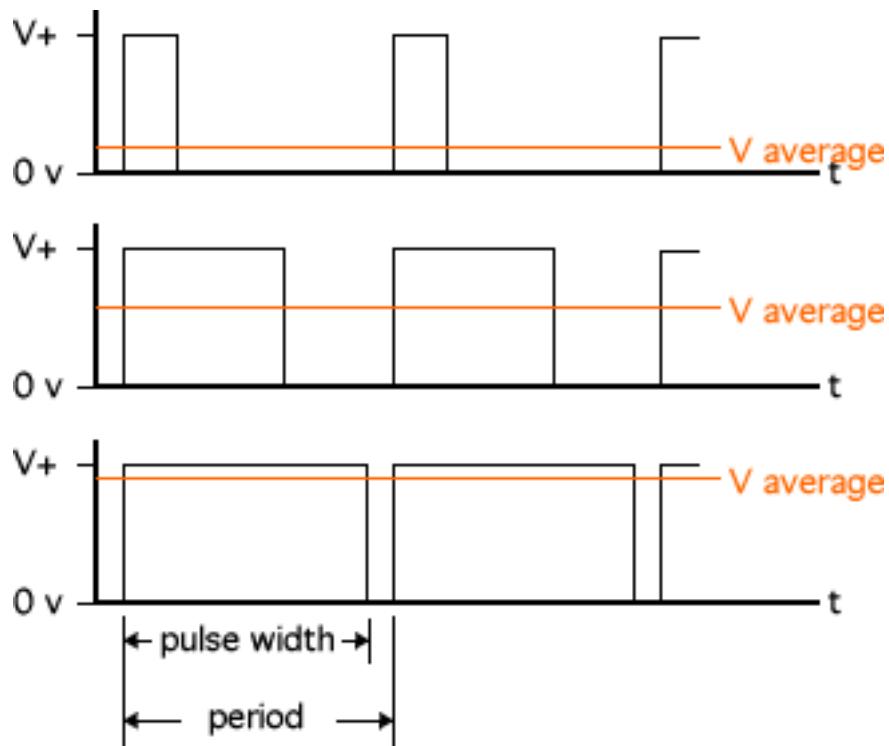


Figura 4.10: Ejemplo de señales PWM

Por lo tanto, el modulo PWM del microcontrolador es el encargado de generar la señal PWM encargada de variar la corriente que consume el freno. También es necesario variar el tiempo de *duty cycle* con tal de conseguir variar la corriente de frenado.

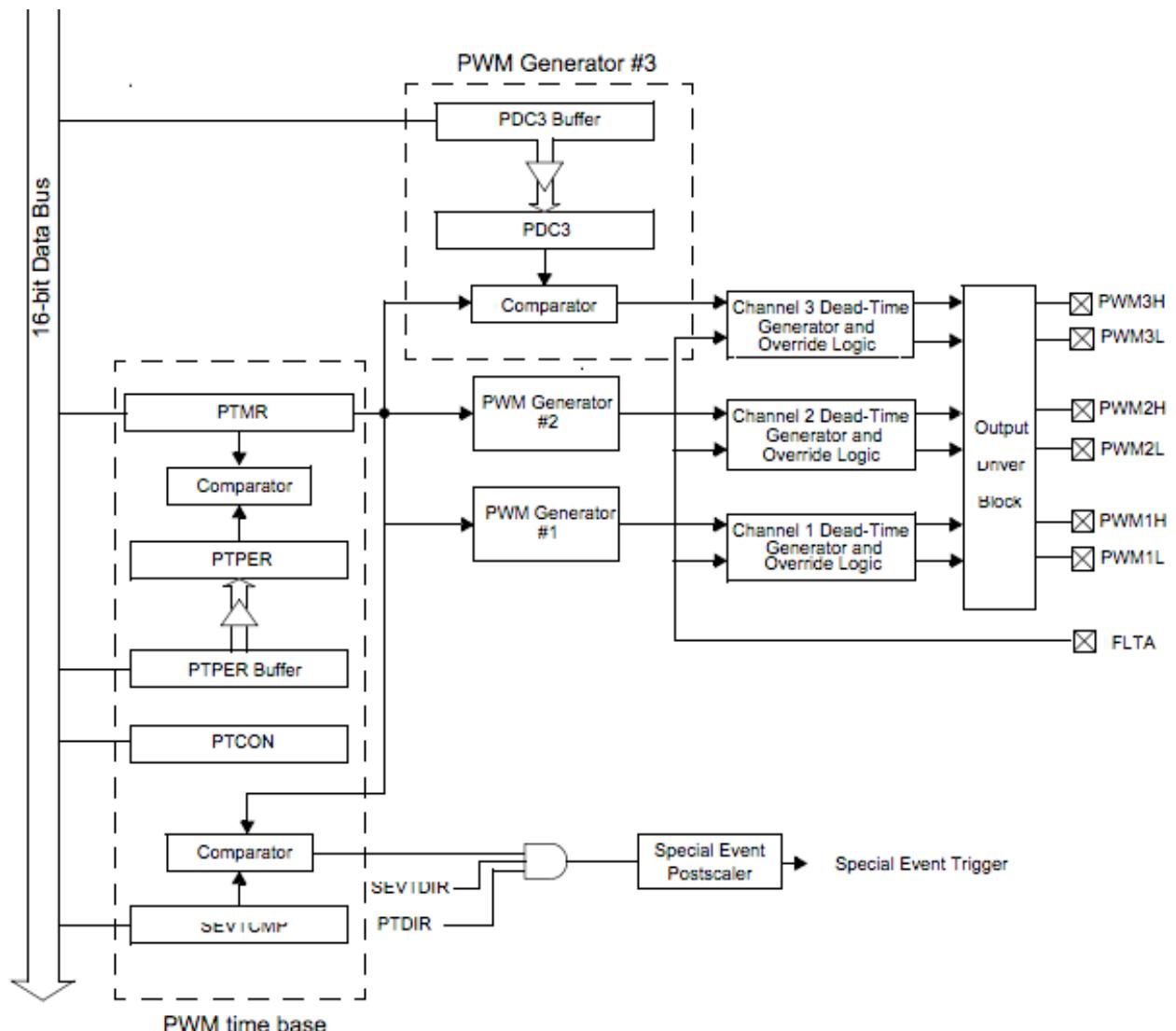


Figura 4.11: Lógica del módulo PWM del dsPIC30F4011

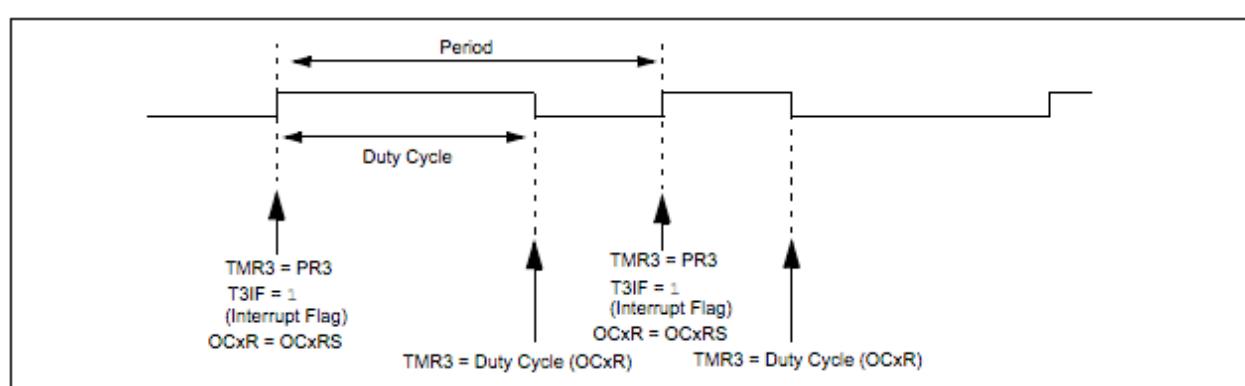


Figura 4.12: Output timing del PWM

Para conseguir el PWM deseado se necesita especificar los siguientes parámetros:

4. Banco de Pruebas

- Definir el periodo: El primer paso a realizar es definir el tiempo de ciclo que se utilizará para generar la señal PWM. Definiremos 50 μ s.
- Definir el *duty cycle*: Una vez definido el periodo, hace falta definir el tiempo en que la señal PWM se mantendrá activa. posteriormente se verá como se calcula.
- Configurar el PTMR: Hace falta configurar el *timer* del PWM para que incremente su valor a la frecuencia suficiente para tener una resolución suficiente en el muestreo de la señal.

El valor del PWM debe ser modificable desde el exterior del microcontrolador. Es por ello que se permite dar una consigna de PWM a través del bus CAN. El dsPIC30F4011 dispone de un módulo CAN, que nos permite filtrar las tramas que queremos recibir.

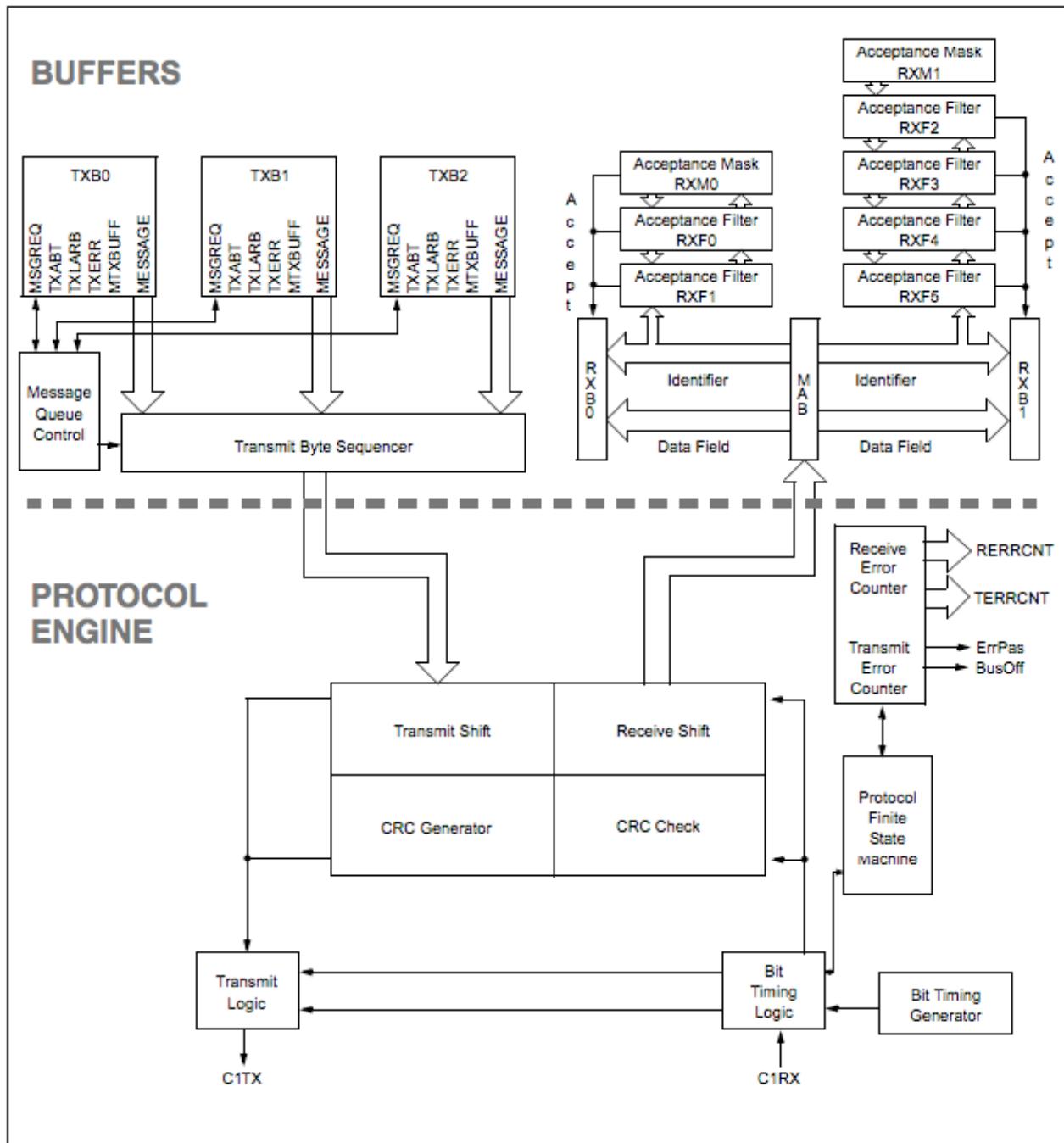


Figura 4.13: Modulo CAN del dsPIC30F4011

Cada vez que llega una trama que tiene por destino la electrónica de control del freno, leemos el dato correspondiente a la consigna de PWM que nos es transmitido y damos al módulo PWM dicho valor de *duty cycle*. A continuación, puede verse el diagrama de flujo del bucle principal de espera activa:

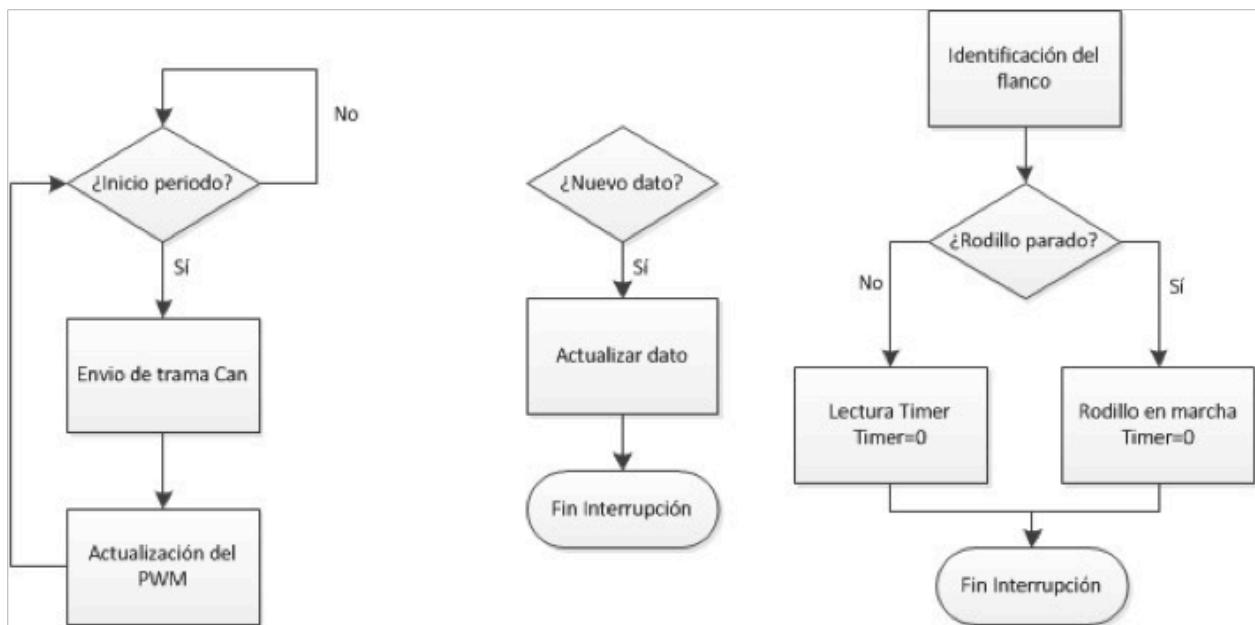


Figura 4.14: Diagrama de flujo del bucle principal(izquierda), interrupción CAN (centro) e interrupción encoder (derecha)

4.3.Electrónica 4PGA

Como ya se ha comentado, los sensores de fuerza que utilizamos son del tipo Wheatstone full-bridge. Como ya hemos visto, la señal de salida de las galgas debe ser amplificada si queremos tener resolución en la medida. Para esta amplificación utilizamos un condicionador programable del señal que nos permitirá configurar el señal con la ganancia deseada, así como corregir el offset en caso que sea necesarios. El modelo escogido es el PGA309. El PGA309 es un acondicionador de señal completa que proporciona excitación del puente, ajuste del rango y offset iniciales, ajuste del rango y offset en función de la temperatura, medida de la temperatura interna o externa, limitación de la salida por sobre y por debajo, detección de errores y calibración digital.

La Figura 4.15 muestra los diagrama de bloques básico del PGA309.

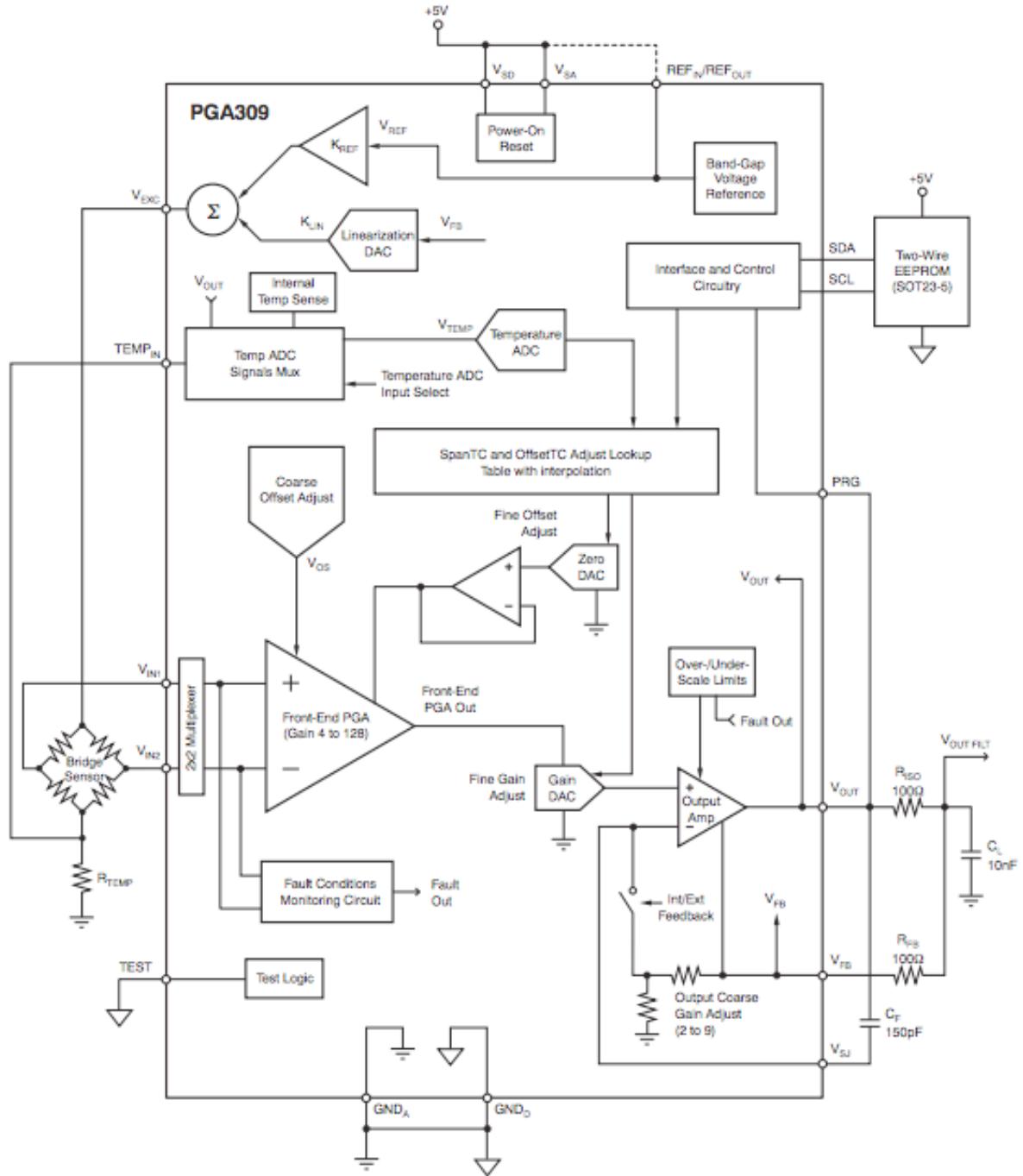


Figura 4.15: diagrama de bloques simplificado de la PGA309

Se utiliza la placa desarrollada en el departamento de ESAlI 4PGA. El funcionamiento y diseño de esta placa se explica en los siguientes apartados. Podemos ver una imagen de la placa ya soldada en la siguiente imagen:

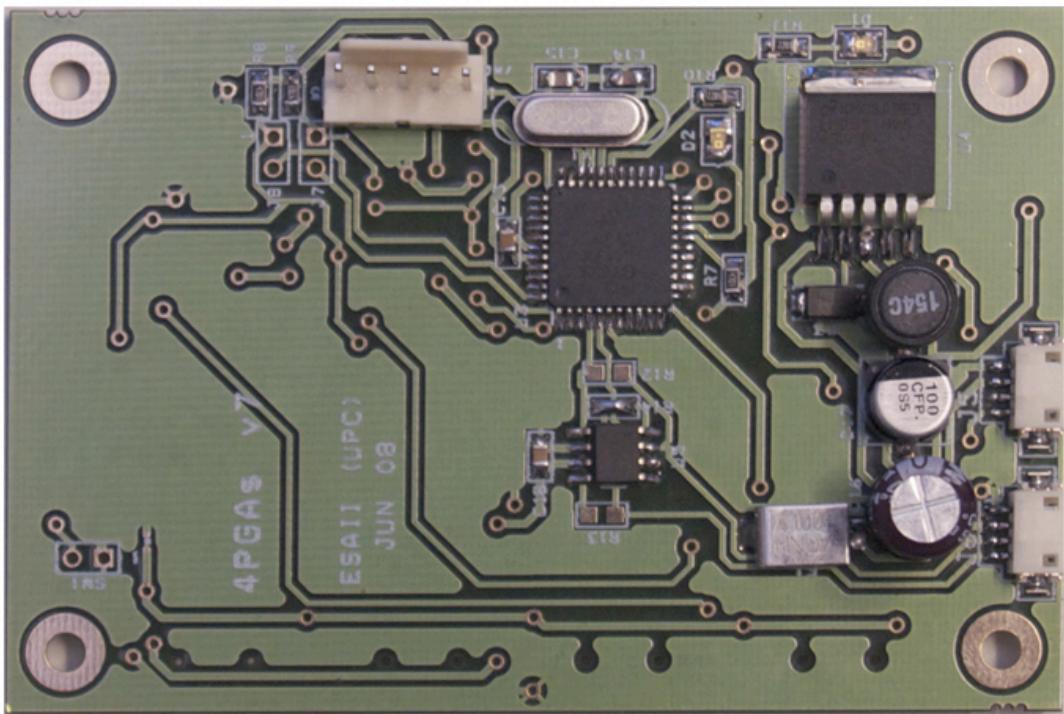


Figura 4.16: Electrónica 4PGA soldada

4.3.1.Escalado de la ganancia

El corazón del PGA309 es el amplificador de entrada de ganancia programable (Front-End PGA). La ganancia total del Front-End PGA más el amplificador de salida (Output Amplifier) puede ser ajustado desde 2.7V/V hasta 1152V/V. La polaridad de las entradas puede ser cambiada usando un multiplexor por tal de conectar fácilmente sensores de puente con polaridad de salida desconocida.

El Front-End PGA proporciona la ganancia tosca (“coarse”) inicial usando un amplificador de instrumentación. La ganancia fina (“fine”) se consigue con un convertidor digital-analógico de ganancia de 16 bits (Gain DAC). El Gain DAC usa como entrada los datos guardados en una tabla de compensación de temperatura accedida por un convertidor digital-analógico de temperatura (Temp ADC). Con el objetivo de compensar las no-linealidades de segundo orden, la ganancia fina se establece usando los coeficientes guardados a una memoria externa no volátil (EEPROM).

A continuación de la ganancia fina viene la etapa del amplificador de salida (Output Amplifier) que proporciona una ganancia programable adicional. Las dos conexiones principales del amplificador de salida, VFB y VSJ, son sacadas del PGA309 para más flexibilidad de las aplicaciones. Estas

conexiones permiten obtener un voltaje de la señal condicionada muy esmerado, controlar los picos de sobrevoltatge y filtrar RFI/EMI.

4.3.2. Ajuste del “offset”

El ajuste del “offset” del sensor se lleva a término en dos fases. El ajuste tosco tiene aproximadamente $\pm 60\text{mV}$ de ajuste para un voltaje de referencia (V_{REF}) de 5V. El ajuste fino se realiza con un convertidor digital-analógico de 16 bits (Zero DAC) que suma la señal a la salida del Front-End PGA. De manera análoga al Gain DAC, los valores digitales d'entrada al Zero-DAC están almacenados en la tabla de compensación de temperatura guardada en la EEPROM externa y direccionada por el TempADC. El rango programable del Zero-DAC es de 0V a V_{REF} , con un rango de salida de 0.1V a VSA-.

4.3.3. Coeficientes de temperatura y EEPROM

La ganancia y el offset finos son ajustados haciendo servir un circuito dedicado a la medición de la temperatura. Se puede seleccionar entre los modos de detección de temperatura interna o externa. La temperatura puede ser medida de una de las formas siguientes:

- Cambio en la impedancia del puente (sensor de corriente de excitación), para sensores con un alto coeficiente de resistencia a la temperatura.
- Temperatura del chip, cuando la PGA está suficientemente próxima al sensor
- Diodo externo, thermistor o RTD colocado en la membrana del sensor. Una fuente interna de $7\mu\text{A}$ tiene que ser habilitada para la excitación de este tipo de sensor de temperatura.

La señal de la temperatura es digitalizada haciendo servir el Temp ADC. La salida de este convertidor es usada por el circuito de control digital para leer los datos de la *lookup table* guardada en la EEPROM externa, y establecer la salida del Gain DAC y del zero DAC según los valores de calibración.

Una función adicional proporcionada por el Temp ADC es la habilidad de leer el pin VOUT por su multiplexor de entrada. Esto proporciona flexibilidad para una salida digital mediante las interfaces de comunicaciones del chip, así como también proporciona la posibilidad de que un microcontrolador externo lleve a cabo el calibrado en tiempo real de la PGA309.

El PGA309 usa una EEPROM externa de conexión Two-Wire, standard en el mundo de la industria. Es necesaria una EEPROM mínima de 1 K-bit cuando se usan los 17 coeficientes de temperatura. Una EEPROM más grande puede ser usada para proporcionar espacio al usuario para guardar datos adicionales.

La primera parte de la EEPROM externa contiene los datos de configuración para la PGA309:

- Registro 3: Control de referencia y linealización
- Registro 4: *Offset* tosco de la PGA y ganancia del *Front End PGA* y *Output Amplifier*
- Registro 5: Configuración de la PGA y límites sobre/sota escala.
- Registro 6: Control del temp ADC

Esta sección contiene su propio *checksum*.

La segunda parte de la EEPROM externa contiene hasta 17 indices de temperatura y los correspondientes coeficientes para el Zero DAC y el Gain DAC. En el caso del corriente proyecto, se utilizan tres coeficientes de temperatura, correspondientes a las tres recalibraciones realizadas.

Esta segunda parte de la EEPROM contiene su propio *checksum*. La lógica de la PGA309 contiene un algoritmo de interpolación para ajustar los valores de los DACs entre los indices de temperatura guardados. Si los *checksums* son incorrectos, la salida de la PGA se mantiene en alta impedancia.

4.3.4. Microcontrolador

En la electrónica 4PGA utilizada en este proyecto, se emula la EEPROM haciendo uso de un microcontrolador. De esta forma, cuando se pida la configuración, el microcontrolador contestará como si de una EEPROM se tratase. Esto nos permitirá cambiar más fácilmente la configuración de la PGA309 simplemente cambiando el firmware del microcontrolador, a parte del ahorro que supone no poner una EEPROM.



Figura 4.17: Emulación de la EEPROM en la electrónica 4PGA

Después de la configuración inicial, la PGA309 habilita su salida con la amplificación de la señal proveniente de la galga. Esta señal, todavía en formato analógico, es digitalizada mediante el conversor A/D ADS1100 de 16 bits, consiguiendo así una alta resolución.

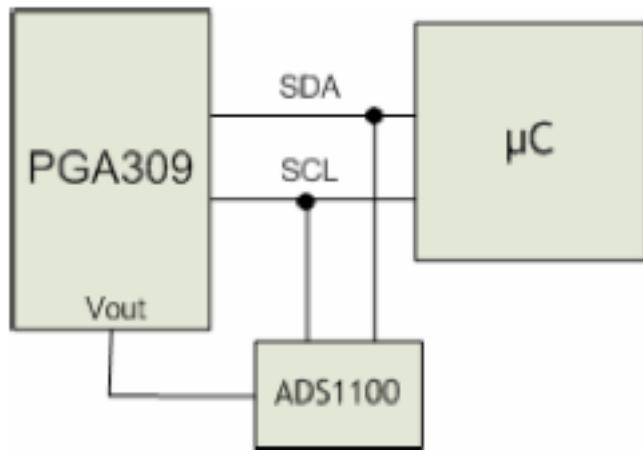


Figura 4.18: Conexión del conversor A/D al bus i2c

4.4. Medición de fuerzas

Como se ha comentado en el apartado 4.1, el rodillo ha sido modificado para incorporar una plancha metálica deformable que permita saber el par ejercido por el freno. Veremos ahora el principio en el que se basa esta medida del par del freno.

4.4.1. Principios de la extensiometría

La extensiometría es una técnica experimental para la medida de fuerzas y deformaciones, basada en el cambio de conductividad eléctrica de un material cuando se le aplica una fuerza. La extensiometría es experimentalmente útil en fuerzas vibracionales y detección de resonancias a altas frecuencias (la respuesta en frecuencia de una galga extensiometrífica es de unos 100kHz) donde otros métodos como por ejemplo los algoritmos de simulación por elementos finitos no ofrecen resultados fiables. En sus múltiples variantes, permite determinar estados tensionales, unidireccionales o completos, medir deformaciones a alta temperatura, fabricar acelerómetros, etc. Su uso requiere la utilización de un amplificador analógico debido a las débiles señales que se generan (normalmente inferiores a 1mV)

4.4.2. Calibración

En LERMA (Laboratorio de Elasticidad y Resistencia de Materiales de la UPC), se ha realizado una calibración estática de las galgas extensiometríficas para comprobar que miden con precisión deformaciones similares a las que se generarán dinámicamente. Esta primera calibración también sirve por comprobar que la plancha trabaja en su zona elástica, es decir, que las deformaciones son proporcionales a la carga y que vuelve a la posición inicial en cuando se encuentra en reposo.

La fuerza que genera la deformación de la plancha depende de la excitación del estator y está limitada por las características técnicas del freno electromagnético; se ha comprobado

4. Banco de Pruebas

experimentalmente y se sitúa a 60N como máximo (correspondiente a un par de freno de 6.45*Nm) por lo tanto la calibración se ha hecho a medidas que van de 0N a 60N.

Para hacer las medidas se han colgado pesos de 0.5kg en la posición donde la palanca se apoya en la plancha y se ha medido la deformación detectada por las galgas para cada situación. En la siguiente tabla se muestran los resultados:

FUERZA [kg]	ENSAÑO				DEFORMACIONES	
	1º [$\mu\epsilon$]	2º [$\mu\epsilon$]	3º [$\mu\epsilon$]	4º [$\mu\epsilon$]	MEDIA EXP. [$\mu\epsilon$]	MEDIA TEORICA [$\mu\epsilon$]
0	0	0	0	0	0	0
0,5	329	330	329	330	329,5	329,8125
1	657	659	657	659	658	659,625
1,5	986	989	985	989	987,25	989,4375
2	1316	1320	1315	1321	1318	1319,25
2,5	1646	1651	1645	1652	1648,5	1649,0625
3	1977	1982	1974	1984	1979,25	1978,875
3,5	2307	2313	2304	2316	2310	2308,6875
4	2638	2643	2634	2647	2640,5	2638,5
4,5	2968	2975	2963	2977	2970,75	2968,3125
5	3297	3304	3291	3307	3299,75	3298,125
5,5	3625	3633	3619	3636	3628,25	3627,9375
6	3955	3962	3948	3966	3957,75	3957,75
0	1	5	1	1	2	0

Figura 4.19: Tabla de calibración proporcionada por LERMA

Notas:

- En gris se representa la deformación medida después de aplicar toda la carga.
- La media teórica se ha hecho considerando a partir de la máxima deformación que todas las deformaciones se comportan linealmente con la carga.

4.4.3. Recalibración en laboratorio

La estructura interior del sensor de fuerza incorporado en el rodillo es un puente de Wheatstone, donde se utilizan 4 líneas, por lo tanto, tenemos que saber la cual es la correspondencia de cada salida del sensor con respecto al puente.

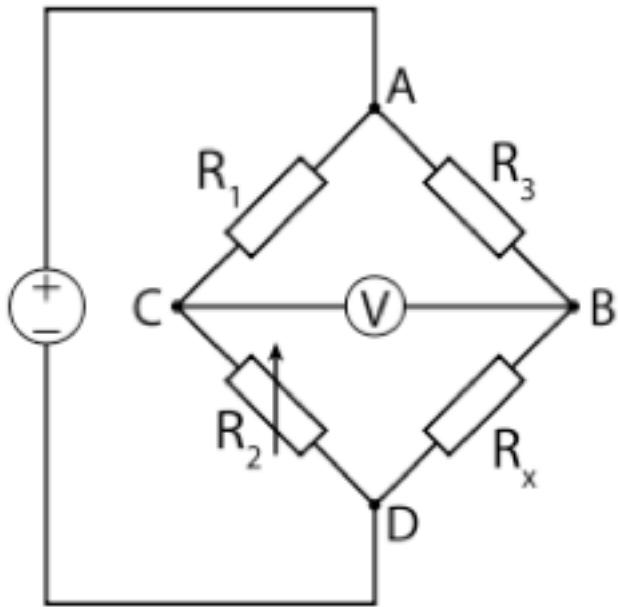


Figura 4.20: Puente de Wheatstone

Como podemos observar en la figura 4.20, dos nodos son para la alimentación del sensor (polo positivo - A y polo negativo - D) y dos más para la señal de salida del sensor (B y C). Conociendo el valor de las resistencias y con un multímetro para medir las resistencias entre dos conexiones podemos deducir la correspondencia entre el cableado físico y las interconexiones del puente de Wheatstone. Una vez conocido el pinout del sensor, podemos proceder a conectarlo con el amplificador de señal programable (PGA) de la placa integrada (explicado en el apartado 4.3) que nos permitirá obtener las lecturas del sensor. Estas lecturas serán una interpretación digital de la señal analógica que obtendríamos del sensor, por lo tanto, no podemos relacionarla directamente con la tabla que nos proporciona el departamento de resistencia de materiales. Debido a esto, hemos tenido que realizar un recalibrado en el laboratorio para saber la relación entre microdeformaciones y la salida en mV de la salida del sensor.

El primer paso es configurar la ventana de trabajo del conversor AD. El conversor utilizado en este proyecto nos proporciona una medida digital de 15 bits, es decir, entre 0x0000 y 0x7FFF. La idea es conseguir colocar la ventana de trabajo para que el valor leído proveniente del sensor sea, para la fuerza máxima, cercano pero inferior a 0x7FFF. De esta forma optimizaremos el funcionamiento del conversor AD y obtendremos una lectura lo más precisa posible. Suponemos, según los datos facilitados por LERMA, que la fuerza máxima que se medirá en la galga del rodillo será de 70N y que la fuerza no será nunca en el sentido negativo. Encontramos entonces, que el sensor tiene que trabajar en un intervalo de 70N. Con el sensor mal calibrado perderemos mucha resolución en la medida, por ejemplo con una ventana de trabajo de 1000N, tendremos una

4. Banco de Pruebas

resolución de 31bits/N en contrapartida a los 468 bits/N que podríamos llegar con una configuración ideal.

El siguiente paso, es conectar las galgas a la electrónica 4PGA (explicada en el siguiente apartado). Conectando la electrónica a su vez a un PC mediante el bus CAN incorporado, podemos saber la lectura del amplificador que realiza el AD. Para la recalibración, se ha seguido el mismo procedimiento que realiza LERMA: colgar pesos de 1 kg en la posición donde la palanca se apoya en la plancha y medir la deformación detectada por las galgas para cada situación. Se ha repetido esta calibración a diferentes temperaturas ambiente, más concretamente se han realizado medidas a 7°C, 21°C y 26°C. Disponer de estas calibraciones permite configurar el amplificador PGA para que compense los efectos de la temperatura sobre el valor leído de las galgas extensiométricas. Una galga encolada tiene variaciones de resistencia causadas por los efectos térmicos sobre la propia galga y por las dilataciones del material sobre el que está colocada. Por ejemplo, en el acero, 5°C de elevación en la temperatura crean la misma deformación que una tensión de 1Kg/mm². En la práctica, eliminar completamente el error es imposible. Los resultados de las calibraciones pueden verse en las siguientes tablas:

Valor AD T=7°C					
Kg	High	Low	Valor	mV/Kg	
0	59	250	15354		
1	57	15	14607	747	
2	54	100	13924	683	
3	51	92	13148	776	
4	48	70	12358	790	
5	45	168	11688	670	
6	42	249	11001	687	

Figura 4.21: Resultado de la calibración a una temperatura ambiente de 7°C

Valor AD T=21°C				
Kg	High	Low	Valor	mV/Kg
0	59	10	15114	
1	54	40	13864	1250
2	49	128	12672	1192
3	45	62	11582	1090
4	41	10	10506	1076
5	36	95	9311	1195
6	31	240	8176	1135

Figura 4.22: Resultado de la calibración a una temperatura ambiente de 21°C

Valor AD T=26°C				
Kg	High	Low	Valor	mV/Kg
0	58	124	14972	
1	53	38	13606	1366
2	47	250	12282	1324
3	43	15	11023	1259
4	38	30	9758	1265
5	33	20	8468	1290
6	28	10	7178	1290

Figura 4.23: Resultado de la calibración a una temperatura ambiente de 26°C

Con los datos obtenidos de las calibraciones a diferentes temperaturas, se puede obtener la siguiente tabla, que relaciona la temperatura ambiente con el valor medio amplificado por la PGA.

Temp °C	Media mv/Kg
7	725
21	1156
26	1299

Figura 4.24: Relación entre temperatura y valor amplificado por la PGA

4. Banco de Pruebas

Gráficamente, los diferentes valores leídos de la galga según la temperatura y la fuerza aplicada pueden representarse por la siguiente gráfica:

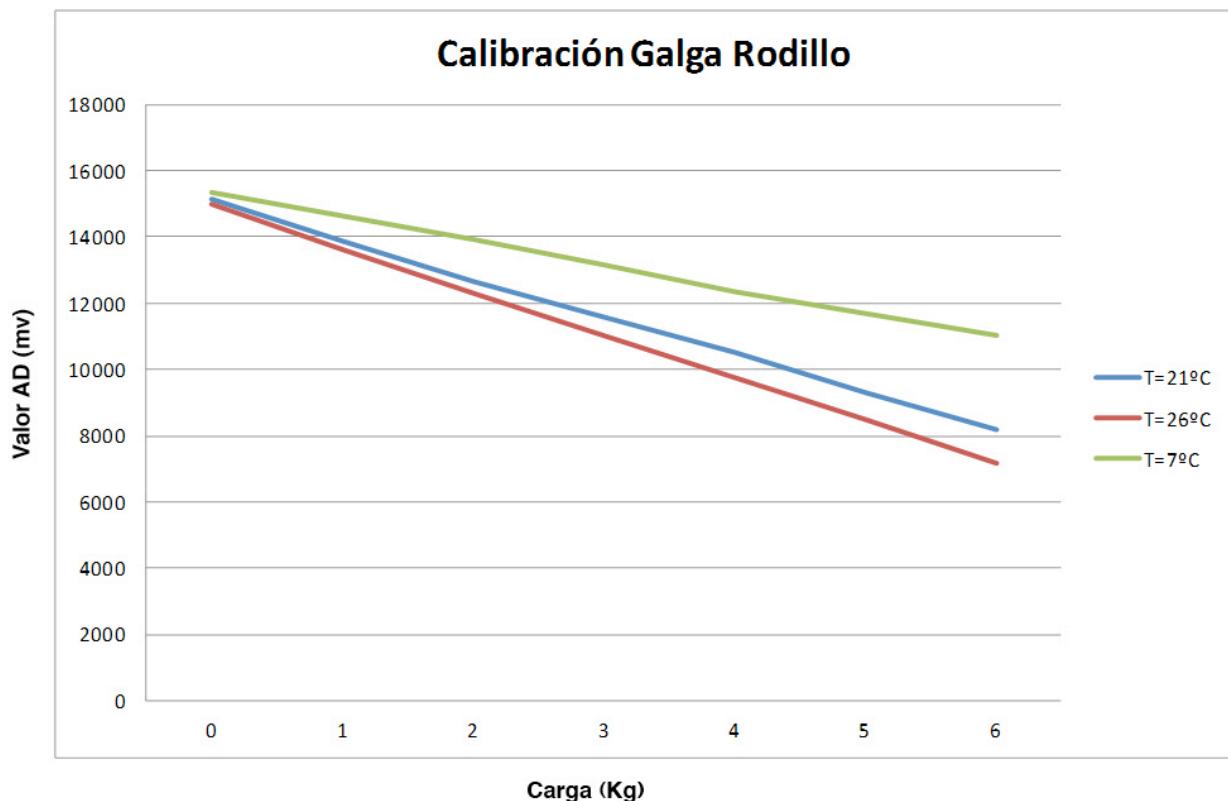


Figura 4.25: Valores leídos por el AD según la temperatura ambiente y la fuerza ejercida en la galga

Todo este proceso permite configurar el valor de la ganancia y offset fino de la PGA309 (Gain Dac y Zero DAC) según la temperatura ambiente detectada.

La salida de la PGA se calcula teniendo en cuenta la siguiente ecuación:

$$V_{OUT} = [(V_{IN} + V_{COURSEDAC}) \cdot GI + V_{ZERODAC}] \cdot GD \cdot GO$$

Donde V_{IN} es la señal de entrada de la PGA309.

$V_{COURSEDAC}$ es el offset tosco del front end PGA.

GI es la ganancia tosca.

$V_{ZERODAC}$ es el offset fino.

GD es la ganancia fina.

G_O es la ganancia de la etapa de output.

Mediante esta función se pueden calcular los coeficientes de temperatura fácilmente utilizando las siguientes expresiones:

$$V_{ZERODAC} = \frac{(A \cdot B - C \cdot D)}{C - A}$$

$$GD = \frac{C}{B + V_{ZERODAC}}$$

Donde C es el valor de la galga en reposo para la temperatura base.

B es el valor de la galga en reposo para la temperatura de la que se está calculando el coeficiente.

A es el valor estándar para la temperatura base en el que se incrementa la lectura de la galga al hacer un paso de calibración más C.

D es el valor estándar en el que se incrementa la lectura de la galga al hacer un paso de calibración más B, para la temperatura de la que se está calculando el coeficiente.

Utilizando este razonamiento, se configuran tres coeficientes de temperatura y la PGA309 se encarga de calcular el valor de ganancia fina según la temperatura ambiente interpolando el valor según los coeficientes dados.

Temp °C	Gain DAC
7	0
21	0,59
26	0,54

Figura 4.26: Coeficientes de Gain DAC calculados según la temperatura

Temp °C	Zero DAC
7	0
21	1,05
26	1,31

Figura 4.27: Coeficientes de Zero DAC calculados según la temperatura

4. Banco de Pruebas

Al utilizar estos coeficientes, se comete un error al leer el esfuerzo realizado por la galga. Partiendo de los 7°C como temperatura base, el error cometido se puede cuantificar y representar según la siguiente gráfica:



Figura 4.28: Error cometido con los coeficientes de temperatura calculados

Cometer un error máximo de 242mV a la salida de la PGA es cometer un error máximo de un 4% es perfectamente asumible, pues el error medio es del 1,5%.

4.5. Encoder

Para obtener el par realizado por la bicicleta, es necesario saber a que velocidad gira el rodillo. Saber la velocidad de giro nos permite también (sabiendo la velocidad de la rueda) detectar si el neumático de la bicicleta hace contacto con el freno o si por lo contrario se están produciendo perdidas de adherencia. El encoder que se instalará en el rodillo es un Osram SFH9240. Se ha elegido este encoder debido a que tiene unas especificaciones suficientes para el propósito que queremos desempeñar y a que se tenían muestras disponibles para ser utilizadas, ahorrando así tiempo y dinero. Se ha diseñado una pcb para alojar la electrónica y poder acoplarla de forma consistente a la estructura del rodillo. Esta operación es de suma importancia, pues de la geometría resultante entre el sensor y el encoder es buena responsable del éxito de los datos obtenidos. En la figura 4.29 puede verse el esquemático de la pcb diseñada.

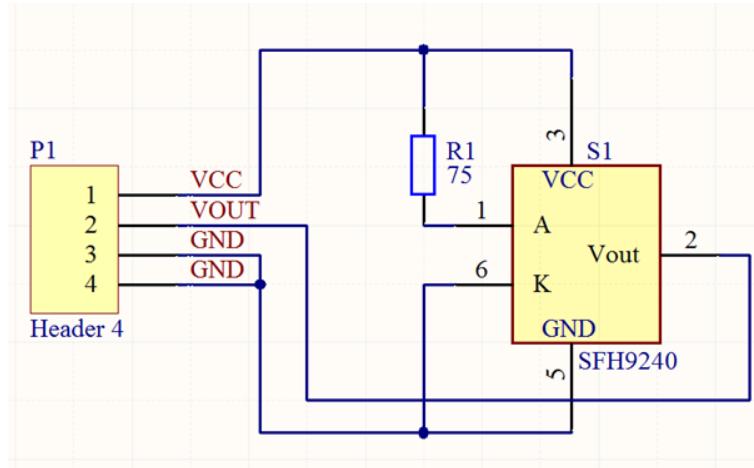


Figura 4.29: Esquemático de la PCB encoder

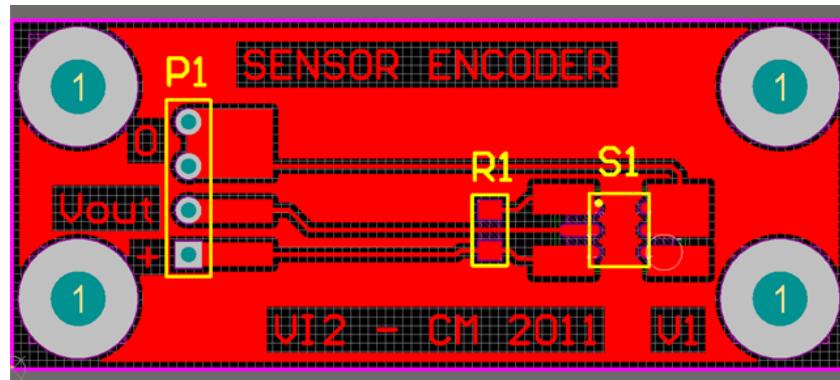


Figura 4.30: Diseño de la PCB del encoder

La pcb ya terminada tiene el siguiente aspecto:

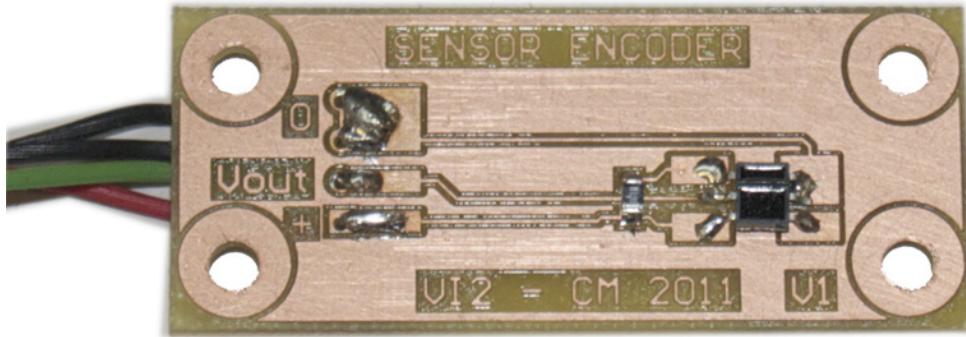


Figura 4.31: PCB encoder construida

Debido a la posición del encoder, muy cercana a elementos en movimiento y sensible a virutas de goma procedentes del desgaste de los neumáticos, se ha encapsulado la electrónica en un

4. Banco de Pruebas

soporte de plástico y se ha recubierto con resina para protegerla. Se muestran a continuación algunas imágenes del proceso de fabricación.

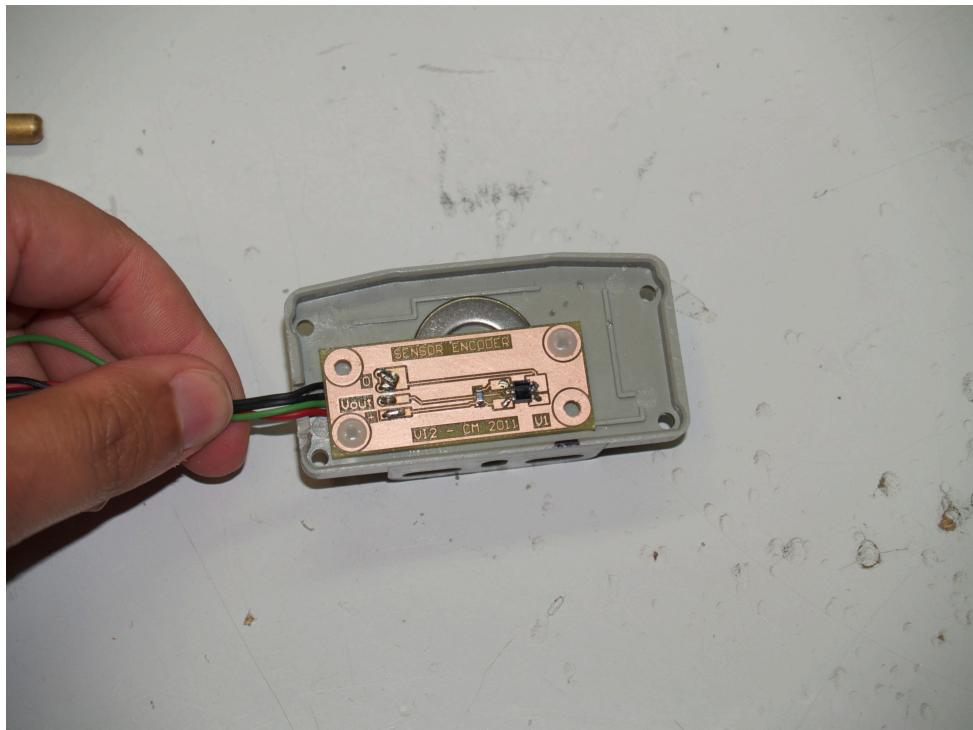


Figura 4.32: Posicionamiento en la carcasa



Figura 4.33: Construcción del protector del sensor



Figura 4.34: PCB cubierta de resina

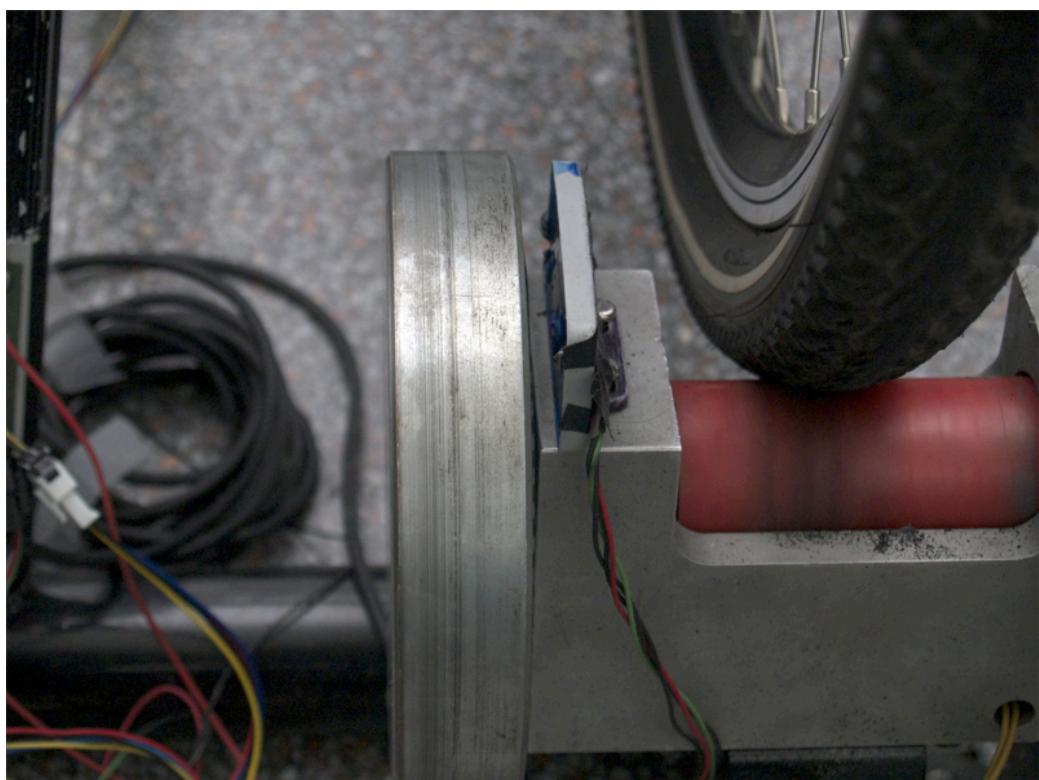


Figura 4.35: Posicionamiento final del encoder

Experimentalmente, se ha probado con diferentes materiales para la construcción del disco del encoder. Finalmente se ha optado por realizar una impresión de un disco de 100 ranuras sobre papel de acetato. Este disco se ha pegado sobre la cara reflectante de un CD y se ha protegido mediante forro transparente adhesivo. Finalmente se ha colocado sobre el disco de inercia del rodillo, respetando la geometría tipificada por el fabricante del sensor. En las siguientes imágenes se puede apreciar el disco encoder ya terminado.



Figura 4.36: Disco encoder reflectante construido.

Se conectará este encoder a la electrónica que controla el rodillo, más concretamente se conecta la salida del encoder a una interrupción externa del dsPIC30F4011 utilizado. Mediante la concatenación de dos timers (el 2 y el 3) se consigue un contador de 32 bits, suficiente para el rango de velocidades que alcanzará el rodillo. El tiempo mínimo entre flancos que se puede calcular es de $8\mu s$ y el máximo es de 10 segundos. El tiempo máximo que permite contar el timer de 32 bits en realidad es mayor, pero no tiene sentido controlar velocidades tan bajas de giro, pudiendo asumir que la velocidad es 0.

4.6.Modificaciones en la estructura del rodillo

Debido al diseño de la bicicleta, donde los cables del motor salen desde el centro del eje de la rueda trasera, se produjo un problema al cortarse los cables del motor cada vez que se montaba la bicicleta en el rodillo y una persona pedaleaba con intensidad.



Figura 4.37: Zona donde se produce el sesgo de los cables del motor.

Para solucionar el problema se han diseñado unos soportes nuevos para el rodillo, especialmente concebidos para el diseño de las bicicletas Ecobike. Con estos nuevos soportes, se da espacio al cable para que no quede aprisionado contra el soporte y se consigue una mejor sujeción de la bicicleta.



Figura 4.38: Proceso de adaptación de la estructura del rodillo.

4. Banco de Pruebas

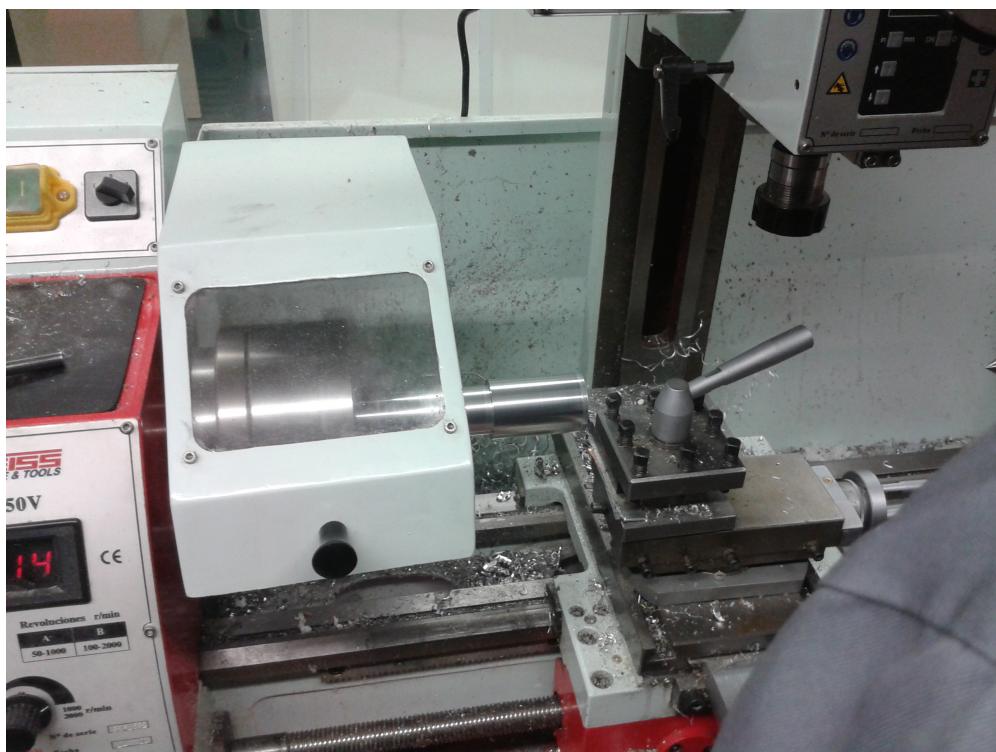


Figura 4.39: Construcción de los nuevos soportes en el torno.

Pueden verse los planos de los soportes diseñados en las figuras 4.40 y 4.41.

Sistema de control y monitorización de datos de una bicicleta eléctrica

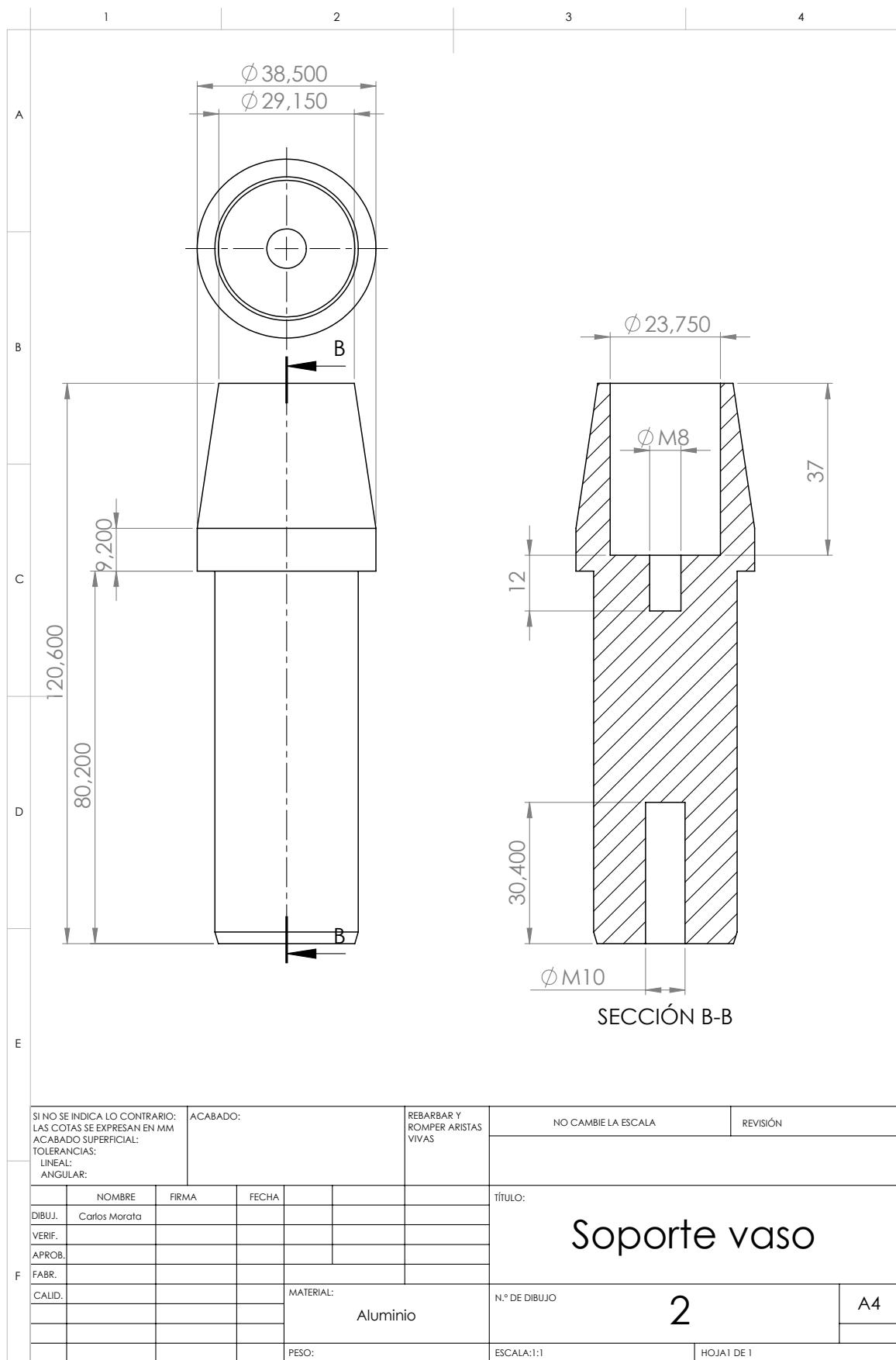


Figura 4.40: Planos del soporte diseñado (I).

4. Banco de Pruebas

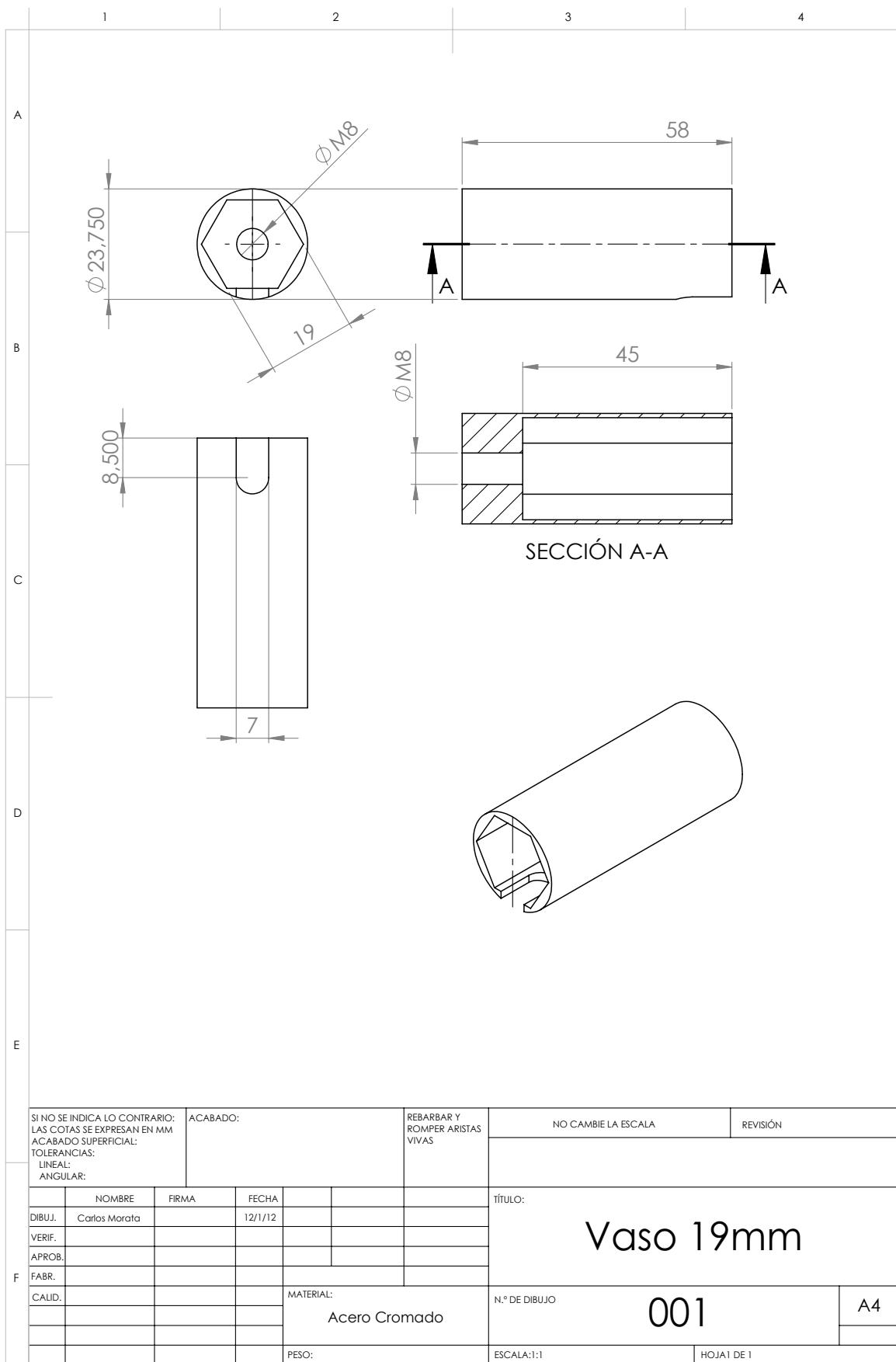


Figura 4.41: Planos del soporte diseñado (II).

El resultado final ha sido muy satisfactorio, pues desde que se utiliza el nuevo soporte no se han vuelto a tener problemas con los cables del motor.

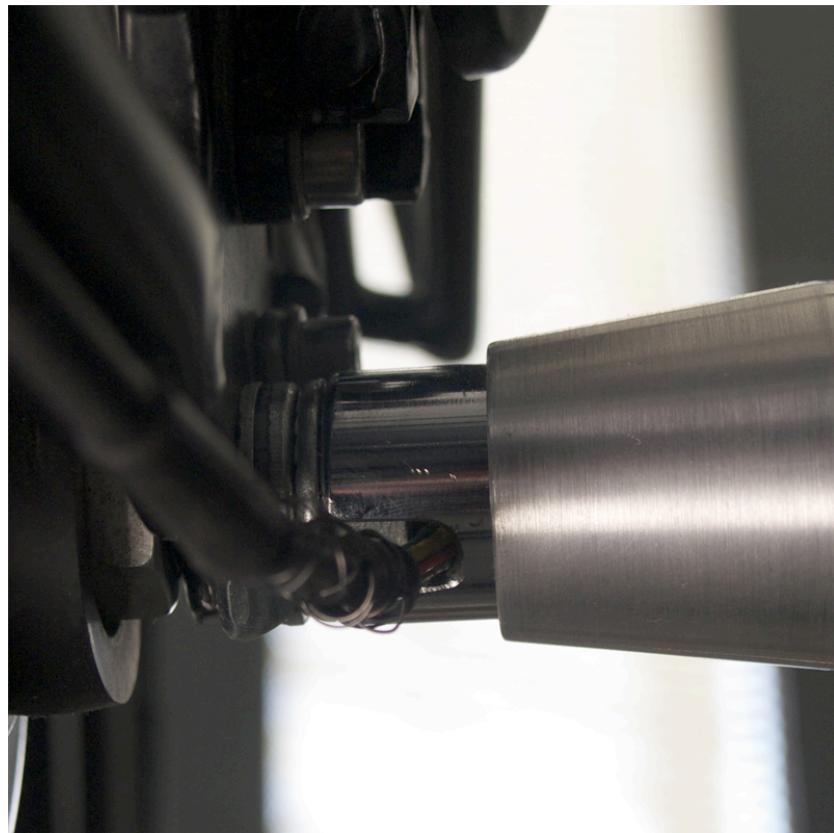


Figura 4.42: Detalle de la salida de cables del motor con el nuevo soporte..

4.7.Cálculo del par realizado por la bicicleta

Una vez se tiene el par realizado por el freno y la velocidad de giro de la rueda de la bicicleta y del disco de inercia del rodillo se puede calcular el par que está realizando en cada momento el motor de la bicicleta. La potencia mecánica realizada por la bicicleta se puede representar de la siguiente forma:

$$P_B = T_B \cdot \omega_B$$

Donde T_B es el torque realizado por el motor de la bicicleta y ω_B es la velocidad angular de la rueda de la bicicleta. De la misma forma, podemos representar la potencia realizada en el rodillo por la siguiente fórmula, variando los términos por los valores calculados en el rodillo:

$$P_R = T_R \cdot \omega_R$$

Se cumple pues que la potencia realizada por la bicicleta es igual detectada en el rodillo:

4. Banco de Pruebas

$$P_B = P_R$$

$$T_B \cdot \omega_B = T_R \cdot \omega_R$$

De donde se deduce que el par realizado por la bicicleta es:

$$T_B = \frac{T_R \cdot \omega_R}{\omega_B}$$

Otro aspecto interesante es conocer el rendimiento (η) de la bicicleta. Conociendo la potencia eléctrica suministrada (P_E) y la potencia mecánica (P_B) de la bicicleta se puede calcular el rendimiento de forma sencilla:

$$P_E = V \cdot I$$

$$P_B = T_B \cdot \omega_B$$

$$P_E \cdot \eta = P_B$$

$$\eta = \frac{T_B \cdot \omega_B}{I \cdot V}$$

Donde V es el voltaje de la batería e I es la corriente consumida por la bicicleta. El rendimiento de la bicicleta puede tomar valores entre 0 y 1

Capítulo 5

Pedalier Thun

El sistema para la medición de par elegido para incorporar en el diseño de la bicicleta ha sido el pedalier sensorizado de la marca alemana Thun. Se ha elegido este pedalier debido a su reducido precio (76€), a que es fácilmente integrable en el diseño actual de la bicicleta. En este capítulo se ahonda en el diseño, las especificaciones y la integración del pedalier.



Figura 5.1: Pedalier Thun

5.1.Especificaciones

El pedalier Thun utilizado es el modelo X-CELL RT que ofrece una medida del par realizado por el ciclista y del giro de los pedales. Incorpora sensores Hall PCME y una electrónica que queda integrada en el diseño del pedalier, haciendo que sea invisible e inaccesible a simple vista. Este aspecto consigue hacerlo más robusto y duradero pues no le afectan las vibraciones, polvo, agua... La electrónica, al no ser accesible no tiene mantenimiento. Permite un montaje similar a la de cualquier otro pedalier, sin necesidad de herramientas específicas.

Specifications	X-CELL RT
Performance 1	Cadence: rotation/min.
Performance 2	Rotational direction
Performance 3	Torque [Nm]
Length of spindles	120K; 120L; 128K; 128L; 133,65K; 133,65L;136L
Certification: EN 14764 (City-Trekking)	Yes
Certification: EN 14766 (MTB)	Yes
Cup threads	BS 1.375 x 24
Right-hand cup	Low profile
Material of cups	PA 6.6 Gf 30 %
Material of sensor shell	Macromelt
Ball Bearings	2 x 61902 2RS
Square	12.73 mm
Surface of spindles	A2B
Assembly tool	Shimano® compatible
Sensory system	2 x Hall-sensors, PCME-sensor
Impulse transmitter 1	Poled ring - 32 impulses/rota-tion
Impulse transmitter 2	Magnetized spindle
Voltage feed	+7...+16 V DC
White cable (input)	Power
Brown cable (output)	Sine signal
Blue cable (output)	Cosine signal
Black cable (ground-connec-tion)	Ground
Grey cable (output)	Torque signal
Length of cable	1100 mm
Signal output: sine	Analogue or digital (open collector)
Signal output: cosine	Analogue or digital (open collector)
Signal output torque: attribute 1	Offset +2500 mV at 0 Nm
Signal output torque: attribute 2	Analogue: ± 10 mV/Nm
Signal output torque: attribute 3	Bandwidth: 250 Hz at -3 dB
Accuracy of signals: sine/cosine	$\pm 3^\circ$ ($\pm 0,8\%$)
Accuracy of signals: torque	Effective range ± 200 Nm
Accuracy of signals: torque	$\pm 2.5\%$
IP level	IP 56 as per EN 60529

Figura 5.2: Especificaciones técnicas del pedalier thun X-CELL RT

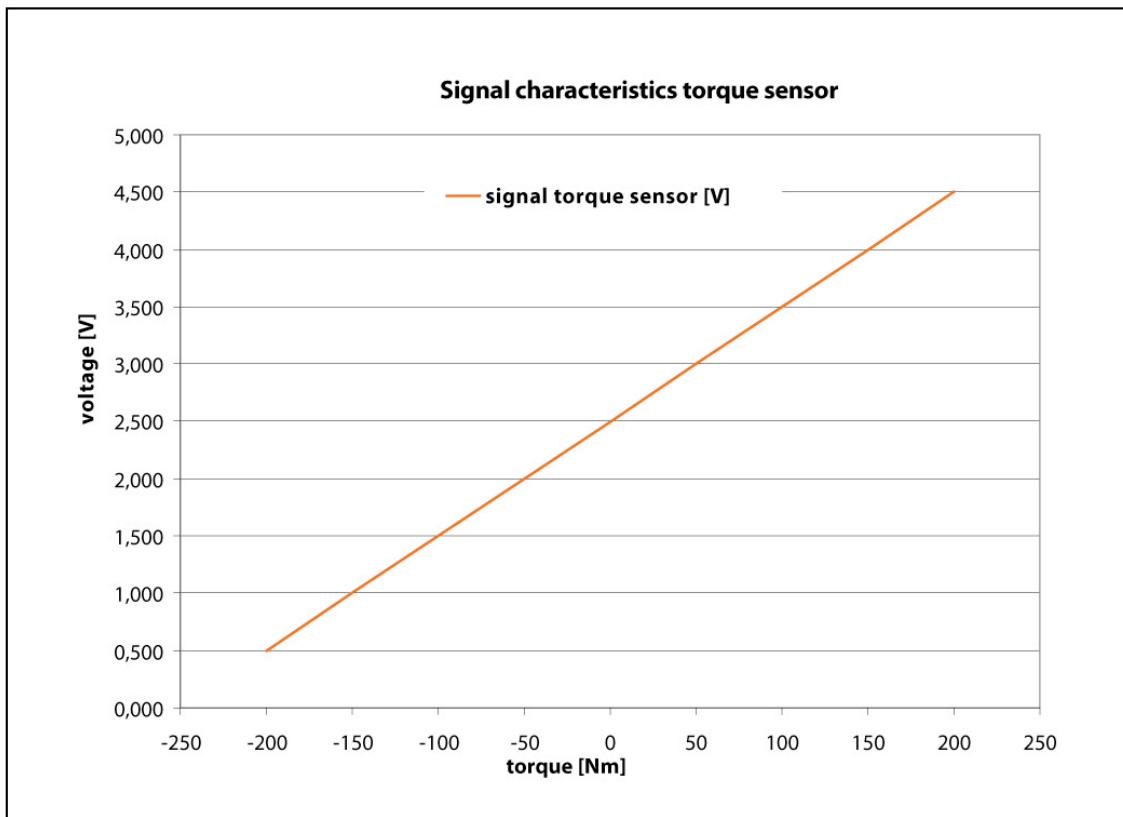


Figura 5.3: Gráfica de la salida de la señal de Par

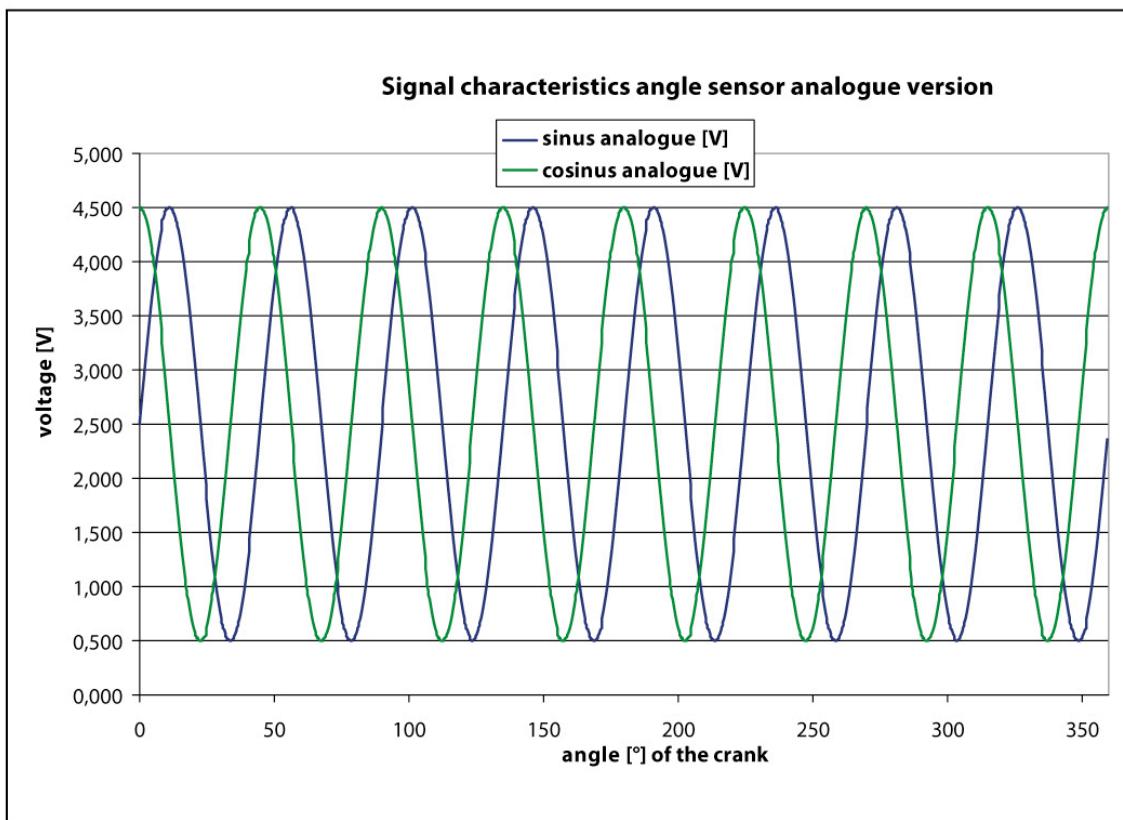


Figura 5.4: Gráfica de la salida sinus/cosinus asociada al giro del pedal

5.2.Principios de funcionamiento

Se puede dividir el funcionamiento del pedalier en dos funciones básicas:

- Detección del giro del pedal
- Medida del par realizado por el ciclista

Para la detección del giro del pedal se utilizan sensores hall con salida analógica. El sensor de efecto Hall o simplemente sensor Hall se sirve del efecto Hall para la medición de campos magnéticos para la determinación de la posición.

Si fluye corriente por un sensor Hall y se aproxima a un campo magnético que fluye en dirección vertical al sensor, entonces el sensor crea un voltaje de salida proporcional al producto de la fuerza del campo magnético y de la corriente. Si se conoce el valor de la corriente, entonces se puede calcular la fuerza del campo magnético; si se crea el campo magnético por medio de corriente que circula por una bobina o un conductor, entonces se puede medir el valor de la corriente en el conductor o bobina.

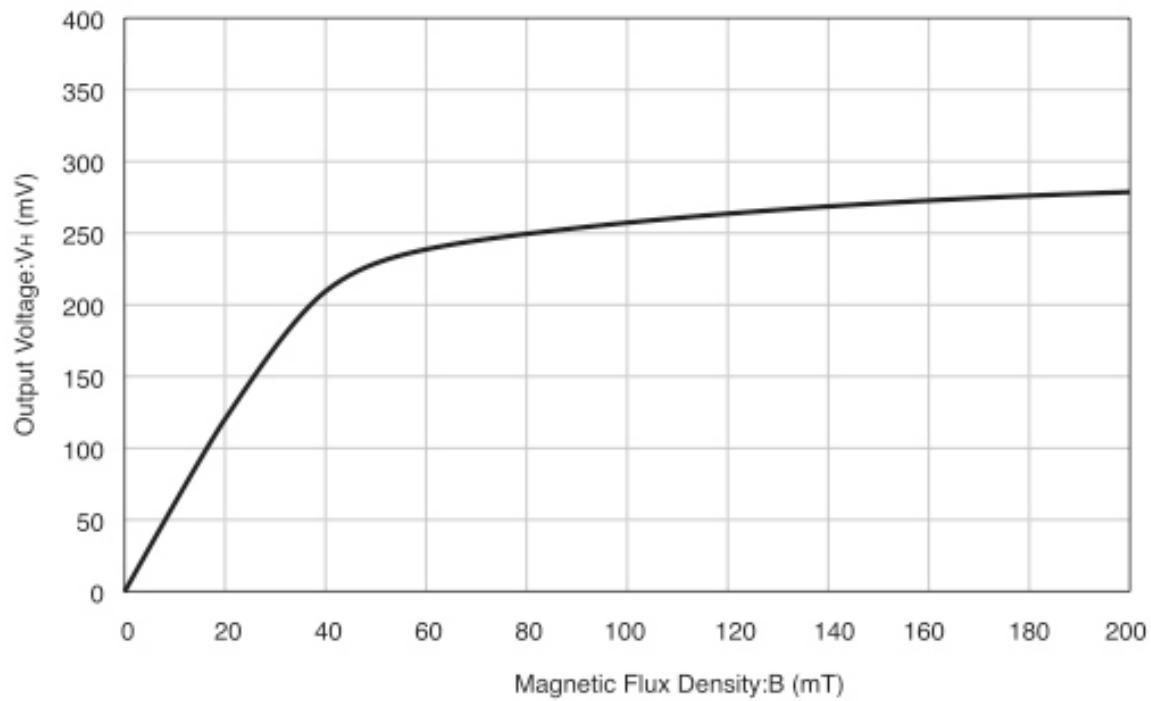


Figura 5.5: Ejemplo de voltaje de salida de un sensor hall analógico

El campo magnético que detectan los sensores hall es generado por un imán de 32 pulsos. La colocación de los sensores de hall con respecto a este imán da como resultado la señal analógica

de salida del pedalier. En el siguiente esquema puede apreciarse la colocación de los sensores con respecto al imán.

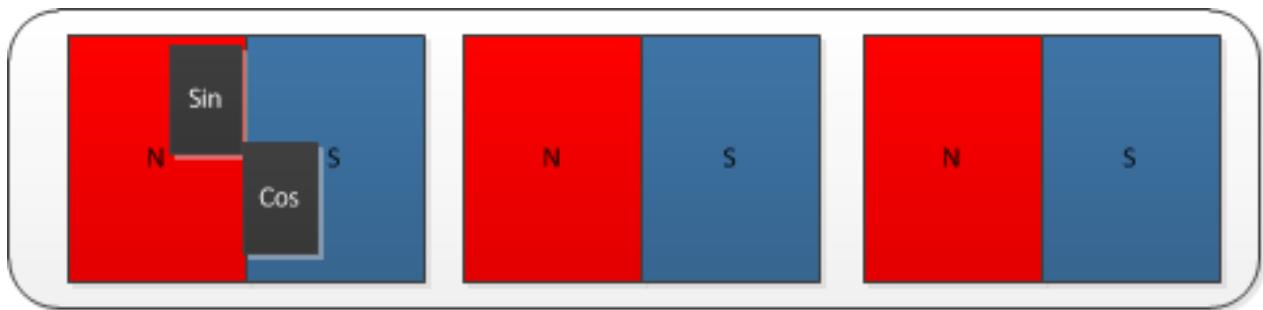


Figura 5.6: Esquema colocación de los sensores hall respecto del imán de 32 pulsos (se visualizan 3 pulsos)

El desfase entre la colocación de un sensor respecto al otro da como resultado la detección del sinus y del cosinus. Por otro lado, al tener el imán 32 pulsos, se observa que el sensor realiza un ciclo de sinus/cosinus entero cada 4 pulsos (norte-sur, sur-norte, norte-sur, sur-norte) es por ello que por cada pedalada el pedalier da 8 sinus completos.

Por lo que respecta a la medida del par se hace uso del un eje ligeramente magnetizado y del principio de magnetoestricción. Se denomina magnetostricción a la propiedad de los materiales magnéticos que hace que estos cambien de forma al encontrarse en presencia de un campo magnético. Las vibraciones en forma de sonido son causadas por la frecuencia de las fluctuaciones del campo. Éste fenómeno es parte de la causa de que se encuentren vibraciones de 100 Hz ó 120 Hz en máquinas eléctricas como motores y transformadores,

Para generar electricidad se utiliza la magnetrostricción inversa, la aplicación de compresión cambia el flujo magnético lo que según la ley de Faraday induce un campo eléctrico. El efecto fue identificado por el científico James Prescott Joule en 1842 cuando observaba níquel puro.

Internamente, los materiales ferromagnéticos tienen una estructura que esta dividida en dos dominios, cada uno de los cuales es una región polarizada magnéticamente. Cuando un campo magnético es aplicado, las fronteras entre los dominios cambian y los dominios rotan, estos dos efectos se ven reflejados en el cambio dimensional del material. El efecto recíproco es el cambio de la susceptibilidad (respuesta a un campo magnético) de un material cuando esta sujeto a deformación mecánica, se le llama efecto magnetoestriktivo inverso, y es el utilizado por el pedalier thun. El efecto Wiedemann que se manifiesta en la torsión de este tipo de materiales cuando un campo magnético helicoidal se aplica en ellos nos permite calcular el ángulo de torsión del eje:

$$\alpha = \frac{j \cdot h}{2G}$$

Donde α es la torsión del eje.

j es la densidad de corriente

h es el parametro de magnetoelasticidad

G es el modulo de Shear (modulo de rigidez)

5.3. Desensamblaje del pedalier

Dado el propósito académico de este proyecto, se ha procedido a desmontar un pedalier sensorizado con tal de ver con más detalle como es en su interior. En las siguientes imágenes se puede observar las diferentes etapas para su desmontado, así como la explicación de los componentes del pedalier.



Figura 5.7: Pedalier fijado al banco de trabajo. Se ha empezado a recortar la cubierta.



Figura 5.8: Comenzando a dejar la electrónica al descubierto

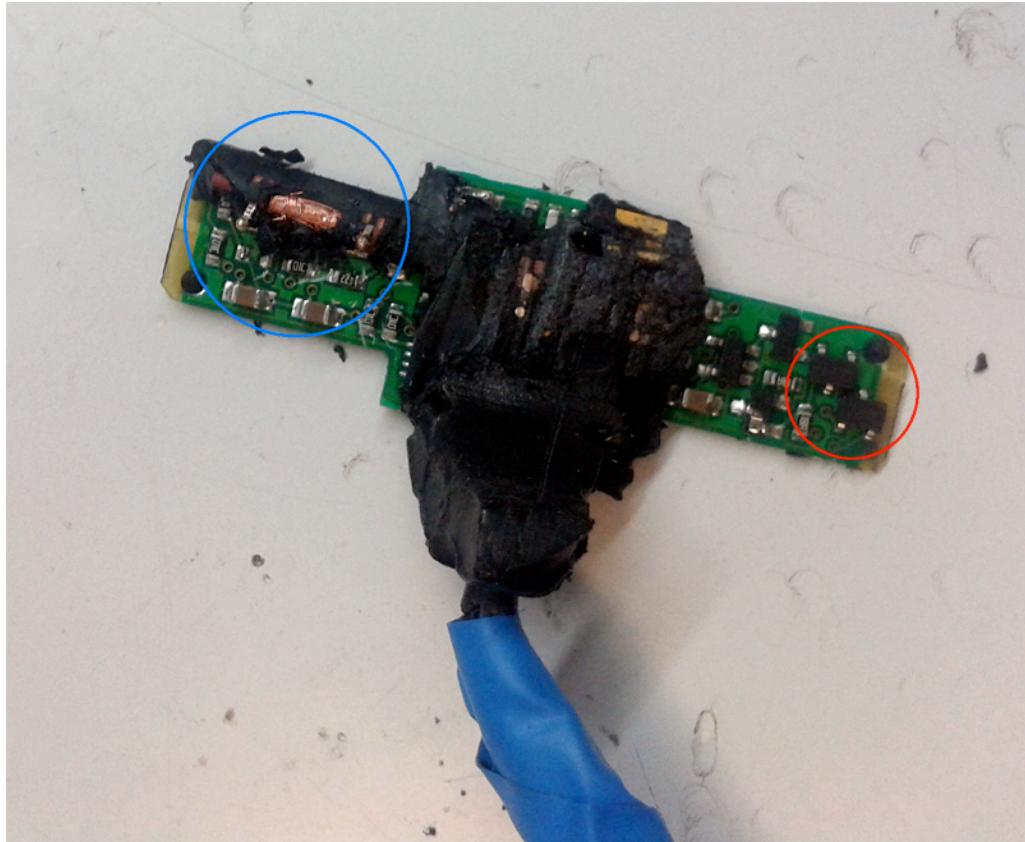


Figura 5.9: Electrónica del pedalier thun. En azul el sensor PCME y en rojo los sensores hall



Figura 5.10: Eje del pedalier Thun

5.4. Incorporación al diseño de la bicicleta

Una vez decidido el sensor de pedalier que se utilizará en este PFC, el siguiente paso es incorporarlo al diseño actual de la bicicleta. Los pasos a seguir son: instalación del pedalier en la bicicleta, captura de los datos mediante la electrónica, calibración y transmisión del dato. La instalación se ha llevado a cabo por los mecánicos de ecobike.



Figura 5.11: instalación del pedalier

Alimentamos eléctricamente el pedalier a 12V. En la figura 5.13 podemos apreciar una captura, en crudo, de las señales que nos da el pedalier (se ha utilizado un osciloscopio para obtenerla)



Figura 5.12: Capturando datos con el osciloscopio

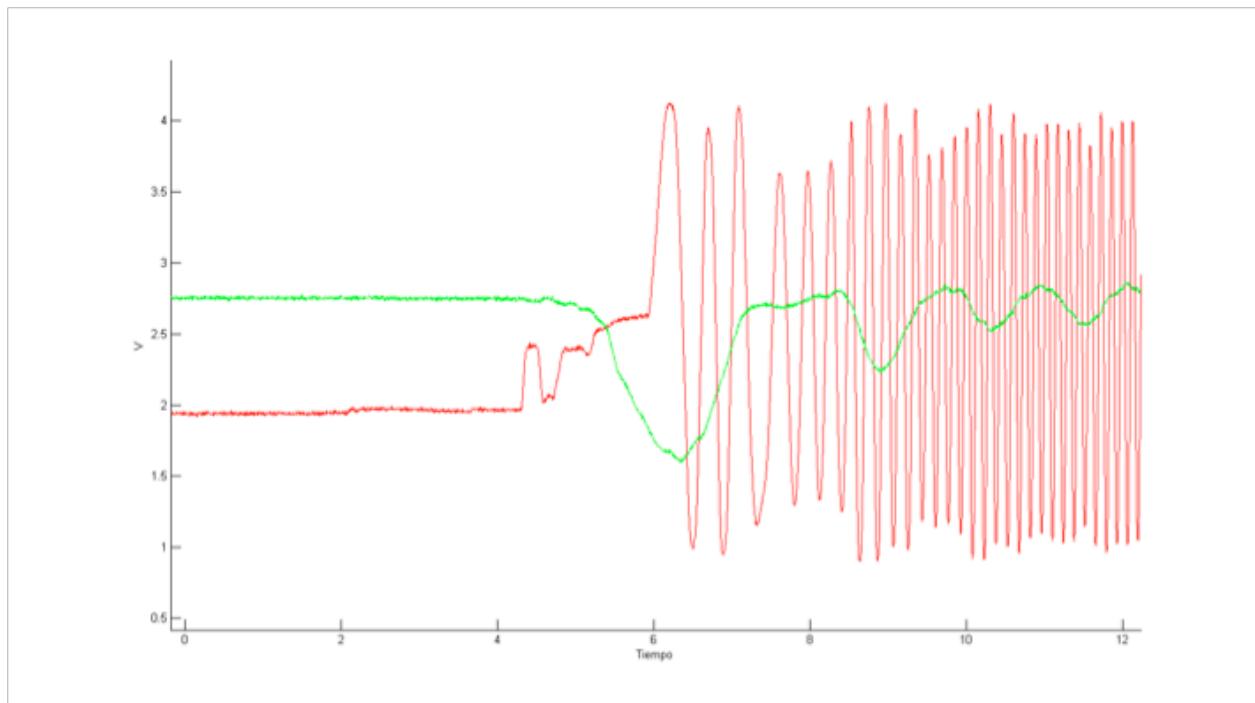


Figura 5.13: ejemplo datos capturados con osciloscopio. En rojo podemos ver la medida del cosinus y en verde el par realizado por el ciclista.

La captura de la señal se realiza mediante los ADs del microcontrolador presente en la electrónica ecocontrol de la bicicleta. Se realiza una calibración con la primera lectura estable del pedalier y

5. Pedalier Thun

posteriormente se envia via CAN el valor leido a cada lectura del AD. Un extracto del codigo que realiza la calibración:

```
HL_Can_TmpTxTrama.Longitud = 2;
HL_Can_TmpTxTrama.FrameID = (3<<8)|(0<<5)|8;
HL_Can_TmpTxTrama.Datos[0] = ((torque)>>8)&0x0FF;
HL_Can_TmpTxTrama.Datos[1] = ((torque))&0x0FF;
HL_Can_Tramas_Tx_Put();
```

La parte del codigo encargada de enviar la lectura por el bus CAN es la siguiente:

```
if (pedalierCalibrado==0) {
    pedalierCalibrado=1;
    pedalier0=lecturaAD;
}
torque=lecturaAD-pedalier0+2500; //en dNm
```

Capítulo 6

Modelo Simulink de una Bicicleta Eléctrica

Matlab es un software matemático creado en 1984, que ofrece un entorno de desarrollo (IDE) y un lenguaje de programación (lenguaje M) disponible en los tres sistemas operativos más populares actualmente: Windows, GNU/Linux y Mac OSX.

La facilidad de manipulación de los datos que ofrece este entorno, hace que sea una herramienta muy utilizada en los procesos de ingeniería que requieren una manipulación de datos a gran escala y su tratamiento con gran rapidez.

Actualmente, este entorno, dispone de herramientas adicionales como Simulink, que permiten aumentar sus funcionalidades. Simulink, es un entorno que permite programar visualmente un sistema en Matlab. Esta herramienta será usada en los siguientes puntos y permitirá modelar de forma fácil el control que se quiere ejercer sobre la bicicleta, así como la monitorización de todos los datos publicados en el bus CAN. Con el modelo creado se podrán crear de forma gráfica prototipos de estrategias de control de forma relativamente fácil, así como debugar de forma cómoda.

6.1.Simulación de la bicicleta

Se ha creado un modelo que simula la física de una bicicleta. Para crear este modelo se ha utilizado como base un modelo realizado en el departamento de ESAII por Xavier Gallego. El modelo de base ya soluciona todo el estudio de las fuerzas implicadas en el movimiento de la bicicleta. La ampliación que se ha realizado en este proyecto a consistido en añadirle la transmisión, cambio de marchas y ruedas libres. El modelo resultante permite simular pruebas sobre el comportamiento de una bicicleta variando los diferentes parámetros implicados en el movimiento:

- Fuerzas en contra (rodamiento, aerodinámica y pendiente)
- Fuerzas a favor (inercias y pares)

Se ha utilizado la *toolbox* Simdrive de Simulink que facilita el diseño de sistemas mecánicos y permite, entre otras cosas, diseñar y simular sistemas de transmisión que permiten transmitir el

6. Modelo Simulink de una bicicleta eléctrica

par de un sistema de ejes de forma sencilla. Estas líneas de transmisión son cuerpos girando alrededor de un eje fijo y sujetos a las leyes de la dinámica de Newton

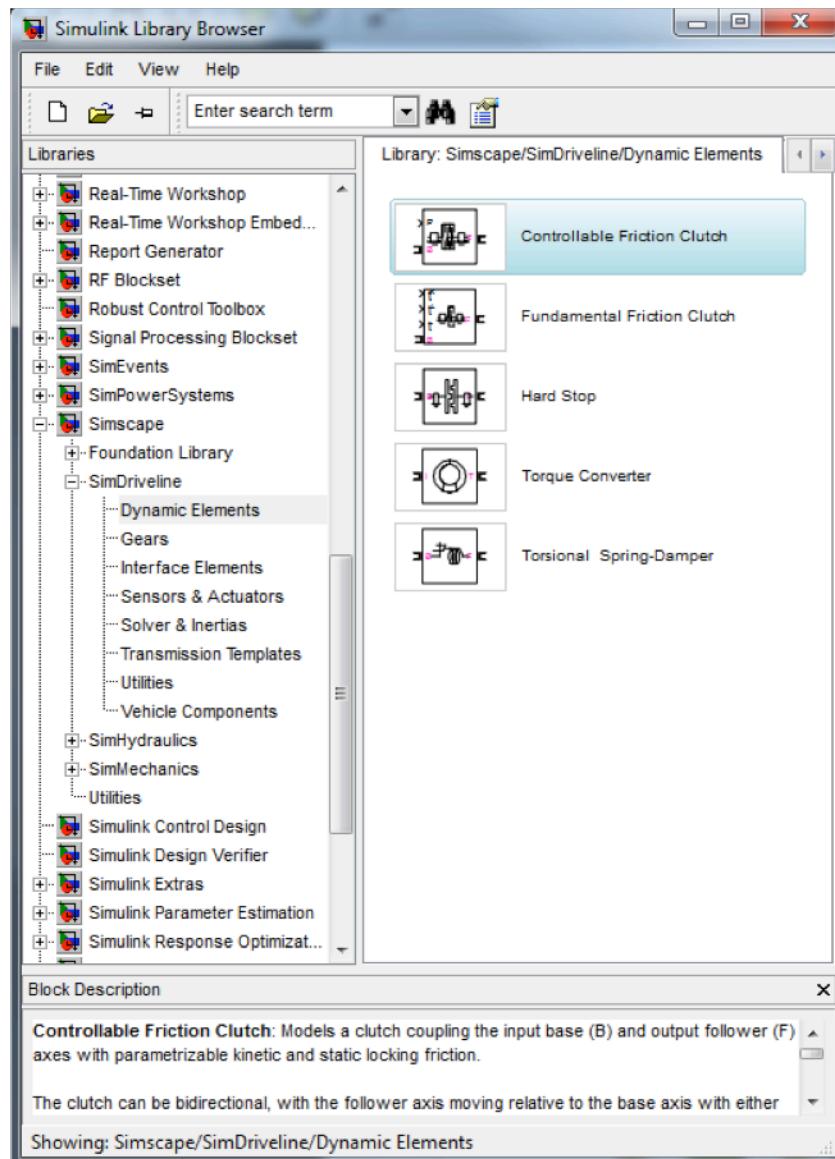


Figura 6.1: Bloques funcionales de Simdrive

El modelo completo es el siguiente:

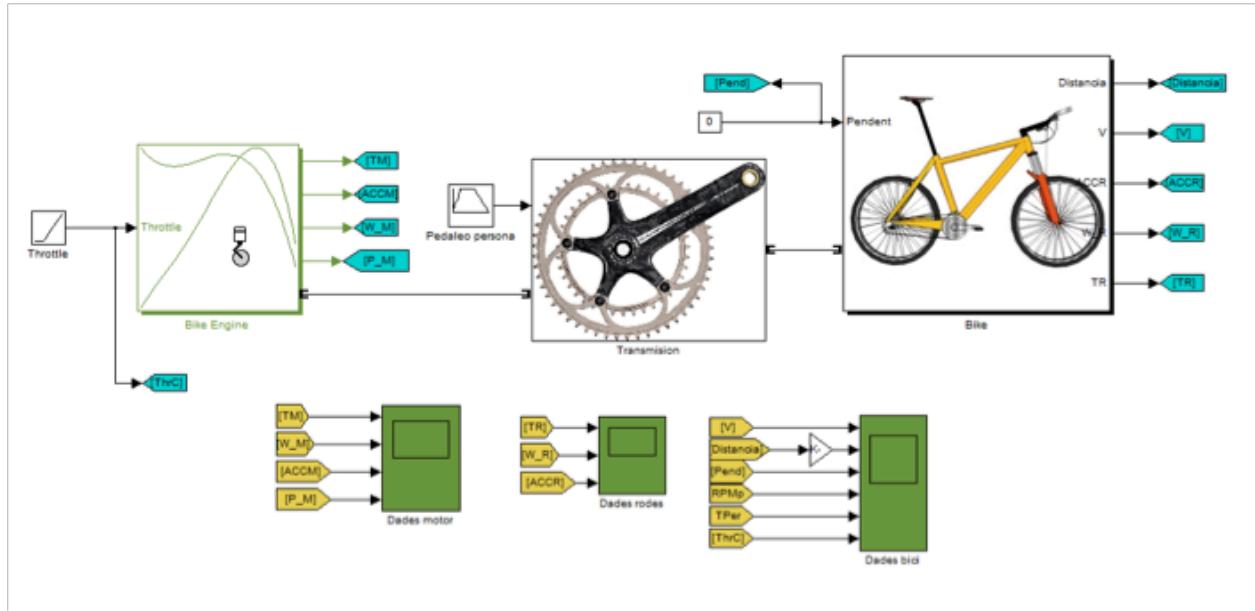


Figura 6.2: Modelo para la simulación de una bicicleta eléctrica

Como se puede observar hay tres bloques principales: el motor, la transmisión y la bicicleta.

Por lo que respecta al bloque del motor consta de cinco salidas:

- Par Motor
- Aceleración angular del motor
- Velocidad angular
- Potencia del motor
- Salida *simdrive* del par motor

Y una entrada:

- Acelerador

Estas cinco salidas son las que permiten controlar el buen funcionamiento del motor, es decir, permiten saber en todo momento si el motor está funcionando de forma correcta. por otro lado, la entrada permite emular el proceso de aceleración que sufriría un motor en la realidad, donde este, no da todo el par de forma instantánea.

Al iniciar el modelo, el motor llama a una rutina (*bike_engine_model*) implementada en Matlab que permite obtener los datos asociados al motor elegido. Estos datos son enviados a las tablas speed

6. Modelo Simulink de una bicicleta eléctrica

y torque utilizadas por una lookup table para dar el par asociado a una velocidad concreta del motor.

Las tablas de velocidad y par, son generadas por aproximación polinómica utilizando las funciones *polyval* y *polyfit* de Matlab. Estas funciones permiten generar un polinomio que representa la tabla motor y nos ayuda a representar todos los valores de este de forma que no sea necesario entrar todos los datos de forma manual.

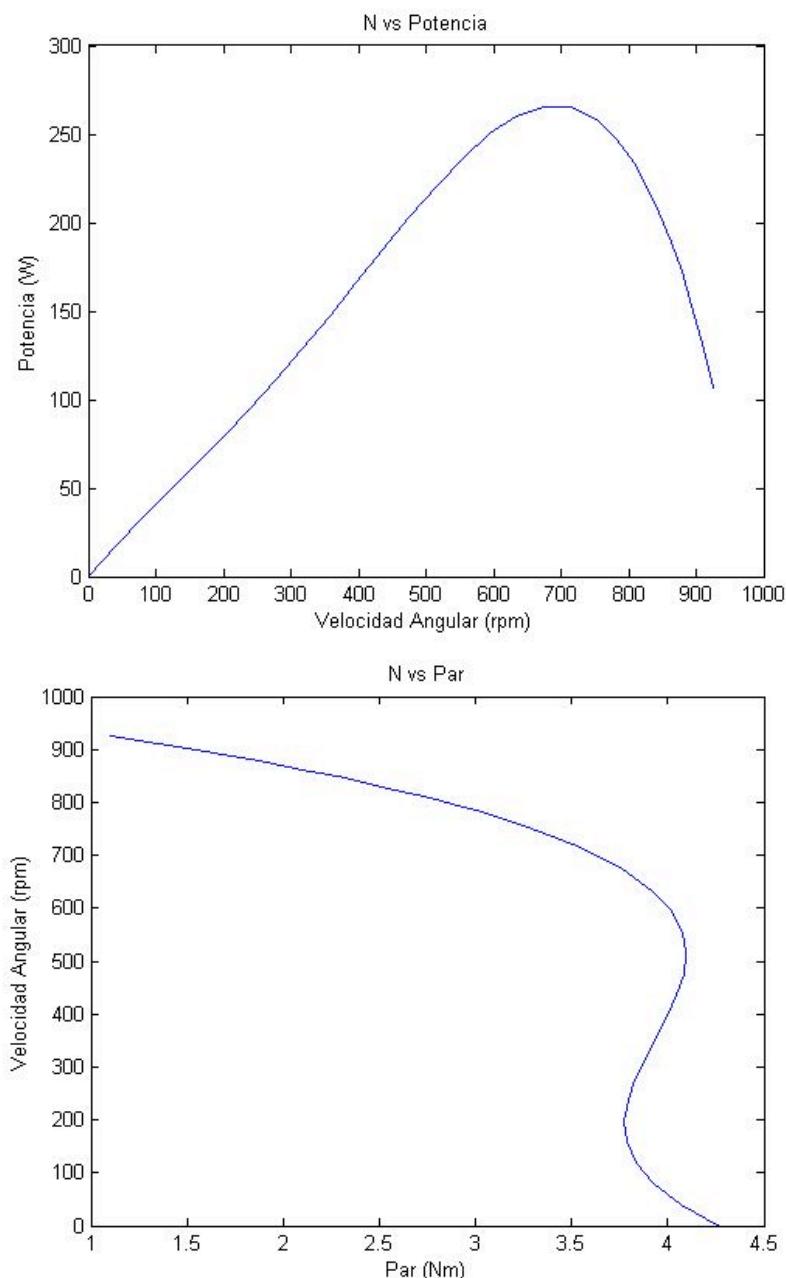


Figura 6.3: Gráficos que caracterizan a un motor BLDC de 250W

El bloque de la bicicleta es el encargado de simular la dinámica de la bicicleta, es por tanto, el encargado de proporcionar el par resistente que sufre la rueda debido a las fuerzas en contra. El

bloque de la bicicleta permite configurar diversos parámetros, como son la masa, el coeficiente aerodinámico, la sección de penetración en el aire y el coeficiente de rodadura.

Consta de cinco salidas:

- Distancia
- Velocidad
- Aceleración angular de las ruedas
- Velocidad angular de las ruedas
- Par de las ruedas

Y de dos entradas:

- Pendiente
- Par en el eje

Estas cinco salidas permiten obtener el estado actual de la bicicleta para comprobar su correcto funcionamiento. Por otro lado, las entradas permiten transmitir el par del eje de la rueda y el pendiente actual en el que se encuentra la bicicleta que permite calcular la fuerza debida a la gravedad que actúa sobre el ciclista.

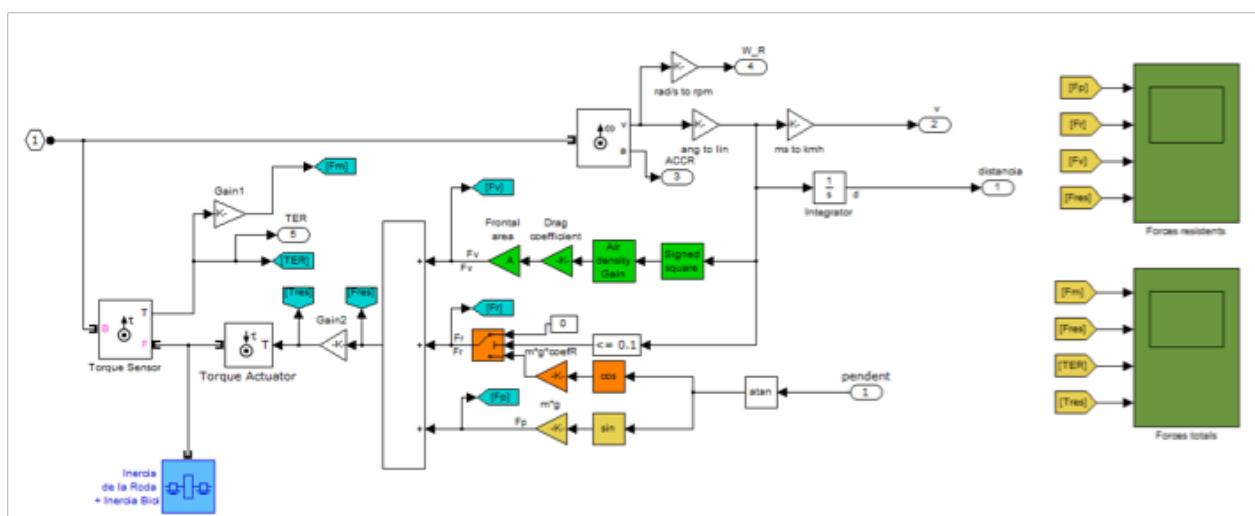


Figura 6.4: Implementación del bloque bicicleta

Por último, el bloque de la transmisión es el encargado de transmitir la fuerza realizada por el ciclista al resto del conjunto. Este bloque incorpora además las ruedas libres que permiten al motor girar sin afectar a los pedales, así como un cambio de marchas.

6. Modelo Simulink de una bicicleta eléctrica

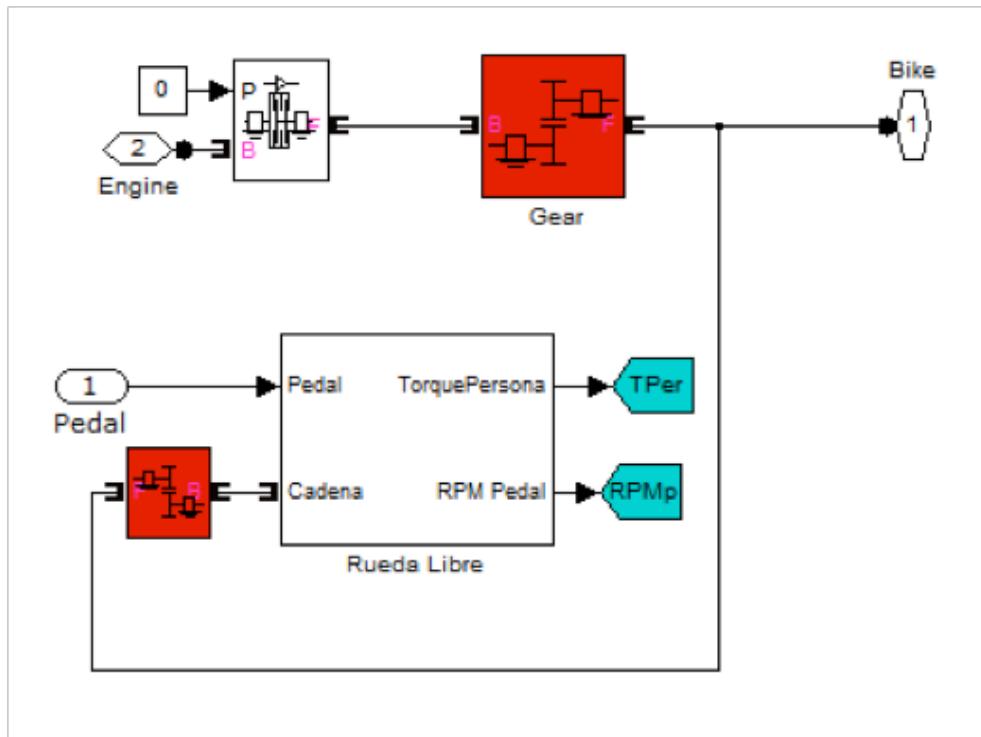


Figura 6.5: Implementación del bloque transmisión

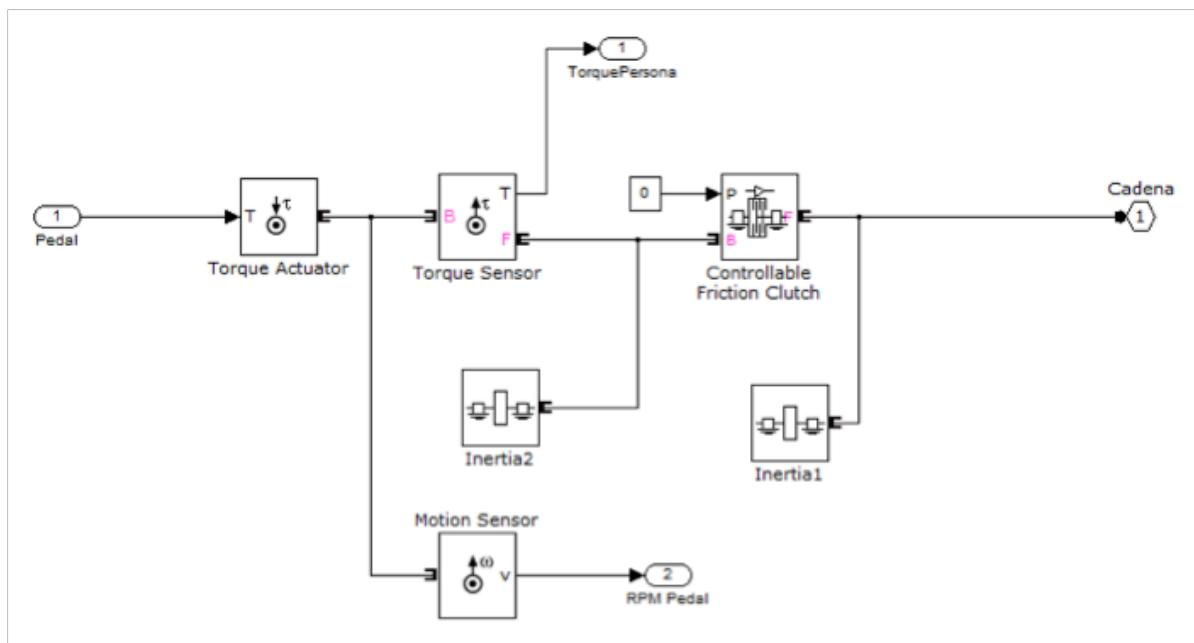


Figura 6.6: Implementación del sistema de ruedas libres

El sistema de transmisión cuenta con dos ruedas libres al igual que en una bicicleta eléctrica real: uno que permite el giro de la rueda sin necesidad de mover el motor, y una segunda rueda libre que permite avanzar a la bicicleta sin necesidad de mover los pedales.

A continuación se muestran algunos ejemplos de ejecuciones del modelo, que permiten a su vez demostrar su funcionamiento:

Funcionamiento en llano. Persona sin pedalear

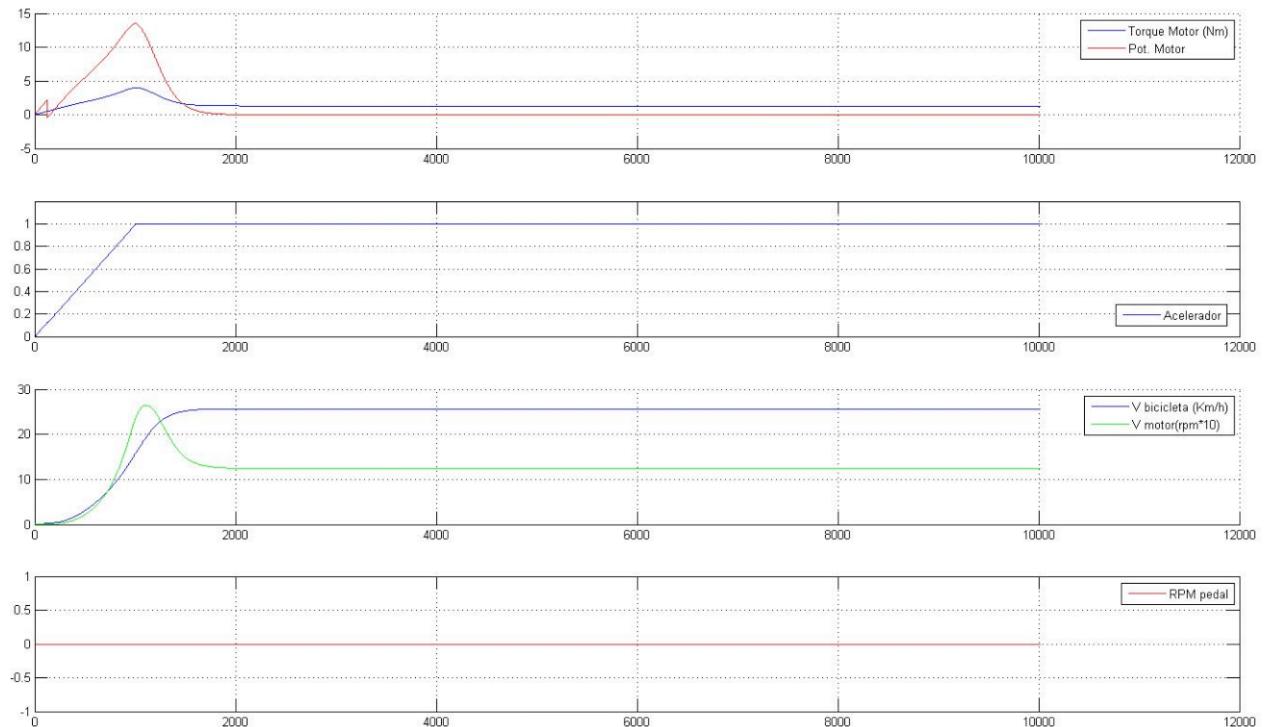


Figura 6.7: Funcionamiento en llano. Datos de la bicicleta

Explicación: Como se puede apreciar en este gráfico, el comportamiento de la bicicleta es el esperado de la bicicleta avanzando sin la ayuda de la persona. Al accionar el acelerador, la bicicleta comienza su marcha, aumentando su velocidad. La aportación de la persona al movimiento de la bicicleta es nula. El fin de este experimento es comprobar como la rueda libre de la carcasa del motor funciona.

Funcionamiento en llano. Persona pedaleando hacia atrás

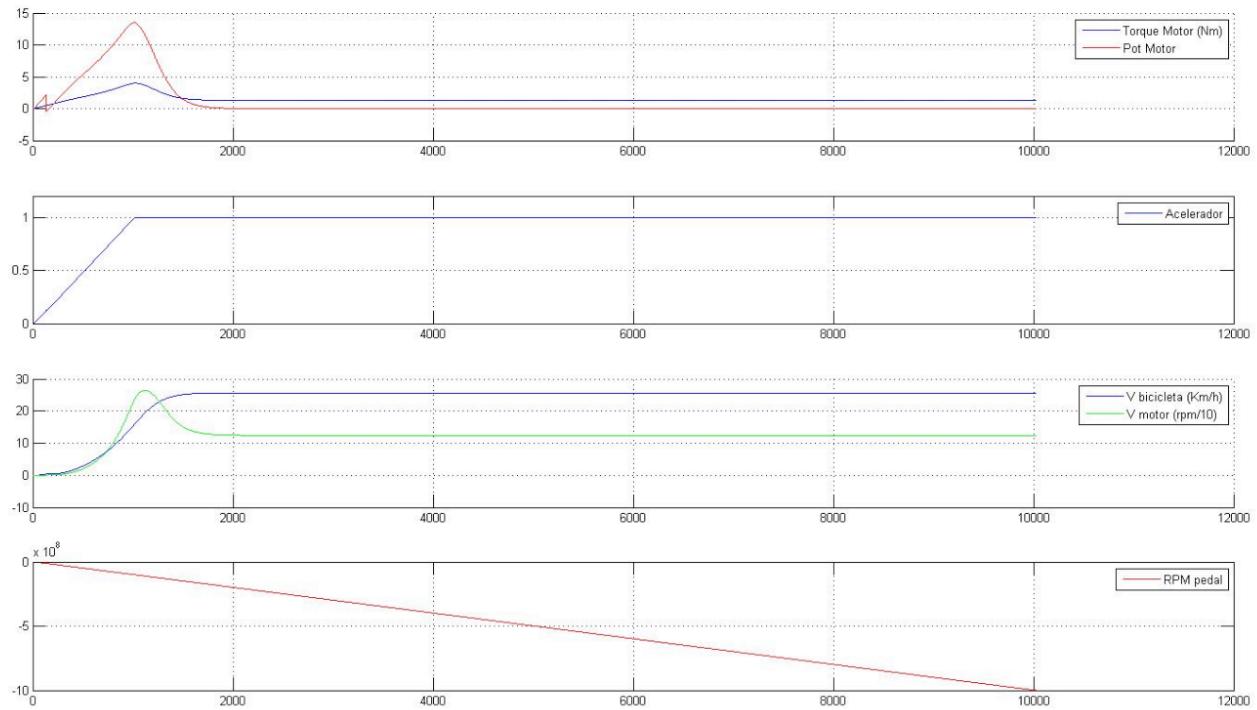


Figura 6.8: Funcionamiento en llano pedaleando hacia atrás. Datos de la bicicleta

Explicación: La persona pedalea hacia atrás sin hacer aportación de par al movimiento de la bicicleta. el comportamiento de la bicicleta es el esperado de la bicicleta avanzando sin la ayuda de la persona. Al accionar el acelerador, la bicicleta comienza su marcha, aumentando su velocidad. La aportación de la persona al movimiento de la bicicleta es nula.. La rueda libre de la transmisión funciona.

Funcionamiento en llano. Persona pedaleando sin motor

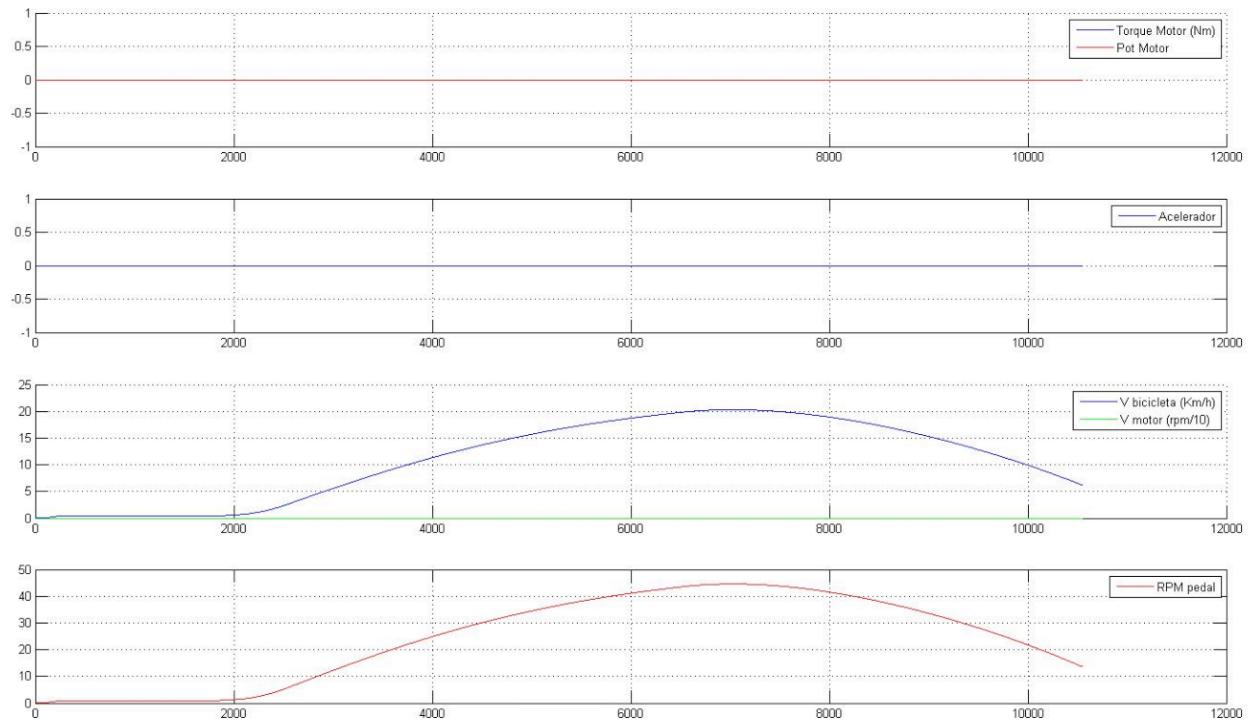


Figura 6.9: Funcionamiento en llano pedaleando sin motor. Datos de la bicicleta

Explicación: La persona comienza a pedalear y la bicicleta se mueve. Se puede apreciar como a más par aplicado por la persona, las rpm del pedal aumentan, y en consecuencia , más velocidad consigue la bicicleta. El objetivo de este experimento es comprobar que la la transmisión y la rueda libre del motor funcionan.

Funcionamiento en llano. Pedal asistido por motor

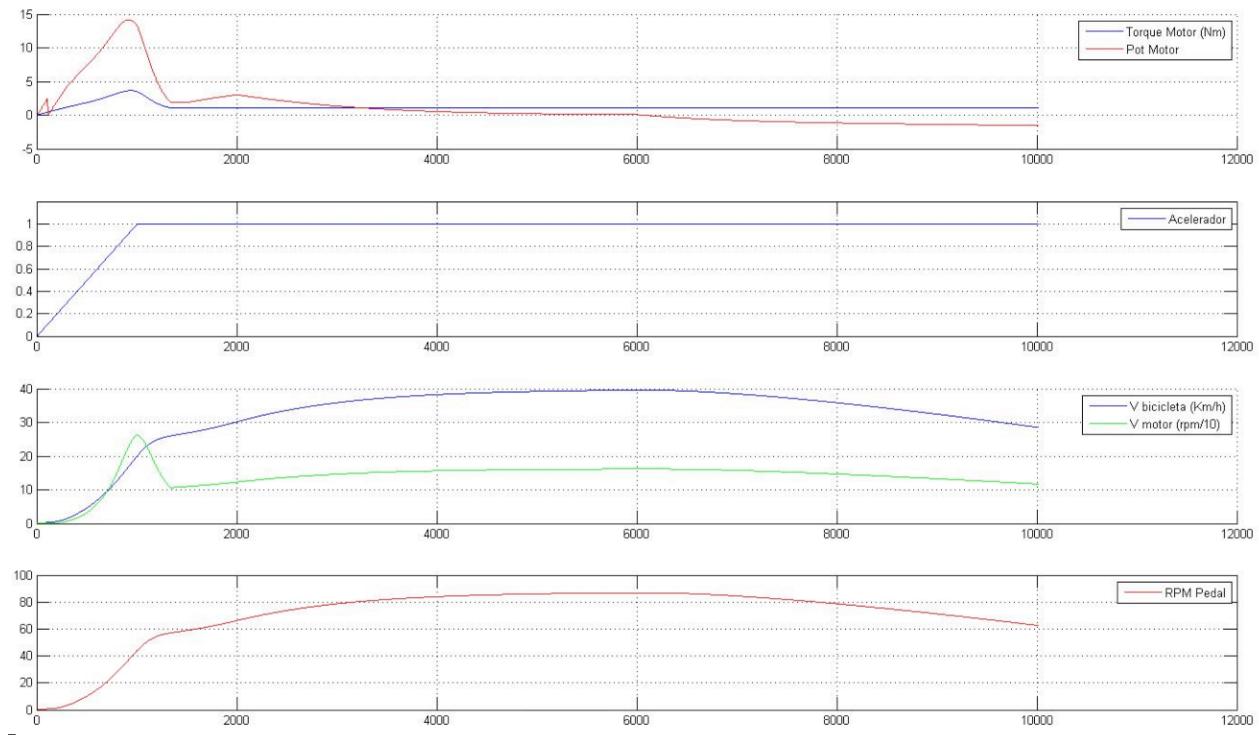


Figura 6.10: Funcionamiento en llano pedal asistido por motor. Datos de la bicicleta

Explicación: En este caso, tanto como la persona como el motor hacen aportación al movimiento de la bicicleta. Se puede apreciar, comparando con los casos anteriores, que la bicicleta acelera más rápido cuando la persona aplica par sobre los pedales y como pierde velocidad cuando el ciclista deja de pedalear. Se puede obtener la conclusión que el conjunto motor transmisión funcionan.

6.2. Modelo para la emulación del control de la bicicleta

Para poder desarrollar el modelo de control compartido (parte emulado en Matlab y parte en la electrónica ecogate de la bicicleta), es necesario contar con un modelo Simulink capaz de conectarse al bus CAN para poder leer las tramas provenientes de la bicicleta, así como tener la capacidad de escribir en el bus para enviar las consignas de control a la bicicleta. En este apartado se explica la implementación de este modelo.

6.2.1. Bloque CAN Interface

Tal y como se ha comentado anteriormente, surge la necesidad de implementar una comunicación bidireccional con el bus can. Para interconectar el modelo creado en Simulink con las diferentes electrónicas presentes en la arquitectura de la bicicleta, se ha elegido una interfaz CAN a USB del fabricante Lawicel.



Figura 6.11: Lawicel Canusb

Lawicel nos proporciona una API que permite desarrollar aplicaciones que interactúen con el dispositivo canusb. Las llamadas esenciales que nos ofrece la API de Lawicel son:

CANLIB Core API Calls
canInitializeLibrary
canOpenChannel
canClose
canSetBusParams
canBusOn
canBusOff
canRead
canReadWait
canWrite
canWriteSync

Figura 6.12: Llamadas implementadas por la API del Lawicel CANUSB

Para poder utilizar estas funciones desde Simulink, se necesita utilizar un bloque de Simulink llamado S-Function. El contenido de este bloque puede ser escrito en el lenguaje de programación C. Las funciones implementadas en este bloque pueden ser utilizadas durante el ciclo de vida del modelo Simulink.

6. Modelo Simulink de una bicicleta eléctrica

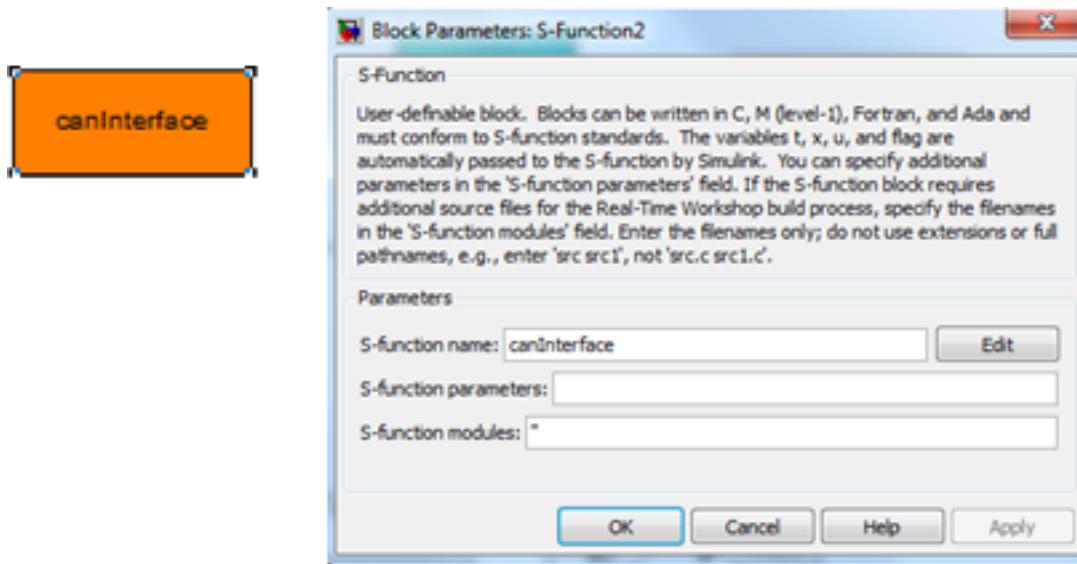


Figura 6.13: Bloque canInterface y sus parámetros

La programación de todas las S-Functions ha sido realizado por Toni Benedico del departamento de ESAII.

6.2.2.Bloques CanWrite y CanRead

Una vez se ha iniciado la comunicación con el modulo Lawicel, se necesita un mecanismo para poder leer las tramas que viajan por el bus, así como poder generar nuestras propias tramas. Los bloques encargados de este cometido, canWrite y canRead están implementados mediante una S-Function. La implementación de los bloques se puede observar en las figuras 6.14 y 6.15

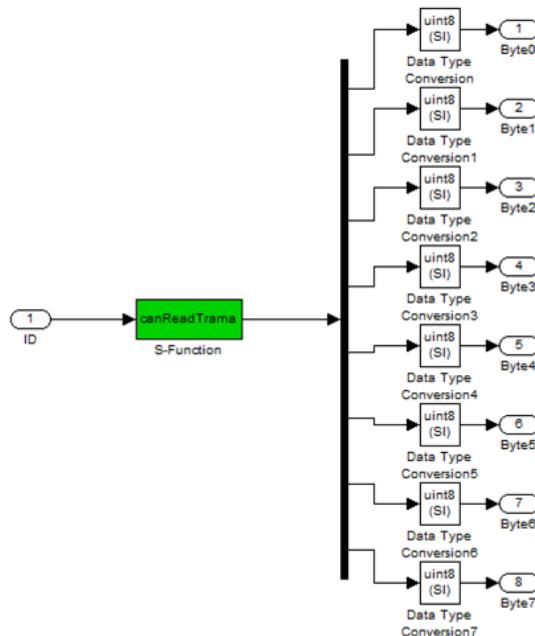


Figura 6.14: Implementación CanRead

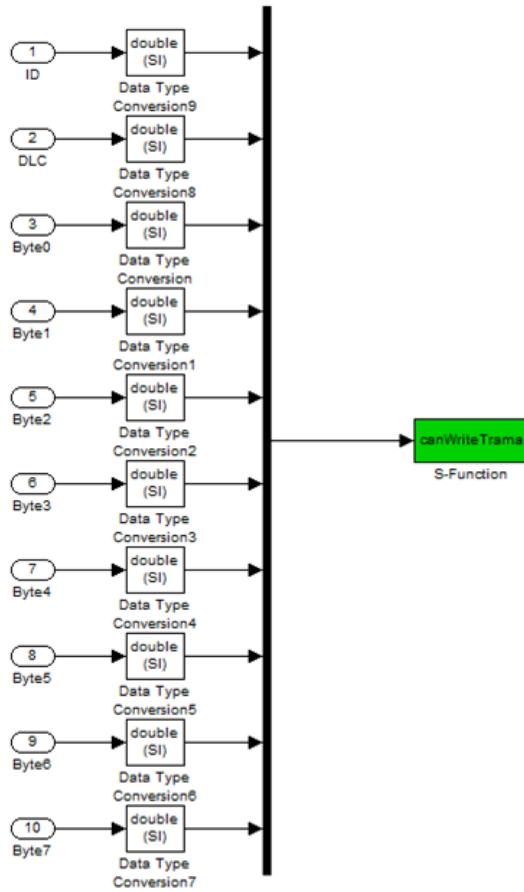


Figura 6.15: Implementación CanWrite

6.2.3.Bloque SampleTimeSync

Este bloque implementa una espera activada 0,01s. De esta forma se consigue una ejecución en pseudotiempo real del modelo Simulink. Este bloque está implementado al igual que el anterior en una S-Function.

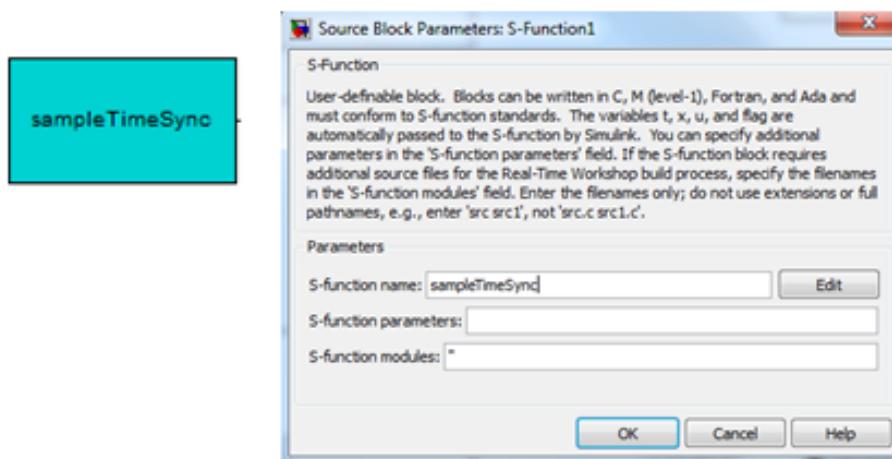


Figura 6.16: Bloque sampleTymeSync y sus parámetros

6. Modelo Simulink de una bicicleta eléctrica

Realizar la espera activa, introduce una serie de dependencias en el sistema. A saber:

- Las inicializaciones del canInterface se tienen que ejecutar antes que cualquier bloque acceda al bus.
- La espera realizada por el bloque sampleTimeSync se tiene que ejecutar al final de cada ciclo de simulación, después de que todos los bloques hayan accedido a CAN.

Simulink permite definir la prioridad de ejecución de cada bloque, forzando la ejecución de forma ordenada. Las prioridades asignadas a cada bloque se presentan a continuación. Los numeros negativos y el 0 son valores validos de prioridad. Mientras más bajo sea el número, más prioritario es el bloque: por ejemplo, -1 es más prioritario que 1.

Tipo de Bloque	Prioridad	Instancias en el modelo
canInterface	-1	1
readCANFrame	0	m
writeCANFrame	0	n
sampleTimeSync	1	1

6.2.4.Bloque Ecocontrol

Este bloque implementa el acceso a todas las tramas CAN que publica la electrónica ecocontrol de la bicicleta.

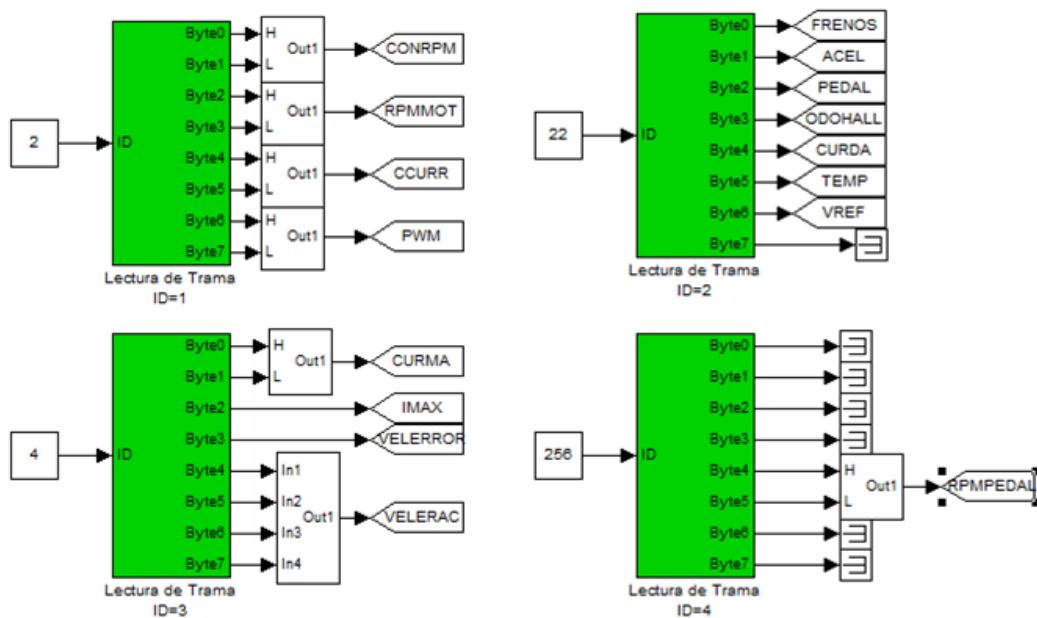


Figura 6.17: Implementación bloque ecocontrol

6.2.5.Bloque Rodillo

Este bloque implementa el acceso a todas las tramas CAN que publica la electrónica ecocontrol modificada que controla el freno. Permite a su vez, dar una consigna de velocidad y de consumo máximo de corriente al motor.

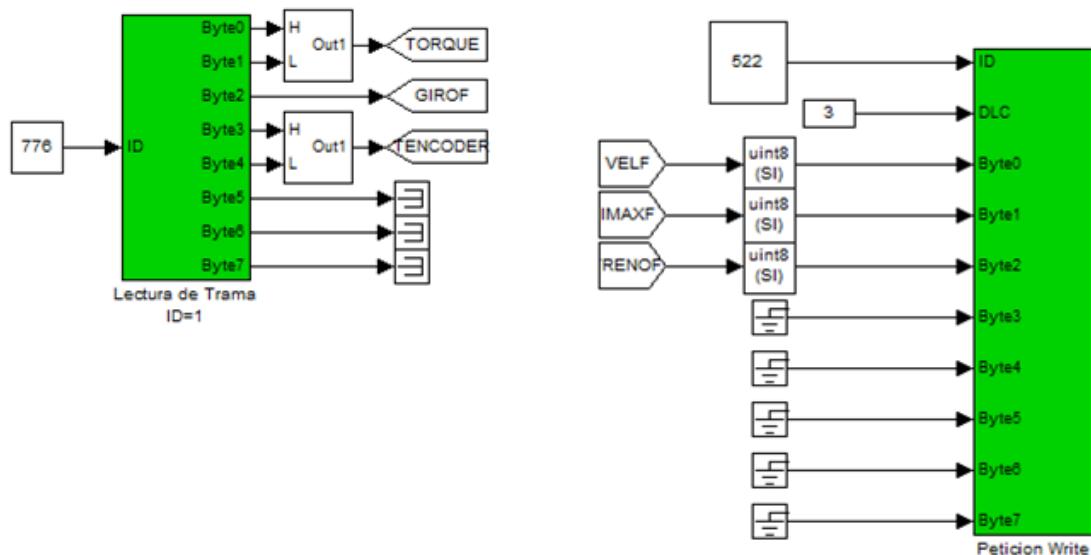


Figura 6.18: Implementación bloques Rodillo

6.2.6.Bloque 4PGA

Este bloque implementa el acceso a todas las tramas CAN que publica la electrónica 4PGA que mide el par realizado por la bicicleta

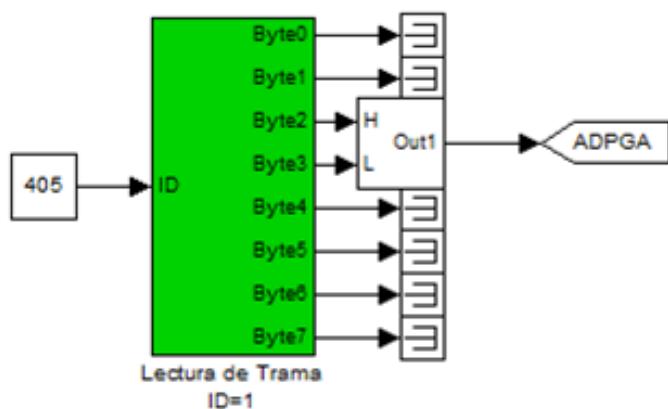


Figura 6.19: Implementación bloques 4PGA

6.2.7.Bloque ecogate

Este bloque implementa el acceso a todas las tramas CAN que publica la electrónica ecogate. Esta electrónica es la encargada de leer el voltaje de la batería, es por ello que se necesita acceder a la información que publica en el bus.

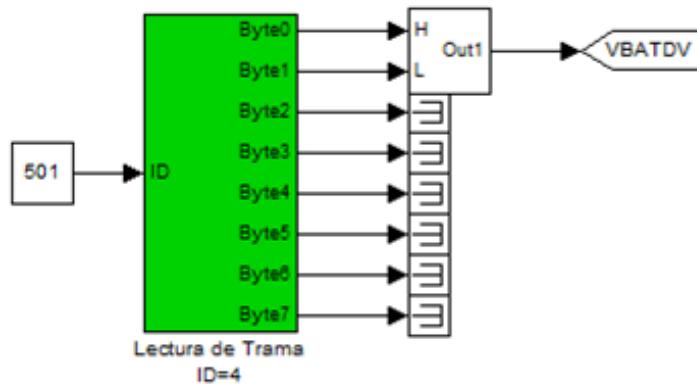


Figura 6.20: Implementación bloque ecogate

6.3. Obtención de la curva motor

Al igual que en muchas otras máquinas, el comportamiento del motor eléctrico puede ser especificado por medio de sus curvas características. Una correcta interpretación de estas curvas para un determinado motor puede brindar abundante información acerca de su comportamiento en distintas condiciones de funcionamiento. Con la potencia de frenado del rodillo elite y el control de la bicicleta mediante Simulink, disponemos de todas las herramientas necesarias para obtener la curva del motor de la bicicleta. En este subapartado se describe el procedimiento para hacerlo.

6.3.1. Entrega de par del freno eléctrico

Debido a las características del freno eléctrico, la entrega de par de este no se realiza de forma lineal. En el siguiente gráfico podemos observar como es el par realizado por el freno como respuesta a una consigna que aumenta de forma lineal. El gráfico se ha obtenido sin realizar control sobre la respuesta del freno con respecto la consigna, es decir, en lazo abierto.

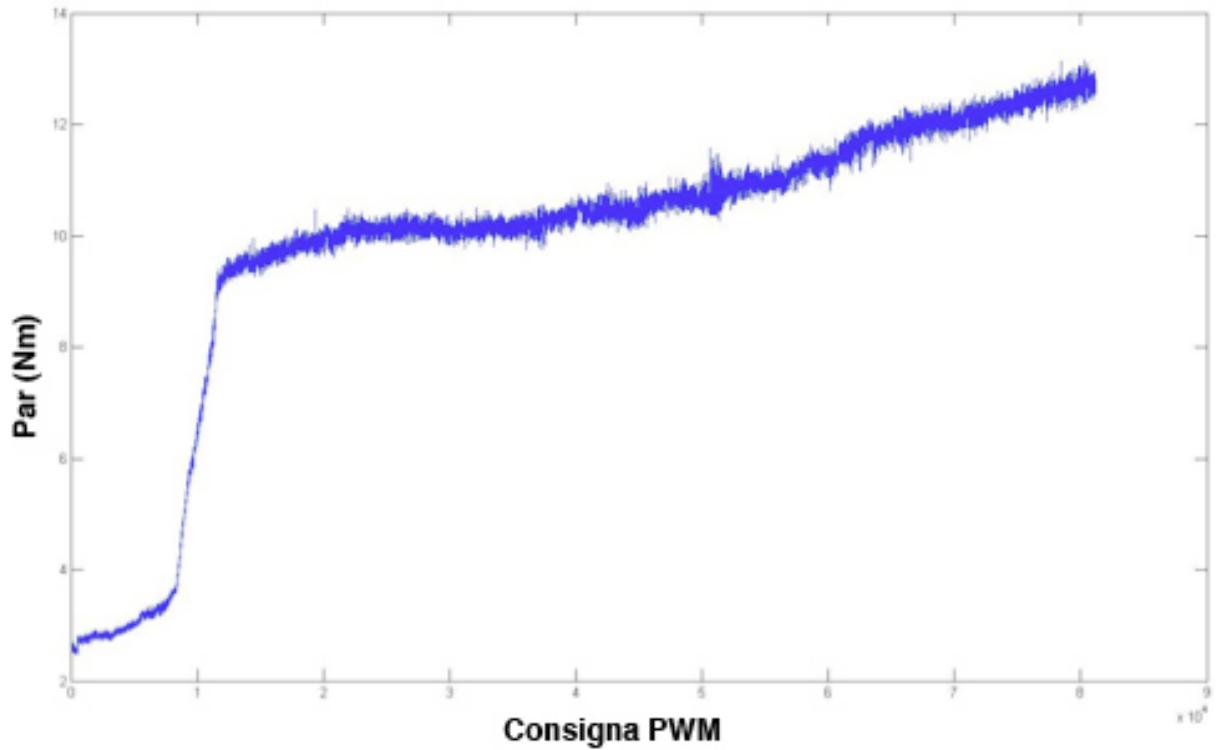


Figura 6.21: Entrga de par del freno eléctrico mediante consigna de PWM en lazo abierto

Podemos observar como el aumento de par es mayormente lineal, exceptuando la zona entre los $3.7\text{N}\cdot\text{m}$ hasta los $9\text{N}\cdot\text{m}$. Para mejorar la linealidad de la entrega de par, se puede ejercer control en lazo cerrado. El esquema de control sería el que se puede observar en la figura 6.22.

6. Modelo Simulink de una bicicleta eléctrica

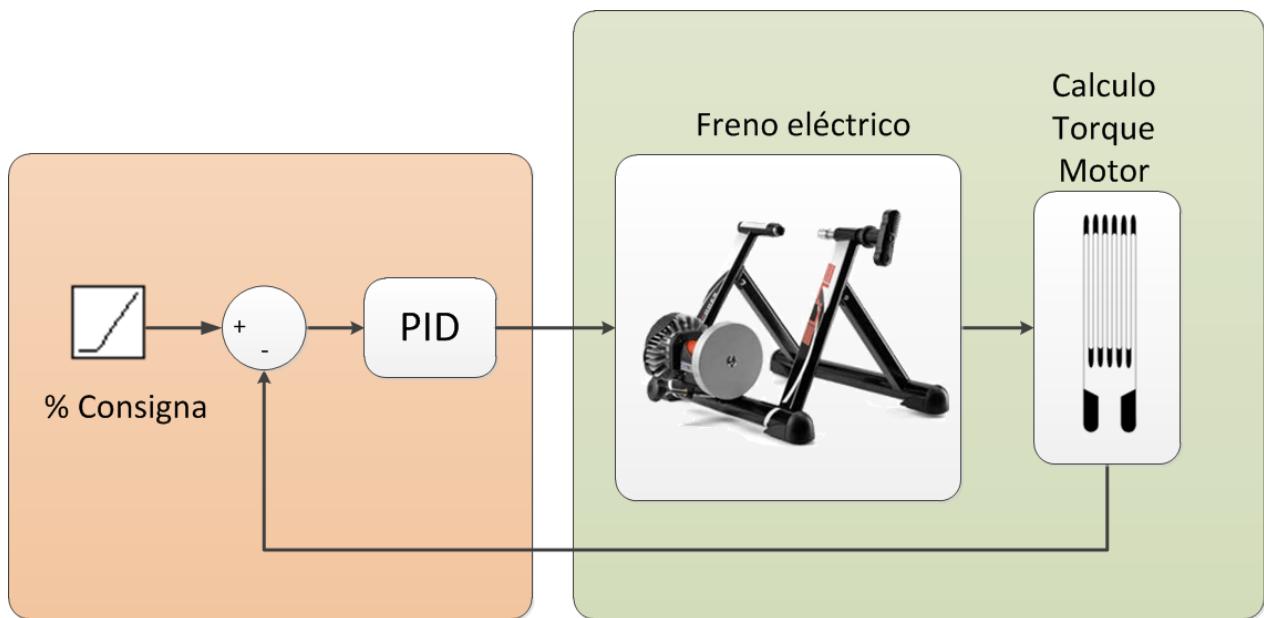


Figura 6.22: Control de par con lazo cerrado. En naranja la parte realizada en Simulink.

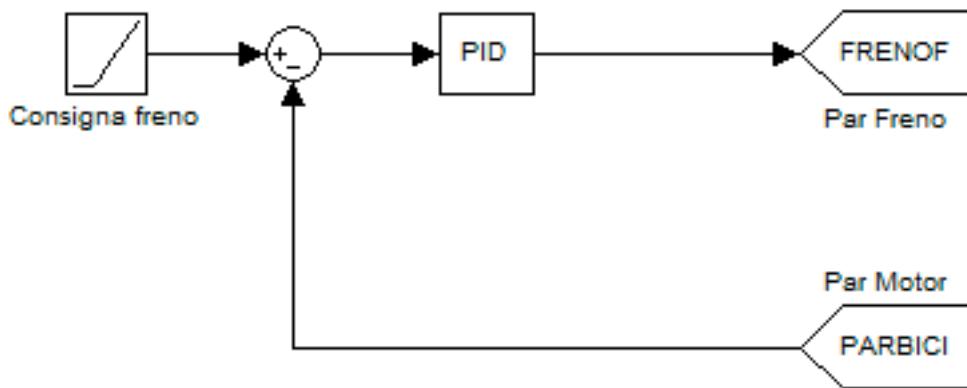


Figura 6.23: Implementación en Simulink del control de par con lazo cerrado

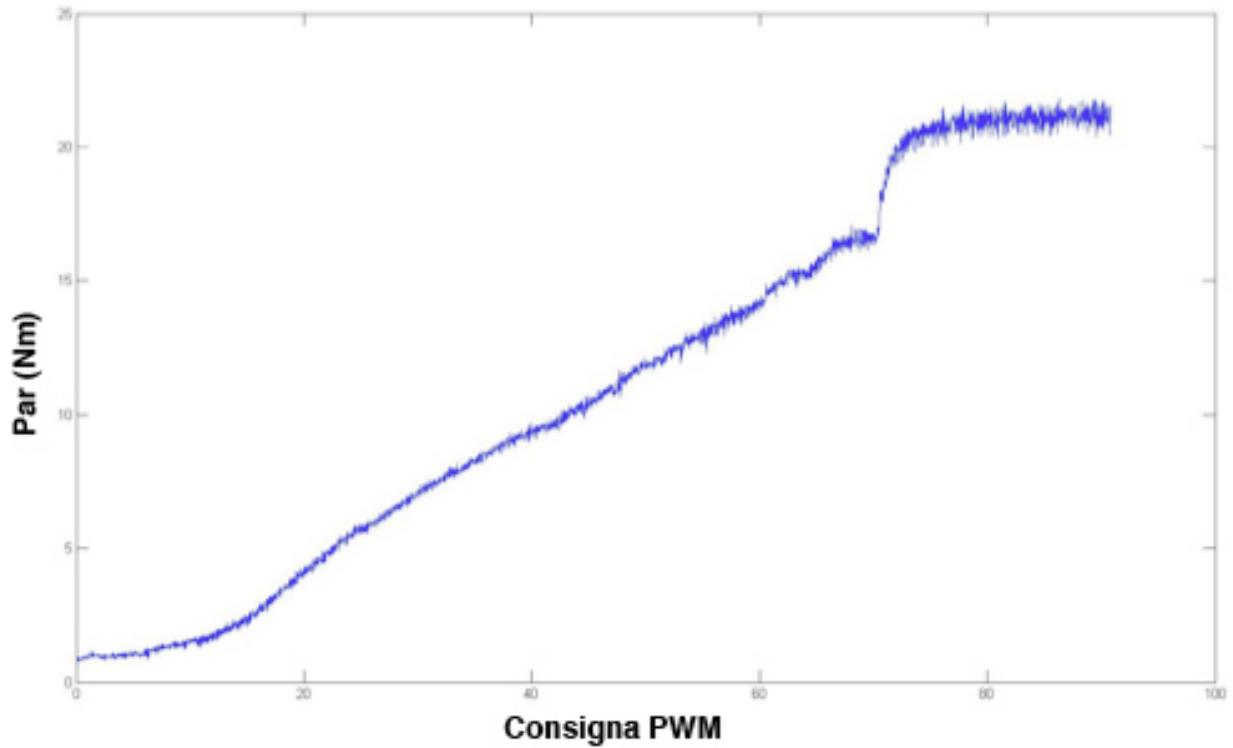


Figura 6.24: Par ejercido por el freno controlado por pid con lazo cerrado

Como se observa en la figura 6.24, el par realizado por el freno eléctrico, controlado mediante PID con lazo cerrado es mucho más lineal. Mediante una sintonización más fina del PID se podría llegar a conseguir una entrega de par un poco más lineal, pero dado a que para este par el motor de la bicicleta gira a muy pocas revoluciones, este grado de control es suficiente para el experimento que queremos realizar.

6.3.2.Curva motor

Una vez controlado el freno eléctrico, es interesante obtener la curva del motor que lleva incorporada la bicicleta

Obtener la curva motor nos permite aproximar el par realizado por el motor a partir de parámetros conocidos como son: la corriente consumida, el voltaje de la batería y las rpm del motor.

El proceso para obtener la curva es el siguiente: acelerar el motor al máximo de rpm con la mínima carga posible. Aplicar una rampa de incremento de par resistente hasta ahogar el motor (bajar sus rpm a prácticamente 0). Esta rampa debe tener un incremento muy lento en el tiempo (en nuestro caso el proceso dura 10 minutos aproximadamente).

Finalmente se obtienen los datos para poder dibujar la curva motor (figura 6.25).

6. Modelo Simulink de una bicicleta eléctrica

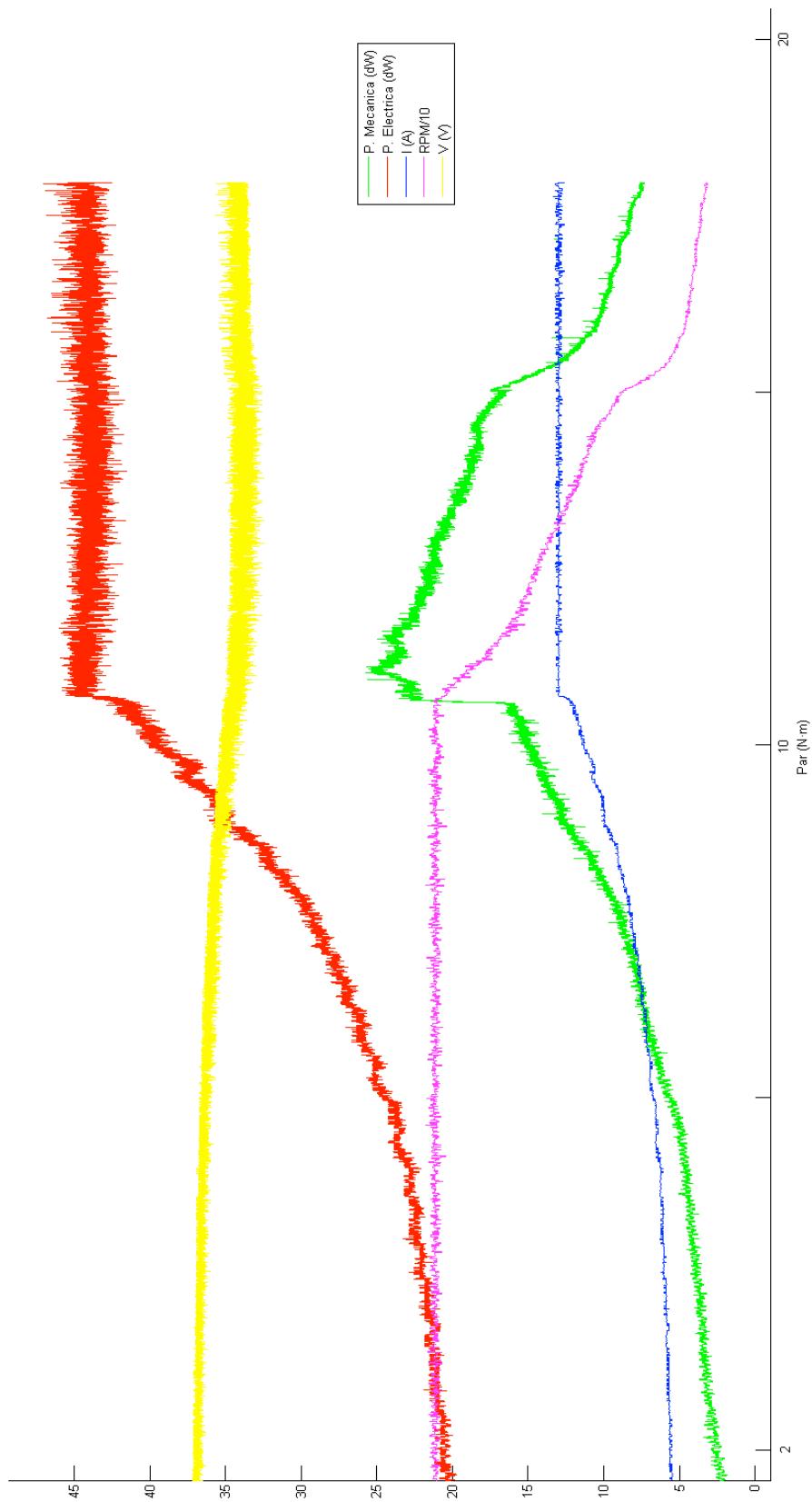


Figura 6.25: Curva del motor BLDC utilizado por ecobike

Capítulo 7

Estrategias de Control: Diseño y Selección

En este capítulo se explican las diferentes estrategias diseñadas para el control del motor de la bicicleta. También se describe de forma detallada la implementación de la estrategia de control elegida para la bicicleta. La variable que se pretende controlar, es la relación entre el trabajo que realiza el ciclista y el trabajo ejercido por el motor. El controlador debe comparar esta relación con la relación de trabajo deseada (consigna) y actuar en consecuencia sobre el motor de la bicicleta. Para testear las estrategias de control diseñadas se hará uso del modelo Simulink desarrollado y explicado en el capítulo 6. El esquema de control compartido entre el modelo Simulink y el controlador de la bicicleta puede observarse en la figura 7.1.

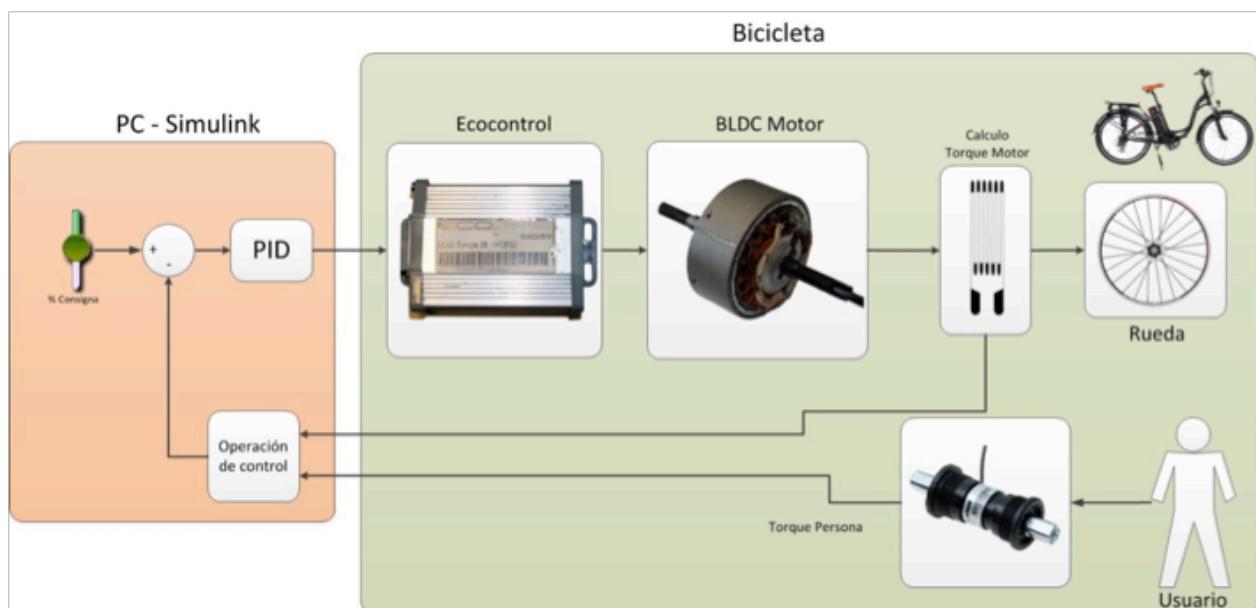


Figura 7.1: Esquema de control compartido

El controlador de la bicicleta ya ofrece un control mediante PID de la velocidad del motor y de la corriente máxima consumida. El doble lazo encadenado de control de velocidad y corriente permite establecer una consigna de velocidad a la que se puede limitar el consumo de corriente a voluntad. El esquema de este doble lazo de control es el siguiente:

7. Estrategias de control: Diseño y selección

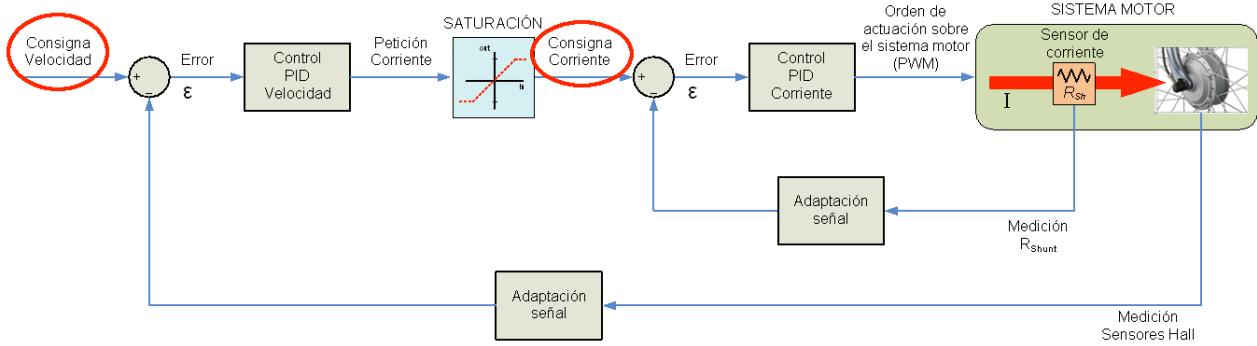


Figura 7.2: Doble lazo de control de velocidad y corriente. Marcadas en rojo las consignas sobre las que se actúa.

El objetivo final de la estrategia de control es que la relación de trabajo, el *business* entre la bicicleta y el usuario, se mantenga constante, independientemente de la ruta seguida. Este concepto queda expresado gráficamente en el siguiente esquema:

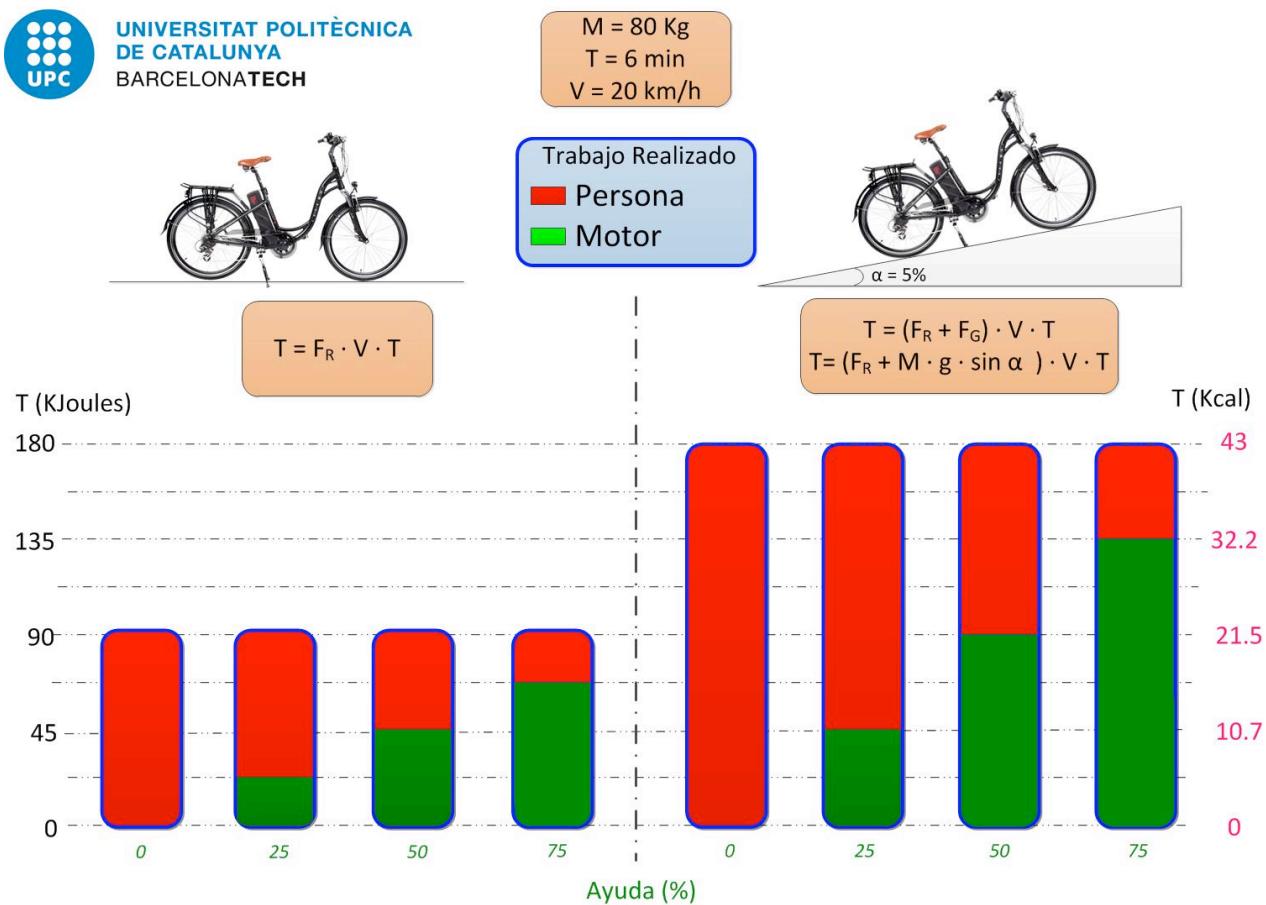


Figura 7.3: Proporciones de trabajo realizados por la bicicleta y el usuario según el nivel de ayuda del motor.

El esquema está dividido en dos casuísticas, el comportamiento de la bicicleta en llano y el comportamiento en subida. El trabajo realizado por el conjunto motor ciclista en el caso de la

conducción en llano viene determinado por la velocidad, el rozamiento que frena la bicicleta y el tiempo que dura el experimento. En el caso de conducción en subida, hay que añadir además, la gravedad como fuerza que se opone al movimiento, con una magnitud dependiente de la pendiente que se sube. Se puede apreciar en el diagrama como el trabajo realizado cuando se supera un desnivel es mayor que cuando se circula en llano, pero la proporción de esfuerzo aportado por el ciclista con respecto al aportado por el motor se mantiene.

7.1.Estrategias diseñadas

Se han diseñado tres estrategias para el control del motor. En los siguientes subapartados se especifican los conceptos básicos de cada estrategia. Para todas las estrategias diseñadas, dado que el pedalier solo da el torque realizado por la pierna izquierda, se supone que el torque realizado por la pierna derecha es igual con un desfase de 180° .

7.1.1.Estrategia 1

La estrategia queda definida según el siguiente esquema:

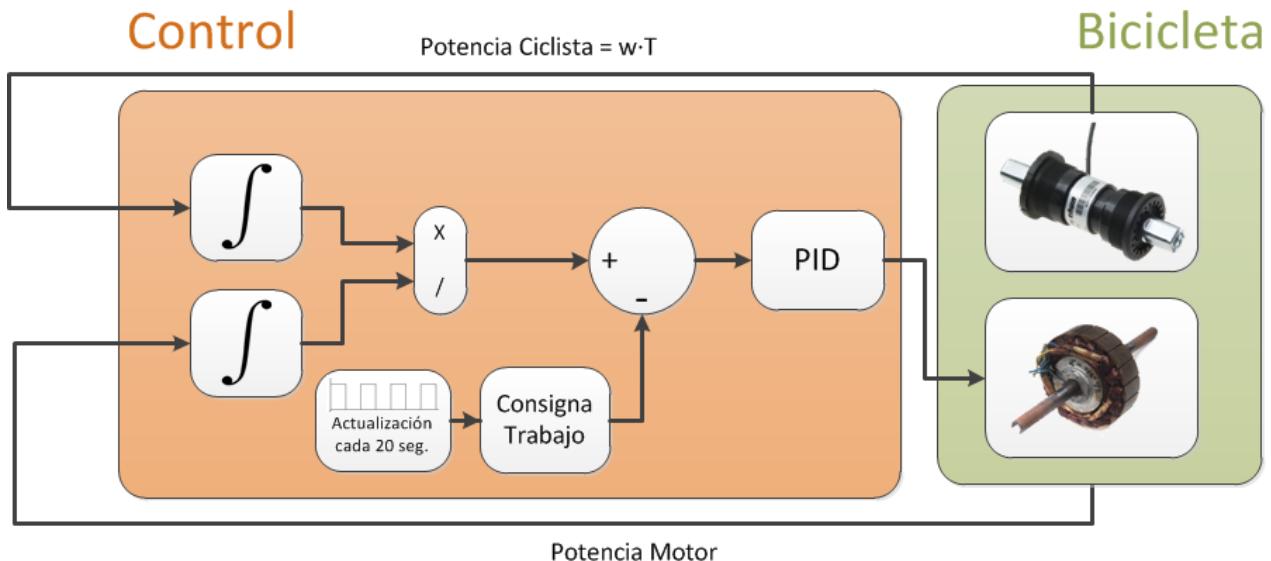


Figura 7.4: Esquema de la estrategia número 1

En esta estrategia se calcula el trabajo realizado por el ciclista integrando el valor de la potencia instantánea obtenida del pedalier Thun. Posteriormente se calcula la relación existente entre el trabajo realizado por el ciclista y el trabajo desarrollado por el motor en un periodo de tiempo concreto (por ejemplo 20 segundos). Según el resultado de la relación trabajo ciclista - trabajo motor, se ajusta la consigna de trabajo para asegurar que la relación deseada se cumple. La principal ventaja de esta estrategia es la precisión que muestra cumpliendo el objetivo de trabajo fijado. Su principal inconveniente es que puede ajustar el nivel de ayuda a la baja en un momento

en el que sea necesario, con la consiguiente molestia para el usuario. Por ejemplo, si durante un periodo concreto, el motor ha realizado más trabajo del que le corresponde, en el siguiente periodo ajustará la consigna para que el ciclista trabaje más. Este ajuste puede coincidir con una subida, dando la sensación al usuario de que no recibe la ayuda que le correspondería.

7.1.2. Estrategia 2

La estrategia queda definida según el siguiente esquema:

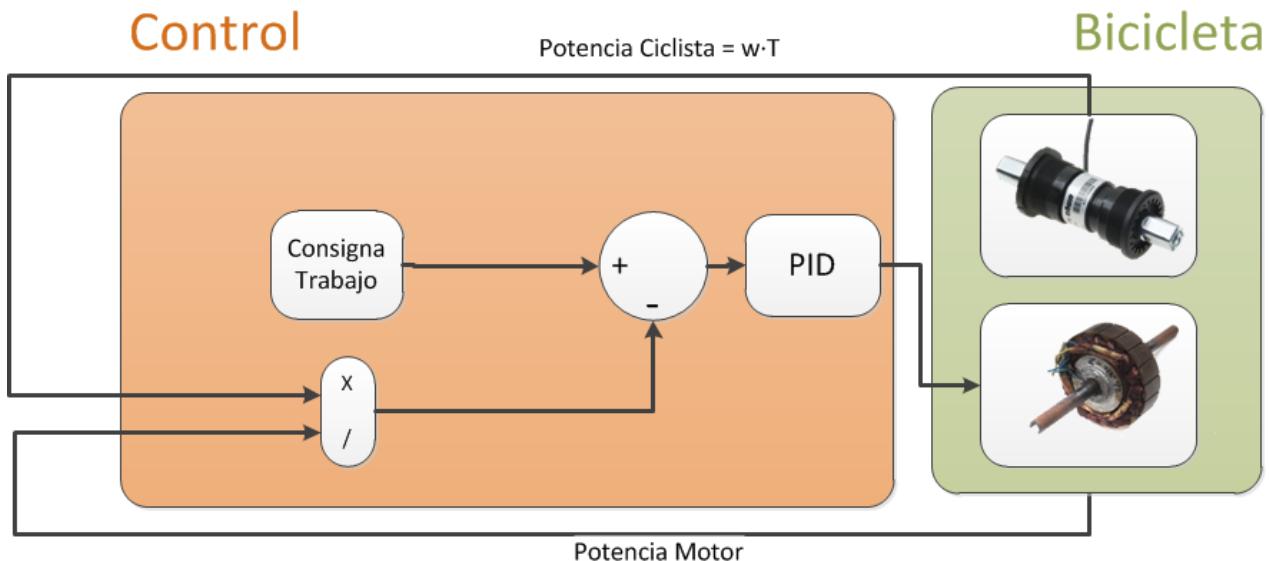


Figura 7.5: Esquema de la estrategia número 2

En esta estrategia se ajusta constantemente el nivel de ayuda que proporciona el motor en relación a la potencia con que pedalea el usuario. El mayor inconveniente de esta estrategia es que el par realizado por el ciclista sobre los pedales no es constante, y en consecuencia, la ayuda que brinda el motor tampoco lo es, dando sensación de “tirones”.

7.1.3.Estrategia 3

La estrategia queda definida según el siguiente esquema:

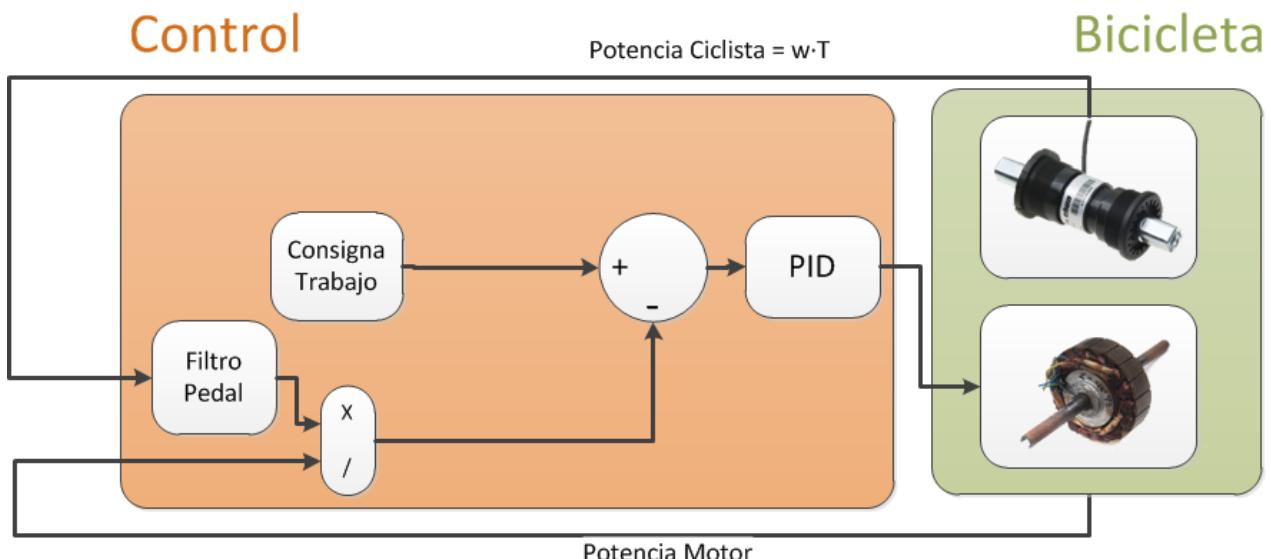


Figura 7.6: Esquema de la estrategia número 3

La estrategia número 3 es, en esencia, igual a la estrategia número 2, con la salvedad que la potencia realizada por el ciclista es filtrada para evitar la sensación de “tirones”. El mayor inconveniente de esta estrategia es la pérdida de precisión debida al filtro de pedaleo, que introduce error en el objetivo de relación entre el trabajo del ciclista y el trabajo realizado por el motor.

7.2.Estrategia seleccionada

Después de valorar las diferentes estrategias diseñadas, se ha optado por implementar la estrategia número 3. Se ha seleccionado esta estrategia por la inmediatez con la que trata la consigna del motor. No se puede olvidar que este PFC tiene un objetivo comercial, y la sensación que produce al ciclista la respuesta de la bicicleta es importante. A falta de un estudio estadístico entre usuarios de bicicletas eléctricas, la estrategia 3 es, la que a mi parecer, resulta más agradable cuando se pedalea.

7.3.Implementación en Simulink

Partiendo del modelo Simulink que se conecta vía can a la bicicleta, se ha implementado la estrategia de control planteada. Emular el trabajo que debe realizar el controlador con el modelo Simulink permite desarrollar un prototipo de forma gráfica y relativamente rápida para poder testear y debuggar la estrategia de control diseñada.

7. Estrategias de control: Diseño y selección

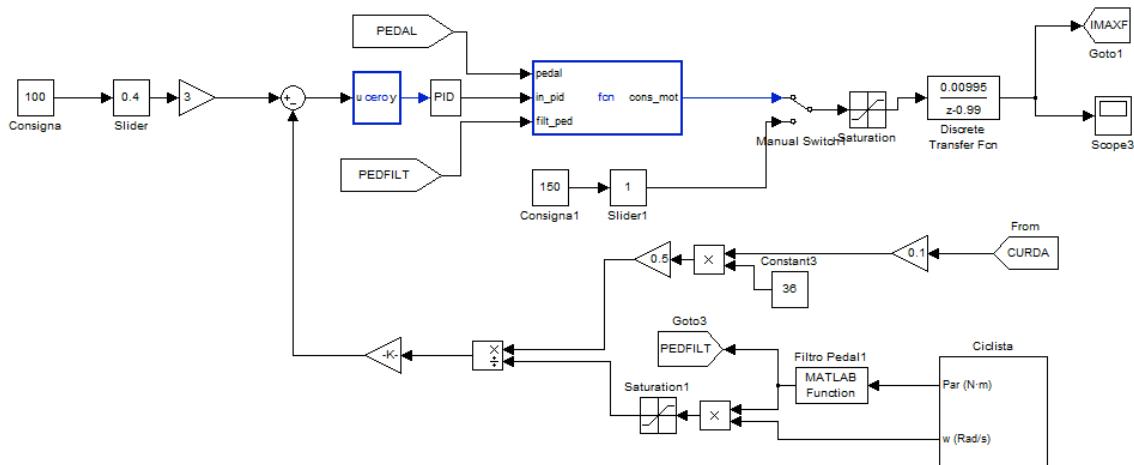


Figura 7.7: Implementación del modelo de control en Simulink

La fuerza ejercida por el ciclista sobre los pedales no es continua, y de hecho varía en cada fase de la pedalada. Es por ello que debe ser filtrada, para que la consigna de control que se envía al motor no se adapte continuamente a la fase de pedaleo en la que se encuentra el ciclista pues resultaría muy incomoda la conducción de la bicicleta. El filtro que se ha implementado en el lenguaje m de Matlab tiene el siguiente código:

```
function [ out ] = filtroPedal( pedal )

global estado;
global potPedal;
global ultimoPotPedal;

vsubida=10;
vbajada=6;

if (estado==0) %detectamos flancos de subida
    if (pedal > vsubida) %comenzamos una subida
        estado=1;
    end
elseif (estado==1) %detectamos flancos de bajada
    if (pedal < vbajada) %acabamos una bajada
        estado=0;
        ultimoPotPedal=potPedal;
        potPedal=0;
    end
end

if (estado==1)
    potPedal=potPedal+(pedal*0.01); %integramos el valor
end
out=ultimoPotPedal;
end
```

El resultado de filtrar el par realizado por el ciclista puede observarse en el siguiente gráfico:

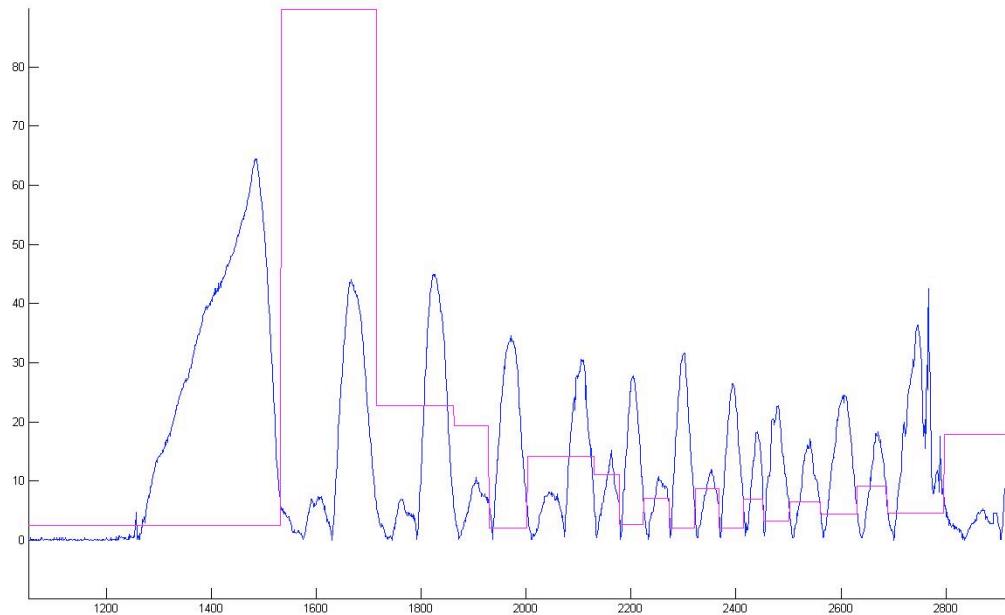


Figura 7.8: Resultado del filtrado del par realizado por el ciclista. El par filtrado es el resultado de contar con las dos piernas.

Se ha decidido en favor de este filtrado porque no introduce una pérdida en el trabajo realizado por el ciclista, cosa que haría errar la estrategia de control.

7.4. Implementación de la estrategia en el controlador de la bicicleta

Una vez testeada la estrategia de control sobre el modelo Simulink, ésta, simplemente se debe programar en el lenguaje C30 para permitir que el controlador de la bicicleta la ejecute. Un extracto de las partes más importantes del código se muestran a continuación.

Tratamiento del pedaleo del ciclista

```
void tratamientoPedalPar(void){
    float relacionPedalMotor=0.0;
    //calibramos el par
    torqueCiclista=torque-2500;

    if (torqueCiclista>0) torqueCiclista=0;
    torqueCiclista=-torqueCiclista;
    filtroPedal();

    potenciaCiclista=torqueFiltrado*((float)rpm_pedal*2*3.14/60.0);
    potenciaMotor=(current_mA/1000)*36.0*((float)rendimiento)/100.0;

    if (potenciaCiclista>0) {
        relacionPedalMotor=(potenciaMotor/potenciaCiclista)*100.0; //lo expresamos
        en tanto por ciento
        accPID=calcPID((3*consigna)-relacionPedalMotor);
        //saturamos el pid
    }
}
```

7. Estrategias de control: Diseño y selección

```
        if (accPID<0) accPID=1;
        if (accPID>150) accPID=150;
    }

    if (pedaleo.estado==PEDALEO_EN_MARCHA && pedaleo.sentido==FW && Simulink==0){
        if (consigna_frenos==0){
            scaBikeSpeed=25.0;
            scaCurrentLimit=((float)accPID)/10.0; //le damos la corriente
calculada por el PID
        } else { //sin ayuda
            scaBikeSpeed=0.0;
            scaCurrentLimit=0.0;
        }
    } else { //sin ayuda
        if (Simulink==0){ //modelo no conectado
            scaBikeSpeed=0.0;
            scaCurrentLimit=0.0;
        }
    }
}
```

Filtrado del par

```
void filtroPedal(void) {

    int vsubida=10;
    int vbajada=6;

    int auxpedal = torqueCiclista*0.1;

    if (auxestado==0){ //detectamos flancos de subida
        if (auxpedal > vsubida){ //comenzamos una subida
            auxestado=1;
        }
    }
    else if (auxestado==1){ //detectamos flancos de bajada
        if (auxpedal < vbajada){ //acabamos una bajada
            auxestado=0;
            torqueFiltrado=potPedal;
            potPedal=0;
        }
    }

    if (auxestado==1){
        potPedal=potPedal+(auxpedal*0.001); //integramos el valor de par
instantaneo
    }
}
```

7.5.Evaluación de la estrategia seleccionada

Una vez seleccionada e implementada la estrategia de control, llega el momento de realizar las pruebas para comprobar su funcionamiento. Se han dividido los tests en dos partes: los realizados sobre el banco de pruebas y los que se han sido hechos circulando por la calle.

7.5.1.Evaluación en el banco de pruebas

Para evaluar la implementación de la estrategia de control, se ha utilizado el banco de pruebas desarrollado y del modelo Simulink para la emulación del control de la bicicleta. Gracias a la galga incorporada en el rodillo se puede saber el trabajo realizado por el conjunto ciclista-motor.

Integrando la potencia del ciclista (obtenida gracias al pedalier Thun) se obtiene el trabajo realizado por éste. La diferencia entre el trabajo calculado en la galga del rodillo y el aportado por el ciclista es el trabajo realizado por el motor. Conocido el trabajo del ciclista y del motor se puede saber la relación que existe entre ellos y evaluar la estrategia de control.

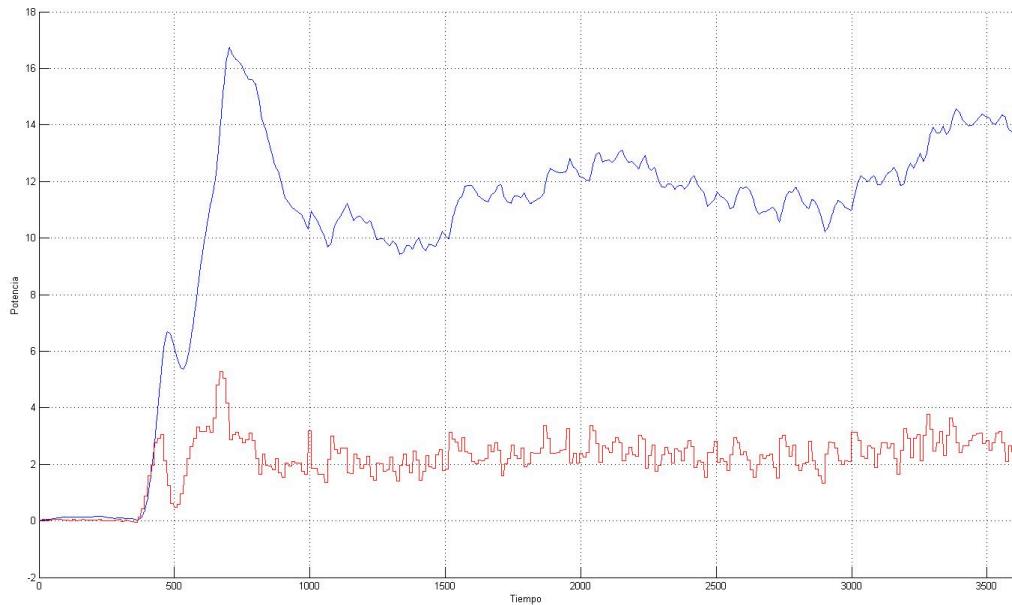


Figura 7.9: Potencia realizada por el ciclista (rojo) y potencia realizada por el motor (azul)

Realizando la integral de la potencia se obtiene el trabajo. La consigna para el test de la figura 7.9 era del 20% de trabajo realizado por el ciclista y un 80% de trabajo realizado por el motor. El resultado de la estrategia de control es de un 19.45% de trabajo realizado por el ciclista y un 80.55% de trabajo realizado por el motor de la bicicleta.

7.5.2. Evaluación en el exterior

El test ha consistido en realizar un recorrido por la calle de seis minutos aproximadamente de duración. Se han dado tres vueltas al siguiente circuito:

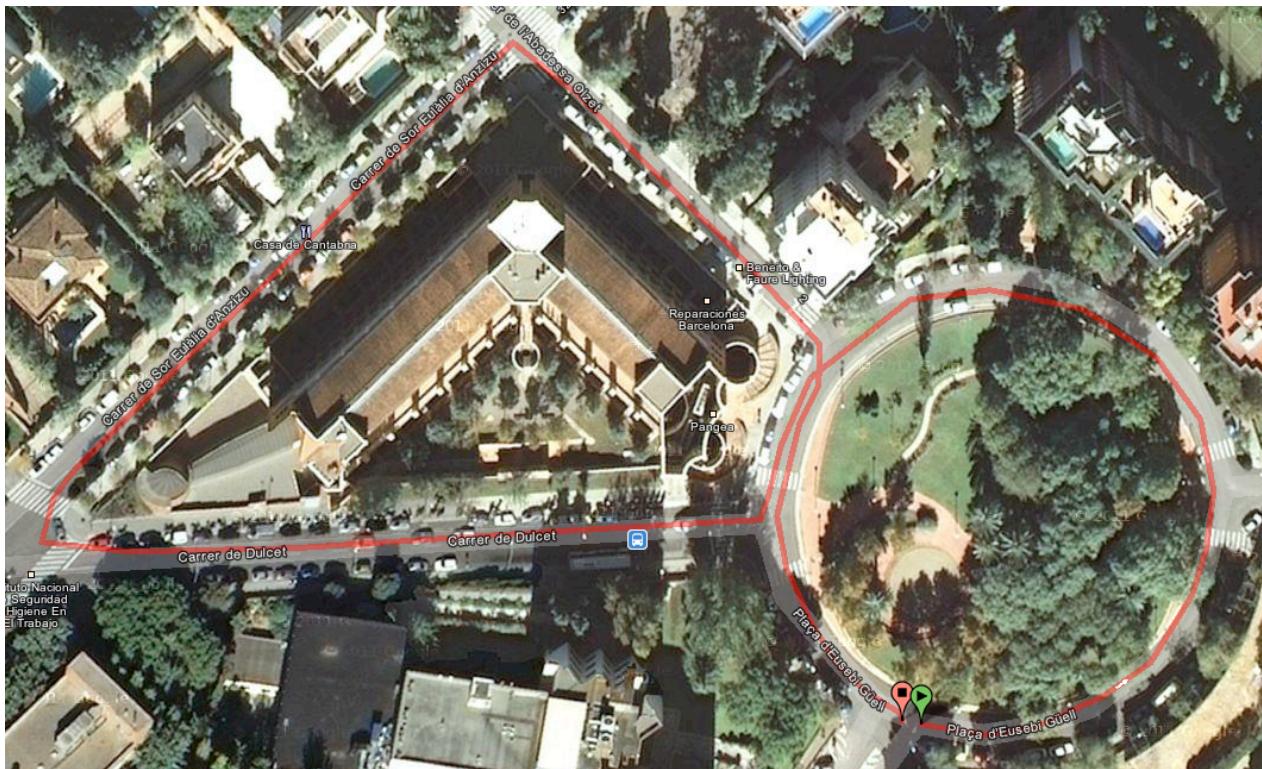


Figura 7.10: Circuito utilizado en los tests en exterior. Punto de inicio en verde y punto final en naranja

Se ha repetido el recorrido tres veces, una para una ayuda del motor del 20%, otra para el 50% y por último para una ayuda del 80%. Los resultados obtenidos se muestran en la siguiente tabla:

Consigna Ayuda	Tiempo Recorrido	Ayuda Obtenida
20%	5m 54s	19,30%
50%	6m 3s	47,74%
80%	5m 45s	82,18%

Para obtener los resultados se ha integrado la potencia realizada por el ciclista y se ha multiplicado por dos para contar también con el trabajo de la pierna derecha. Se ha integrado la potencia mecánica estimada realizada por el motor. Se ha obtenido la relación entre el trabajo realizado por el ciclista y el trabajo realizado por el motor. La obtención de estos datos se ha conseguido gracias al uso de la aplicación Android desarrollada. En conclusión se puede observar que la estrategia de control funciona con un error aceptable, teniendo en cuenta que el circuito cuenta con subida, llano y bajada.

Capítulo 8

Monitorización

Uno de los puntos marcados en los objetivos de este proyecto es crear una aplicación para poder visualizar los datos de la bicicleta eléctrica sobre teléfonos móviles de última generación, más concretamente sobre aquellos que funcionan sobre la plataforma Android de Google. Android es un sistema operativo móvil basado en Linux, que está enfocado para ser utilizado en dispositivos móviles como teléfonos inteligentes, tabletas, Google TV y otros dispositivos.

Fue desarrollado inicialmente por Android Inc., una firma comprada por Google en 2005. Es el principal producto de la Open Handset Alliance, un conglomerado de fabricantes y desarrolladores de hardware, software y operadores de servicio. Las unidades vendidas de teléfonos inteligentes con Android se ubican en el primer puesto en los Estados Unidos, en el segundo y tercer trimestres de 2010, con una cuota de mercado de 43,6% en el tercer trimestre. A nivel mundial alcanzó una cuota de mercado del 50,9% durante el cuarto trimestre de 2011, más del doble que el segundo sistema operativo (iOS de iPhone) con más cuota.

Tiene una gran comunidad de desarrolladores escribiendo aplicaciones para extender la funcionalidad de los dispositivos. A la fecha, se han sobrepasado las 400.000 aplicaciones (de las cuales, dos tercios son gratuitas) disponibles para la tienda de aplicaciones oficial de Android: Google Play, sin tener en cuenta aplicaciones de otras tiendas no oficiales para Android, como pueden ser la App Store de Amazon o la tienda de aplicaciones Samsung Apps de Samsung. Los programas están escritos en el lenguaje de programación Java.

La estructura del sistema operativo Android se compone de aplicaciones que se ejecutan en un framework Java de aplicaciones orientadas a objetos sobre el núcleo de las bibliotecas de Java en una máquina virtual Dalvik con compilación en tiempo de ejecución. Las bibliotecas escritas en lenguaje C incluyen un administrador de interfaz gráfica (surface manager), un framework OpenCore, una base de datos relacional SQLite, una Interfaz de programación de API gráfica OpenGL ES 2.0 3D, un motor de renderizado WebKit, un motor gráfico SGL, SSL y una biblioteca estándar de C Bionic.

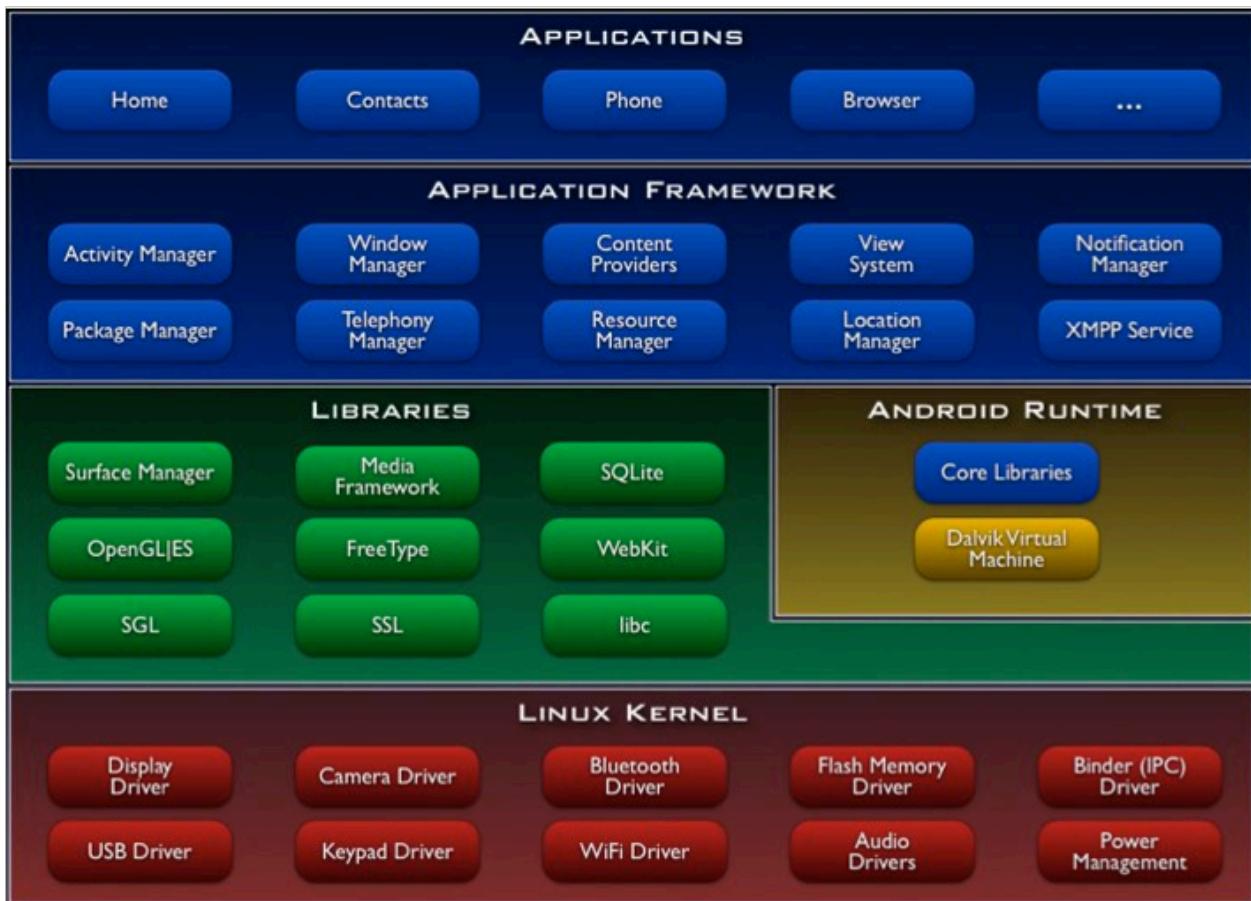


Figura 8.1: Arquitectura de un sistema Android

8.1. Bluetooth

La tecnología utilizada para la transmisión de datos entre la bicicleta y el teléfono móvil es el Bluetooth. Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz. Los principales objetivos que se pretenden conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles y fijos.
- Eliminar cables y conectores entre éstos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

8.1.1.Electronica ecoGate

Se utiliza una electronica diseñada en el departamento de ESAII para hacer uso de la comunicación via bluetooth. Esta elecctónica, de nobre ecoGate, contiene un microcontrolador dsPIC30F4011, conectado al bus CAN y un modulo bluetooth bluegiga WT12. El diagrama de bloques que define el funcionamiento de la electrónica se puede observar en la figura 8.2.

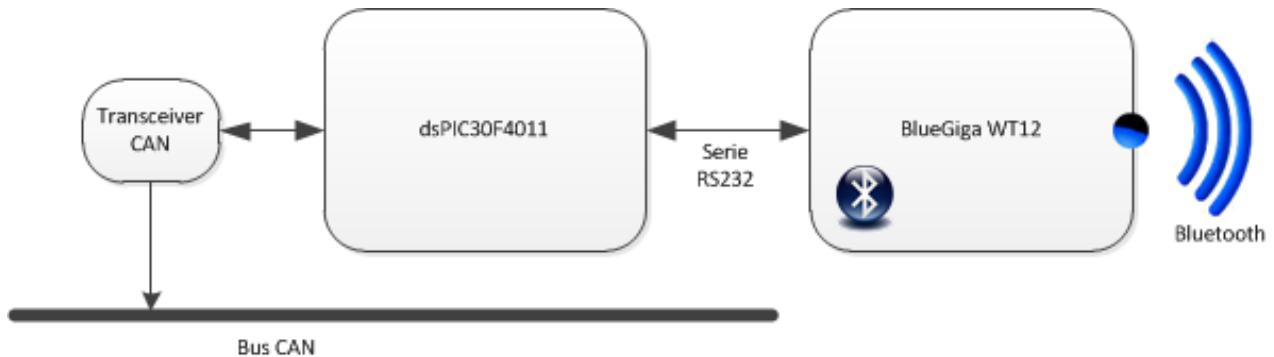


Figura 8.2: Diagrama de bloques de la electrónica ecogate

El microcontrolador captura los datos que viajan por el bus can y actúa como servidor de modbus. Los comandos modbus le llegan procedentes del modulo bluegiga a través de la conexión serie que comparten. El modulo bluegiga nos abstrae una conexión serie sobre bluetooth para poder conectarnos mediante diferentes dispositivos y obtener los datos que viajan por el bus CAN a través de modbus.

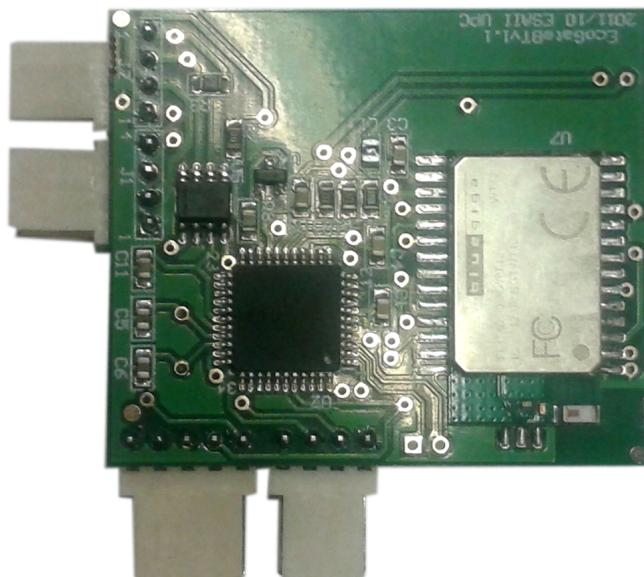


Figura 8.3: Electrónica ecoGate

8.1.2. Modbus serie

El Modbus es un protocolo de estructura de mensajes utilizado para establecer comunicaciones de tipo cliente/servidor entre dispositivos. Gracias a su formato, los dispositivos pueden estar conectados y funcionar de diferentes maneras.

El protocolo Modbus fue creado por la empresa Modicon (hoy Schneider Electric) el año 1979. Modbus es una marca registrada por la empresa Schneider Electric, que se encargó de especificar el protocolo, implementarlo y mantenerlo. Originalmente, se creó para transferir datos de control entre controladores y sensores utilizando el puerto serie RS232. Rápidamente se fue extendiendo y actualmente centenares de fabricantes lo implementan en sus productos con tal de transferir datos, siendo un protocolo ampliamente soportado.

Modbus ha acabado convirtiéndose en un estándar poco a poco, y se utiliza en múltiples aplicaciones cliente/servidor, para monitorizar y programar dispositivos, para comunicar dispositivos con sensores e instrumentos, etc. Modbus también es un protocolo ideal para las aplicaciones RTU (Remote Terminal Unit) donde se necesita comunicación wireless.

El protocolo Modbus funciona transmitiendo un mensaje simple que se envía independientemente del medio con el que se transmite, ya que es un protocolo que trabaja a nivel de la capa de aplicación del modelo OSI.

El mensaje consta de un código de función y de sus datos.

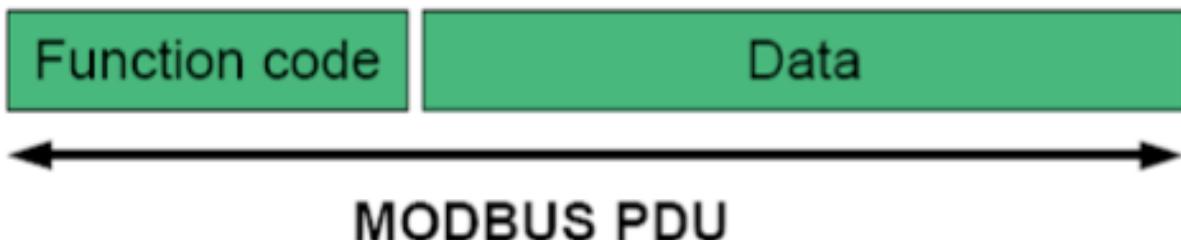


Figura 8.4: Estructura del PDU modbus

Las funciones que se envían son básicamente leer y escribir datos (bits o registros de 16 bits). También se pueden consultar errores. Los datos que se leen y se escriben, dependen del dispositivo del que se trata. Es decir, una misma dirección puede tener un significado diferente en dos dispositivos diferentes.

El Modbus se puede transmitir por diferentes medios. El mensaje Modbus, que corresponde a la capa de aplicación, se encapsula según el medio de transmisión utilizado. En el siguiente gráfico vemos los diferentes tipos de transmisión.

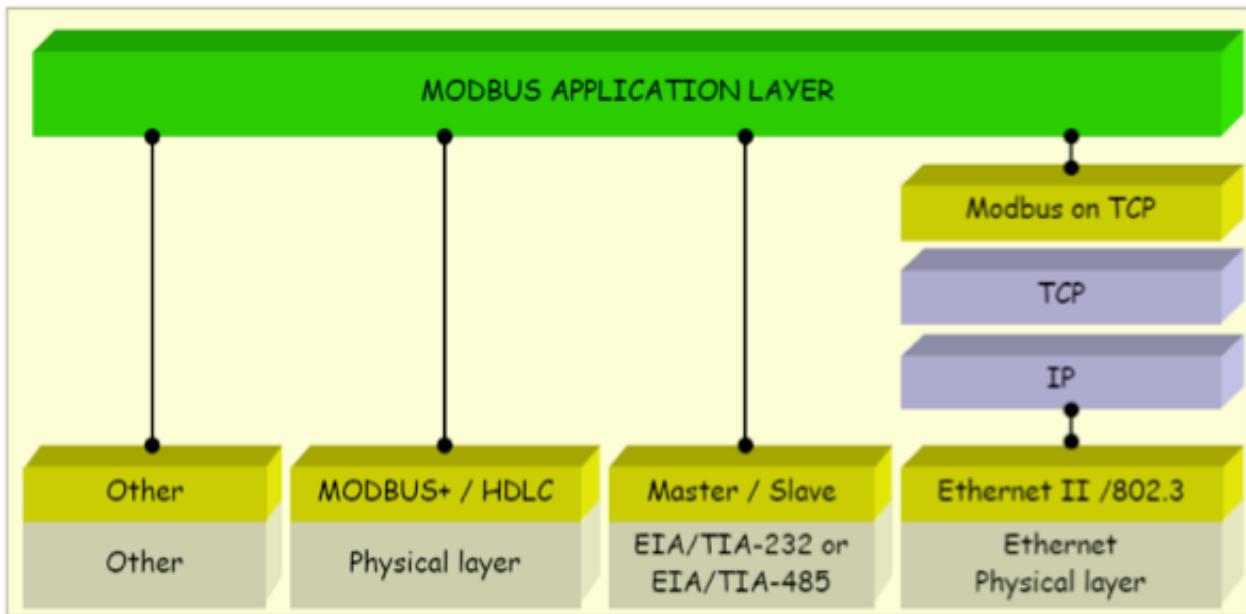


Figura 8.5: Diferentes tipos transmisión de un paquete Modbus

Los dos tipos de transmisión de datos más utilizados son el Modbus RTU, que utiliza el puerto serie i el Modbus TCP/IP.

El protocolo Modbus sobre el puerto serie tiene dos métodos de transmisión: el Modbus ASCII y el Modbus RTU. El más utilizado es el RTU (Remote Terminal Unit).

La medida máxima de los paquetes que se envían es de 256 bytes, distribuidos de la siguiente manera:

- 1 byte para el campo de dirección.
- 253 bytes para el MODBUS PDU.
- 2 bytes para el control de errores (CRC).

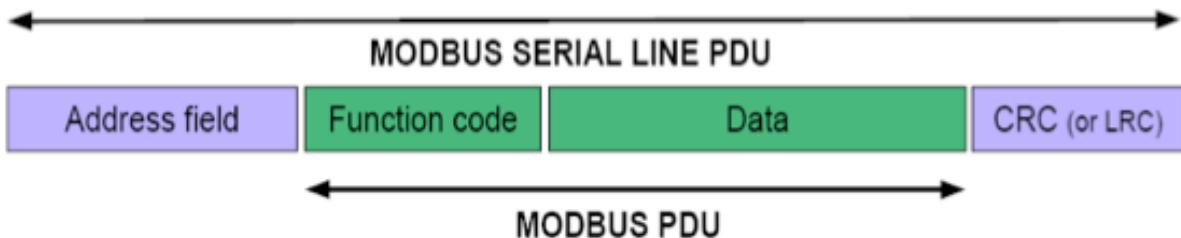


Figura 8.6: Campos de un paquete Modbus

En la figura 8.7 puede verse un diagrama de estados de las transacciones Modbus, en el que se tratan las funciones que llegan.

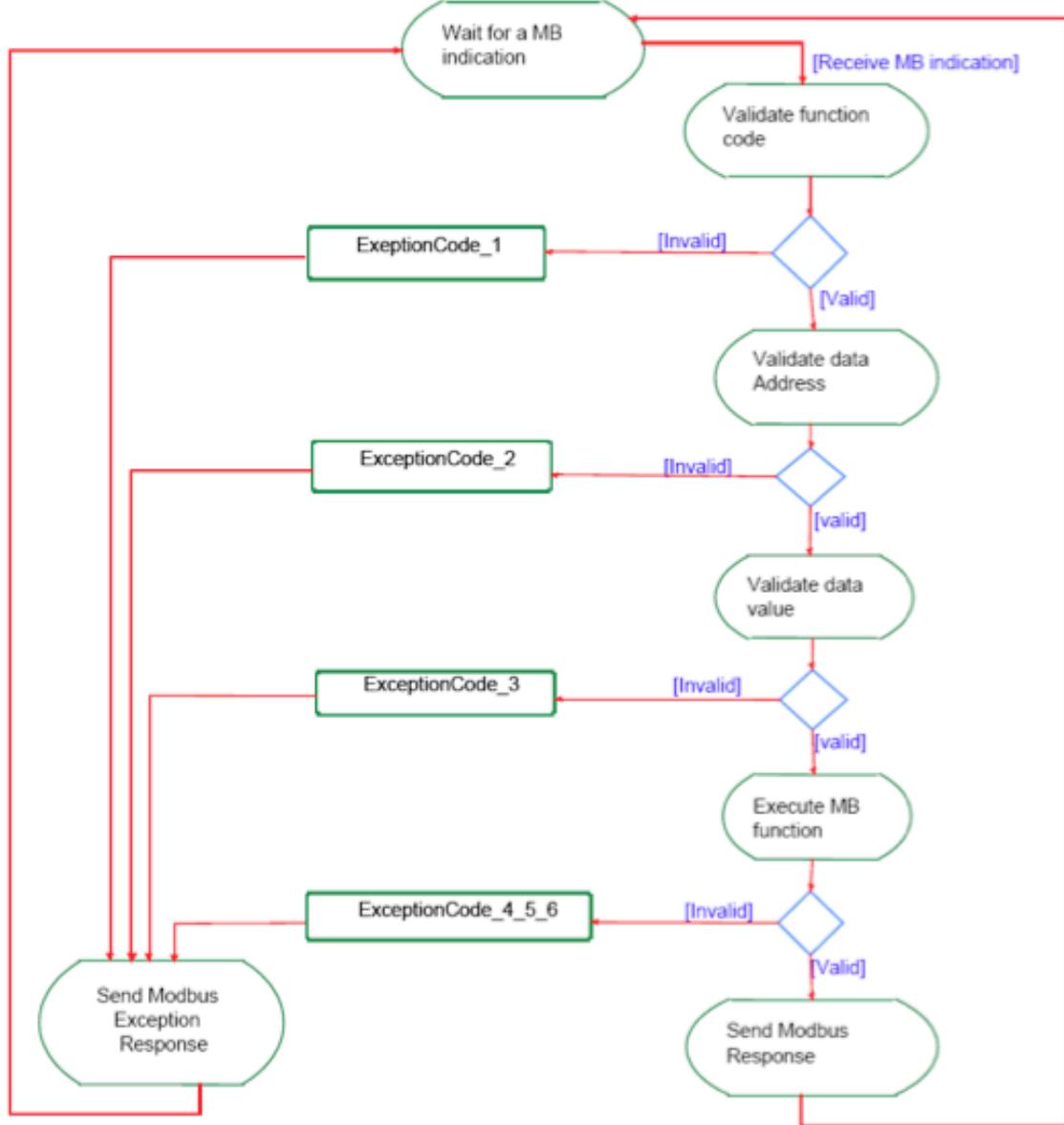


Figura 8.7: Diagrama de estados de las transacciones Modbus

Las funciones modbus implementadas son las siguientes:

Read Holding Registers (codigo de función: 03)

Esta función se utiliza para leer el contenido de un bloque de registros consecutivos de un dispositivo remoto. Permite leer hasta 125 registros. Los datos se empaquetan utilizando 2 bytes por registro.

Request

Function code	1 Byte	0x03
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	1 to 125 (0x7D)

Response

Function code	1 Byte	0x03
Byte count	1 Byte	2 x N*
Register value	N* x 2 Bytes	

*N = Quantity of Registers

Error

Error code	1 Byte	0x83
Exception code	1 Byte	01 or 02 or 03 or 04

Figura 8.8: Datos de los paquetes read/response/error para función código 03 de Modbus

Write Single Register (código de función: 06)

Esta función permite escribir el valor de un único registro en un dispositivo remoto. El request PDU especifica la dirección del registro que debe ser escrito. Los registros son direccionados a partir del 0.

La respuesta normal de esta función es un echo de la request, añadiendo al final de la PDU el valor del registro escrito.

Request

Function code	1 Byte	0x06
Register Address	2 Bytes	0x0000 to 0xFFFF
Register Value	2 Bytes	0x0000 to 0xFFFF

Response

Function code	1 Byte	0x06
Register Address	2 Bytes	0x0000 to 0xFFFF
Register Value	2 Bytes	0x0000 to 0xFFFF

Error

Error code	1 Byte	0x86
Exception code	1 Byte	01 or 02 or 03 or 04

Figura 8.9: Datos de los paquetes read/response/error para función código 06 de Modbus

Write Multiple registers (código de función: 16)

8. Monitorización

Esta función es utilizada para escribir un bloque contiguo de registros (hasta 123) en un dispositivo remoto. Los valores a ser escritos están especificados en el request. Los datos se empaquetan utilizando 2 bytes por registro.

La respuesta normal de esta función, retorna el function code, la dirección de inicio y la cantidad de registros leídos.

Request

Function code	1 Byte	0x10
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	0x0001 to 0x007B
Byte Count	1 Byte	2 x N*
Registers Value	N* x 2 Bytes	value

*N = Quantity of Registers

Response

Function code	1 Byte	0x10
Starting Address	2 Bytes	0x0000 to 0xFFFF
Quantity of Registers	2 Bytes	1 to 123 (0x7B)

Error

Error code	1 Byte	0x90
Exception code	1 Byte	01 or 02 or 03 or 04

Figura 8.10: Datos de los paquetes read/response/error para función código 16 de Modbus

El mapa de memoria implementado es el siguiente:

Dirección	R/W	Variable	Unidades	Rango
0	R	Motor Speed	rpm/10	0...2000
1	R	Pedal Speed	rpm/10	-1200...1200
2	R	Bike Speed	(Km/h)/10	0...600
3	R	Motor Temp	d°C	-10...40
4	R	External Temp	d°C	Sin usar
5	R	Battery Voltage	dV	24...41
6	R	Battery Current	dA	'0...15
7	R/W	Control Mode		0,1,2,3
8	R	Accelerator	dV	'0...5

Dirección	R/W	Variable	Unidades	Rango
9	R	Left Brake		0..1
10	R	Right Brake		0..1
11	R/W	Dist. Traveled	Dm	0...2^16-1
12	R	Pedal Torque	dNm	-200...200
13	R	Motor Torque	dNm	0...25

8.1.3.Modbus sobre java

Como se ha mencionado con anterioridad, las aplicaciones para Android, están desarrolladas en su mayoría utilizando el lenguaje de programación java. Se necesita pues, una forma de acceder al modbus serie desde la aplicación Java. Para conseguir este fin, se ha utilizado la librería Modbus4Java desarrollada en el departamento de ESAII por Xavier Gallego.

Modbus4Java nos ofrece una interfaz para poder trabajar con modbus de una forma cómoda. Las funcionalidades implementadas se pueden observar en el siguiente extracto de código:

```
//Read Register --> Function Code, Starting @, Qty
public ModbusPacket newRequestReadRegisterPacket(short startingAddr, short
quantity);
//Write Register --> Function Code, Register @, Value
public ModbusPacket newRequestWriteRegisterPacket(short register, short value);
public ModbusPacket newResponsePacket(byte[] data, int bytes);
```

Modbus4Java nos permite crear paquetes modbus para los tres functions codes implementados en la electrónica ecoGate. La creación de la cabecera y del CRC del paquete, así como la validación del CRC de los paquetes recibidos es gestionada por esta librería.

8.2.Aplicación para teléfono Android

Para cumplir el segundo gran objetivo de este PFC, la visualización de los datos de la bicicleta eléctrica, se tiene que conseguir un cliente Bluetooth que se conecte a un servidor y escuche por su servicio de puerto serie (SPP). Para gestionar toda la API de Android para la utilización del Bluetooth, se ha partido de base del ejemplo Bluetooth Chat, incluido en la documentación oficial de Android. Esta aplicación permite a dos dispositivos Android comunicarse mediante mensajes de texto enviados vía Bluetooth. Demuestra todas las capacidades fundamentales de la API Bluetooth de Android, tales como:

- Búsqueda de otros dispositivos Bluetooth
- Consulta al adaptador local de los dispositivos vinculados.

8. Monitorización

- Establecimiento de los canales/sockets RFCOMM
- Conexión a un dispositivo remoto
- Transferencia de datos vía Bluetooth

En este apartado, se muestran y comentan las diferentes funcionalidades implementadas, pero no se da el código programado. Se pueden ver las partes más importantes del código de la aplicación en el anexo 2.

8.2.1.Ciclo de vida de una aplicación Android

El ciclo de vida de una aplicación en Android es manejado por el sistema operativo, basándose en las necesidades del usuario, los recursos disponibles, etc. Si se tiene una aplicación que está consumiendo muchos recursos y se arranca otra nueva, el sistema operativo probablemente le diga a la aplicación que se queda en segundo plano que libere todo lo que pueda, y si es necesario la cerrará. En Android los recursos son normalmente muy limitados y por eso el sistema operativo tiene más control sobre las aplicaciones que en programas de escritorio.

En la mayoría de los casos, cada aplicación Android corre en su propio proceso de Linux. Este proceso es creado para la aplicación cuando se arranca y seguirá corriendo hasta que no sea necesario y el sistema reclame recursos para otras aplicaciones y se los de a estas.

Android puede en cualquier momento pausar, parar, destruir nuestra aplicación según las necesidades del momento. Las llamadas que nos permiten controlar el destino de nuestra aplicación durante su ciclo de vida son:

- `onCreate()` Llamada cuando la actividad es creada por primera vez. Después de esta llamada siempre se llama al `onStart()`.
- `onRestart()` Llamada cuando la actividad ha sido parada, antes de volver a ser ejecutada. Siempre viene después un `onStart()`.
- `onStart()` Llamada cuando la actividad está siendo visible por el usuario. Después de ésta, se puede ir al `onResume()` si la actividad va a ser visible o se puede ir al `onStop()` si se ejecuta en segundo plano.
- `onResume()` Llamada cuando la actividad va a empezar a interactuar con el usuario.
- `onPause()` Llamada cuando el sistema va a empezar una nueva actividad.. Después de esta llamada puede venir un `onResume()` si la actividad vuelve a primer plano o un `onStop()` si se hace invisible para el usuario.

- `onStop()` Llamada cuando la actividad ya no es visible al usuario, porque otra actividad ha pasado a primer plano. Desde aquí se puede ir al `onRestart()` si vuelve a primer plano o al `onDestroy()` si se destruye del todo.
- `onDestroy()` Esta es la llamada final a la actividad, después de ésta, es totalmente destruida.

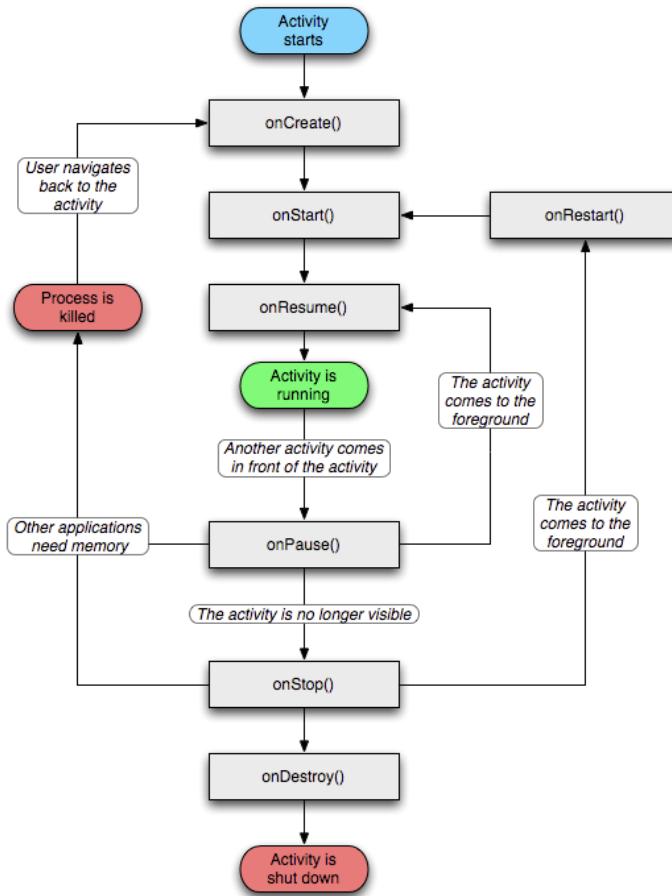


Figura 8.12: Ciclo de vida de una aplicación Android.

8.2.2. Estructura de la aplicación

La cantidad de datos a mostrar en la aplicación desarrollada, hacen que no quepan todos en una pantalla, o como mínimo no con el tamaño suficiente para ser leíbles. Para este caso se pueden utilizar varias soluciones, como por ejemplo el scroll, pero la solución que se ha utilizado en este proyecto es la creación de pestañas.

Para crear pestañas en Android se necesita una clase `TabActivity` que será la encargada de crear todas las pestañas necesarias y una clase del tipo `Fragment` necesitado con su correspondiente `layout(xml)` por cada pestaña que se quiera introducir. Este sistema tiene muchas ventajas como pueden ser tener el código más dividido y limpio, que una pestaña pueda ser una lista y otra una simple

Activity, que cada pestaña(clase) tenga su propio menú, etc. Las pestañas que componen la aplicación son:

- Speed: pestaña donde se presentan los datos correspondientes a la velocidad de la bicicleta, el pedal y el motor
- Energy: muestra los datos referentes a la batería y al consumo de corriente del motor. Permite seleccionar el grado de ayuda que queremos del motor.
- Map: mapa online de Google Maps con posicionamiento GPS
- Summary: otros datos no recogidos en otras pantallas, como pueden ser el estado de los frenos, distancia recorrida, etc.

Una descripción más detallada de las funcionalidades de éstas pantallas es dada en los siguientes subapartados.

Tal y como se ha comentado, cada pestaña implementa una activity independiente. Para que el controlador de la tarea modbus pueda interactuar con cada pestaña e intercambiar los datos que se reciben por Bluetooth, cada activity hereda de una clase abstracta llamada EcoBikeActivity y es obligada a implementar el método update, que es donde el controlador de mobus se comunica para poder enviar los datos que se deberán mostrar por pantalla.

8.2.3. Conexión de un dispositivo

Al iniciar la aplicación, ésta comprueba mediante el BluetoothAdapter, si el teléfono tiene activado el Bluetooth. Si está apagado, se pide permiso para activarlo y se enciende. Si no se concede el permiso la aplicación se cierra. El primero paso para poder visualizar los datos en la aplicación, es conectar con el Bluetooth de una bicicleta. Dentro del menú implementado, se encuentra la opción para conectar con un dispositivo. Se muestran por defecto los dispositivos que han sido vinculados con anterioridad al smartphone, así como también la posibilidad de iniciar una búsqueda de dispositivos y vincular nuevas electrónicas ecogate.



Figura 8.13: Conectar con un dispositivo.

8.2.4. Pestaña Speed

En esta pantalla se muestran de forma gráfica los velocímetros que presentan los datos de las diferentes velocidades que se pueden medir en la bicicleta: velocidad de la bicicleta, velocidad del motor y cadencia de pedaleo.



Figura 8.14: Pantalla speed.

8. Monitorización

La velocidad del motor y la cadencia de pedaleo son valores que calcula la electrónica ecocontrol de la bicicleta, y que por lo tanto solo se deben leer y mostrar. Por el contrario, la velocidad de la bicicleta no se calculaba por defecto en la bicicleta de serie, es por ello que a la bicicleta utilizada en este PFC se le ha instalado un sensor hall que lee los pasos de un imán instalado en la rueda trasera. Calculando el tiempo entre los flancos generados por el imán y sabiendo la circunferencia de la rueda se puede calcular la velocidad de la bicicleta.



Figura 8.15: Sensor hall e imán instalados en la bicicleta para calcular su velocidad.

8.2.5.Pestaña Energy

En esta pestaña, se muestra de forma gráfica mediante dos barras de nivel el voltaje de la batería (medido por la electrónica ecoGate) y el consumo de corriente en amperios del motor (medido por la electrónica ecoControl). También se nos permite en esta pantalla elegir el grado de ayuda que queremos que nos proporcione el motor. Más concretamente, este parámetro nos permite elegir la proporción de trabajo que realiza el motor respecto al trabajo que realiza el usuario pedaleando. Los diferentes valores que se permite seleccionar son:

- Off: el motor no ayuda.
- Eco: el trabajo realizado por el motor representa el 30% del total
- Normal: el trabajo realizado por el motor representa el 60% del total
- Full: el trabajo realizado por el motor representa el 90% del total

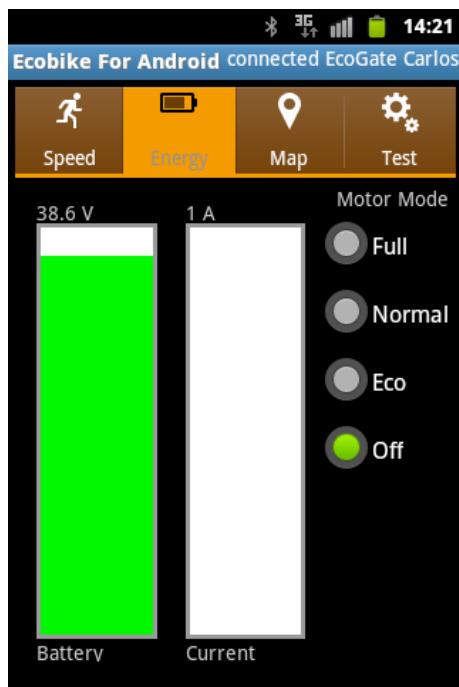


Figura 8.16: Pantalla Energy.

8.2.6.Pestaña Map

En esta pestaña se muestra un mapa utilizando la API que proporciona Android para utilizar los mapas de Google Map. Si se activa la opción de localización mediante GPS, se dibuja una bicicleta en la posición donde se localiza el SmartPhone. Se da también la información sobre el error cometido por el GPS en la medida de la posición. Esta pantalla permite realizar zoom de forma multitáctil (con dos dedos), adaptando la proyección de la información del error cometido en la localización al nuevo nivel de zoom. Esta adaptación de la proyección, permite hacerse una idea visual y realista de la zona en la que puede encontrarse el usuario.

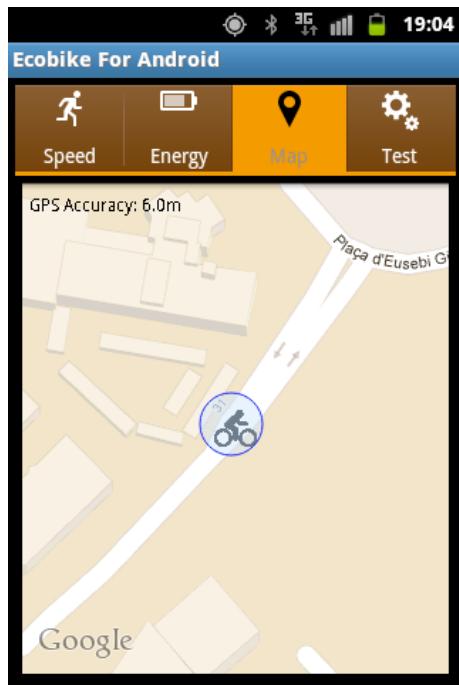


Figura 8.17: Pantalla Map.

8.2.7.Pestaña Summary

En esta pantalla se muestran el resto de datos que se obtienen de la bicicleta y que no tienen cabida en el resto de pestañas. Los datos mostrados son:

- El estado de los frenos.
- La distancia recorrida.
- La temperatura del motor
- El torque realizado por el ciclista
- La relación entre el trabajo realizado por el ciclista y el trabajo estimado que realiza el motor.

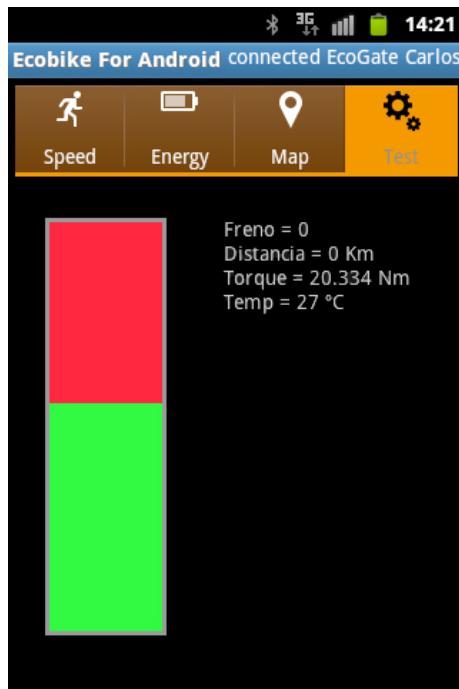


Figura 8.18: Pantalla Summary.

8.3. Aplicación para tablet Android

Una vez se tenía desarrollado el prototipo de aplicación para Android, y aprovechando que en el departamento se disponía de una tablet Motorola Xoom con Android 3.0 se decidió realizar la adaptación de la aplicación para poder funcionar de forma nativa sobre esta nueva plataforma.



Figura 8.19: Tablet Motorola Xoom.

La principal diferencia en la programación de aplicaciones para Android 2.X y 3.X es el nuevo paradigma para construir la aplicación que Google está favoreciendo en detrimento de otras estrategias como pueden ser los TabActivitys: los Fragments. El objetivo del uso de Fragment es

justamente hacer mas fácil el desarrollo para apps que funcionen bien en tablets y smartphones. Los Fragments son fragmentos de una Activity. Un Fragment fuera de una Activity no existe. Son similares a los paneles y justamente por el hecho de tener un código y una vista son altamente reutilizables. Se puede tener un Fragment usado en varios lugares, mostrando en cada sitio la información deseada. La filosofía de diseño queda ejemplificada en la siguiente imagen:

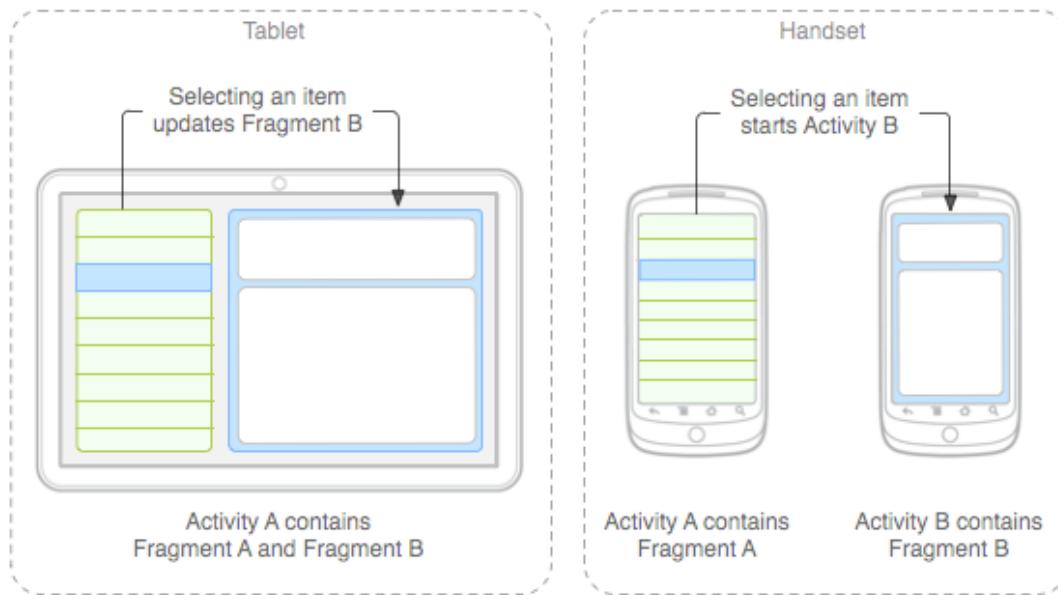


Figura 8.20: Ejemplo de dos interfaces creadas mediante la reutilización de fragments para tablet (izquierda) y para smartphone (derecha).

Se ha pasado pues de las pestañas de la aplicación para teléfono a los fragments. Los fragments implementados son los siguientes:

- BatteryFragment: Permite visualizar los datos de voltaje de la batería y la corriente consumida por el motor
- ExperimentFragment: Este fragment es el encargado de grabar los datos de la bicicleta en un archivo para poder ser visualizados a posteriori.
- FolderFragment: Permite seleccionar de entre todas las capturas de datos realizadas para su visualización.
- ForcesFragment: Muestra el par ejercido por el ciclista. así como la relación entre el trabajo realizado por éste y el trabajo realizado por el motor.
- ControlModeFragment: Permite seleccionar el nivel de ayuda del motor.

Dado que muchas de las funcionalidades implementadas en la tablet son iguales a las implementadas para los smartphones, solo se explicarán de forma más detallada las capacidades exclusivas de la aplicación para tablets.

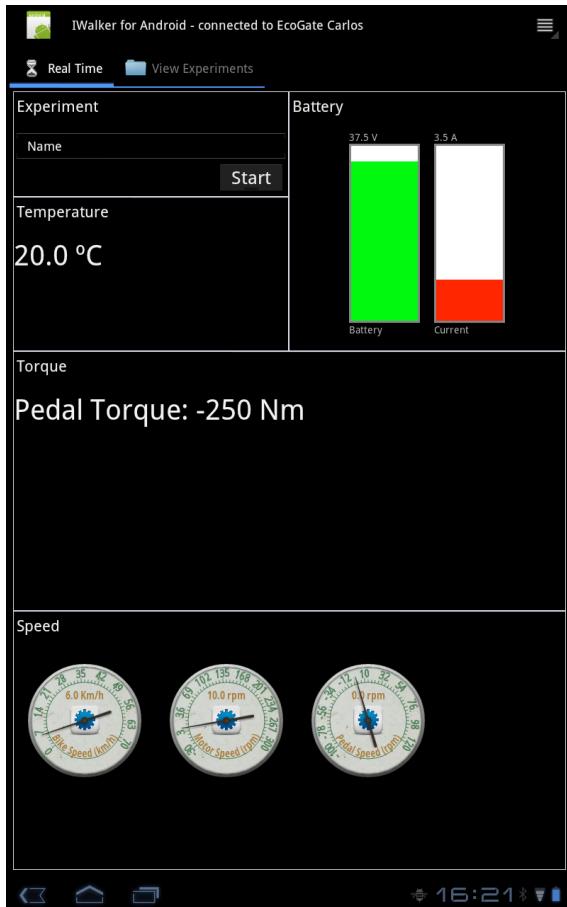


Figura 8.21: Vista general de la aplicación

8.3.1. Grabación de datos

La aplicación desarrollada para funcionar sobre tablets Android, dada la potencia que suelen tener este tipos de dispositivos, permite grabar todos los datos que se monitorizan provenientes de la bicicleta. Esta función es de gran utilidad, pues permite grabar los datos para poder analizarlos posteriormente. La aplicación permite dar el nombre que se desee al experimento y decidir mediante un botón cuando se quiere comenzar a grabar y cuando se quiere parar.

8. Monitorización

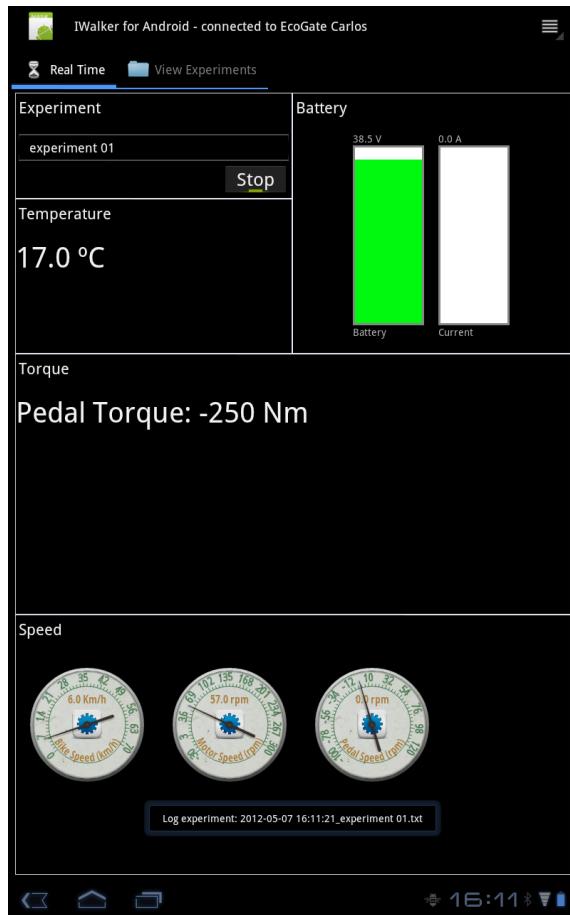


Figura 8.22: Grabación de datos.

8.3.2. Visualización de experimentos

En el FolderFragment se muestran en forma de lista todos los logs de experimentos que se han realizado con la tablet. Se dispone de un botón que permite refrescar la lista por si se da el caso en que el último experimento realizado no aparezca en la lista.

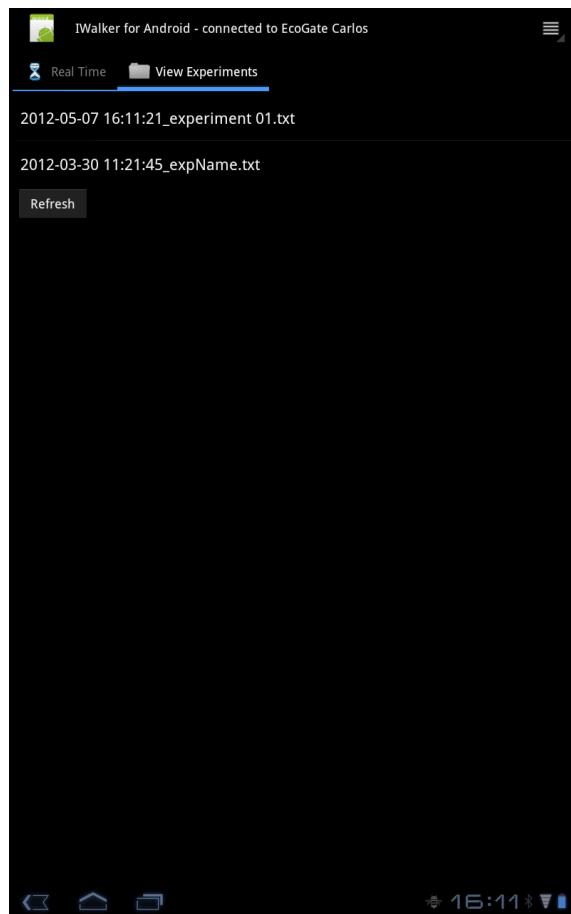


Figura 8.22: Lista de experimentos grabados.

Seleccionando el experimento que se desee, se inicia la activity que permite visualizar de forma gráfica los datos grabados en el log seleccionado (se visualiza la velocidad, la velocidad del motor, el par realizado por la persona y la cadencia de pedaleo)

8. Monitorización

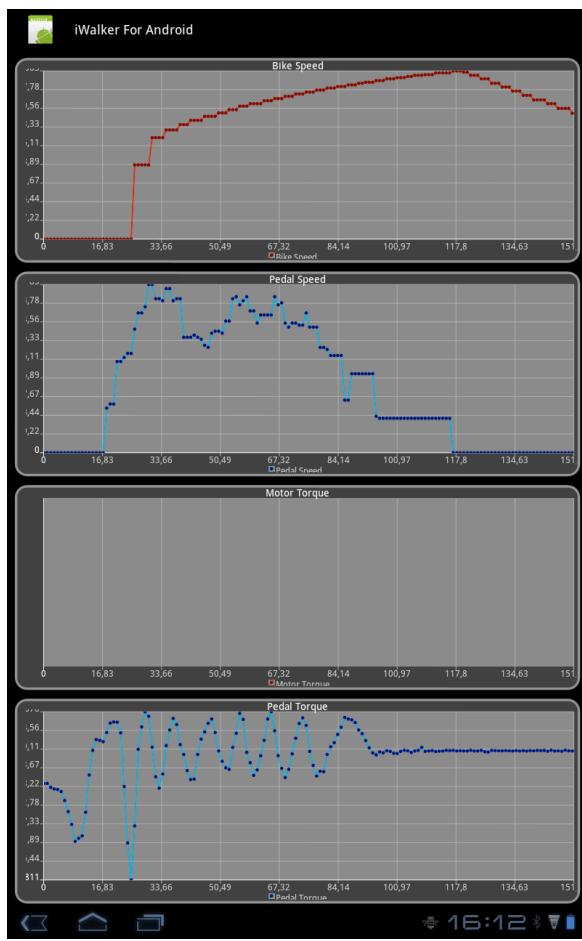


Figura 8.23: Representación gráfica de los datos grabados durante un experimento.

Estas gráficas permiten la realización de zoom de forma multitáctil (dos dedos), así como la posibilidad de moverse en todo el dominio de los datos guardados.

Capítulo 9

Evaluación de Resultados

En este apartado se hace reflexión sobre las metas conseguidas al finalizar este PFC. Se tratan también las líneas de trabajo que hay que seguir para mejorar el sistema desarrollado y conseguir ampliar sus funcionalidades.

9.1.Metas conseguidas

Con el sistema desarrollado se ha conseguido:

- Crear un entorno de trabajo que permite trabajar de forma fácil con la bicicleta eléctrica, así como conectarla mediante el bus CAN a un modelo Simulink.
- Se ha diseñado, implementado y testeado una estrategias de control del motor dependiente del par ejercido sobre los pedales por un usuario.
- Mostrar de forma gráfica el trabajo realizado por el ciclista en comparación al aportado por el motor, utilizando un Smartphone. Se muestran también los datos más importantes relacionados con la odometría de la bicicleta. Se ha implementado este sistema también sobre tablets con Android 3.0, permitiendo además la grabación y posterior visualización de los datos.

Considero entonces conseguidos los objetivos marcados al inicio del proyecto.

9.2.Ampliaciones/mejoras

Hay varios aspectos en los que se podría seguir trabajando. Por ejemplo, solo se ha explorado una estrategia para el control del motor, y con el banco de pruebas desarrollado se podría experimentar fácilmente con nuevas posibilidades.

La interfaz gráfica desarrollada para la aplicación Android es visualmente muy poco atractiva. Mejorar la apariencia, sobretodo si se pretende conseguir un producto comercial, debería ser una de las líneas de trabajo principales a seguir.

Se podría completar la aplicación desarrollada, además, incorporando la grabación de los tracks proporcionados por el GPS, así como permitir la incorporación de tracks externos para poder realizar rutas mediante la navegación GPS.

9. Evaluación de resultados

Por último, se podría incorporar también la funcionalidad de pulsómetro a la aplicación para tener todos los datos relativos al esfuerzo realizado por el ciclista. Para incorporar esta funcionalidad se debería utilizar una banda pulsómetro Bluetooth como por ejemplo las fabricadas por Polar o Zephyr.

Capítulo 10

Valoración Económica

Al realizar la valoración económica del proyecto se puede diferenciar entre dos tipos de recursos: materiales y humanos.

10.1. Recursos materiales

Son los recursos hardware y software que se han utilizado y que se muestran en la siguiente tabla:

Hardware

Concepto	Coste
Electrónica ecocontrol	€ 70
Electrónica ecogate	€ 60
Electrónica 4pga	€ 50
Electrónica encoder	€ 20
Rodillo elite powermag	€ 1.200
ICD2	€ 120
Motorola Xoom	€ 525
Total	€ 2.045

Software

Concepto	Coste
ORCAD (estudiante)	Gratuito
MPLAB 8	Gratuito
MPLAB X	Gratuito
Microchip C30	Gratuito
Microsoft Office 2007 (estudiante)	Gratuito

Concepto	Coste
Matlab 2008 (Estudiante)	Gratis

10.2. Recursos humanos

Por lo que respecta al trabajo realizado en mi PFC, podemos diferenciarlo en distintas tareas. Estas tareas tienen un valor diferente dentro de un proyecto de ingeniería. Los diferentes perfiles que pueden tomar parte son:

- Analista: Encargado de realizar el estudio y diseño general. Toma decisiones que determinan el buen funcionamiento del sistema.
- Programador: Se encarga de diseñar e implementar el software del sistema tal y como ha especificado el analista
- Técnico: Se encarga de diseñar e implementar el hardware a desarrollar.

En la siguiente tabla se puede ver el resumen de horas según cada tarea:

Tarea	Horas	Coste Hora	Coste
Análisis y diseño software	100	€ 35	3500
Implementación software	180	€ 25	4500
Análisis y diseño hardware	240	€ 35	4900
Implementación hardware	420	€ 25	10500
Total			23400

10.3. Coste Total

El coste total del proyecto es la suma de los diferentes coste descritos en las tablas anteriores:

Recurso	Coste (€)
Materiales	2045
Humanos	23400
Total	25445

Capítulo 11

Planificación

Al inicio del proyecto se realizó una planificación temporal que ha sido modificada iterativamente a lo largo de todo el desarrollo de este. Esta planificación está separada en las siguientes tareas:

- Estado del arte.
- Integración del pedalier Thun.
- Creación del banco de pruebas.
- Modelo Simulink control banco de pruebas.
- Implementación modbus.
- Emulación estrategias de control en Simulink.
- Implementación estrategias de control en la electronica de la bicicleta.
- Test estrategias de control y correcciones.
- Desarrollo de la aplicación Android.
- Documentación.

El diagrama de Gannt para la planificación ha sido el siguiente:

11. Planificación

Id.	Nombre de tarea	Comienzo	Fin	T3 11			T4 11			T1 12			T2 12	
				jul	ago	sep	oct	nov	dic	ene	feb	mar	abr	
1	Estado del arte	01/07/2011	05/08/2011											
2	Integración pedalier Thun	05/09/2011	16/09/2011											
3	Creación banco de pruebas	05/09/2011	23/01/2012											
4	Modelo Simulink control banco de pruebas	07/11/2011	25/11/2011											
5	Implementación Modbus	16/01/2012	27/01/2012											
6	Emulación estrategia de control en Simulink	20/02/2012	13/03/2012											
7	Implementación estrategia de control en la electronica de la bicicleta	14/03/2012	27/03/2012											
8	Test estrategia y correcciones	28/03/2012	20/04/2012											
9	Aplicación android	27/01/2012	08/03/2012											
10	Documentación	01/07/2011	11/05/2012											

Durante todo el proyecto se ha ido cumpliendo de forma razonable con la planificación. Debido a una serie de problemas hardware, algunas tareas se ha retrasado una semana, acarreando retrasos también en el resto de tareas, sobre todo en la documentación.

Bibliografía

- [1] Margaret Estivatel, Pierre Brisso. The Engineering of Sport 7, Volume 1. 2008
- [2] Mateu Martín i Batlle, Francesc Roure i Fernández, Jesús Sanz i Rubies. Extensiometría. Maig 1992
- [3] <http://www.srm.de>
- [4] <http://www.cycleops.com/>
- [5] <http://www.cyclepowermeters.com/>
- [6] <http://www.amtriathlon.com/2010/09/look-keo-power-polar-p5.html>
- [7] <http://www.ergomousa.com>
- [8] Thun. BB X-CELL RT datasheet
- [9] Texas Instrument. PGA309 datasheet
- [10] Texas Instrument. PGA309 user's manual
- [11] Texas Instrument. PGA
- [12] <http://www.kalkhoffusa.com/pedal-assisted-bikes.php>
- [13] <http://www.microchip.com>
- [14] Microchip. dsPIC30F family reference manual
- [15] Microchip. dsPIC30F4011 datasheet
- [16] Open Handset Alliance. <http://developer.Android.com>
- [17] <http://www.wikipedia.org>
- [18] Osram. sfh9240 datasheet
- [19] <http://www.electricbikesales.co.uk/info/buyingguide/>
- [20] <http://www.kalkhoffusa.com/pedal-assisted-bikes.php>
- [21] http://imagenes.w3.racc.es/uploads/file/8364_PEDELECS_test_2010.pdf
- [22] <http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:32002L0024:ES:HTML>

Agradecimientos

Me gustaría dedicar este pequeño hueco en la memoria de este PFC para dar mi más sincero agradecimiento a todas aquellas personas que me han ayudado a llegar a escribir estas palabras.

En primer lugar me gustaría dar las gracias a todos aquellos que comenzaron siendo compañeros de trabajo para acabar convirtiéndose en amigos: Laure, Toni, Xisco, Miquel, Jose, Valentí, Dani, Atia i Guiem. Gracias por todos los momentos.

Gracias a Joan Vidos por iniciarme en el mundo del diseño de PCBs.

Gracias a mi director, Antonio B. Martínez por darme el privilegio de aprender en este entorno.

Gracias a mis padres, José y Paqui, sin su soporte nada de esto hubiese sido posible.

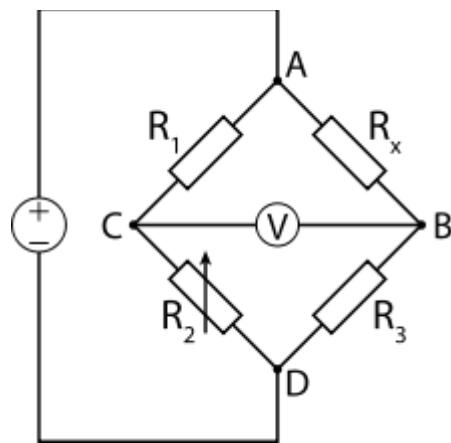
Gracias a mi mujer Raquel y a mi hija Ada por darme fuerzas y una sonrisa siempre que lo he necesitado.

¡Gracias a todos!

Anexos

Anexo 1: El puente de Wheatstone

Un puente de Wheatstone Se utiliza para medir resistencias desconocidas mediante el equilibrio de los brazos del puente. Estos están constituidos por cuatro resistencias que forman un circuito cerrado, siendo una de ellas la resistencia bajo medida.



Disposición eléctrica de un puente de Wheatstone

En la figura vemos que, R_x es la resistencia cuyo valor queremos determinar, R_1 , R_2 y R_3 son resistencias de valores conocidos, además la resistencia R_2 es ajustable. Si la relación de las dos resistencias del brazo conocido (R_1/R_2) es igual a la relación de las dos del brazo desconocido (R_x/R_3), el voltaje entre los dos puntos medios será nulo y por tanto no circulará corriente alguna entre esos dos puntos C y B.

Para efectuar la medida lo que se hace es variar la resistencia R_2 hasta alcanzar el punto de equilibrio. La detección de corriente nula se puede hacer con gran precisión mediante el voltímetro V.

La dirección de la corriente, en caso de desequilibrio, indica si R_2 es demasiado alta o demasiado baja. El valor de la F.E.M. (E) del generador es indiferente y no afecta a la medida.

Cuando el puente esta construido de forma que R_3 es igual a R_2 , R_x es igual a R_1 en condición de equilibrio.(corriente nula por el galvanómetro).

Asimismo, en condición de equilibrio siempre se cumple que:

Si los valores de R₁, R₂ y R₃ se conocen con mucha precisión, el valor de R_x puede ser determinado igualmente con precisión. Pequeños cambios en el valor de R_x romperán el equilibrio y serán claramente detectados por la indicación del galvanómetro.

De forma alternativa, si los valores de R₁, R₂ y R₃ son conocidos y R₂ no es ajustable, la corriente que fluye a través del galvanómetro puede ser utilizada para calcular el valor de R_x siendo este procedimiento más rápido que el ajustar a cero la corriente a través del medidor.

Variantes del puente de Wheatstone se pueden utilizar para la medida de impedancias, capacitancias e inductancias

La disposición en puente también es ampliamente utilizada en instrumentación electrónica. Para ello, se sustituyen una o más resistencias por sensores, que al variar su resistencia dan lugar a una salida proporcional a la variación. A la salida del puente (en la Figura 1, donde está el galvanómetro) suele colocarse un amplificador .

Anexo 2: Fragmentos de código de las aplicaciones Android

En este anexo, se adjuntan algunos fragmentos de código, aquellos considerados más importantes o más representativos, de las aplicaciones desarrolladas para los dispositivos Android.

Smartphone - Android 2.2

Código de la clase abstracta Ecobike Activity:

```
package com.ecobike.Android.app;
import android.app.Activity;
public abstract class EcoBikeActivity extends Activity{
    public abstract void update(short[] data, int reg);
}
```

Código de la actividad que implementa el TabActivity:

```
package com.ecobike.Android.app;
import java.util.ArrayList;

import com.ecobike.Android.app.R;
import com.upc.ecobike.Android.modbus.ByteUtils;
import android.app.Activity;
import android.app.TabActivity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.content.Context;
import android.content.Intent;
import android.content.res.Resources;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.Window;
import android.widget.EditText;
import android.widget.TabHost;
import android.widget.TabHost.OnTabChangeListener;
import android.widget.TabWidget;
import android.widget.TextView;
import android.widget.Toast;

import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;

import com.google.Android.maps.MapActivity;
import com.upc.ecobike.Android.modbus.ByteUtils;

public class Ecobike4AndroidActivity extends TabActivity {
    // Debugging
    private static final String TAG = "EcoBike4Android";
    private static final boolean D = true;
```

```
// Message types sent from the BluetoothChatService Handler
public static final int MESSAGE_STATE_CHANGE = 1;
public static final int MESSAGE_READ = 2;
public static final int MESSAGE_WRITE = 3;
public static final int MESSAGE_DEVICE_NAME = 4;
public static final int MESSAGE_TOAST = 5;

// Key names received from the BluetoothChatService Handler
public static final String DEVICE_NAME = "device_name";
public static final String TOAST = "toast";

// Intent request codes
//private static final int REQUEST_CONNECT_DEVICE_SECURE = 1;
private static final int REQUEST_CONNECT_DEVICE_INSECURE = 2;
private static final int REQUEST_ENABLE_BT = 3;

//Layout views
private TextView mTitle;
private String mTabSelected;

// Name of the connected device
private String mConnectedDeviceName = null;
// Local Bluetooth adapter
private BluetoothAdapter mBluetoothAdapter = null;
// Member object for the chat services
private BluetoothService mBTService = null;

private boolean gps=false;
private LocationManager locManager;
private LocationListener locListener;

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    if(D) Log.d(TAG, "+++ ON CREATE +++");

    requestWindowFeature(Window.FEATURE_CUSTOM_TITLE);
    setContentView(R.layout.main);

    tabConfig();

    //Title with 2 text fields (App title text and App state text)
    getWindow().setFeatureInt(Window.FEATURE_CUSTOM_TITLE, R.layout.custom_title);

    mTitle = (TextView) findViewById(R.id.title_left_text);
    mTitle.setText(R.string.app_name);
    mTitle = (TextView) findViewById(R.id.title_right_text); //App status message
text field

    // Get local Bluetooth adapter
    mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();

    // If the adapter is null, then Bluetooth is not supported
    if (mBluetoothAdapter == null) {
        Toast.makeText(this, "Bluetooth is not available", Toast.LENGTH_LONG).show
();
        finish();
        return;
    }
}

private void tabConfig() {
    Resources res = getResources(); // Resource object to get Drawables
    TabHost tabHost = getTabHost(); // The activity TabHost
    TabHost.TabSpec spec; // Reusable TabSpec for each tab
    Intent intent; // Reusable Intent for each tab
```

```

// Create an Intent to launch an Activity for the tab (to be reused)
intent = new Intent().setClass(this, SpeedActivity.class);

// Initialize a TabSpec for each tab and add it to the TabHost
spec = tabHost.newTabSpec("speed").setIndicator("Speed",
    res.getDrawable(R.drawable.ic_speed))
    .setContent(intent);
tabHost.addTab(spec);

// Do the same for the other tabs
intent = new Intent().setClass(this, EnergyActivity.class);
spec = tabHost.newTabSpec("energy").setIndicator("Energy",
    res.getDrawable(R.drawable.ic_energy))
    .setContent(intent);
tabHost.addTab(spec);

intent = new Intent().setClass(this, Map_Activity.class);
spec = tabHost.newTabSpec("map").setIndicator("Map",
    res.getDrawable(R.drawable.ic_map))
    .setContent(intent);
tabHost.addTab(spec);

intent = new Intent().setClass(this, TestActivity.class);
spec = tabHost.newTabSpec("test").setIndicator("Test",
    res.getDrawable(R.drawable.ic_test))
    .setContent(intent);
tabHost.addTab(spec);

tabHost.setCurrentTab(1);
mTabSelected = tabHost.getCurrentTabTag();

tabHost.setOnTabChangedListener(new OnTabChangeListener() {
    public void onTabChanged(String tabId) {
        mTabSelected = tabId;
    }
});

@Override
public void onStart() {
    super.onStart();
    if(D) Log.d(TAG, "++ ON START ++");

    // If BT is not on, request that it be enabled.
    // setupChat() will then be called during onActivityResult
    if (!mBluetoothAdapter.isEnabled()) {
        Intent enableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(enableIntent, REQUEST_ENABLE_BT);
    } else {
        if (mBTService == null) setupApp();
    }
}

@Override
public synchronized void onResume() {
    super.onResume();
    if(D) Log.d(TAG, "+ ON RESUME +");

    // Performing this check in onResume() covers the case in which BT was
    // not enabled during onStart(), so we were paused to enable it...
    // onResume() will be called when ACTION_REQUEST_ENABLE activity returns.
    if (mBTService != null) {
        // Only if the state is STATE_NONE, do we know that we haven't started
already
        if (mBTService.getState() == BluetoothService.STATE_NONE) {
            // Start the Bluetooth chat services
            //mBTService.start();
        }
    }
}

```

```

}

private void setupApp() {
    Log.d(TAG, "setupApp()");
    // Initialize the BluetoothService to perform bluetooth connections
    mBTService = new BluetoothService(this, mHandler);
}

@Override
public synchronized void onPause() {
    super.onPause();
    if(D) Log.d(TAG, "- ON PAUSE -");
}

@Override
public void onStop() {
    super.onStop();
    if(D) Log.d(TAG, "-- ON STOP --");
}

@Override
public void onDestroy() {
    super.onDestroy();
    if(D) Log.d(TAG, "--- ON DESTROY ---");
    if (locManager!=null) locManager.removeUpdates(locListener);

    // Stop the Bluetooth Services
    if (mBTService != null) mBTService.stop();
}

// The Handler that gets information back from the BluetoothService
private final Handler mHandler = new Handler() {
    @Override
    public void handleMessage(Message msg) {
        switch (msg.what) {
            case MESSAGE_STATE_CHANGE:
                if(D) Log.i(TAG, "MESSAGE_STATE_CHANGE: " + msg.arg1);

                switch (msg.arg1) {
                    case BluetoothService.STATE_CONNECTED:
                        //Set device title text
                        mTitle.setText(R.string.title_connected_to);
                        mTitle.append(" ");
                        mTitle.append(mConnectedDeviceName);
                        //TODO aixo s'hauria de canviar per tal que el
                        //      tingues un handler que interactues amb el
                        ModbusControllerTask
                        mBTService
                        (mBTService);
                        break;
                    case BluetoothService.STATE_CONNECTING:
                        mTitle.setText(R.string.title_connecting);
                        break;
                    case BluetoothService.STATE_LISTEN:
                    case BluetoothService.STATE_NONE:
                        mTitle.setText(R.string.title_not_connected);
                        ModbusControllerTask.getInstance().stop();
                        break;
                }
                break;
            case MESSAGE_WRITE:
                //Don't need message write for the moment
                //Message_write is the message sended from APP to Accessory
                break;
            case MESSAGE_READ:
                byte[] readBuf = (byte[]) msg.obj;

                updateView(readBuf,(msg.arg1-5)/2);
                break;
        }
    }
}

```

```

        case MESSAGE_DEVICE_NAME:
            // save the connected device's name
            mConnectedDeviceName = msg.getData().getString(DEVICE_NAME);
            Toast.makeText(getApplicationContext(), "Connected to"
                + mConnectedDeviceName, Toast.LENGTH_SHORT).show();
            break;
        case MESSAGE_TOAST:
            Toast.makeText(getApplicationContext(), msg.getData().getString
(TOAST),
                Toast.LENGTH_SHORT).show();
            break;
    }
};

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data){
    switch (requestCode) {
        case REQUEST_CONNECT_DEVICE_INSECURE:
            if (resultCode == Activity.RESULT_OK) {
                connectDevice(data, false);
            }
            break;
        case REQUEST_ENABLE_BT:
            if(resultCode == Activity.RESULT_OK){
                setupApp();
            }else{
                Log.e(TAG,"BT not enabled");
                Toast.makeText(this, R.string.bt_not_enabled_leaving,
Toast.LENGTH_SHORT).show();
                finish();
            }
            break;
        default:
            break;
    }
}

protected void updateView(byte[] readBuf, int registers) {
    short reg[] = new short[registers]; //Size of registers
    int i = 0;

    for(int j=0; j < registers; j++){
        short regVal = ByteUtils.byteArrayToShort(readBuf[i], readBuf[i+1]);
        reg[j] = regVal;
        i+=2;
    }
    if (!mTabSelected.equals("map")) {
        EcoBikeActivity activity = (EcoBikeActivity) getLocalActivityManager
        ().getActivity(mTabSelected);
        activity.update(reg, registers);
    }
}

private void connectDevice(Intent data, boolean secure) {
    // Get the device MAC address
    String address = data.getExtras()
        .getString(DeviceListActivity.EXTRA_DEVICE_ADDRESS);
    // Get the BluetoothDevice object
    BluetoothDevice device = mBluetoothAdapter.getRemoteDevice(address);
    // Attempt to connect to the device
    mBTService.connect(device, secure);
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.option_menu, menu);
    return true;
}

```

```

@Override
public boolean onPrepareOptionsMenu(Menu menu) {
    if (gps){
        MenuItem gpsmenu = menu.getItem(1);
        gpsmenu.setTitle(R.string.disablegps);
    } else {
        MenuItem gpsmenu = menu.getItem(1);
        gpsmenu.setTitle(R.string.enablegps);
    }
    return super.onPrepareOptionsMenu(menu);
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    Intent serverIntent = null;
    switch (item.getItemId()) {
        case R.id.connect_scan:
            // Launch the DeviceListActivity to see devices and do scan
            serverIntent = new Intent(this, DeviceListActivity.class);
            startActivityForResult(serverIntent,
REQUEST_CONNECT_DEVICE_INSECURE);
            return true;
        case R.id.enable_gps:
            if(!gps){
                gps=true;
                comenzarLocalizacion();
            } else {
                gps=false;
                locManager.removeUpdates(locListener);
            }
            return true;
    }
    return false;
}

private void comenzarLocalizacion()
{
    Toast.makeText(getApplicationContext(), "Enable gps ",
Toast.LENGTH_SHORT).show();

    //Obtenemos una referencia al LocationManager
    locManager =
        (LocationManager) getSystemService(Context.LOCATION_SERVICE);

    //Obtenemos la 'ltima posiciÛn conocida
    Location loc =
        locManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);

    //Mostramos la 'ltima posiciÛn conocida
    //mostrarPosicion(loc);

    //Nos registramos para recibir actualizaciones de la posiciÛn
    locListener = new LocationListener() {
        public void onLocationChanged(Location location) {
            //mostrarPosicion(location);
            if (mTabSelected.equals("map")) {
                Activity activity = (Activity) getLocalActivityManager
().getActivity("map");
                ((Map_Activity) activity).mostrarPosicion(location);
            }
        }
        public void onProviderDisabled(String provider){
            Toast.makeText(getApplicationContext(), "Provider OFF",
Toast.LENGTH_SHORT).show();
        }
    }
}

```

```

        public void onProviderEnabled(String provider){
            Toast.makeText(getApplicationContext(), "Provider ON",
Toast.LENGTH_SHORT).show();
        }
        public void onStatusChanged(String provider, int status, Bundle
extras){
            Log.i("", "Provider Status: " + status);
            Toast.makeText(getApplicationContext(), "Provider status: "+status,
Toast.LENGTH_SHORT).show();
        }
    };

    locManager.requestLocationUpdates(
        LocationManager.GPS_PROVIDER, 30000, 0, locListener);
}
}

```

Código ejemplo de una implementación de una EcobikeActivity:

```

package com.ecobike.Android.app;

import com.ecobike.Android.app.R;

import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;

public class EnergyActivity extends EcoBikeActivity{
    private LevelBarView currentBar = null;
    private LevelBarView batteryBar = null;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.energy_layout);

        View energy_layout = (View) findViewById(R.id.energy_lin_layout);
        energy_layout.setDrawingCacheBackgroundColor(Color.WHITE);

        currentBar = (LevelBarView) findViewById(R.id.levelBar_current);
        currentBar.setLevel(0);

        batteryBar = (LevelBarView) findViewById(R.id.levelBar_battery);
        batteryBar.setLevel(0);
    }

    @Override
    public void update(short[] data, int reg) {
        double voltage = ((double) data[5] / (double) 10);
        int vPercent = (int) (voltage * 100) / 42;
        double current = ((double) data[6] / (double) 10);
        int cPercent = (int) (current * 100) / 15;

        TextView vText = (TextView) findViewById(R.id.txt_battery_num);
        TextView cText = (TextView) findViewById(R.id.txt_current_num);

        batteryBar.setLevel(vPercent);
        currentBar.setLevel(cPercent);
        vText.setText(voltage + " V");
        cText.setText(current + " A");
    }
}

```

}

Código del controlador modbus:

```
package com.ecobike.Android.app;

import java.util.concurrent.Executors;
import java.util.concurrent.ScheduledExecutorService;
import java.util.concurrent.ScheduledFuture;
import android.util.Log;
import com.upc.ecobike.Android.modbus.ModbusPacket;
import com.upc.ecobike.Android.modbus.ModbusService;
import com.upc.ecobike.Android.modbus.ModbusServiceImpl;
import static java.util.concurrent.TimeUnit.*;

public class ModbusControllerTask implements Runnable{
    private static final String TAG = "ModbusControllerTask";
    private static final boolean D = true;

    private final int mInitDelay = 100; //Time in ms
    private final int mPeriod = 100; //Time in ms

    private static ModbusControllerTask instance = null; //Singleton instance
    private BluetoothService mBTService = null;
    private ModbusService mdbService = null;

    private final ScheduledExecutorService scheduler = Executors.newScheduledThreadPool
(1);
    private ScheduledFuture mdbControllerTaskHandle = null;

    private short mStartingAddr = 0x0000;
    private short mQuantity = 7;
    private ModbusPacket mdbPacket = null;

    private ModbusControllerTask(){
        mdbService = new ModbusServiceImpl();
        mdbControllerTaskHandle = scheduler.scheduleAtFixedRate(this, mInitDelay,
mPeriod, MILLISECONDS);
    }

    private synchronized static void createInstance() {
        if (instance == null) instance = new ModbusControllerTask();
    }

    public static ModbusControllerTask getInstance(){
        if (instance == null) createInstance();

        return instance;
    }

    public void run() {
        mdbPacket = mdbService.newRequestReadRegisterPacket(mStartingAddr,
mQuantity);

        this.sendModbusPacket(mdbPacket);
    }

    public short getmStartingAddr() {
        return mStartingAddr;
    }

    public void setmStartingAddr(short mStartingAddr) {
        this.mStartingAddr = mStartingAddr;
    }

    public short getmQuantity() {
        return mQuantity;
    }
}
```

```

    }

    public void setmQuantity(short mQuantity) {
        this.mQuantity = mQuantity;
    }

    /**
     * Stop modbus controller task using future task created on start
     * method.
     */
    public boolean stop(){
        if(mdbControllerTaskHandle == null) return false;

        mdbControllerTaskHandle.cancel(true);
        mBTService = null;
        return true;
    }

    /**
     * Send a modbus packet.
     * @param message The modbus packet that you want to send
     */
    private void sendModbusPacket(ModbusPacket mdbPacket) {
        // Check that there's actually something to send
        if (mdbPacket.length() > 0) {
            if(D) Log.i(TAG, "Sending modbus packet: "+mdbPacket.toString());

            // Get the message bytes and tell the BluetoothChatService to write
            byte[] mdbPacketBytes = mdbPacket.toByteArray();
            mBTService.write(mdbPacketBytes);
        }
    }

    public BluetoothService getmBTService() {
        return mBTService;
    }

    public void setmBTService(BluetoothService mBTService) {
        this.mBTService = mBTService;
    }
}

```

Código que implementa la capa para dibujar la posición de la bicicleta sobre el mapa:

```

package com.ecobike.Android.app;

import Android.content.Context;
import Android.graphics.Bitmap;
import Android.graphics.BitmapFactory;
import Android.graphics.Canvas;
import Android.graphics.Color;
import Android.graphics.Paint;
import Android.graphics.Point;
import Android.widget.Toast;

import com.google.Android.maps.GeoPoint;
import com.google.Android.maps.MapView;
import com.google.Android.maps.Overlay;
import com.google.Android.maps.Projection;

import Android.location.Location;

public class OverlayMapa extends Overlay {

```

```

    private Double latitud = 41.40*1E6;
    private Double longitud = 5.99*1E6;

```

```

private Float error = (float) 100;
//private int zoom=1;
int radio=0;

@Override
public void draw(Canvas canvas, MapView mapView, boolean shadow)
{
    Projection projection = mapView.getProjection();
    GeoPoint geoPoint =
        new GeoPoint(latitud.intValue(), longitud.intValue());

    if (shadow == false)
    {
        Point centro = new Point();
        projection.toPixels(geoPoint, centro);

        //Definimos el pincel de dibujo
        Paint p = new Paint();
        p.setColor(Color.BLACK);

        canvas.drawText("GPS Accuracy: "+error+"m", 5, 20, p);
        if (radio<3) radio=0;

        Bitmap bm = BitmapFactory.decodeResource(
            mapView.getResources(),
            R.drawable.ic_bike);

        canvas.drawBitmap(bm, centro.x - bm.getWidth()/2,
                          centro.y - bm.getHeight()/2, p);

        Paint innerCirclePaint;

        innerCirclePaint = new Paint();
        innerCirclePaint.setARGB(120, 198, 226, 255);
        innerCirclePaint.setAntiAlias(true); //
        innerCirclePaint.setStyle(Paint.Style.FILL);
        canvas.drawCircle(centro.x- bm.getWidth()/8, centro.y- bm.getHeight()/8,
        radio*2, innerCirclePaint);

        innerCirclePaint.setARGB(255, 0, 0, 255);
        innerCirclePaint.setAntiAlias(true); //
        innerCirclePaint.setStyle(Paint.Style.STROKE);
        canvas.drawCircle(centro.x- bm.getWidth()/8, centro.y- bm.getHeight()/8,
        radio*2, innerCirclePaint);

    }
}

private int metersToRadius(float meters, MapView map, double latitude) {
    return (int) (map.getProjection().metersToEquatorPixels(meters) *Math.abs( (1/
Math.cos(Math.toRadians(latitude)))));      //
}

public void setLocation (Location l){
    latitud=l.getLatitude()*1E6;
    longitud=l.getLongitude()*1E6;
    error=l.getAccuracy();

}

public void setZoom (MapView map){
    radio=metersToRadius(error, map, latitud);
}

```

```

    }
}
```

Tablet - Android 3.0

Ejemplo de programación de un fragment:

```

package com.Android.upc.tablet;

import android.app.Fragment;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.TextView;

public class BatteryFragment extends Fragment {
    private LevelBarView currentBar = null;
    private LevelBarView batteryBar = null;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.battery, container, false);
    }

    public void repintar(double v, double a) {
        int vPercent = (int) (v*100)/42;
        int cPercent = (int) (a*100)/15;

        TextView vText = (TextView) getActivity().findViewById(R.id.txt_battery_num);
        TextView cText = (TextView) getActivity().findViewById(R.id.txt_current_num);

        batteryBar.setLevel(vPercent);
        currentBar.setLevel(cPercent);
        vText.setText(v+" V");
        cText.setText(a+" A");

    }

    public void setCero() {
        currentBar = (LevelBarView) getActivity().findViewById(R.id.levelBar_current);
        currentBar.setLevel(0);
        batteryBar = (LevelBarView) getActivity().findViewById(R.id.levelBar_battery);
        batteryBar.setLevel(0);

        TextView vText = (TextView) getActivity().findViewById(R.id.txt_battery_num);
        TextView cText = (TextView) getActivity().findViewById(R.id.txt_current_num);

        batteryBar.setLevel(0);
        currentBar.setLevel(0);
        vText.setText("0 V");
        cText.setText("0 A");

    }
}
```

Código de la activity para la visualización de gráficas:

```
package com.Android.upc.tablet;

import Android.os.Bundle;
import Android.util.FloatMath;
import Android.view.MotionEvent;
import Android.view.View;
import Android.view.View.OnTouchListener;
import Android.widget.Button;
import Android.widget.Toast;
import Android.app.Activity;
import Android.content.res.Configuration;
import Android.drm.DrmStore.RightsStatus;
import Android.graphics.Color;
import Android.graphics.PointF;
import Android.os.Bundle;
import com.Androidplot.xy.SimpleXYSeries;
import com.Androidplot.series.XYSeries;
import com.Androidplot.xy.*;

import java.text.DecimalFormat;
import java.util.Arrays;
import java.util.Timer;
import java.util.TimerTask;

public class BikeActivity extends EcoBikeActivity implements OnTouchListener{

    public static Number[] bikeSpeed;
    public static Number[] pedalSpeed;
    public static Number[] pedalTorque;
    public static Number[] motorTorque;

    private XYPlot bikeSpeedPlot;
    private XYPlot pedalSpeedPlot;
    private XYPlot pedalTorquePlot;
    private XYPlot motorTorquePlot;

    private PointF minXY;
    private PointF maxXY;
    private float absMinX;
    private float absMaxX;
    private float minNoError;
    private float maxNoError;
    private double minDif;

    final private double difPadding = 0.1;

    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.bike);
        bikeSpeedPlot = (XYPlot) findViewById(R.id.bikespeedplot);
        bikeSpeedPlot.setOnTouchListener(this);
        pedalSpeedPlot = (XYPlot) findViewById(R.id.pedalspeedplot);
        pedalSpeedPlot.setOnTouchListener(this);
        pedalTorquePlot = (XYPlot) findViewById(R.id.pedaltorqueplot);
        pedalTorquePlot.setOnTouchListener(this);
        motorTorquePlot = (XYPlot) findViewById(R.id.motortorqueplot);
        motorTorquePlot.setOnTouchListener(this);

        layoutConfig(bikeSpeedPlot);
        layoutConfig(pedalSpeedPlot);
        layoutConfig(pedalTorquePlot);
        layoutConfig(motorTorquePlot);
    }
}
```

```

// Turn the above arrays into XYSeries:
XYSeries bspeed = new SimpleXYSeries(
    Arrays.asList(bikeSpeed), // SimpleXYSeries takes a
List so turn our array into a List
    SimpleXYSeries.ArrayFormat.Y_VALS_ONLY, // Y_VALS_ONLY means
use the element index as the x value
    "Bike Speed"); // Set the display
title of the series
XYSeries pspeed = new SimpleXYSeries(Arrays.asList
(pedalSpeed),SimpleXYSeries.ArrayFormat.Y_VALS_ONLY, "Pedal Speed");
XYSeries ptorque = new SimpleXYSeries(Arrays.asList
(pedalTorque),SimpleXYSeries.ArrayFormat.Y_VALS_ONLY, "Motor Torque");
XYSeries mtorque = new SimpleXYSeries(Arrays.asList
(motorTorque),SimpleXYSeries.ArrayFormat.Y_VALS_ONLY, "Pedal Torque");

// Create a formatter to use for drawing a series using
LineAndPointRenderer:
LineAndPointFormatter leftFormat = new LineAndPointFormatter(
    Color.rgb(255, 0, 0), // line color
    Color.rgb(128, 0, 0), // point color
    null); // fill color (optional --> no el volumen)
LineAndPointFormatter rightFormat = new LineAndPointFormatter(Color.rgb(0,
191, 255),Color.rgb(0, 0, 128),null);

// Add series1 to the xyplot:
bikeSpeedPlot.addSeries(bspeed, leftFormat);
pedalSpeedPlot.addSeries(pspeed, rightFormat);
pedalTorquePlot.addSeries(ptorque, leftFormat);
motorTorquePlot.addSeries(mtorque, rightFormat);

//Enact all changes
bikeSpeedPlot.redraw();
pedalSpeedPlot.redraw();
pedalTorquePlot.redraw();
motorTorquePlot.redraw();

trackBoundaries(bikeSpeedPlot, bspeed);
trackBoundaries(pedalSpeedPlot, pspeed);
trackBoundaries(pedalTorquePlot, ptorque);
trackBoundaries(motorTorquePlot, mtorque);

//Check x data to find the minimum difference between two neighboring
domain values
//Will use to prevent zooming further in than this distance
double temp1 = bspeed.getX(0).doubleValue();
double temp2 = bspeed.getX(1).doubleValue();
double temp3;
double thisDif;
minDif = 1000000; //increase if necessary for domain values
for (int i = 2; i < bspeed.size(); i++) {
    temp3 = bspeed.getX(i).doubleValue();
    thisDif = Math.abs(temp1 - temp3);
    if (thisDif < minDif)
        minDif = thisDif;
    temp1 = temp2;
    temp2 = temp3;
}
minDif = minDif + difPadding; //with padding, the minimum difference
//Toast.makeText(this, "girar", Toast.LENGTH_LONG).show();
}

public void layoutConfig(XYPlot plot) {
    //Plot layout configurations --> Forces X
    plot.getGraphWidget().setTicksPerRangeLabel(1);
    plot.getGraphWidget().setTicksPerDomainLabel(1);
}

```

Anexo 2: Fragmentos de Código de las Aplicaciones Android

```
plot.getGraphWidget().setRangeValueFormat(new DecimalFormat("#####.##"));
plot.getGraphWidget().setDomainValueFormat(new DecimalFormat("#####.##"));
plot.getGraphWidget().setRangeLabelWidth(25);
plot.setRangeLabel("");
plot.setDomainLabel("");
plot.disableAllMarkup();
}

public void trackBoundaries(XYPlot plot, XYSeries series) {
    //Set of internal variables for keeping track of boundaries
    plot.calculateMinMaxVals();
    minXY = new PointF(plot.getCalculatedMinX().floatValue()
(),plot.getCalculatedMinY().floatValue());
    absMinX = minXY.x;
    minNoError = Math.round(series.getX(1).floatValue()+2);
    maxXY = new PointF(plot.getCalculatedMaxX().floatValue()
(),plot.getCalculatedMaxY().floatValue());
    absMaxX = maxXY.x;
    maxNoError = (float) Math.round(series.getX(series.size()-1).floatValue())
-2;
}

static final private int NONE = 0;
static final private int ONE_FINGER_DRAG = 1;
static final private int TWO_FINGERS_DRAG = 2;
private int mode = NONE;

private PointF firstFinger;
private float lastScrolling;
private float distBetweenFingers;
private float lastZooming;

public boolean onTouch(View arg0, MotionEvent event) {
    switch(event.getAction() & MotionEvent.ACTION_MASK) {
        case MotionEvent.ACTION_DOWN:
            firstFinger = new PointF(event.getX(),event.getY());
            mode = ONE_FINGER_DRAG;
            break;
        case MotionEvent.ACTION_UP:
            firstFinger = new PointF(event.getX(),event.getY());
            mode = ONE_FINGER_DRAG;
            break;
        case MotionEvent.ACTION_POINTER_UP:
            final Timer t = new Timer();
            t.schedule(new TimerTask(){
                public void run(){
                    while(Math.abs(lastScrolling) > 1f || Math.abs
(lastZooming-1)<1.01) {
                        lastScrolling *= .8;
                        scroll(lastScrolling);
                        lastZooming += (1- lastZooming) * .2;
                        zoom(lastZooming);
                        checkBoundaries();
                        try{
                            bikeSpeedPlot.postRedraw();
                            pedalSpeedPlot.postRedraw();
                            pedalTorquePlot.postRedraw();
                            motorTorquePlot.postRedraw();
                        }catch (final InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                }
            }, 0);
            break;
        case MotionEvent.ACTION_POINTER_DOWN: //second finger
            if(event.getPointerCount() >= 2) {
                distBetweenFingers = spacing(event);
                //System.out.println("ACTION_POINTER_DOWN ---> " +
distBetweenFingers);
                if(distBetweenFingers > 5f){

```

```

        mode = TWO_FINGERS_DRAG;
    }
}
break;
case MotionEvent.ACTION_MOVE:
    if(mode == ONE_FINGER_DRAG) {
        final PointF oldFirstFinger = firstFinger;
        firstFinger = new PointF(event.getX(),event.getY());
        lastScrolling = oldFirstFinger.x - firstFinger.x;
        scroll(lastScrolling);
        lastZooming = (firstFinger.y - oldFirstFinger.y) /
bikeSpeedPlot.getHeight();
        if(lastZooming < 0) {
            lastZooming = 1 / (1 - lastZooming);
        } else {
            lastZooming += 1;
            zoom(lastZooming);
            checkBoundaries();
            bikeSpeedPlot.redraw();
            pedalSpeedPlot.redraw();
            pedalTorquePlot.redraw();
            motorTorquePlot.redraw();
        }
    } else if (mode == TWO_FINGERS_DRAG) {
        final float oldDist = distBetweenFingers;
        if(event.getPointerCount() >= 2) {
            distBetweenFingers = spacing(event);
            //System.out.println("ACTION_MOVE ---> " +
distBetweenFingers);
            lastZooming = oldDist / distBetweenFingers;
            zoom(lastZooming);
            checkBoundaries();
            bikeSpeedPlot.redraw();
            pedalSpeedPlot.redraw();
            pedalTorquePlot.redraw();
            motorTorquePlot.redraw();
        }
    }
}
break;
}
return true;
}

private void zoom(float scale) {
    final float domainSpan = maxXY.x - minXY.x;
    final float domainMidPoint = maxXY.x - domainSpan / 2.0f;
    final float offset = domainSpan * scale / 2.0f;
    minXY.x = domainMidPoint - offset;
    maxXY.x = domainMidPoint + offset;
}

private void scroll(float pan) {
    final float domainSpan = maxXY.x - minXY.x;
    final float step = domainSpan / bikeSpeedPlot.getWidth();
    final float offset = pan * step;
    minXY.x += offset;
    maxXY.x += offset;
}

private float spacing (MotionEvent event) {
    final float x = event.getX(0) - event.getX(1);
    //System.out.println("getX(0): " + event.getX(0) + "   getX(1): " +
event.getX(1));
    final float y = event.getY(0) - event.getY(1);
    //System.out.println("getY(0): " + event.getY(0) + "   getY(1): " +
event.getY(1));
    return FloatMath.sqrt(x*x + y*y);
}

private void checkBoundaries() {
    if (minXY.x < absMinX) minXY.x = absMinX;
}

```

```

        else if (minXY.x > maxNoError) minXY.x = maxNoError;
        if (maxXY.x > absMaxX) maxXY.x = absMaxX;
        else if (maxXY.x < minNoError) maxXY.x = minNoError;
        if (maxXY.x - minXY.x < minDif) maxXY.x = maxXY.x + (float) (minDif -
(maxXY.x - minXY.x));
        bikeSpeedPlot.setDomainBoundaries(minXY.x, maxXY.x, BoundaryMode.AUTO);
        pedalSpeedPlot.setDomainBoundaries(minXY.x, maxXY.x, BoundaryMode.AUTO);
        pedalTorquePlot.setDomainBoundaries(minXY.x, maxXY.x, BoundaryMode.AUTO);
        motorTorquePlot.setDomainBoundaries(minXY.x, maxXY.x, BoundaryMode.AUTO);
    }

    @Override
    public void update(short[] data, int reg) {
        // nothing to do
    }
}

```

Ontología para el tratamiento de los datos procedentes de la bicicleta:

```

package com.Android.upc.tablet;

import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

public class DataParserGui {

    /* ----- BIKE -----*/
    static final String MotorSpeed = "MotorSpeed";
    static final String PedalSpeed = "PedalSpeed";
    static final String BikeSpeed = "BikeSpeed";
    static final String MotorTemp = "MotorTemp";
    static final String ExternalTemp = "ExternalTemp";
    static final String BatteryVoltage = "BatteryVoltage";
    static final String BatteryCurrent = "BatteryCurrent";
    static final String Mode = "Mode";
    static final String Accelerator = "Accelerator";
    static final String LeftBrake = "LeftBrake";
    static final String RightBrake = "RightBrake";
    static final String DistTraveled = "DistTraveled";
    static final String PedalTorque = "PedalTorque";
    static final String MotorTorque = "MotorTorque";

    public static List<iWalkerStatus> readConfig(String configFile){
        List<iWalkerStatus> items = new ArrayList<iWalkerStatus>();

        FileInputStream fis = null;
        try {
            fis = new FileInputStream(configFile);
        } catch (FileNotFoundException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
        }
        DataInputStream dis = new DataInputStream(fis);
        BufferedReader br = new BufferedReader(new InputStreamReader(dis));
        String strLine;
        try {

```

```

        while ((strLine = br.readLine()) != null) {
            iWalkerStatus item = new iWalkerStatus();
            String[] data = strLine.split(",");
            /* ----- BIKE ----- */
            item.setMotorSpeed(Double.parseDouble(data[0]));
            item.setPedalSpeed(Double.parseDouble(data[1]));
            item.setBikeSpeed(Double.parseDouble(data[2]));
            item.setMotorTemp(Double.parseDouble(data[3]));
            item.setExternalTemp(Double.parseDouble(data[4]));
            item.setBatteryVoltage(Double.parseDouble(data[5]));
            item.setBatteryCurrent(Double.parseDouble(data[6]));
            item.setMode(Double.parseDouble(data[7]));
            item.setAccelerator(Double.parseDouble(data[8]));
            item.setLeftBrake(Double.parseDouble(data[9]));
            item.setRightBrake(Double.parseDouble(data[10]));
            item.setDistTraveled(Double.parseDouble(data[11]));
            item.setPedalTorque(Double.parseDouble(data[12]));
            item.setMotorTorque(Double.parseDouble(data[13]));
            items.add(item);
        }
        dis.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    return items;
}
}

```

Fragment encargado de la grabación de los experimentos:

```

package com.Android.upc.tablet;

import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import android.app.Fragment;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.TextView;
import android.widget.Toast;

public class ExperimentFragment extends Fragment {
    public static final String DATE_FORMAT_NOW = "yyyy-MM-dd HH:mm:ss";
    private boolean grabando=false;
    private File file;

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                           Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.experiment, container, false);
    }
}

```

```

public void setGrabando(boolean b){
    grabando=b;
}

public boolean getGrabando() {
    return grabando;
}

public void grabarLog() {

    if (!grabando){
        EditText nombre = (EditText) getActivity().findViewById(R.id.nombre_experimento);

        File path = getActivity().getExternalFilesDir(Environment.DIRECTORY_DOWNLOADS);
        String date = DateUtils.now();
        String name = date + "_" + nombre.getText() + ".txt";
        file = new File(path, name);

        Toast.makeText(getActivity().getApplicationContext(), "Log experiment: " + name , Toast.LENGTH_SHORT).show();

    } else {
        Toast.makeText(getActivity().getApplicationContext(), "End Log experiment" , Toast.LENGTH_SHORT).show();
    }
    grabando=!grabando;
}

public void repintar(){
    if (grabando) {
        try{
            InputStream is = new ByteArrayInputStream(TabletActivity.packet.getBytes("UTF-8"));
            OutputStream os = new FileOutputStream(file, true); // MODE_APPEND = ON
            byte[] data = new byte[is.available()];

            is.read(data);
            os.write(data);
            is.close();
            os.close();
        } catch (IOException e) {
            Log.w("ExternalStorage", "Error writing " + file, e);
        }
    }
}

boolean hasExternalStoragePrivatePicture() {
    File path = getActivity().getExternalFilesDir(Environment.DIRECTORY_DOWNLOADS);
    if (path != null) {
        EditText nombre = (EditText) getActivity().findViewById(R.id.nombre_experimento);
        String name = nombre.getText() + ".iwl";
        File file = new File(path, name);
        return file.exists();
    }
    return false;
}

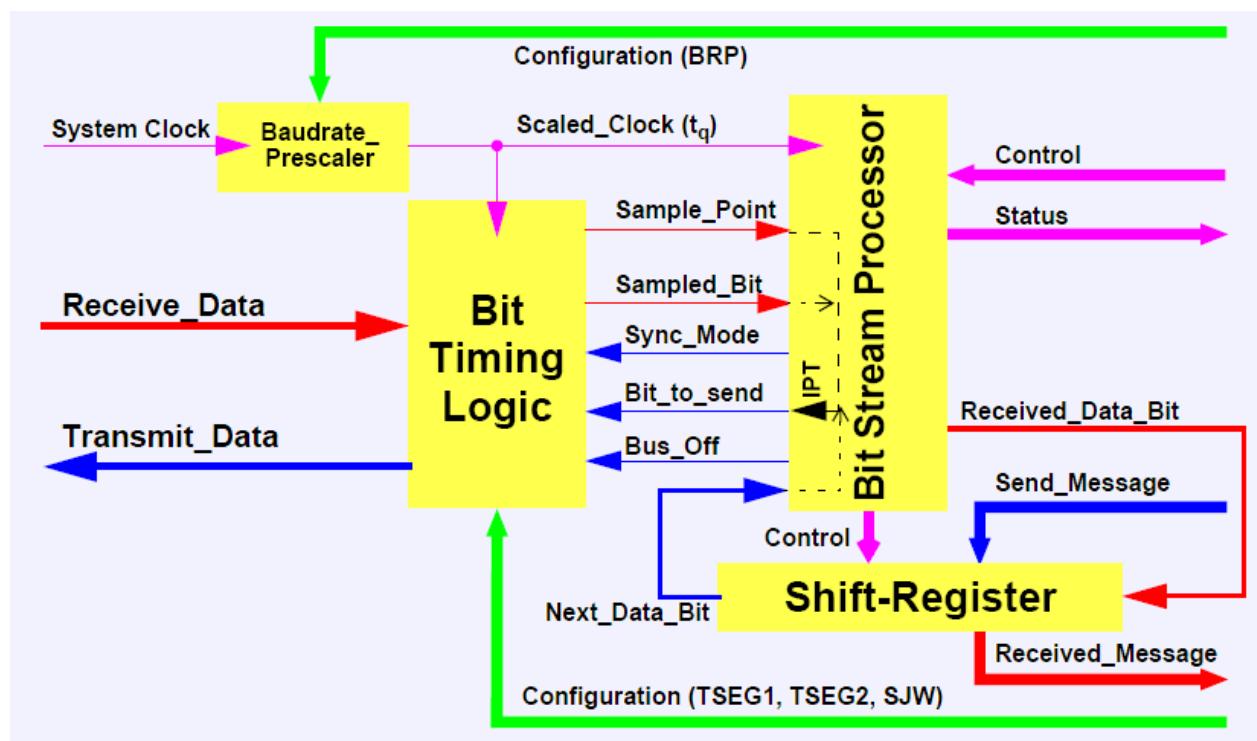
```

Anexo 3: Bus CAN

CAN (acrónimo del inglés Controller Area Network) es un protocolo de comunicaciones desarrollado por la firma alemana Robert Bosch GmbH, basado en una topología bus para la transmisión de mensajes en entornos distribuidos. Además ofrece una solución a la gestión de la comunicación entre múltiples CPUs (unidades centrales de proceso).

El protocolo de comunicaciones CAN proporciona los siguientes beneficios:

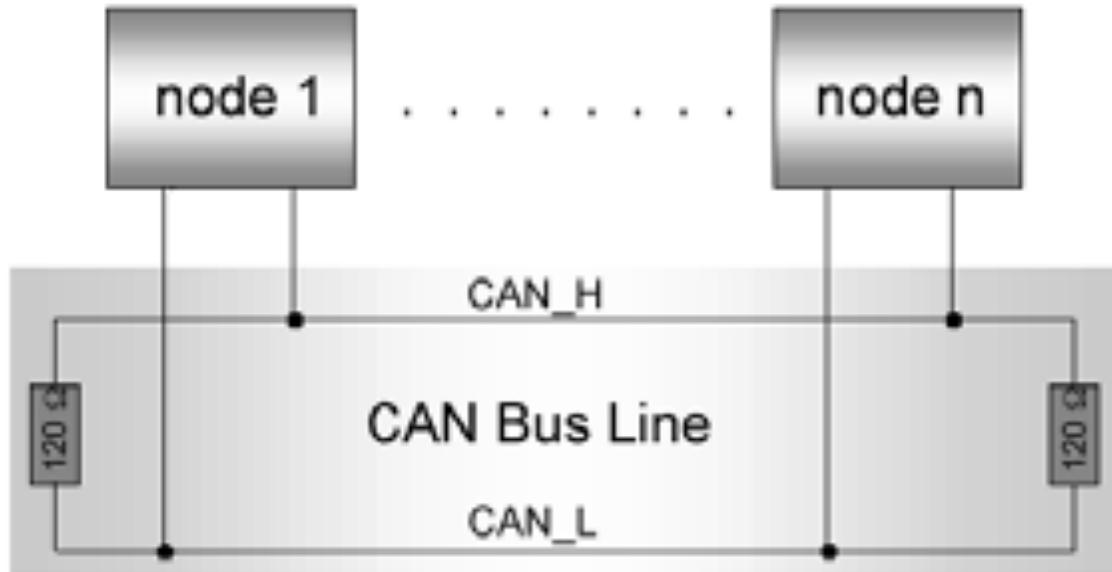
- Es un protocolo de comunicaciones normalizado, con lo que se simplifica y economiza la tarea de comunicar subsistemas de diferentes fabricantes sobre una red común o bus.
- El procesador anfitrión (host) delega la carga de comunicaciones a un periférico inteligente, por lo tanto el procesador anfitrión dispone de mayor tiempo para ejecutar sus propias tareas.
- Al ser una red multiplexada, reduce considerablemente el cableado y elimina las conexiones punto a punto, excepto en los enganches.



Principales características de CAN

CAN se basa en el modelo productor/consumidor, el cual es un concepto, o paradigma de comunicaciones de datos, que describe una relación entre un productor y uno o más consumidores. CAN es un protocolo orientado a mensajes, es decir la información que se va a intercambiar se descompone en mensajes, a los cuales se les asigna un identificador y se encapsulan en tramas para su transmisión. Cada mensaje tiene un identificador único dentro de la red, con el cual los nodos deciden aceptar o no dicho mensaje. Dentro de sus principales características se encuentran:

- Prioridad de mensajes.
- Garantía de tiempos de latencia.
- Flexibilidad en la configuración.
- Recepción por multidifusión (multicast) con sincronización de tiempos.
- Sistema robusto en cuanto a consistencia de datos.
- Sistema multimaestro.
- Detección y señalización de errores.
- Retransmisión automática de tramas erróneas
- Distinción entre errores temporales y fallas permanentes de los nodos de la red, y desconexión autónoma de nodos defectuosos.



Esquema de cableado del bus CAN

CAN fue desarrollado, inicialmente para aplicaciones en los automóviles y por lo tanto la plataforma del protocolo es resultado de las necesidades existentes en el área de la automoción. La Organización Internacional para la Estandarización (ISO, International Organization for Standardization) define dos tipos de redes CAN: una red de alta velocidad (hasta 1 Mbps), bajo el estándar ISO 11898-2, destinada para controlar el motor e interconectar las unidades de control electrónico (ECU); y una red de baja velocidad tolerante a fallos (menor o igual a 125 Kbps), bajo el estándar ISO 11519-2/ISO 11898-3, dedicada a la comunicación de los dispositivos electrónicos internos de un automóvil como son control de puertas, techo corredizo, luces y asientos.

Protocolo de comunicaciones CAN

CAN es un protocolo de comunicaciones serie que soporta control distribuido en tiempo real con un alto nivel de seguridad y multiplexación.

El establecimiento de una red CAN para interconectar los dispositivos electrónicos internos de un vehículo tiene la finalidad de sustituir o eliminar el cableado. Las ECUs, sensores, sistemas antideslizantes, etc. se conectan mediante una red CAN a velocidades de transferencia de datos de hasta 1 Mbps.

De acuerdo al modelo de referencia OSI (Open Systems Interconnection, Modelo de interconexión de sistemas abiertos), la arquitectura de protocolos CAN incluye tres capas: física, de enlace de datos y aplicación, además de una capa especial para gestión y control del nodo llamada capa de supervisor.

Capa física: define los aspectos del medio físico para la transmisión de datos entre nodos de una red CAN, los más importantes son niveles de señal, representación, sincronización y tiempos en los que los bits se transfieren al bus. La especificación del protocolo CAN no define una capa física, sin embargo, los estándares ISO 11898 establecen las características que deben cumplir las aplicaciones para la transferencia en alta y baja velocidad.

Capa de enlace de datos: define las tareas independientes del método de acceso al medio, además debido a que una red CAN brinda soporte para procesamiento en tiempo real a todos los sistemas que la integran, el intercambio de mensajes que demanda dicho procesamiento requiere de un sistema de transmisión a frecuencias altas y retrasos mínimos. En redes multamaestro, la técnica de acceso al medio es muy importante ya que todo nodo activo tiene los derechos para controlar la red y acaparar los recursos. Por lo tanto la capa de enlace de datos define el método de acceso al medio así como los tipos de tramas para el envío de mensajes

Cuando un nodo necesita enviar información a través de una red CAN, puede ocurrir que varios nodos intenten transmitir simultáneamente. CAN resuelve lo anterior al asignar prioridades mediante el identificador de cada mensaje, donde dicha asignación se realiza durante el diseño del sistema en forma de números binarios y no puede modificarse dinámicamente. El identificador con el menor número binario es el que tiene mayor prioridad.

El método de acceso al medio utilizado es el de Acceso Múltiple por Detección de Portadora, con Detección de Colisiones y Arbitraje por Prioridad de Mensaje (CSMA/CD+AMP, Carrier Sense Multiple Access with Collision Detection and Arbitration Message Priority). De acuerdo con este método, los nodos en la red que necesitan transmitir información deben esperar a que el bus esté libre (detección de portadora); cuando se cumple esta condición, dichos nodos transmiten un bit de inicio (acceso múltiple). Cada nodo lee el bus bit a bit durante la transmisión de la trama y comparan el valor transmitido con el valor recibido; mientras los valores sean idénticos, el nodo continúa con la transmisión; si se detecta una diferencia en los valores de los bits, se lleva a cabo el mecanismo de arbitraje.

CAN establece dos formatos de tramas de datos (data frame) que difieren en la longitud del campo del identificador, las tramas estándares (standard frame) con un identificador de 11 bits definidas en la especificación CAN 2.0A, y las tramas extendidas (extended frame) con un identificador de 29 bits definidas en la especificación CAN 2.0B.

Para la transmisión y control de mensajes CAN, se definen cuatro tipos de tramas: de datos, remota (remote frame), de error (error frame) y de sobrecarga (overload frame). Las tramas remotas también se establecen en ambos formatos, estándar y extendido, y tanto las tramas de

datos como las remotas se separan de tramas precedentes mediante espacios entre tramas (interframe space).

En cuanto a la detección y manejo de errores, un controlador CAN cuenta con la capacidad de detectar y manejar los errores que surjan en una red. Todo error detectado por un nodo, se notifica inmediatamente al resto de los nodos.

Capa de supervisor: La sustitución del cableado convencional por un sistema de bus serie presenta el problema de que un nodo defectuoso puede bloquear el funcionamiento del sistema completo. Cada nodo activo transmite una bandera de error cuando detecta algún tipo de error y puede ocasionar que un nodo defectuoso pueda acaparar el medio físico. Para eliminar este riesgo el protocolo CAN define un mecanismo autónomo para detectar y desconectar un nodo defectuoso del bus, dicho mecanismo se conoce como aislamiento de fallos.

Capa de aplicación: Existen diferentes estándares que definen la capa de aplicación; algunos son muy específicos y están relacionados con sus campos de aplicación. Entre las capas de aplicación más utilizadas cabe mencionar CAL, CANopen, DeviceNet, SDS (Smart Distributed System), OSEK, CANKingdom.

