UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Diseño y construcción de un prototipo para monitoreo con interfaz en smartphone de una bicicleta eléctrica

TESIS

Que para obtener el título de

Ingeniero eléctrico electrónico

P R E S E N T A

Michael Ivan Cedillo Lira

DIRECTOR DE TESIS

Dr. Jaime Baltazar Morales Sandoval

Ciudad Universitaria, Cd. Mx., 2019

Designación de sinodales de examen profesional

Presidente: Ing. Alejandro Sosa Fuentes

Vocal: Dr. Jaime Baltazar Morales Sandoval

Secretario: M.I. Jesús Álvarez Castillo

1er. Suplente: M.I. Ricardo Mota Marzano

2do. Suplente: Dr. Saúl de la Rosa Nieves

Agradecimientos

De todo corazón a mis padres, por haberme dado tanto, por ser el más grande motivo de que me encuentre ahora escribiendo estas líneas, en este punto tan importante de mi vida, de la cual les estaré eternamente en deuda.

A mis abuelos, por ser el más grande ejemplo de humildad, trabajo y perseverancia. Y de quienes no he carecido ni un segundo de sus atenciones.

A mis hermanos, por el aliento que me dan al demostrarme que están orgullos de mí.

A mi director de tesis, por su inagotable paciencia, por las largas horas de platica dedicadas a formarme de la mejor manera, tanto como ingeniero como persona.

A mis sinodales, quienes en su momento fueron también mis profesores, por darme el honor de revisar y corregir mi trabajo, así como también de contribuir enormemente en los conocimientos y desarrollo académico que ahora poseo.

A cada uno de los colegas de clases, con quienes compartí notas, tareas y trabajos. Pero en especial a mi fraternidad "Deshfashamiento", por caminar juntos esta dura y larga carrera cuesta arriba, por no dejar a nadie solo cuando se le necesitara, por empujarnos los unos a los otros, por ser alegres y llegar a la cima juntos.

Al equipo de Buceo de la UNAM, por instruirme en tan fascinante actividad deportiva, por ser una escapada hacia lo profundo cuando las actividades en superficie se tornaban agobiantes. Por darme amigos de otros círculos y mostrarme un mundo lleno de color.

Al moto club "Super biker's" por recorrer tantas y tantas carreteras en su compañía. Por dejarme formar parte de la camaradería más leal que he conocido.

A cada una de las personas que este leyendo esto… ¡Gracias!

Resumen

Como resultado de un estudio, que realizamos como parte de este proyecto, sobre las características de las bicicletas eléctricas y dispositivos relacionados, ofertadas en la ciudad de México a finales del año 2017, entendimos que existe la necesidad de que éstas se equipen con un sistema de adquisición de datos confiable que permita el uso seguro y cómodo del vehículo. En el mercado existen tanto dispositivos muy sofisticados capaces de medir muchas de las variables involucradas en el uso de las bicicletas, principalmente las utilizadas en el deporte de alto rendimiento, así como sistemas muy simples, que solo miden los giros que da la rueda. Sin embargo, existe un nicho de oportunidad para sistemas como el descrito en este proyecto.

La oportunidad de poder desarrollar un sistema intermedio, que permita registros confiables del uso energético enfocado en bicicletas tanto eléctricas como convencionales, con materiales de fácil adquisición en México y bajo costo. Que mida parámetros como: voltaje, corriente, temperaturas y velocidades de rotación de los ejes de pedaleo y rueda.

Por ser un vehículo eléctrico, los principales parámetros que deseamos medir son el voltaje y la corriente de consumo. Con ambas se podrá calcular posteriormente parámetros como la potencia instantánea, carga en la batería y autonomía. Medir los giros de la rueda es indispensable para conocer velocidad y distancia recorrida, pero hemos agregado un sensor de giro más, ubicado en eje del pedaleo. Con esto podremos después conocer la cadencia de pedaleo y un estimado de la energía aportada por el usuario y el motor. Como sensores adicionales que nos puedan proveer información para el uso seguro y correcto del vehículo. Instalamos cuatro sensores de temperatura, el primero medirá la temperatura ambiental que nos sirve de referencia. Los otros tres, medirán las temperaturas de los tres principales componentes de la bicicleta eléctrica, el motor, el controlador, y la batería. Que como sabremos más adelante, soportan potencias de hasta 1,100 [W] a 53 [VDC] y 27 [A].

Toda esta información será recibida por un microcontrolador, que después la enviará vía Bluetooth a un dispositivo móvil (Smartphone) el cual la despliega para el usuario. Por lo tanto, también se desarrolla un software para el dispositivo capaz de recibir, decodificar, procesar y desplegar los datos recibidos.

Esto último se planteó como la característica más atractiva del sistema, pues nos encontramos en una etapa industrial en donde todo se puede conectar con todo (Internet de las cosas). Y al tener un hardware lo suficientemente robusto instalado en

el vehículo, los trabajos a futuro tanto en actualización como en funciones serían mucho más fácil de aplicar únicamente en el software de la aplicación.

Contenido

INDICE DE TABLAS

INDICE DE FIGURAS

INDICE DE ECUACIONES

Antecedentes

En los últimos años los motores HUB (termino en ingles que hace referencia a motores colocados en la misma masa de la rueda) son la solución más frecuente encontrada en vehículos eléctricos ligeros, como bicicletas y triciclos. Principalmente por su economía, versatilidad y bajo mantenimiento.

Los controladores para estos motores también son económicos y su funcionalidad y características dependen de su costo. Las baterías son los componentes que entre más ligeros y con mejores prestaciones se deseen, más onerosos resultan en este momento.

Los modelos económicos carecen de una pantalla que muestre información confiable referente a la carga actual de la batería, velocidad instantánea, distancia recorrida y otras funciones básicas. El conocer cómo se evoluciona la batería en corriente y voltaje, así como la respuesta de las partes mecánicas del vehículo y del ciclista o usuario, son de gran interés no solo para optimizar el funcionamiento de la bicicleta sino el posicionamiento y cadencia del ciclista. Prácticamente ningún sistema de interfaz gráfica o *display* indica las condiciones del vehículo y/o componentes, así como tampoco un estimado de energía consumida entre el vehículo y el usuario.

Planteamiento del problema

Los sistemas comerciales actuales disponibles en México son de baja resolución y en general costosos. La reparación o mantenimiento de estos no están disponibles en cualquier tienda mexicana. Por ello se considera bastante atractivo analizar la construcción de un sistema que pudiese comercializarse en nuestro entorno y que sea de mejor acceso para nuestros usuarios.

OBJETIVOS

Proponer, diseñar y construir la instrumentación que realice el monitoreo del estado de una bicicleta tradicional o bien asistida con un motor eléctrico; una aplicación para Smartphone que monitoree el vehículo y procese los datos, y también un sistema inalámbrico que los acople:

1) **El sistema deberá medir continuamente la velocidad de desplazamiento del vehículo, velocidad de pedaleo, temperatura ambiental, del driver, del motor y de la batería, *voltaje* de la batería y corriente de consumo.**
2) **La aplicación mostrará de manera sencilla los parámetros de velocidad y nivel de la batería e incluirá funcionas específicas tales como: calorías quemadas, avisos de temperaturas críticas de los componentes y un registro histórico de las distancias recorridas.**

ALCANCE

Se pretende que los dispositivos usados para la obtención de datos sean los que se encuentran actualmente en el mercado y de bajo costo, como lo son los *Shields* ya desarrollados y listos para usarse en diferentes microcontroladores (Mc). Así mismo el Mc deberá ser también de bajo costo y contar con las capacidades indispensable para la realización de este proyecto como pueden ser: número de puertos, velocidad de procesamiento, canales ADC, etc.

En cuanto a la interfaz, ésta se desarrollará para dispositivos Android por su mayor popularidad en el mercado. Dicha aplicación será programada en la plataforma de App Inventor para aprovechar las ventajas y simplificación que ofrece hacia los desarrolladores, además de que es de acceso libre. El objetivo es poder ofrecer en el mercado nacional un producto con las características generales descritas pero que pueda mantenerse con elementos de fácil reemplazo en México.

CAPITULO 1 BICICLETAS ELÉCTRICAS

1.1 Situación actual y futura de su aplicación

Con el gran crecimiento poblacional en las principales urbes alrededor del mundo, el problema de la movilidad se ha convertido en uno de los grandes temas a tratar por los gobiernos, así como la contaminación ambiental que esto conlleva por tratarse en su gran mayoría de vehículos impulsados por combustibles fósiles. Agregando a esto el encarecimiento de las gasolinas y de más servicios asociados a los automotores como son impuestos, estacionamiento, verificaciones, etc. Han hecho que tanto gobiernos como ciudadanos promuevan el uso de auto transportes ecológicamente limpios y económicos, tal como lo es la bicicleta.

En la CDMX, son cada vez más los recursos destinados para el desarrollo de la infraestructura ciclista, como lo son ciclovías, carriles confinados, estacionamientos, espacios exclusivos en cruceros, y políticas de cultura vial.

Una importante categoría de estos vehículos y sobre la cual trabajaremos en la presente tesis es la bicicleta eléctrica. Estas combinan algunas ventajas tanto de las bicicletas convencionales como de los vehículos eléctricos, como es el bajo costo de auto transporte, típico de los vehículos de dos ruedas y la posibilidad de la asistencia eléctrica tanto en rebases, pendientes y durante fatiga o cansancio. Y la posibilidad de regeneración de energía aprovechando los procesos de frenado y descenso de pendientes.

[i]De acuerdo a un estudio publicado por la *Universidad de Rusia y el Departament of Engines and Vehicles titulado "A study on electric bicycle energy efficiency"* en donde se somete a diversas pruebas una bicicleta eléctrica convencional, es decir con motor *Brushless* DC de 500 [W], batería de iones de litio de 36[V] y 9 [Ah], con un peso total de 24,4 [Kg] y un sistema capaz de recargar energía durante el frenado y descenso, concluyen lo siguiente:

- **Destacan que la regeneración de energía está íntimamente ligado a las condiciones de movimiento y las pendientes de las calles, oscilando entre el 6 y 14 % de recarga.**
- **En un descenso controlado a 25 [km/h], durante 60 s, en la batería se regeneraron ~0.17 [Ah], que en una batería de 9 [Ah] representa un ~2%.**

- **Para los experimentos realizados obtuvieron un promedio de autonomía de 34.77 [Km] en condiciones urbanas y sin regeneración de energía. Velocidad máxima de 35.4 [km/h]. Y una eficiencia energética de 11.2 [Wh/Km].**

En resumen, el estudio muestra que la bicicleta eléctrica podría reducir la contaminación del aire en hasta 10 veces comparado con otros vehículos eléctricos monoplaza o hasta en 15 veces con un automóvil tradicional.

Por último y no menos importante, muestra que a velocidades de 15 a 35 [Km/h], el motor funciona entre 100 y 300 [W], teniendo un consumo energético de 7 a 12 [Wh/Km], lo que representa 6 a 23 veces menos que el consumo de energía de los automóviles eléctricos actuales.

MODELO	CONSUMO DE ENERGÍA POR KILÓMETRO [Wh/Km]
1997 GM EV1 (Lead Acid)	102
1996 Toyota RAV4 EV (Lead Acid)	146
2014 BMW i3 BEV	168
2015 VW e-Golf	180
2015 Nissan Leaf	187
2015 Tesla Model S 85D	211

Tabla 1 [ii]Diferentes consumos de energía para vehículos eléctricos

[iii]Las emisiones de contaminación que se le pueden asociar a un vehículo eléctrico radican únicamente en la recarga de las baterías. Tal energía en el peor de los casos provendría de una central eléctrica de carbón y aun así generaría mucho menos emisiones que las asociadas a un automotor de gasolina. Una simulación del año 1995, en la que se reemplazaron todos los vehículos de gasolina por eléctricos, demostró que se podrían reducir enormemente las emisiones: NMOG 98%, NOX 92%, CO 99%. Además de que las personas de las ciudades no serían expuestas a las emisiones de las centrales eléctricas, ya que estas no se encuentran en las áreas metropolitanas, además de que es más viable que instalarles sistemas de reducción de emisores de contaminantes a cada vehículo.

Si a todo lo anterior le sumamos el rápido crecimiento que están teniendo las tecnologías de las baterías y el abaratamiento de costos de motores eléctricos importados del extranjero (principalmente China y de lo cual hablaré más adelante), resulta evidente el gran impacto que se espera tendrán estos vehículos, una vez que se dispongan de ciclovías más seguras.

Un mundo conectado

En los últimos años, hemos conocido una nueva forma en que los dispositivos electrónicos han sido diseñados para satisfacer los gustos y necesidades de la población. Es lo que conocemos como "Internet de las cosas" (o "IoT" por sus siglas en ingles). Este fenómeno ha llegado a tal punto, que hoy casi cualquier cosa se puede conectar a la red, ya sean TV, relojes, refrigeradores, video cámaras de seguridad, automóviles, etc. Y las bicicletas no podrían ser la excepción. [iv]El ciclismo se ha convertido en uno de los mayores deportes urbanos, al mismo tiempo con el desarrollo de IoT y de los sistemas embebidos, ambos se han fusionado para proveer al ciclista información durante el ejercicio y un análisis post-ejercicio.

Estos sistemas adquieren datos en tiempo real de las condiciones de movimiento del vehículo mediante sensores integrados en el dispositivo (GPS, acelerómetros, Giroscopios) o usando los sensores de un teléfono inteligente (en algunos casos se puede medir la cadencia de pedaleo[v]). Después de la ruta el ciclista, podría ver información como ruta tomada, distancia total, calorías quemadas, así como también la evolución durante cada parte del recorrido de cantidades como velocidad, cadencia, aceleración, frenado, cambio de dirección y pendientes. El ciclista no solo podrá analizar sus resultados, sino que también los comparte en la red.

Es importante mencionar que dichos sistemas están enfocados en el análisis del rendimiento deportivo de un ciclista, es decir a un público muy específico. Y que tales sistemas son únicamente para bicicletas tradicionales y no eléctricas. Por lo que resulta muy atractivo construir un sistema dirigido a dichos vehículos y a un público más general.

1.1.1 Características y Componentes del vehículo

Una bicicleta eléctrica prácticamente se compone de 4 partes principales: bicicleta, motor eléctrico, batería y controlador.

La bicicleta puede ser de cualquier tipo (urbana, montaña, ruta, etc.). Actualmente en el mercado se ofrecen bicicletas eléctricas de todos los tipos, pero son las urbanas y de montaña las que más importancia tienen en este segmento. A continuación, muestro una tabla con las diferentes bicicletas ofrecidas en la CDMX, sus características y precios. Esta tabla fue elaborada en una investigación en conjunto con el LAIRN

(Laboratorio de investigación de reactores nucleares) y en ella se observan BE (Bicicletas eléctricas) de las 4 principales empresas de este sector en la ciudad.

MODELO	POTENCIA Y MOTOR	AUTONOMIA km	BATERIA	OTRAS CARACTERISTICAS	PRECIO MXN
EASY MOTION [vi]					
EVO 27.5	BLDC HUB 350w	85	Litio 435Wh	montaña	48,965
GO RACE	250 W BLDC con sistema de engranes	45	Litio 216Wh	Urbana de velocidad	25,404
GO STRET	250 W DC con sistema de engranes	45	Litio 216Wh	Urbana y compacta	27,434
EVO JUMPER	350 W BLDC HUB	105	Litio 460Wh	Montaña, batería dentro del chasis	58,178
EASY EVO VOLT	350 W BLDC HUB	65	Litio 316Wh	Plegable, urbana, compacta	26,605
PRODECOTECH [vii]					
GENESIS V5	600 W BLDC HUB	40	Litio 518Wh	Urbana, batería integrada en el chasis	50,992
MARINER 8 V5	300W BLDC HUB	40	Litio 360Wh	Urbana, plegable, compacta	43,342
OUTLAW SS	750W BLDC HUB	27	Litio 464Wh	Montaña	43,342
STRIDE 500 V5	500W BLDC HUB	40	Litio 418Wh	Urbana	45,892
ELECTROBIKE [viii]					

TRIP	250 W BLDC HUB	15-25	Acido-plomo	Urbana	12,900
CLICK	250 W BLDC HUB	20-25	Ion litio	Urbana plegable	13,900
DASH	350 W BLDC en el eje de pedaleo	20-40	Ion litio	Urbana, batería oculta en el cuadro	18,900
STEEL	350 W BLDC HUB	20-40	Ion litio	Montaña	19,900
DELTA	350 W BLDC HUB	20-30	Ion litio	Urbana, compacta, plegable	23,900
MAGNOS	350 W BLDC HUB	20-40	Ion litio	Urbana, compacta, plegable	28,900
CROSS	350 W BLDC HUB	20-40	Ion litio	Montaña, batería oculta en el cuadro	30,900
SEAL 500	500 W BLDC HUB	20-40	Ion litio	montaña	34,900
ELECTROPEDALEO[ix]					
SUMMIT	350 W BLDC HUB	30	Litio 216Wh	Montaña	10,100
PRO LIGHT	1000 W BLDC HUB	30	Litio 396Wh	Montaña	17,300
FAT TIRE	1500 W BLDC HUB	30	Litio 528Wh	Urbana	20,400

Tabla 2 Lista de bicicletas eléctricas ofertadas en la CDMX 2017[x]

De la tabla 2 podemos destacar algunas cosas como son:

- **Los precios varían de entre $10,100 y $58,178 MXN**
- **Las potencias van desde 250 [W] hasta 1500 [W]**
- **La autonomía promedio es de 36.8 [KM]**
- **El 100% de los motores son BLDC de estos 85% son HUB**
- **El 95% de las baterías son de Litio con una capacidad promedio de 379[Wh]**

1.2 Motores eléctricos en bicicletas.

En bicicletas eléctricas comerciales podemos encontrar básicamente dos configuraciones de motores. Ambas utilizan motores de corriente directa sin escobillas (BLDC) por las ventajas que explicaremos a continuación y diferenciándose únicamente por la posición del motor y la forma en que este aplica la fuerza de movimiento.

1.2.1 El motor BLDC de imanes permanentes

[xi]Las ventajas que representa usar un motor BLDC son las siguientes

- **Fácil control de variación de velocidad y dirección de giro**
- **Bajo mantenimiento al no contar con conmutadores físicos que causen desgaste (escobillas)**
- **La posibilidad de frenado regenerativo**
- **No generan interferencias de radio por conmutación**
- **Relación torque/corriente lineal**
- **Bajo costo de manufactura**
- **Mayor durabilidad y confiabilidad**

En estos motores tenemos imanes permanentes en el rotor, por lo tanto, el estátor es el encargado de generar el flujo de campo magnético necesario para producir el movimiento. En estos motores la corriente es activada en tres fases y conmutada por una serie de transistores excitados sincronizada mente respecto a la posición del rotor, dicha posición se conoce gracias al uso de sensores de efecto Hall colocados dentro del motor.

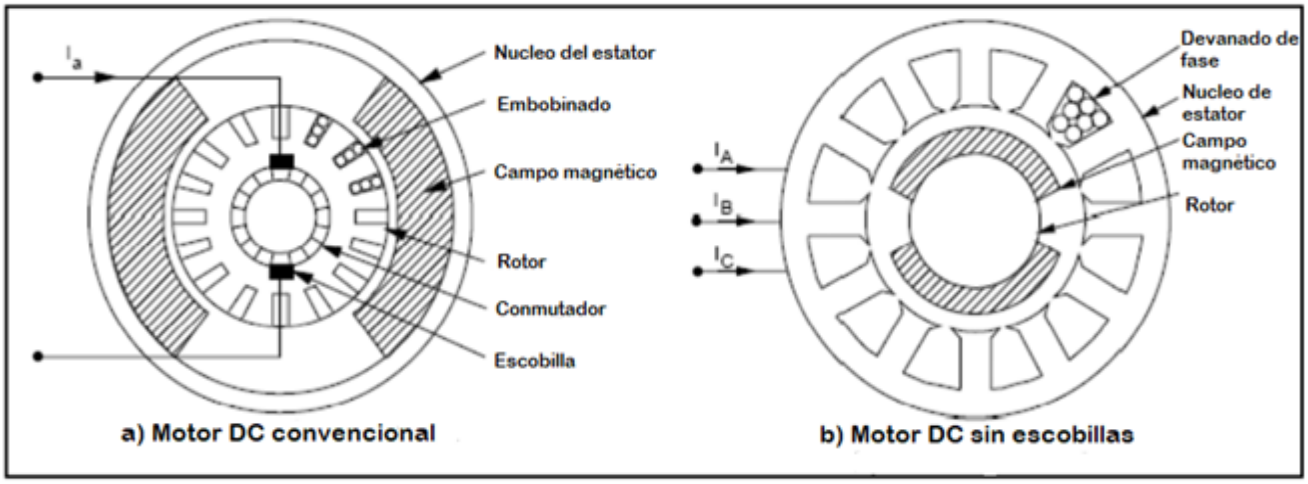


Figura 1.1 Comparación entre motor convencional y sin escobillas

Las dos configuraciones disponibles en el mercado para el posicionamiento del motor son las siguientes:

La primera es un motor BLDC posicionado cerca del mismo eje de pedaleo, y que transmite el torque mediante un sistema de engranes a este. [xii]Aprovechando así los propios cambios de marcha de la bicicleta o "sproks". Al hacer uso de la relación de cambios de la bicicleta, los hace más eficientes, pudiendo comparar el rendimiento de un motor central de 250 [W] con uno HUB de 500 [W]. Otra ventaja es que mantiene el centro de gravedad del vehículo cerca del centro geométrico, lo que representa una conducción más estable y segura, ideal para bicicletas de campo o de alto rendimiento. Sus principales desventajas es tener precios elevados y que el chasis debe ser diseñado para el motor en particular para poder acoplarse, lo que lo hace poco adaptable.

Figura 1.2 Motor eléctrico en posición central

El segundo tipo de sistemas son los motores BLDC acoplados en la misma rueda y son los denominados "HUB". Estos son de gran popularidad en el mercado por su bajo costo y sobre todo por la posibilidad de ser adaptado a cualquier medida de rueda y por consiguiente a cualquier bicicleta. Sin embargo, no está exento de desventajas.

Al estar colocado en el centro de la rueda este no cuenta con ningún tipo de amortiguación contra golpes causados por las imperfecciones del terreno, causando así mayor probabilidad de daños al sistema de rodamientos y rayos de la rueda, incluso al mismo motor. Es evidente que el centro de masas se desplaza hacia adelante o hacia atrás según sea el caso en donde se instale el motor, cambiando por completo las características de manejo típicas del vehículo. Incluso si se ha instalado en la rueda delantera, no se aprovechará su torque al máximo en una pendiente, pues esta pierde tracción.

Figura 1.3 Motor BLDC montado en rueda (HUB)

1.3 Controladores de potencia y sistemas típicos de una BE

El controlador es una de las partes más importantes de la BE pues es el encargado de enviar potencia eléctrica desde la batería hacia el motor. Esto lo hace mediante la interpretación de las señales provenientes de los sensores Hall colocados en diferentes ángulos en el motor para así determinar la posición del rotor y excitar los transistores correspondientes a la fase.

[xiii]Los impulsos de voltaje del motor BLDC deben aplicarse a las dos fases del sistema de embobinado trifásico para que el ángulo entre el estátor, el flujo y el rotor se mantengan cerca de 90 grados para así generar el par máximo del motor.

La figura 1.4 muestra la etapa de potencia estándar para la fase trifásica del motor BLDC.

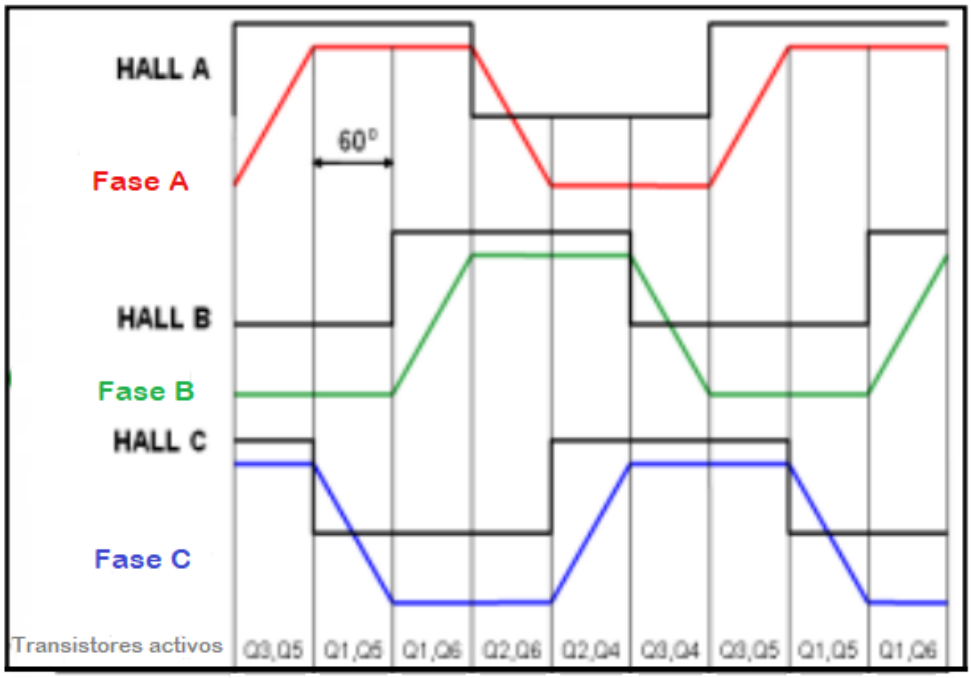


Figura 1.4 Diagrama de tiempos de disparo entre fases

La etapa de potencia del motor BLDC utiliza seis transistores para encender y desactivar las señales que se están entregando a cada fase individual del motor. Algún desplazamiento (offset) en las señales de sincronización arruinará los tiempos de disparo de los transistores, por lo tanto, el motor no funcionará a su máxima eficiencia.

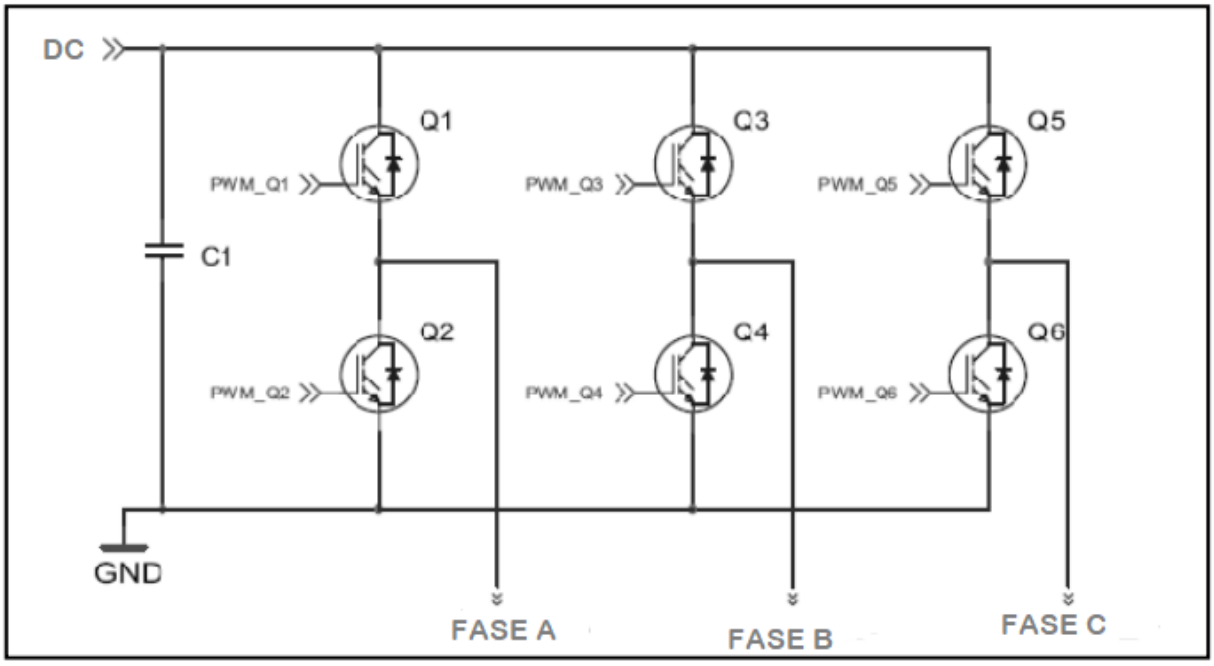


Figura 1.5 Etapa de potencia de un controlador para BLDC

El sistema general de un controlador BLDC está representado en la figura 1.6. Las entradas al controlador son las señales de corriente y velocidad que recibe del acelerador y el PAS (sistema de asistencia al pedaleo, por sus siglas en inglés), respectivamente. La fuente de alimentación de DC que alimenta al controlador. Los sensores de efecto Hall proporcionan la retro alimentación necesaria para que el controlador conozca la posición del rotor y para determinar cuándo suministrar el voltaje a las diferentes fases del motor BLDC. Las salidas son las 6 líneas de disparo a cada uno de los transistores de potencia.

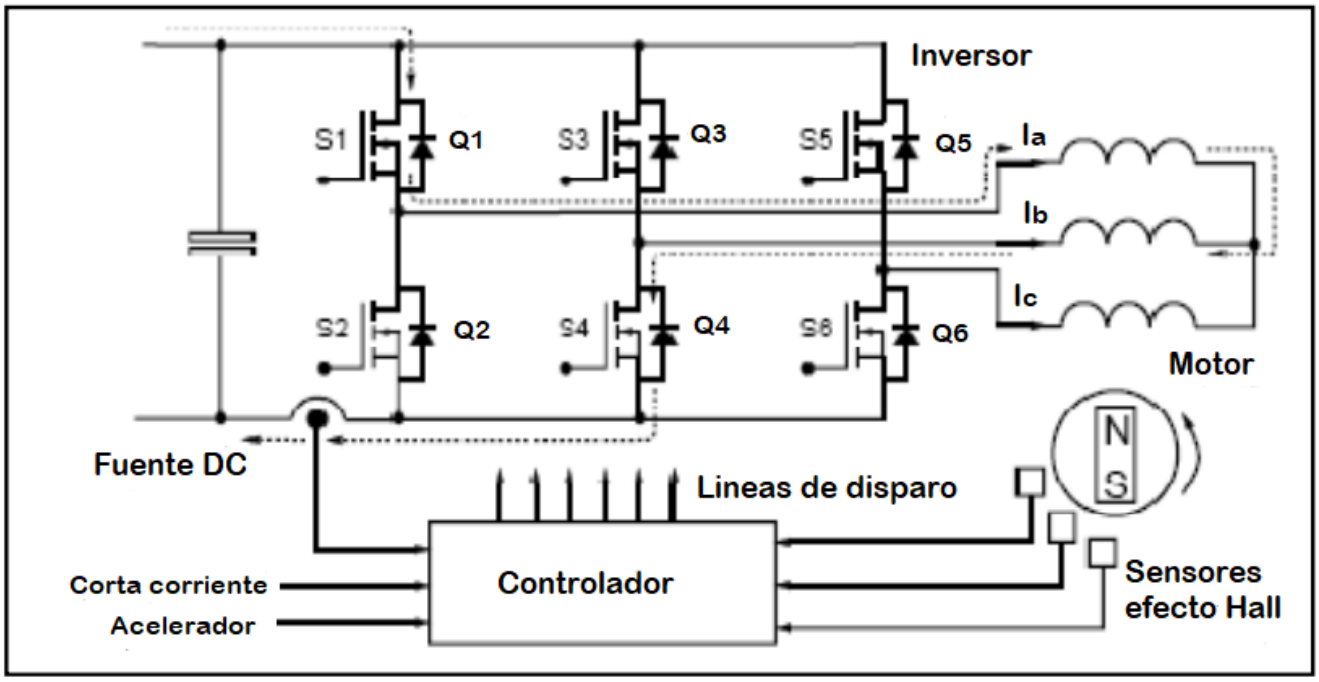


Figura 1.6 Sistema eléctrico de una bicicleta eléctrica

Para saber en qué momento acelerar el motor, el controlador debe recibir una señal del usuario que le indique el momento en que debe de entregar par y la cantidad. Esto se hace mediante un acelerador manual en el puño y/o el asistente de pedaleo (PAS).

El PAS no es más que un sensor de efecto Hall colocado en el eje de pedaleo que mide la velocidad de rotación del eje y con base en eso el controlador entrega la potencia indicada. En la figura 1.7 se observan las funciones básicas de un controlador típico.

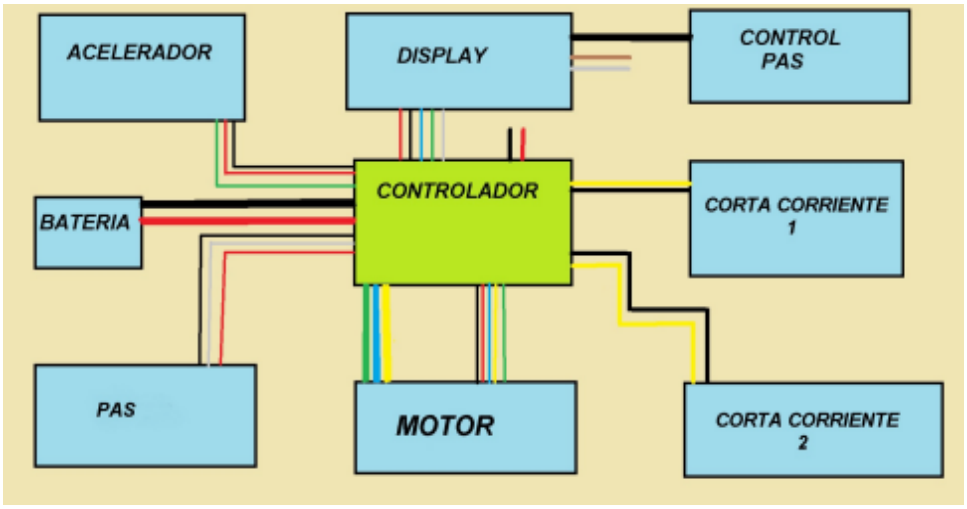


Figura 1.7 Funciones básicas de una BE de gama media

Dependiendo del costo y la marca del controlador este puede incluir otras características como son: indicador de nivel de batería simple (leds), cortacorrientes en las palancas de freno, diferentes niveles de asistencia de pedaleo e interruptor de luces. En algunos modelos de gama media se incluye un display digital que muestra velocidad, nivel de batería, distancia recorrida y nivel de PAS seleccionado. (ver figura 1.8)

Figura 1.8 Pantalla de informaciòn de una BE convencional

1.4 Baterías

Sin lugar a dudas la batería es la parte más costosa de un vehículo eléctrico, también es a la que más atención se le debe prestar a la hora de elegir un modelo, pues tiene varios parámetros fundamentales para un mejor aprovechamiento. Los cuales pueden ser: capacidad de almacenamiento de energía (Wh), material de fabricación, ciclos de carga útil (vida media), empaquetado (seguridad), peso, costo, etc.

Los distribuidores de BE no suelen ofrecer todos estos datos mencionados a excepción de la capacidad y material.

Como ya vimos en la investigación realizada, el 95% de las baterías son de iones de Litio con una capacidad media de 379 [Wh].

Estas baterías las podemos encontrar en paquetes de 24, 36 y 48 Volts con capacidades desde 9 a 14 [Amperes/hora].

Las razones por la que se usan baterías de iones de Litio son básicamente dos, su alta densidad energética y su bajo peso respecto a otros materiales.

Además, presenta la ventaja de poder obtener altos valores de voltajes en circuito abierto, lo que beneficia al transferir potencia en cantidades bajas de corriente.

Al ser el Litio un material altamente reactivo, este permite almacenar una gran cantidad de energía en sus enlaces atómicos, lo que lo convierte en la mejor configuración energía/peso.

[xiv]Su capacidad de retención de energía cuando no está en uso es muy alta, alcanzando valores de pérdida de energía del 5% al mes, que comparado con otras fórmulas como NiMH alcanzan el 20%, lo que representa cerca de 4 veces más retención.

A todo lo anterior agregamos que no tienen el llamado "efecto memoria", lo que significa que la batería no se tiene que descargar completamente para comenzar con el proceso de recarga y sin que esto afecte sus ciclos de vida.

Sin embargo, presentan una desventaja notoria. Su alta densidad energética las convierte en un objeto con el cual se debe tener mucha precaución, pues esta puede explotar o incendiarse si las condiciones no son las adecuadas, es decir, si es golpeada o sometida a altas temperaturas. Por lo que es imprescindible un monitoreo constante de las condiciones de la batería.

CAPITULO 2
DISEÑO DE LA INSTRUMENTACIÓN PARA ADQUISICIÓN DE DATOS DE LA BE

Recordemos que una BE convencional de gama baja, carece de información confiable de los diferentes parámetros del vehículo, y es por eso que en esta tesis y en concreto en este capítulo, nos centraremos en desarrollar la instrumentación para la adquisición de datos de los diferentes parámetros dela BE como son: temperatura del motor, de la batería, del controlador y del ambiente, velocidad de la BE y de pedaleo, corriente instantánea de la batería al controlador y nivel de voltaje en la batería, así como también la programación del microcontrolador para recibir, procesar y desplegar los datos.

En este capítulo, el envío de datos se realizará en una pantalla LCD de 2x24 caracteres y dejaremos él envió y recepción de datos a través de Bluetooth al smartphone para el capítulo 3.

Los parámetros por medir se eligieron por su importancia y relevancia para el usuario promedio, así como para un uso correcto y seguro del vehículo. Los requerimientos que debe tener cada uno de los sensores se abordaran individualmente en cada sección.

2.1 Elección del microcontrolador

Para seleccionar la tarjeta sobre la cual se basaría este proyecto tomamos en cuenta diferentes aspectos como son: velocidad del procesador, memoria *flash,* memoria RAM, numero de puertos, *timers*, ADC's, protocolos de comunicación, costo.

Las tarjetas sometidas en consideración fueron las siguientes: Texas Instruments MSP430G2553IN20, Cypress Semiconductor CY8CKIT-145-4000S PsoC, y Texas Instruments EK-TM4C1294XL TIVA

En la tabla 3 se muestra la comparación entre las 3 tarjetas.

	MSP430G2553IN20[xv]	4000S PsoC	EK-TM4C1294XL TIVA[xvi]
MCU	MSP430G2553IN20	ARM CORTEX M0	ARM CORTEX M4F
RELOJ	16MHZ	48 MHZ	120MHZ
BUS DE DATOS	16 BITS	32 BITS	32 BITS
FLASH	16 KB	32 KB	1024 KB
RAM	512 B	4KB	256KB
PUERTOS GPIO	16	36	>60
TIMERS	2 X 16 bits	5 X 16 BITS	8 X 32 BITS
ADC	8 CH 10 bits	8 CH 10 bits	2 CH 12 BITS
BLUETOOTH	NO	SI	NO
(SPI,I2C,UART)	1X	2X	2X
COSTO	14 USD	18 USD	27 USD

Tabla 3 Opciones de microcontroladores

De acuerdo con nuestras necesidades que son las siguientes:

- 6 canales de convertidor analógico digital (4 para los sensores de temperatura LM35, uno para el de corriente ACS, el último para el de voltaje)
- 2 entradas digitales para los codificadores de velocidad
- 7 salidas digitales para el LCD, que en total suman 15 puertos GPIO en uso.

Una velocidad de reloj tal que se pueda estar desplegando la información de todo el sistema en tiempo real cada 250[mS] mínimo y la cual será calculada más adelante.

Observando nuestras propuestas nos damos cuenta, que el MSP430 queda demasiado ajustado a nuestras necesidades y que el TIVA sobrepasa por mucho los requerimientos, lo que nos deja con la opción intermedia, el PSoC de Cypress.

Una característica especial y que es indispensable en el prototipo es el uso de comunicación vía Bluetooth, algo que le da mucha ventaja a la placa de Cypress, ya que esta ya tiene incorporado un módulo Bluetooth Low Energy (BLE), esto nos ahorra dos puertos para comunicación UART (Rx-Tx) y la adquisición de un Módulo BLE de aproximadamente 7 USD.

La tecnología PSoC (Progrmable Sistem on Chip) de la placa de Cypress resulta muy interesante por incorporar dentro del mismo chip, comparadores, amplificadores, filtros, multiplexores y demás componentes electrónicos, todos estos totalmente programables y dirigidos a cualquier puerto, además de que cuenta con una IDE de acceso libre (Psoc Creator[xvii]) que permite desarrollar software en lenguaje C y también de manera gráfica con el uso de bloques con una amplia gama de bibliotecas para cada uno de sus componentes.

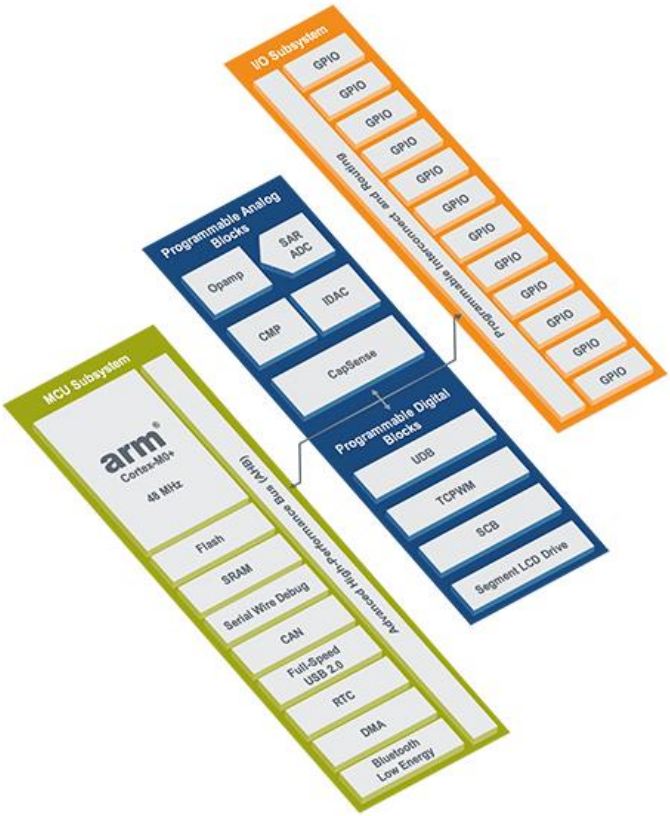


Figura 2.1 Arquitectura de PSoC 4

Es por todo lo anterior que se ha tomado la decisión de usar la tarjeta de Cypress.

Cabe señalar que, aunque no es una tecnológica nueva, esta no tiene la popularidad que debería merecer por razones que desconocemos, y que con el desarrollo de la presente tesis deseamos dar más proyección a dicha tecnología.

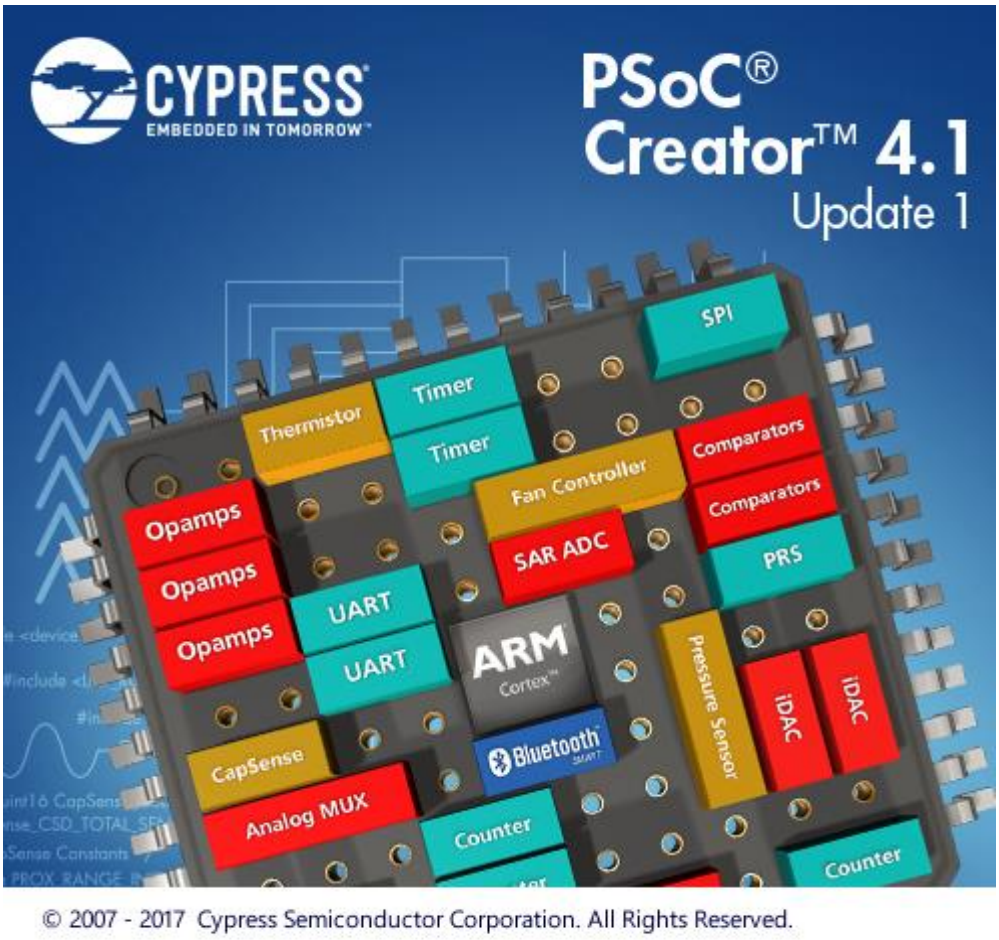


Figura 2.2 PSoC Creator 4.1 IDE

2.2 Sensores y adquisición de datos

Para el desarrollo del prototipo se usaron sensores comerciales y de bajo costo, en el caso del sensor de corriente y el codificador de velocidad ocupamos dispositivos con su circuitería prefabricada en placa de cobre o como se les conoce popularmente

como “Shields”. Para los otros parámetros se desarrolló por separado la electrónica necesaria.

2.2.1 Temperatura

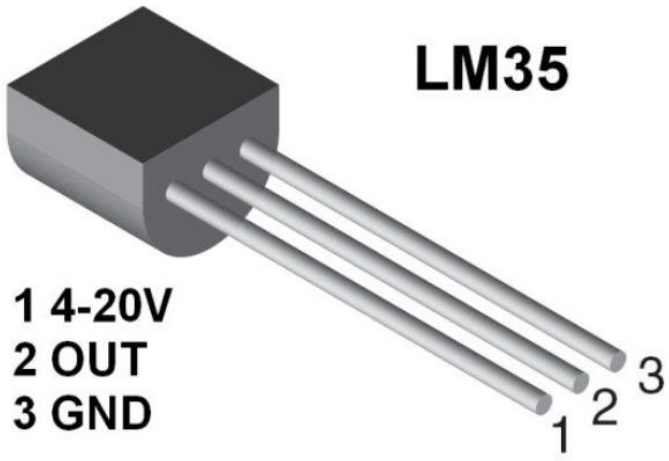


Figura 2.3 Sensor de temperatura LM35

Para medir temperatura usamos el sensor LM35 de Texas Instruments, ya que es muy versátil, económico y de lectura muy práctica. La configuración aplicada es la mostrada en la figura 2.4.

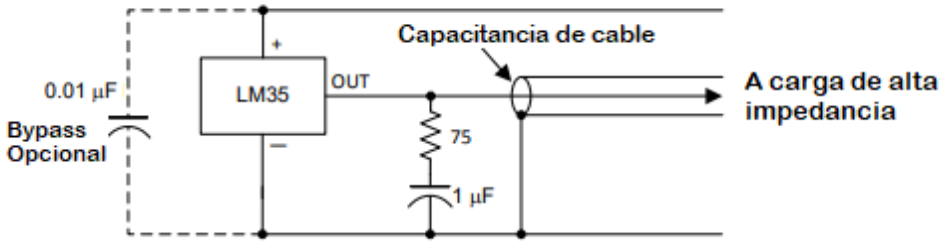


Figura 2.4 Conexión de LM35 con estabilizador de señal

Con esta configuración, podemos usar una fuente de alimentación desde 4[V] y hasta 20[V], tomar lecturas de temperatura entre 2[°C] y 150[°C], suficientes para los parámetros que deseamos monitorear, las cuales se estiman en hasta 50 [°C] por encima de la temperatura ambiente. Es importante mencionar que dicho sensor entrega medidas en escala Celsius en un orden de 10[mV/°C], un factor de error de ± 0.5[°C] y no requiere calibración. Por lo que solo es necesario activar una entrada ADC en el microcontrolador por cada sensor instalado.

2.2.2 Velocidades de rueda y eje de pedaleo

Es necesario conocer la velocidad de alguna de las dos ruedas del vehículo para después con unos simples cálculos conocer la velocidad de la bicicleta. En lo que respecta a la velocidad del eje de pedaleo, esta se mide para tener un estimado de la cadencia de pedaleo del ciclista y así poder calcular la potencia y/o energía entregada por este.

Figura 2.5 Shield con opto-interruptor

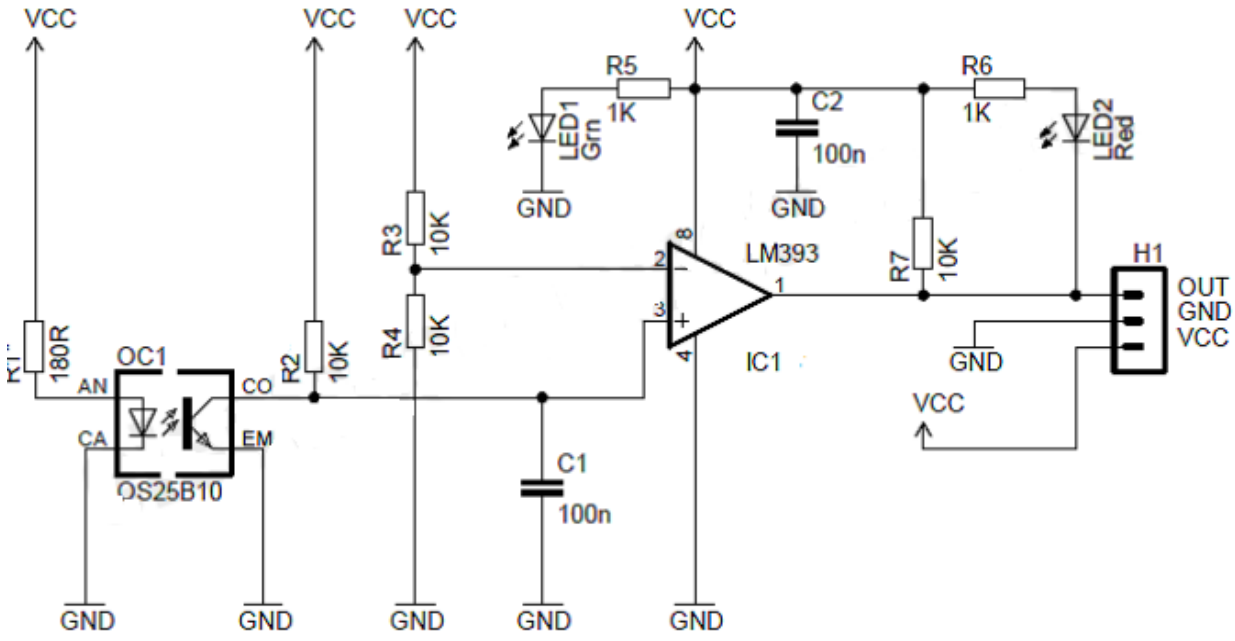


Figura 2.6 Diagrama eléctrico de opto-interruptor

Para medir dichas velocidades hicimos uso de un disco de opto interruptores diseñados por nosotros mismos y el cual se instala directamente en los ejes.

El sistema funciona de la siguiente manera: el disco opto-interruptor diseñado tiene 18 "ventanas" (una cada 20°), las cuales dejan pasar el haz del infrarrojo instalado en el Shield, cada que esto sucede, el dispositivo entrega un voltaje positivo a la salida, lo que será interpretado como un '1' lógico en el controlador. El microcontrolador llevara la cuenta de pasos por ventanas y de tiempo transcurrido para poder hacer el cálculo de la velocidad angular de dichos ejes de rotación.

El disco ha sido diseñado para acoplarse directamente en la rueda con la tornillería estándar, donde en algunos modelos va colocado el disco de freno. En el caso del sensor del eje de pedaleo, se tomó en cuenta la medida del eje para poder montar este disco. (ver figura 2.7)

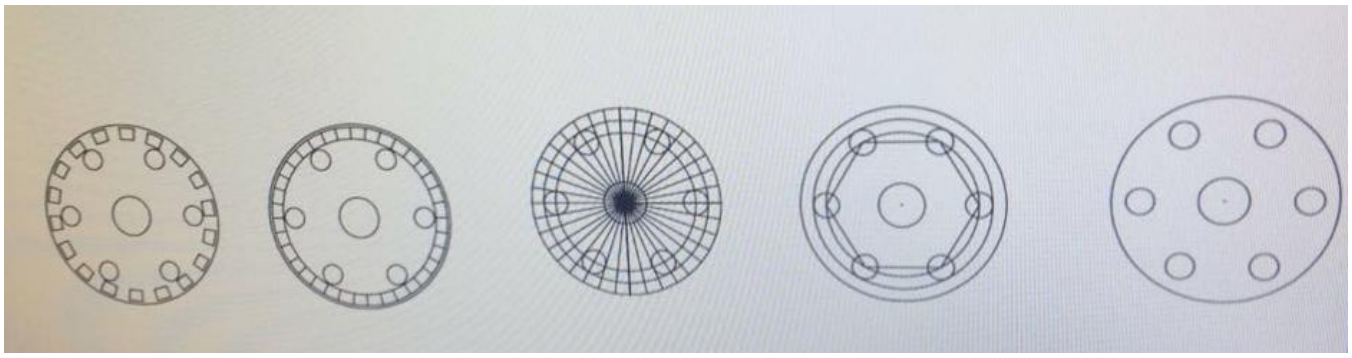


Figura 2.7 Proceso de diseño del disco

Figura 2.8 Disco terminado

2.2.3 Corriente instantánea

Aquí hemos hecho uso del sensor ACS712T ELC-30A de efecto Hall, también en *Shield* prefabricado. (ver figura 2.9)

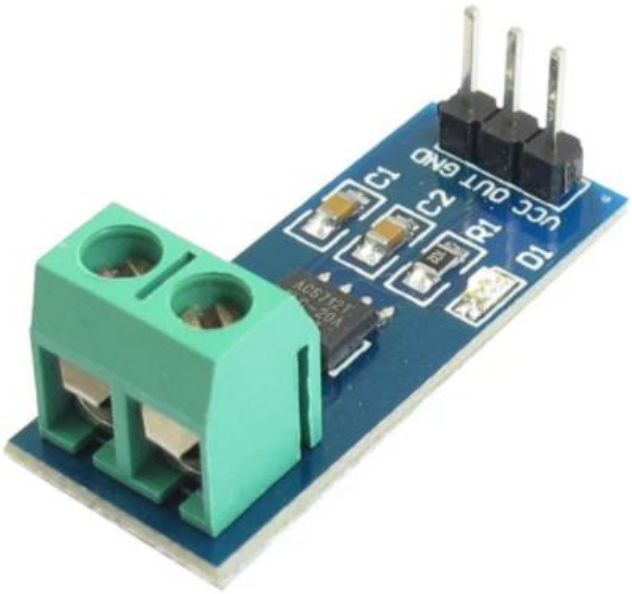


Figura 2.9 Shield para censado de corriente

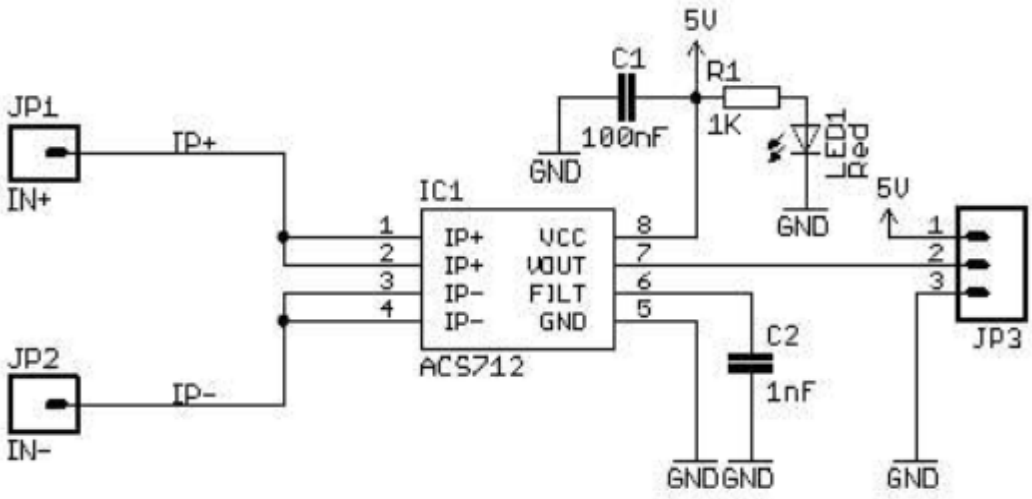


Figura 2.10 Diagrama eléctrico de shield de censado de corriente con ACS712

Recordemos brevemente cómo funciona el efecto Hall dentro de este sensor. En el circuito integrado se crea un campo magnético de un valor conocido, el cual atraviesa perpendicularmente una placa por donde circulara la corriente a medir, a los extremos

de esta placa se producirá una diferencia de potencial (voltaje Hall) directamente proporcional a la cantidad de corriente que circule.

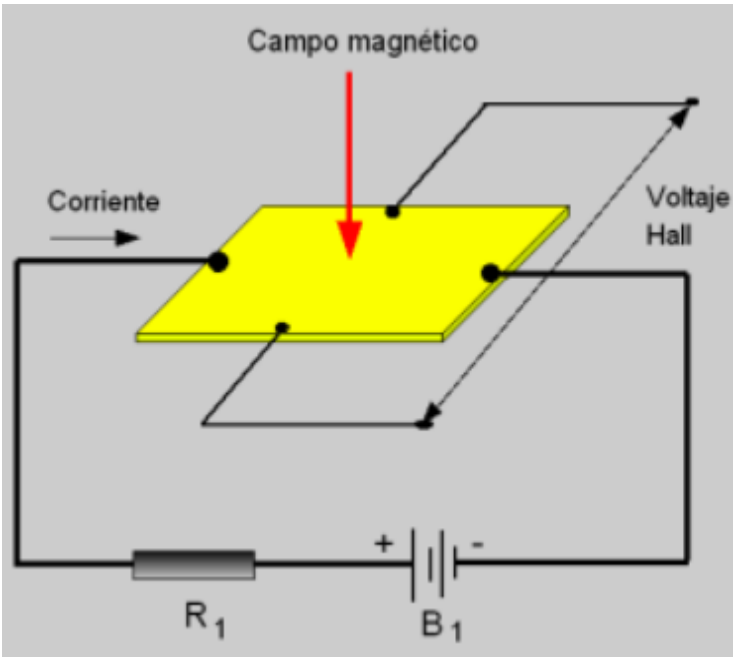


Figura 2.11 Efecto Hall

El sensor ACS712 ELC-30A está diseñado para soportar la medición de hasta 30 Amperes de corriente, que resulta suficiente para nuestro prototipo, pues este funciona con una batería de 48 [V] y un motor de 1000 [W], lo que representa una corriente nominal de 20.83 [A]

$$1000\ [W]/48\ [V] = 20.83\ [A]$$

Ecuación 1 Corriente nominal de la BE

La sensibilidad de este encapsulado es de 66 [mV/A] y un factor de error de 1.5%.

En la figura 2.12 se puede apreciar cómo se comporta la salida de voltaje dependiendo de la corriente de entrada, es importante notar que cuando no hay corriente el voltaje permanece en (Vcc/2).[xviii]

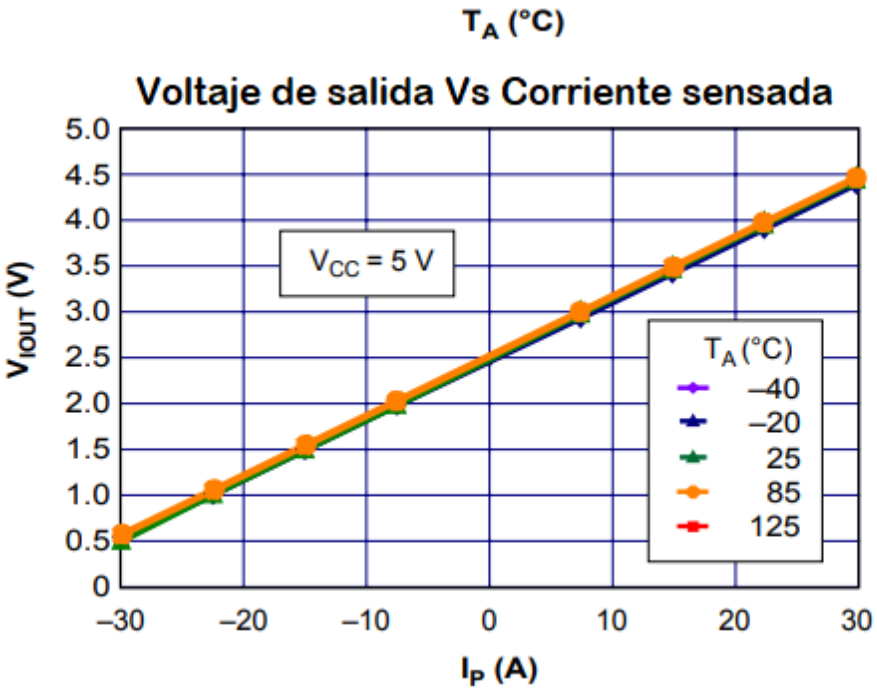


Figura 2.12 Voltaje de salida del sensor ACS712

2.2.4 Voltaje en batería

Por último, para medir voltaje, tenemos que tener en cuenta que la batería a plena carga tiene un voltaje nominal de 53 [V] aproximadamente. Por lo que tenemos que recurrir a diseñar un divisor de voltaje que nos permita tomar lecturas en nuestro ADC, pues este tiene un máximo voltaje de entrada de 5[V]. Para una mejor lectura del ADC, después del dicho divisor es necesario colocar un amplificador operacional en modo seguidor para poder acoplar las impedancias y aislar el circuito En la figura 2.13 se muestran los pasos seguidos.

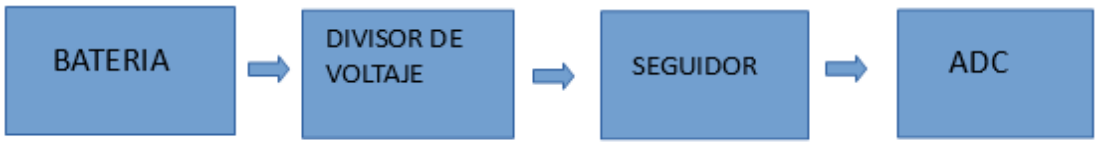


Figura 2.13 Diagrama de bloques de censado de voltaje

Para el diseño del divisor de voltaje obtenemos una fracción veinte veces menor al voltaje real de la batería. Así cuando la batería este a plena carga 53[V], en el divisor tendremos 2.65[V] y no sobrepasará el límite de voltaje de entrada del microcontrolador, el cual es VCC + 0.5 [V]. Es indispensable que el valor de impedancia total del divisor de voltaje sea alto, en el orden de los 100[kΩ] para así no tener problemas de calentamiento o destrucción de componentes por una corriente alta.

La ecuación general de un divisor de voltaje queda expresada de la siguiente forma:

$$V2=((R2/R1+R2))\ (V1)$$

Ecuación 2 Divisor de voltaje general

De la cual R1 la propondremos de 100KΩ, y V2 y V1 las sustituiremos de acuerdo a nuestros requerimientos de que V2 sea 20 veces menor a V1. Quedando solamente como incógnita R2.

$$R2=(R1/((V1/V2)-1))$$

$$R2=(100K\Omega/((53V/2.65V)-1)) =5.26[K\Omega]$$

Ecuación 3 Calculo de las resistencias para el divisor de voltaje

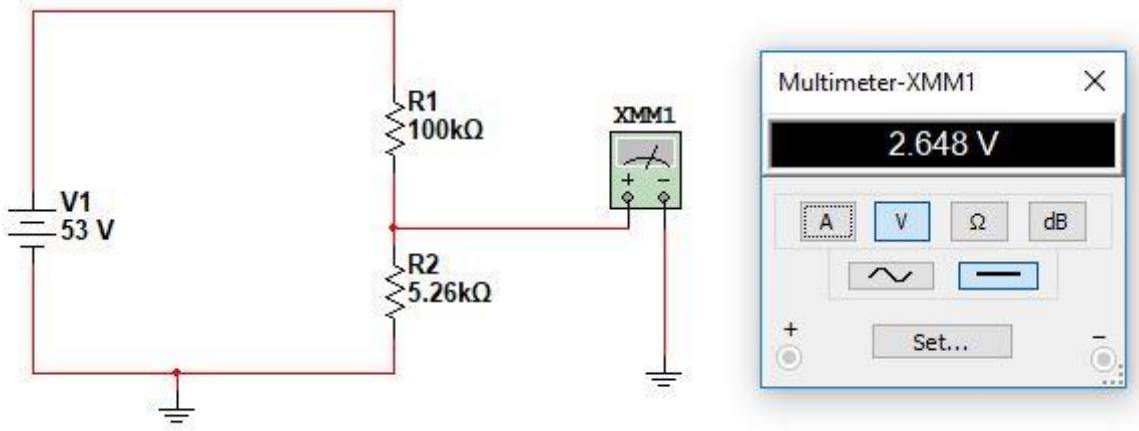


Figura 2.14 Divisor de voltaje relación 20:1

2.3 Diagrama de flujo del software principal de adquisición de datos

En la figura 2.15 se muestran los pasos seguidos para la elaboración de la primera versión del software, el cual explicaremos paso a paso más adelante.

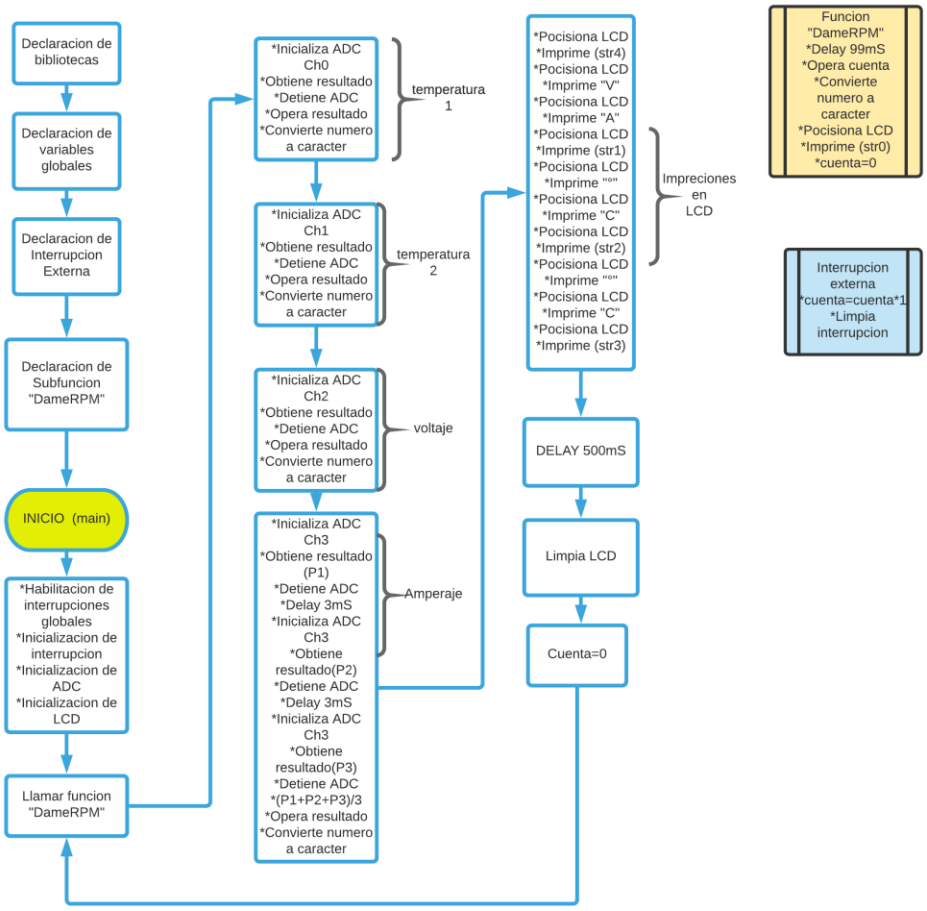


Figura 2.15 Primer versión de software

En la primera parte del diagrama, antes de la sentencia de Inicio (main), se declaran las bibliotecas, las variables globales, las interrupciones y las funciones que usaremos en el sistema. Aquí es importante mencionar que tanto la interrupción como la función "Dame RPM" son para medir la velocidad de giro del disco instalado en la rueda como codificador.

Después del "main", se habilitan las interrupciones globales y se inicializan tanto la interrupción declarada y las funciones ADC Y LCD, que cabe mencionar son nativas del controlador y no hace falta desarrollarlas. Después de esto comienza nuestro "loop" o ciclo infinito del programa.

Como primer paso dentro de este loop, llamamos a la función "Dame RPM", que a su vez utilizara la interrupción externa creada para poder medir la velocidad de giro del disco. Esto funciona de la siguiente forma: la interrupción externa se activa cada que

hay un cambio de bajada en el nivel lógico, es decir de "1" a "0", en el pin de entrada de dicha interrupción, esto sucede cada que las ventanas del disco impiden el paso del infrarrojo del encoder. Es entonces cuando una variable denominada "cuenta" se incrementa en más uno. Es importante mencionar que durante cualquier momento o línea que se esté ejecutando en el programa esta interrupción está funcionando, pero solo cuando se llama a la función "Dame RPM" es cuando toma importancia, pues es aquí cuando se va a leer a la variable "cuenta".

Durante la función "DameRPM", lo primero que sucede es un retardo de 99[mS], esto para dejar "operar" por dicho tiempo la interrupción y tener un numero de pasos por ventanas conocido dentro de un intervalo de tiempo también conocido y así calcular sin problemas la velocidad de giro del disco. Los siguientes son para operar tales números y obtener RPM, convertir el número a carácter, ya que, la función LCD solo imprime caracteres, posicionar el cursor en el LCD donde deseamos imprimir para después realizar esta impresión en pantalla, por último, reiniciamos el valor de "cuenta". Los pasos para convertir un número a carácter, posicionarlo en el LCD e imprimir, los tenemos que hacer con cada uno de los valores que deseamos mostrar en pantalla.

Los siguientes bloques son un tanto más sencillos y similares entre sí. Se encargan de tomar las lecturas del ADC en cuatro canales diferentes, temperatura 1, temperatura 2, voltaje, corriente. Cada "bloque" funciona así: se inicializa el ADC en el canal deseado, se obtiene conversión en milivolts, se detiene la conversión, se opera el resultado y se convierte dicho número a carácter. Esto se repite para los cuatro parámetros, con la excepción que para la corriente se tomó un camino un poco diferente.

Para la medición de corriente se tomaron tres lecturas del ADC separadas por un pequeño intervalo de tiempo y después se promediaron, ya que el sensor entrega medidas un tanto variables, y con esto tratamos de compensarlo, obteniendo como resultado una medida más estable.

El siguiente bloque solo se refiere a los pasos seguidos para la impresión de los caracteres en pantalla LCD. Después de esto hacemos un retardo (delay) de 500[mS] para darle tiempo al ojo humano de observar la información antes de limpiar la pantalla e iniciar el ciclo de nuevo. Por último, se iguala la "cuenta" a cero, pues recordemos que durante todo el proceso esta variable se está incrementando, y la función que le procede la necesita en cero.

2.4 Conexiones en PsoC (Programable Sistem on Chip)

A continuación, se muestra el diagrama de bloques del código implementado en el IDE PSOC Creator 4.1 y el cual corresponde a la primera versión del software explicado anteriormente. Las instrucciones específicas se presentan al final.

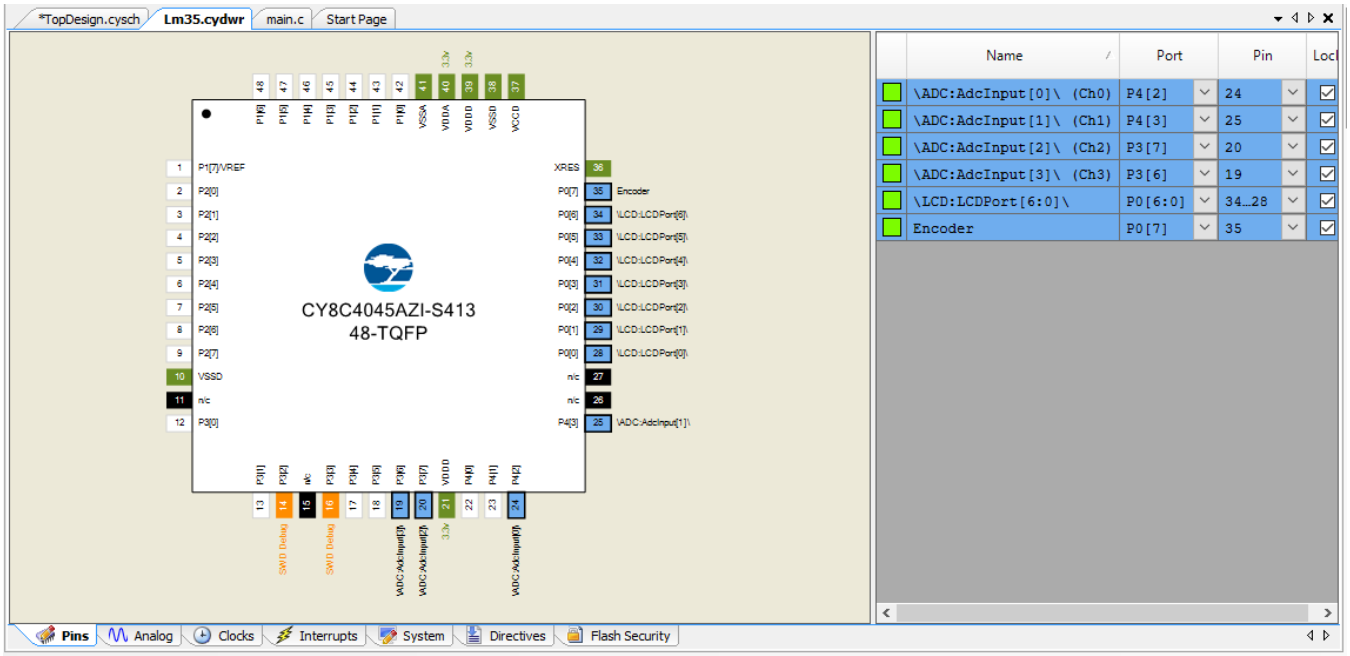


Figura 2.16 Mapa de asignación de pines del microcontrolador

En la figura 2.16 se muestra el mapa de asignación de pines del microcontrolador, como se puede apreciar todo es de manera gráfica, agilizando por completo este proceso.

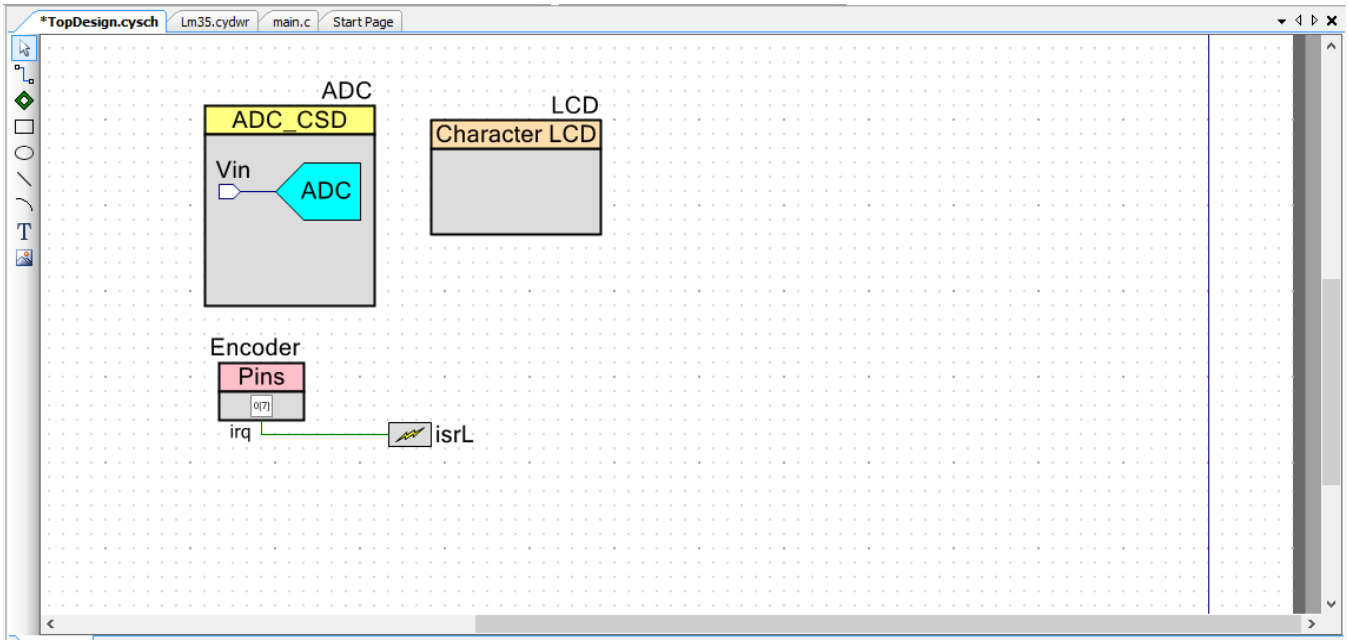


Figura 2.17 Esquema gráfico de las funciones utilizadas en PSoC Creator

En la figura 2.17, podemos observar el diseño de las funciones en modo gráfico. Como mencionamos en un principio, esta una de las grandes ventajas de utilizar la tecnología Psoc, pues de manera gráfica podemos hacer conexiones de pines, comparadores, amplificadores, y demás funciones nativas del sistema como es en este caso los bloques del ADC y del LCD, con estos dos bloques agregamos las bibliotecas necesarias para hacer funcionar estos dispositivos dentro del sistema. Por otra parte, se observa un pin asignado al codificador y al mismo una conexión que lo relaciona directamente a la interrupción externa "isrL".

En la figura 2.18 se muestra el diagrama de bloques de las conexiones hechas del proyecto durante esta primera etapa de despliegue de datos sobre el LCD.

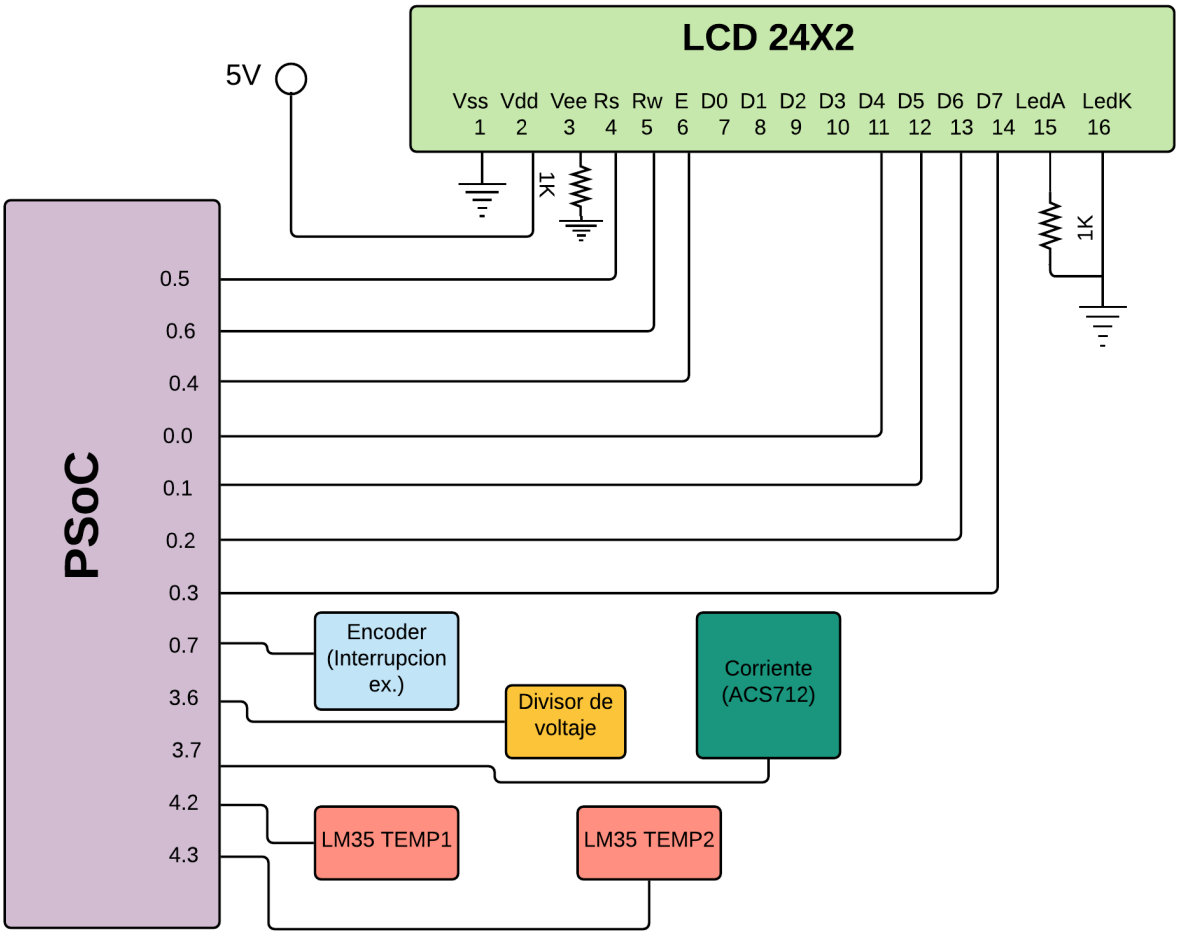


Figura 2.18 Diagrama de bloques del sistema

Pantalla LCD para el despliegue de datos

2.5 Conclusiones del capitulo

Cabe mencionar que durante este capítulo solo se instalaron y programaron dos sensores de temperatura LM35, por lo que harían falta otros dos, para poder medir las cuatro temperaturas de nuestro interés (batería, motor, controlador, ambiente), y un solo sensor codificador para la velocidad de la rueda, por lo tanto, hace falta otro codificador para la velocidad del eje de pedaleo. Dichos sensores se instalarán para el siguiente capítulo, aunque su funcionamiento y programación han quedado explicados en el presente.

Por otra parte, tengo que mencionar que el software implementado en esta primera versión funciona de manera muy aceptable, aunque es evidente que requiere una revisión en lo que respecta a optimización de recursos y tiempos de ejecución para un desempeño aún mejor.

En cuanto al censado de corriente y de voltaje, estos dos requieren un filtrado más adecuado, pues ambos presentan demasiadas variaciones en sus lecturas. Esto se atacará más adelante, tanto con filtros pasivos en hardware y digitales en el programa. Recordemos que, en esta primera etapa del proyecto, se implementó el despliegue de datos a través de la pantalla LCD para verificar la correcta lectura de los mismos antes de poder ser enviada al módulo Bluetooth.

Por otra parte, en el tema de los bloques analógicos con los que cuenta la tecnología PsOC. Estos no pudieron ser implementados ya que la placa seleccionada (4000S) no cuenta con ellos por ser de las más accesibles de la compañía.

CAPITULO 3 DISEÑO Y PROGRAMACION DEL MODULO BLUETOOTH

En este capítulo daremos una breve introducción a la tecnológica BLE, también explicaremos como esta se utiliza en la tarjeta de desarrollo en este proyecto y la posterior programación del módulo. Y por último daré un repaso a la versión mejorada y complementada del software de adquisición de datos del capítulo anterior.

3.1 La tecnología Bluetooth Low Energy (BLE) 4.2

El Bluetooth es un protocolo de comunicaciones por radiofrecuencia para dispositivos electrónicos en la banda ISM de 2.4 [GHz].

Esta tecnología existe desde el año 1998 y con el tiempo ha ido evolucionando en diferentes versiones hasta la 5.0 que existe en la actualidad. Mejorando tanto en velocidad de transmisión de datos, alcance, y estabilidad, así como también en los protocolos de comunicación, seguridad y conexión.

Por su parte la versión 4.2 (desarrollada en 2010 y también llamada Bluetooth Smart), se diferencia por un protocolo de bajo consumo de energía, diseñado para instalarse en dispositivos de muy reducido tamaño y bajo costo. Esto lo hace mediante él envió intermitente de datos "empaquetados" solo cuando es necesario en vez de un flujo constante de información. Además, es capaz de cambiar a un estado de "descanso" cuando no se está usando la transmisión de datos. Por lo anterior esta tecnología se ha abierto paso en sectores, como el de salud, deportivo, y domótica. Además de que gracias a su sencilla configuración cada vez son más las aplicaciones en nuestra vida cotidiana.

3.1.1 Protocolo

El protocolo BLE 4.2 se caracteriza por dos principales términos: GAP y GATT.

GAP (Generic Access Profile)

Para establecer una conexión entre dos dispositivos BLE es necesario identificar cual es el dispositivo central y cual el dispositivo periférico. El GAP (Generic Access Profile) nos permite definir dichos roles entre los dispositivos. Así como también otros parámetros como son: contraseñas, nombres, velocidades de conexión etc.

Principalmente se define al dispositivo como “central” cuando este tiene el poder de iniciar y finalizar la conexión, así como la capacidad de búsqueda de otros dispositivos. El periférico es el dispositivo que “anuncia” la disponibilidad de un nuevo dato o “paquete” y lo envía, pero es el dispositivo central el que decide cuando recibir dicho paquete. En nuestro caso el dispositivo central será el teléfono móvil y el periférico será el microcontrolador. En la versión BLE 4.2 es posible conectar varios dispositivos periféricos a un dispositivo central.

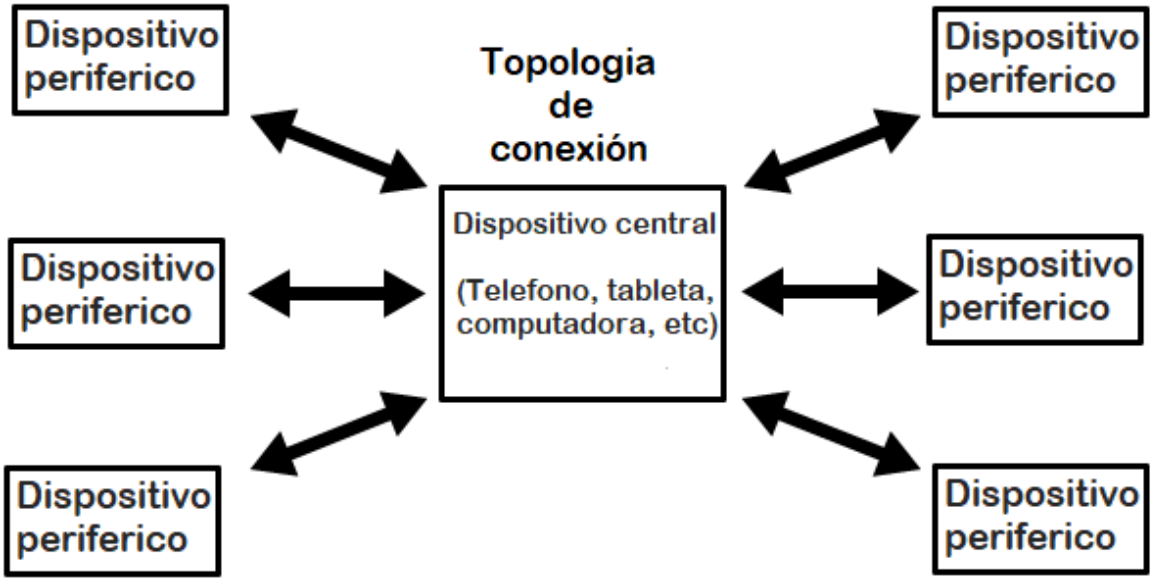


Figura 3.1 Topología de conexión BLE 4.2

GATT (Generic Atribute profile)

En el intercambio de datos es importante definir a un “cliente” y a un “servidor”. Esto lo hacemos mediante el GATT (Generic Atribute Profile). En este protocolo el servidor es quien proporciona, almacena y escribe en la base de datos y el cliente es quien recibe dichos datos. Cabe mencionar que, aunque en nuestro proyecto el intercambio de datos es unidireccional, con este protocolo es posible que ambos dispositivos cumplan los dos roles y la comunicación sea bidireccional. Para nuestro proyecto el servidor será el microcontrolador y el cliente el teléfono móvil.

Figura 3.2 Topología BLE entre BE y teléfono móvil

El empaquetado de los datos a enviar por el servidor se hace mediante un método de "servicios y características".

Un servicio es la manera en cómo el periférico anuncia el tipo de papel que desempeña. Un servicio puede contener una o más características. Se le llama característica al dato "empaquetado" que se enviara. Cada servicio y característica posee un número único de identificación (UUID) para poder comunicarse fácilmente entre dispositivos sin interferencia de otros datos, ya que como mencionamos el dispositivo central tiene la facultad de decidir cuándo y que datos quiere leer de un dispositivo deseado.

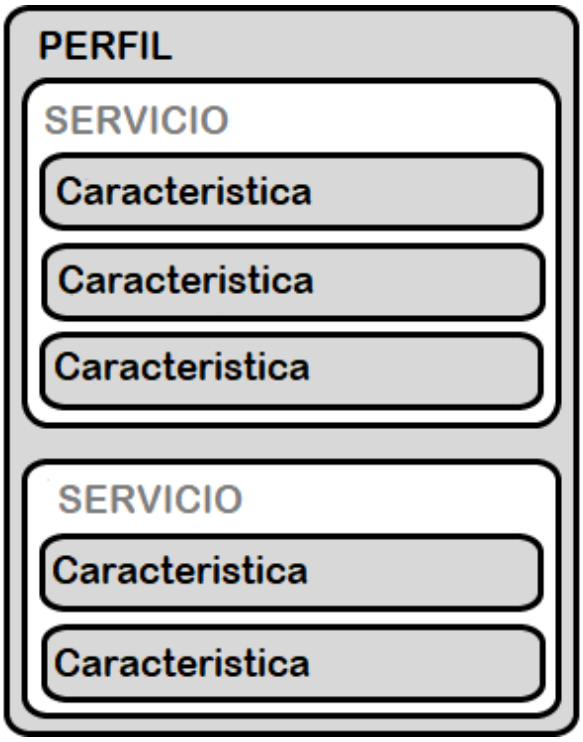


Figura 3.3 Organización de perfiles, servicios y características BLE

Existen perfiles de "servicios y características" aceptados oficialmente por desarrolladores BLE. Un ejemplo puede ser el perfil de Medición de frecuencia cardíaca, el cual contiene un servicio llamado "Frecuencia cardíaca" y este a su vez contiene tres características: Medición de la frecuencia cardíaca, Ubicación del sensor corporal, Punto de control de la frecuencia cardíaca. Estos perfiles oficiales comparten los mismos identificadores (UUID) para facilitar el desarrollo de aplicaciones.

3.2 Arquitectura del módulo BLE en la tarjeta PsOC 4000s

Dentro de la tarjeta de desarrollo encontramos que el módulo BLE y el microcontrolador están separados física y lógicamente. Es decir, cada uno requiere una programación por separado de cada una de sus funciones. Ambas partes se programan en el sistema de PsoC Creator, y comparten la misma interfaz de programación por USB (KitProg2). Únicamente un interruptor físico es el que nos permite seleccionar que dispositivo es el que deseamos programar. Estos dos pueden intercambiar datos tanto con el protocolo de I2C y el modo serial.

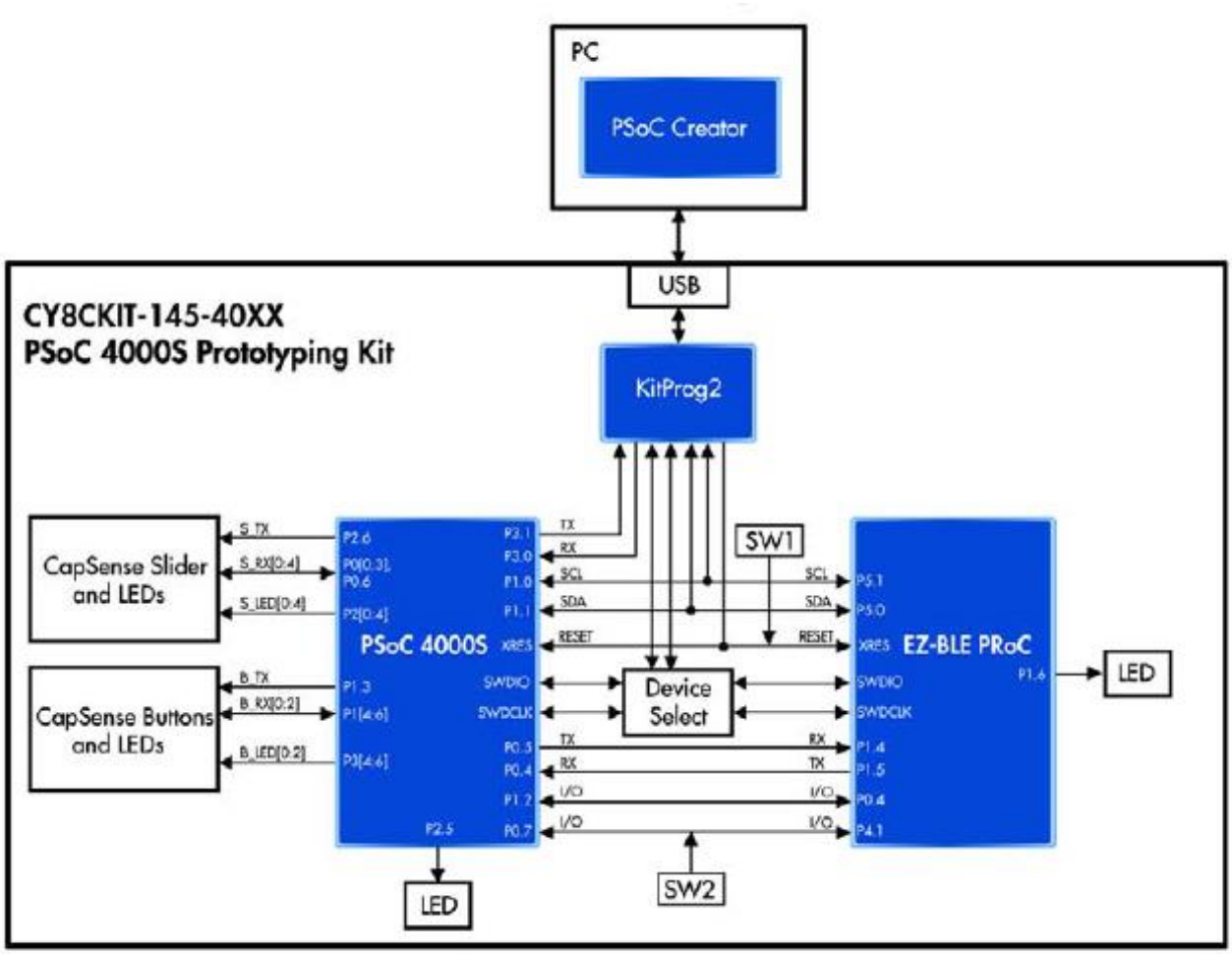


Figura 3.4 Arquitectura PSoC 4000S

Por lo tanto, en la siguiente sección de este capítulo desarrollaremos el programa encargado de controlar el módulo BLE y definir su servicios y características.

3.2.1 Diseño y programación del módulo BLE

Este capítulo lo dividiremos en dos partes, primero nos ocuparemos del manejo del GAP y del GATT y en la segunda parte nos enfocaremos en el flujo del programa que controlara el BLE.

GAP Y GATT

Dentro del IDE PsoC Cretor accedemos al bloque dedicado al módulo BLE para ajustar su configuración, tal cual como le hemos hecho con los bloques anteriores. (ver figura 3.5)

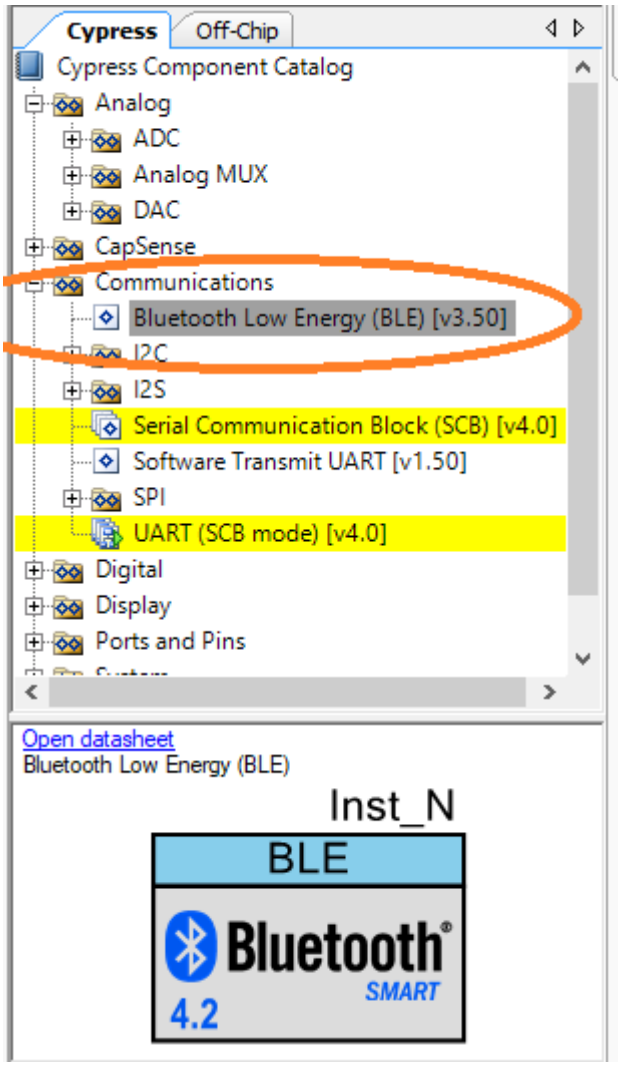


Figura 3.5 Pestaña de recursos en PSoC Creator

En la primera pestaña de este cuadro de configuración elegimos las opciones deseadas que ya mencionamos: un perfil personal (custom), GATT server y GAP periférico. (ver figura 3.6)

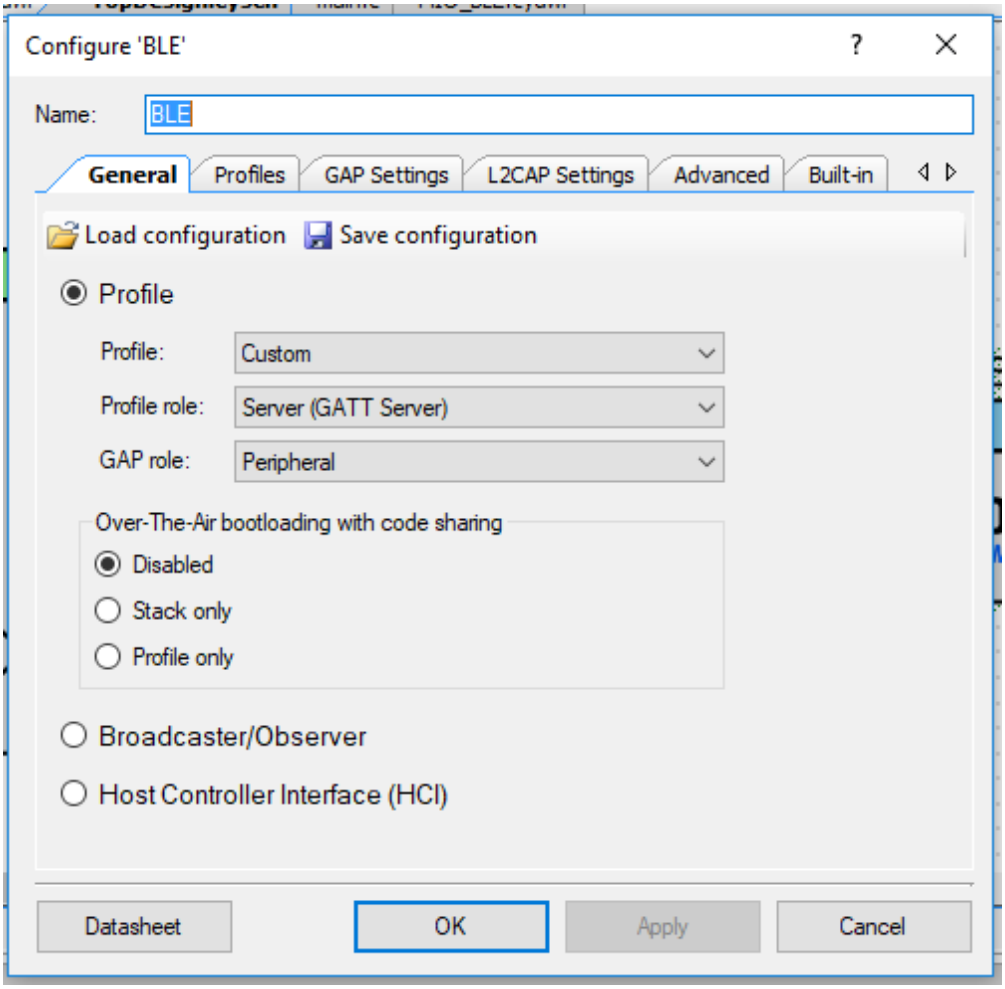


Figura 3.6 Configuración general BLE

Hecho esto, pasamos a configurar nuestros servicios y características en la pestaña "profiles" de la misma ventana. (ver figura 3.7)

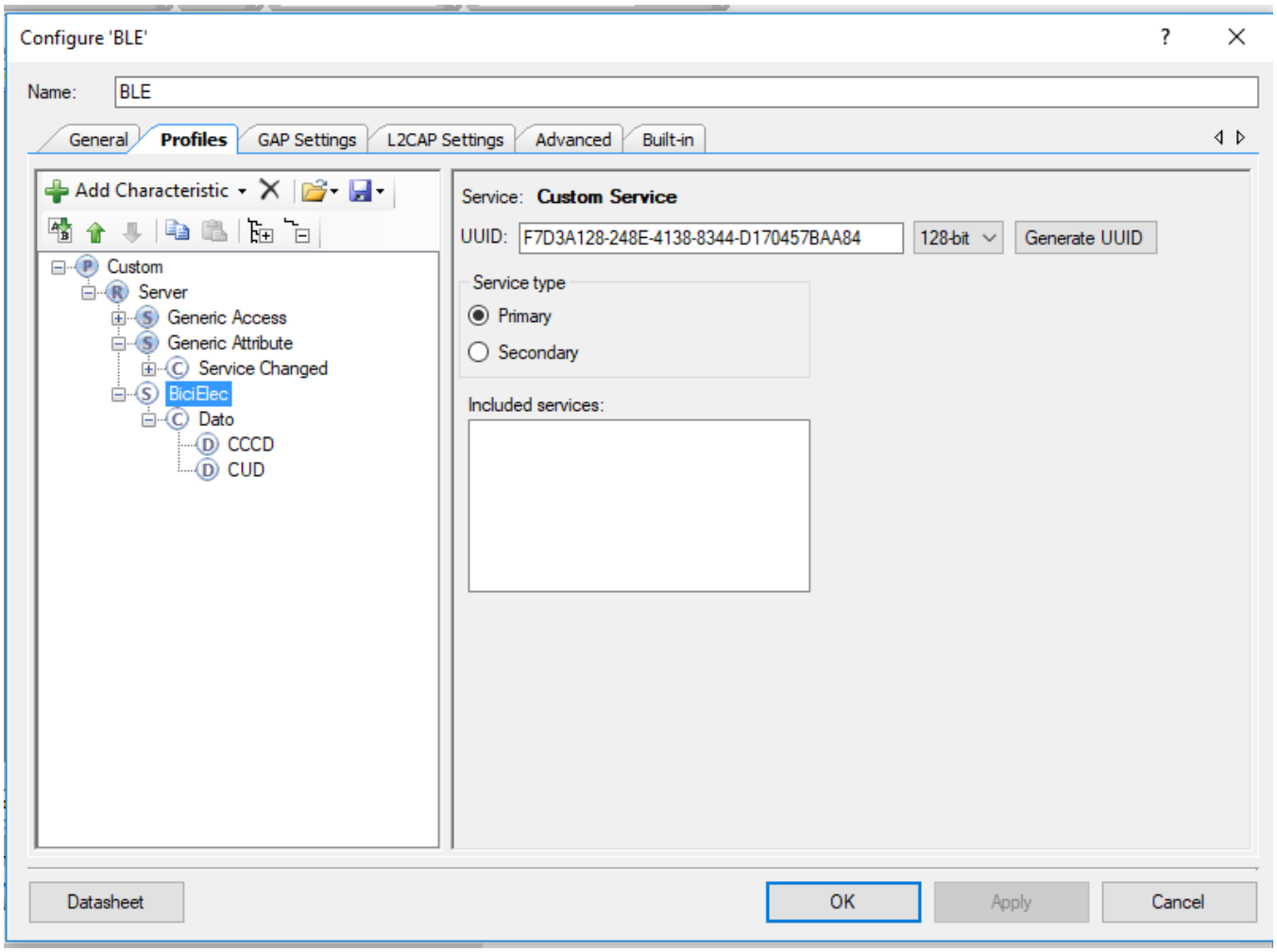


Figura 3.7 Configuración de perfile BLE

Al crear un perfil personalizado o "Custom", la plataforma nos crea por defecto dos servicios: "Generic Acces" y "Generic Attribute", cada uno con sus propias características las cuales contienen los parámetros de conexión de dicho perfil creados también por defecto, a excepción de nombre de dispositivo y contraseñas creadas por nosotros.

En la figura 3.7 hemos creado un servicio llamado "BiciElec", mismo que contiene una característica llamada "Dato"

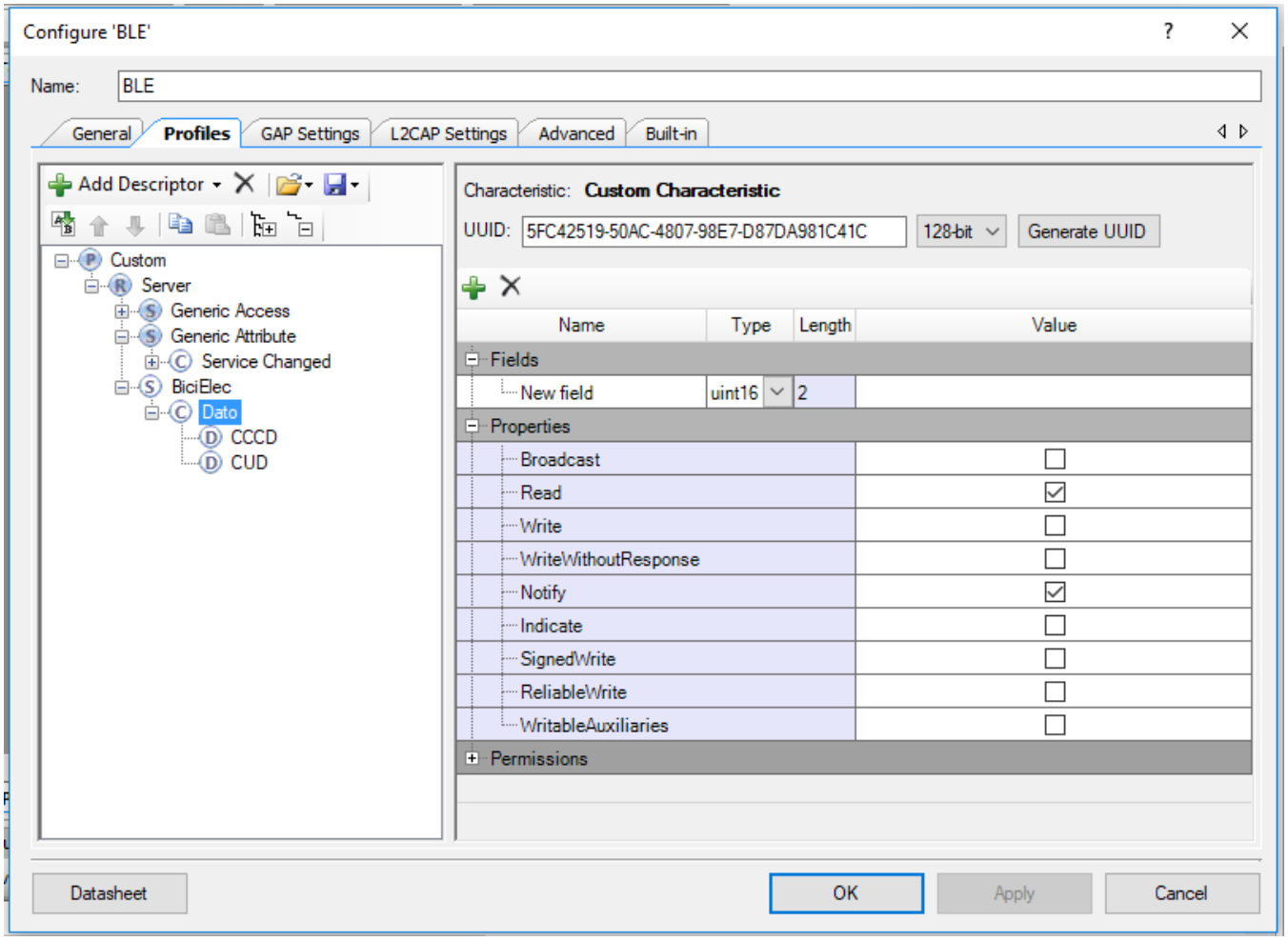


Figura 3.8 Configuración de características BLE

En la configuración de "Dato" le indicamos que es una variable sin signo de 16 bits y marcamos las casillas de "Read" y "Notifi", esto último indica que el paquete es de solo lectura y además que al actualizarse en la base de datos envié una notificación al cliente. Para terminar los campos CCCD (Client Characteristic Configuration Descriptor) y CUD (Characteritic User Descriptor) nos sirven para agregar información textual al paquete que estamos enviando sobre (longitud del dato, tamaño y nombre) y este sea de fácil identificación al cliente. (ver figura 3.8)

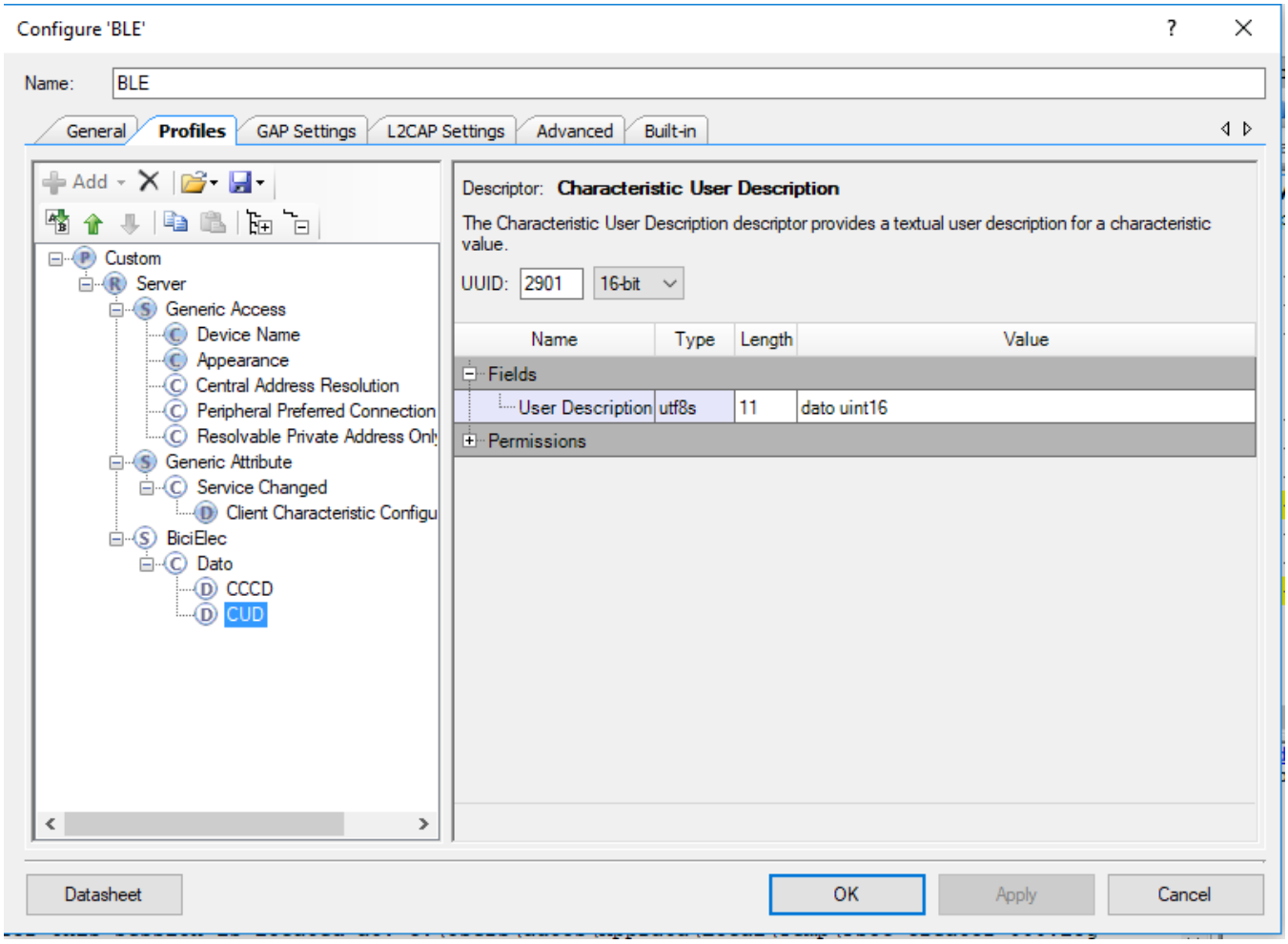


Figura 3.9 Configuraciones específicas de las características

Llegando a este punto tenemos listo nuestro servicio y característica. Bien podríamos haber hecho una característica por cada parámetro que deseamos transmitir de la bicicleta (temperaturas, voltaje, corriente etc.) y así tener bien identificado el dato que estamos recibiendo y en un principio se deseaba fuera así, pero después nos dimos cuenta de que la plataforma APP Inventor 2 en la cual se desarrollara la interfaz, solo soporta la lectura de una sola característica, ya que sus librerías están aún en desarrollo. Por este motivo nos vemos en la necesidad de usar una sola característica para los 8 datos que enviaremos, añadiendo una codificación de dichos datos para poder identificarlos y separarlos en la interfaz una vez que sean recibidos. Este proceso de codificación se explicará más adelante.

Hasta aquí hemos terminado con la configuración del módulo BLE

3.2.2 Diagrama de flujo del programa BLE

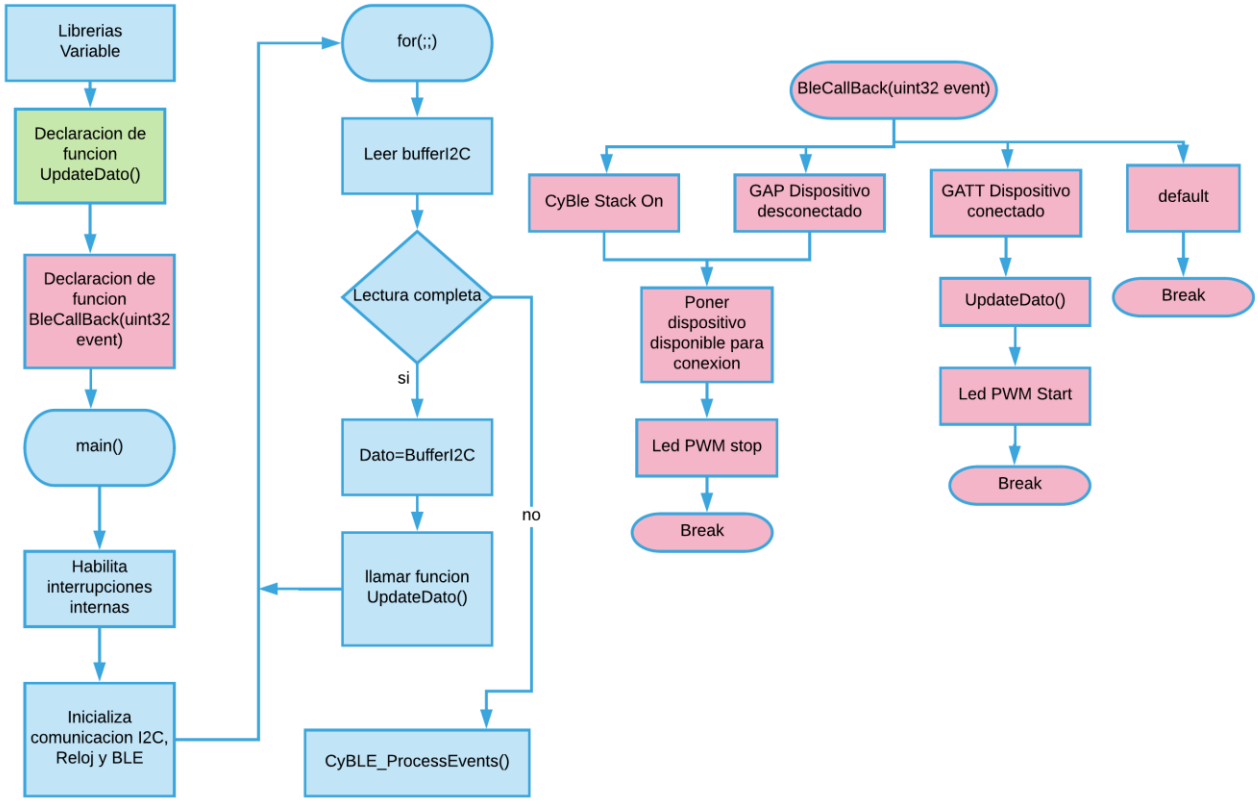


Figura 3.10 Diagrama de flujo del software del BLE

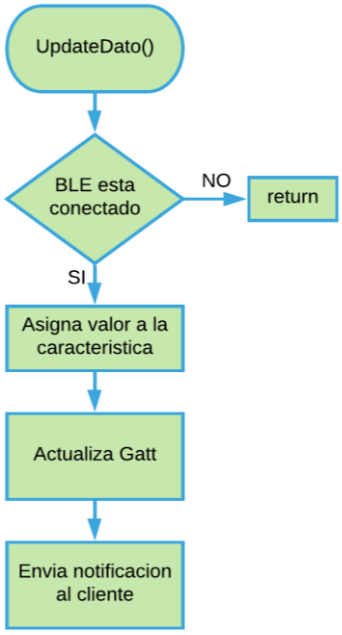


Figura 3.11 Diagrama de flujo de la función UpdateDato()

Este programa es relativamente sencillo, se compone de tres funciones, main (), updateDato () y BleCallBack (). Main () se encarga del flujo general del programa, UpdateDato () envía el dato y BleCallBack controla y administra los procesos del BLE. Esta última función es nativa de las librerías del IDE, pero adaptada y simplificada a las necesidades de nuestro proyecto.

3.3 Codificación y envió de datos a través de I2C al módulo BLE

Como ya dijimos antes, él envió de los diferentes datos vía BLE se hará por la misma característica, lo cual requiere una codificación o mejor dicho la inserción de un identificador en dicho dato para poder diferenciarlos.

El proceso de agregar un identificador es muy simple. Después de hacer la lectura del ADC o del codificador, tenemos un número entero sin signo que contiene la información de cada parámetro medido. A este número le agregamos un dígito en la posición de las unidades. Tal dígito es el identificador que necesitamos para reconocer el dato y lo agregamos multiplicando por diez y sumando el dígito identificador. Por ejemplo:

El ADC nos entrega una lectura del sensor lm35 de 256[mV] (25.6°C), a 256 lo multiplicamos por 10 y sumamos 1.

$$(256*10) +1 = 2561$$

Ecuación 4 Codificación

Este último dígito será el que tenga que decodificar nuestra interfaz en la aplicación Android para reconocer el dato.

En la tabla 4 muestro como se organizaron los parámetros con sus respectivos identificadores.

Parámetro	Rango	Identificador
Temperatura Aire	(20[mV]-1500[mV])	1
Temperatura Motor	(20[mV]-1500[mV])	2
Temperatura Driver	(20[mV]-1500[mV])	3
Temperatura Batería	(20[mV]-1500[mV])	4
Voltaje	(0[mV]-5000[mV])	5
Corriente	(520[mV]-4480[mV])	6
RPM Encoder Rueda	(0 rpm- 547.8 rpm aprox.)	7
RPM Encoder Pedales	…	8

Tabla 4 Identificadores

El rango de nuestros datos es muy importante, pues el número a enviar es un entero sin signo de 16 bits. Lo que representa un 65,536 máximo. Por lo que ninguno de nuestros datos después de agregarle el identificador debe supera este valor.

En el caso de las RPM, este es el único valor que en algún momento es igual a cero, por lo tanto, el dato enviado será únicamente su identificador y después la interfaz lo interpretará como cero.

La lectura máxima de RPM de la rueda se calculó suponiendo una velocidad máxima de la bicicleta de 70[Km/h] (lo cual está muy por arriba del valor real). Convertimos ese valor a metros por minuto.

$$(70) *([km/h]) *(1000[m/1Km]) *(1[h/60min]) = 1167[m/min]$$

Ecuación 5 Conversión de Km/h a m/min máximos

El radio de la rueda cuyo eje va solidario al disco codificador es de 34[cm]. Al calcular el diámetro de dicha rueda, obtenemos el avance lineal por cada revolución de la misma.

$$D= (2*\pi)\ (0.34[m]) = 2.13[m/revolución]$$

Ecuación 6 Avance lineal por cada giro de la rueda

Si dividimos los metros por minuto entre metros por revolución, obtenemos las revoluciones por minuto del codificador y de la rueda.

$$RPM= (1167[m/min]) / (2.13[m/rev]) = 547.8\ RPM$$

Ecuación 7 RPM máximas estimadas

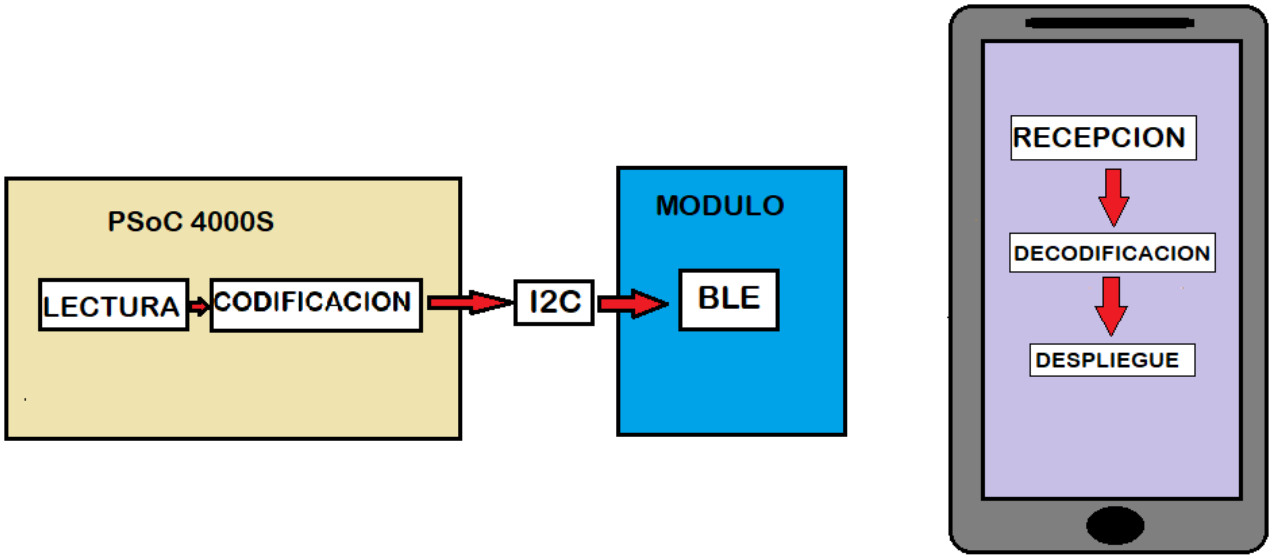


Figura 3.12 Interconexión del sistema

Él envío de datos a través de I2C se hace mediante la creación de un buffer que almacena los datos a transmitir. Dicho buffer es una variable que puede ser una estructura o arreglo. En nuestro proyecto, dado que el BLE transmitirá un dato a la vez, decidimos también enviar un dato a la vez por el I2C. Para lo que creamos un arreglo de buffer de una localidad. El ciclo del programa de adquisición y envió de datos a través del I2C se explica en el siguiente subcapítulo.

3.4 Actualización y reestructuración del software de adquisición de datos.

En este subcapítulo, mostrare el diagrama de flujo del programa actualizado para la adquisición de datos y envió de estos por I2C al módulo BLE y explicaremos como funciona cada bloque y/o función.

En esta versión del software se le ha agregado la comunicación I2C y los tres sensores que nos faltaban en el capítulo anterior (dos de temperatura y un codificador). Las lecturas de cada parámetro se han dividido en funciones para una mejor estructura y entendimiento del funcionamiento general.

Respecto a las lecturas del ADC, estas se han optimizado en código para una mayor rapidez.

Las interrupciones dedicadas a los codificadores se rediseñaron para solo operar cuando se les necesitan y la impresión de datos en pantalla LCD se sigue utilizando para poder observar los datos que se están enviando a la aplicación y poder detectar errores.

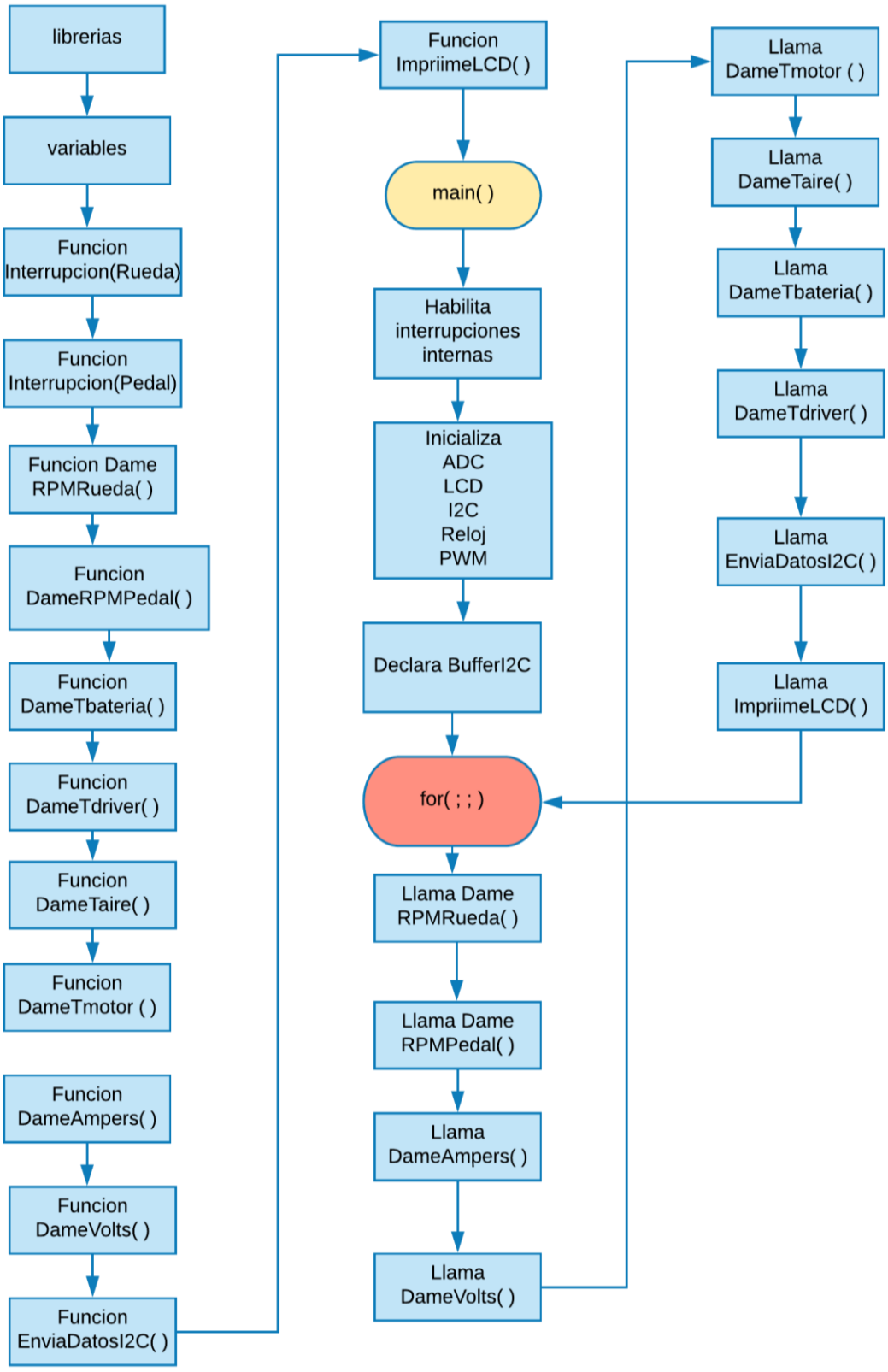


Figura 3.13 Diagrama de flujo del software principal

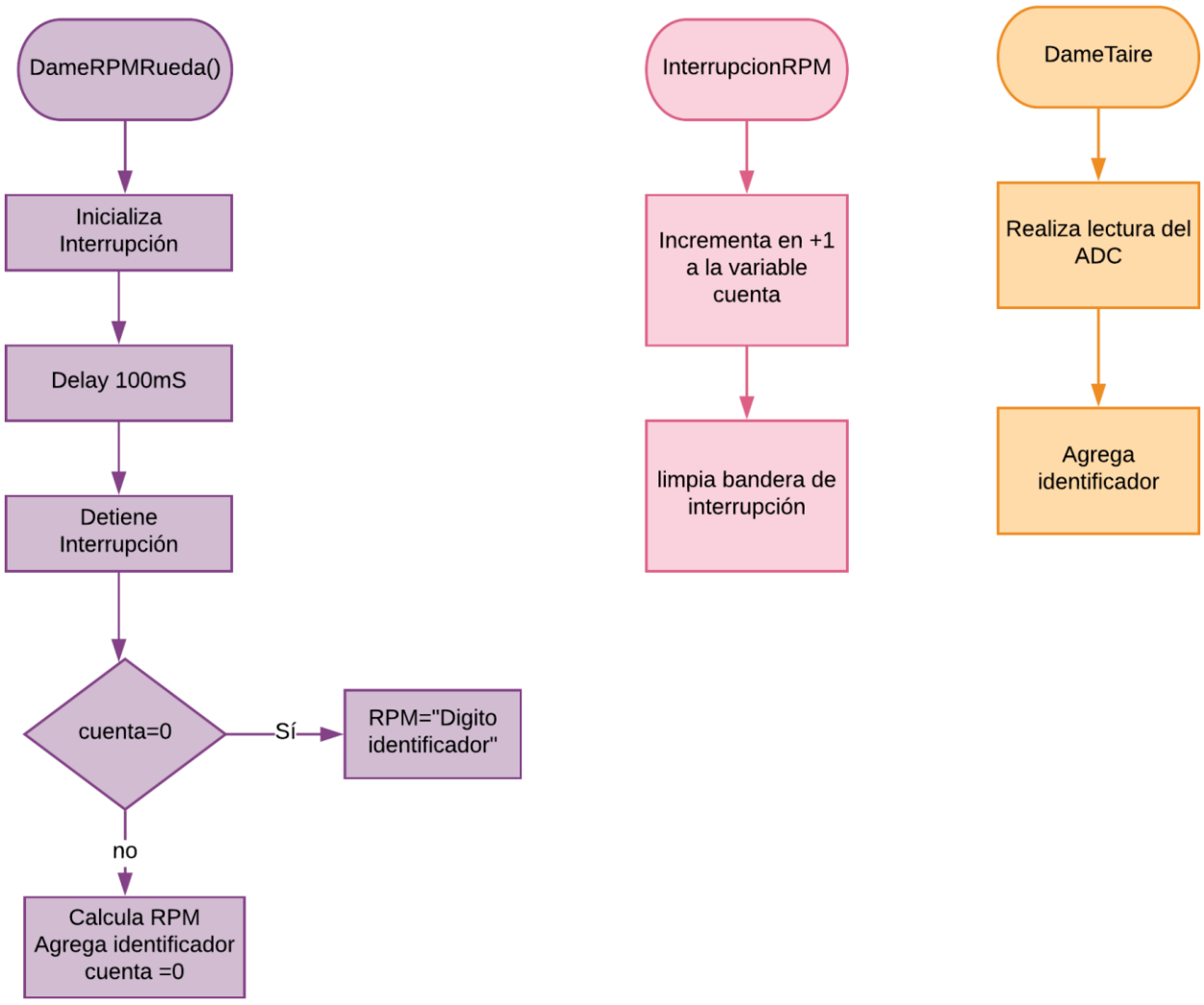


Figura 3.14 Diagrama de flujo de funciones principales de registro de valores de los sensores

En la figura 3.14 observamos las funciones DameRPMRueda, InterrupcionRPM y DameTaire. Cabe mencionar que las primeras dos son idénticas tanto para medir las revoluciones de la rueda como del pedaleo por lo que solo mostraremos una vez para no ser repetitivos. En cuanto a la función DameTaire, esta es idéntica para los cuatro parámetros de temperatura y para el voltaje de la batería.

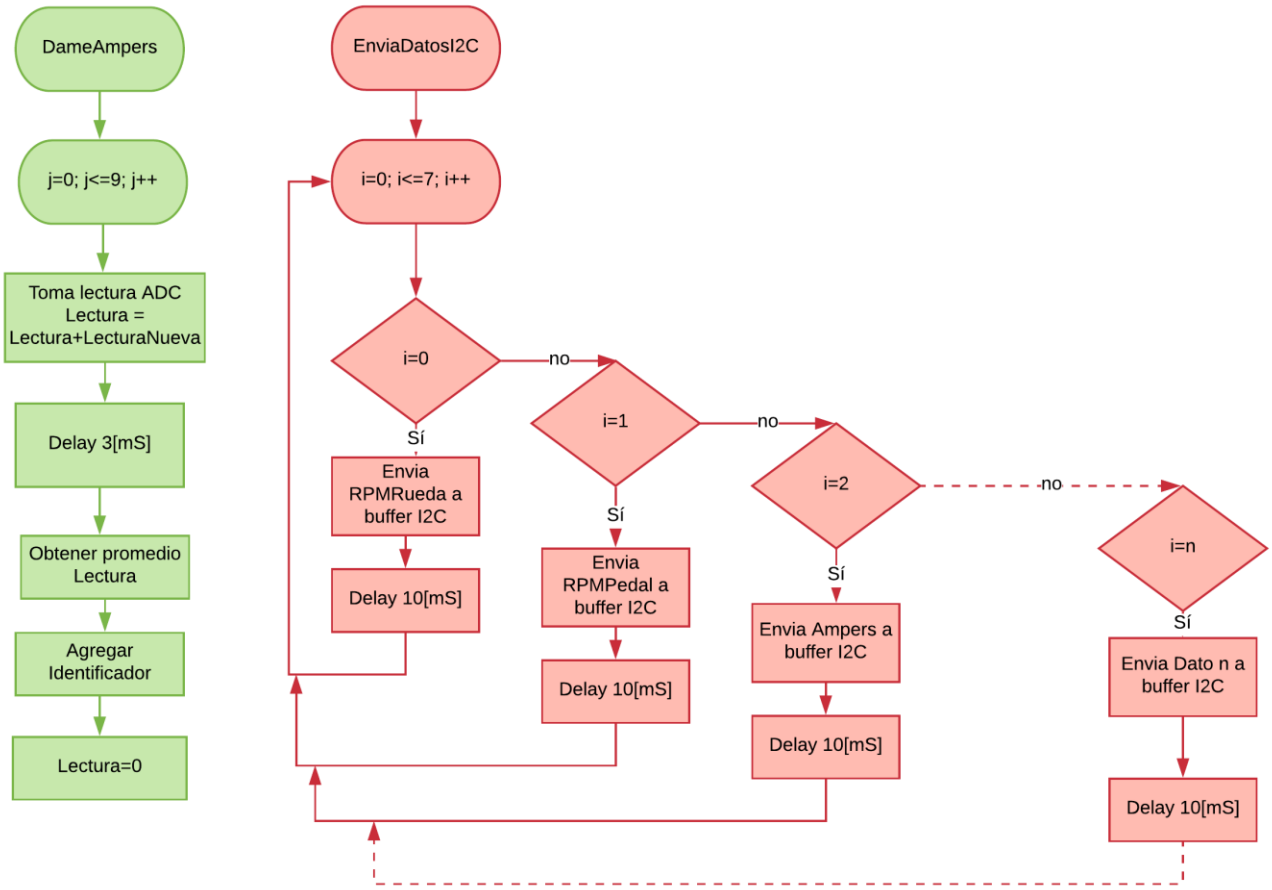


Figura 3.15 Diagrama de flujo de funciones principales

Para obtener la lectura de corriente se diseñó una función llamada DameAmpers () la cual toma 10 lecturas del ADC y las promedia, de este modo ejecutamos un filtro digital y solucionamos el problema que comentamos en el capítulo dos, el cual era que el sensor entregaba valores un poco dispersos. (ver figura 3.15)

Para él envió de datos por I2C, se programó la función EnviaDatosI2C (), la cual se encarga de poner en el buffer un dato a la vez. Después de cargar cada dato, agregamos un retardo para darle tiempo al BLE y a la aplicación de recibir el dato y poder pasar al siguiente. De esta manera evitamos perder información en el transcurso de ejecución de esta función.

3.5 Conclusiones del capitulo

El principal logro en este capítulo fue crear el perfil BLE con su servicio y característica para poder iniciar una comunicación BLE. Aun que lo deseable era enviar cada parámetro mediante una característica diferente, la plataforma de desarrollo de la aplicación no nos lo permitió, por razones que veremos más a fondo en el capítulo siguiente. Esto nos obligó a implementar una serie de identificadores para cada dato, y él envió secuencial de estos. Dejamos para el capítulo cinco dedicado a las pruebas, la optimización de dichos tiempos de envió para obtener la mayor fluidez posible en el despliegue de datos en la aplicación.

CAPITULO 4 DISEÑO Y PROGRAMACIÓN DE LA INTERFAZ PARA ANDROID

En este capítulo, nos centraremos en desarrollar la aplicación para teléfonos móviles cuyo sistema operativo sea alguna versión de Android. Dicha aplicación en su primera etapa de construcción deberá recibir, decodificar y desplegar los datos de censado obtenidos de la bicicleta en tiempo real. Tal aplicación se desarrollará en el ambiente gráfico llamado App Inventor 2. Como primer paso, presentare muy brevemente y a grandes rasgos ¿Qué es? Y ¿Cómo funciona *app inventor*?, para después pasar a un diagrama de flujo del funcionamiento de la aplicación y el desarrollo de esta en la plataforma.

4.1 Plataforma de desarrollo App Inventor

App inventor es un entorno de desarrollo de software para dispositivos móviles Android creado por Google Labs. Este ambiente es de acceso libre y administrado por el MIT.

Fue pensado para incrementar el desarrollo de aplicaciones en un modo sencillo y práctico, al alcanzo de usuarios que no necesariamente sean programadores. De esta manera, de forma visual, el usuario puede crear aplicaciones a partir de un conjunto de herramientas básicas o "bloques".

La simplicidad de dicho entorno juega un papel importante al limitar la complejidad de las aplicaciones creadas, aunque otorga lo suficiente para hacer prototipos de aplicaciones completamente funcionales.

Esta plataforma se divide principalmente en dos partes: *Designer* y *Blocks*.

En "Designer" se diseña la interfaz de la aplicación, es decir, la parte que se mostrara en la pantalla del móvil y con la cual interacciona el usuario. Aquí mismo se eligen los recursos que usara la aplicación, como pueden ser: botones, cuadros de texto, imágenes, relojes, sensores, cámara, micrófono, etc.

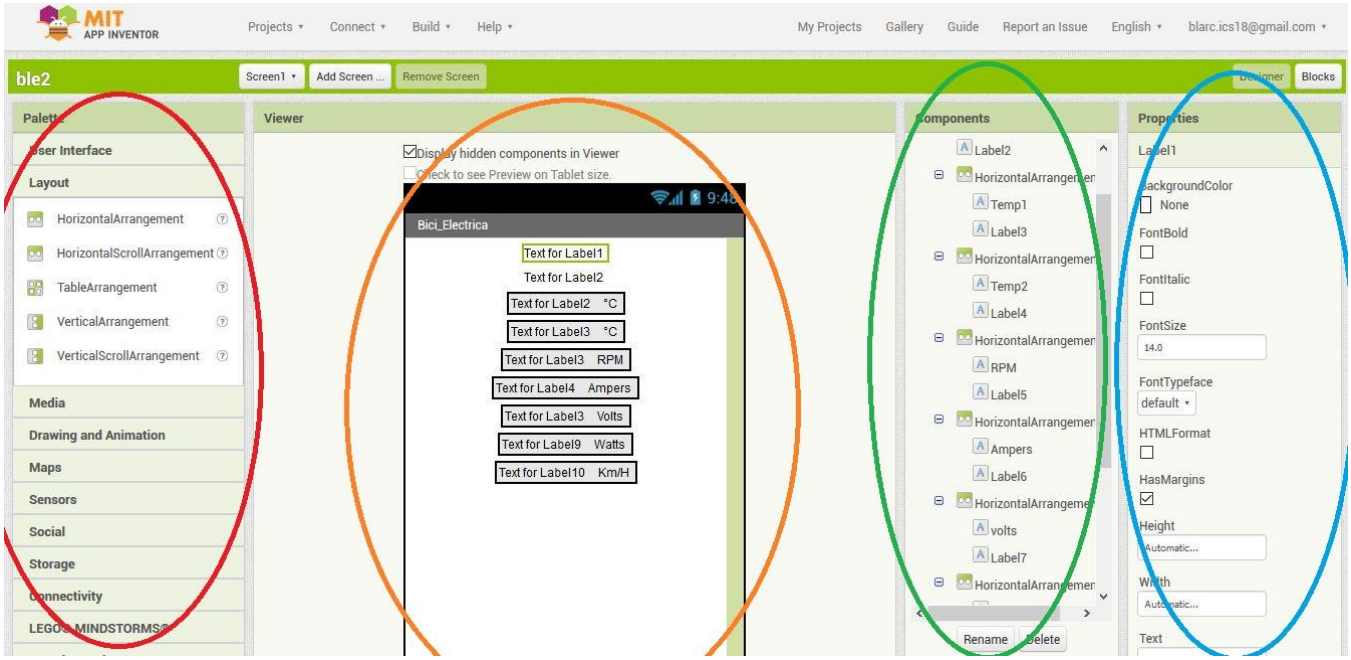


Figura 4.1 Arquitectura IDE App Inventor 2

En la figura 4.1 se observan como se distribuyen los diferentes menús en la sección "designer". Enmarcado en rojo, tenemos la paleta de recursos disponibles para desarrollar la app. En naranja, la previsualización de la pantalla de usuario del móvil. En verde, la lista de componentes que hemos seleccionado de la paleta de recursos. Por último, en azul tenemos las propiedades básicas de los componentes anteriormente seleccionados. En síntesis, en esta parte de la plataforma se seleccionan los recursos a usar y se diseña la parte gráfica.

En "Blocks" está el corazón de la aplicación, es decir, las instrucciones de cómo debe funcionar el sistema. Es aquí donde se unen los bloques que obedecen a la serie de pasos que deseamos que ejecute el programa. Existen bloques para cada uno de los recursos seleccionados anteriormente, así como otros más, dedicados a funciones lógicas, matemáticas, de texto y procedimientos. (ver figura 4.2)

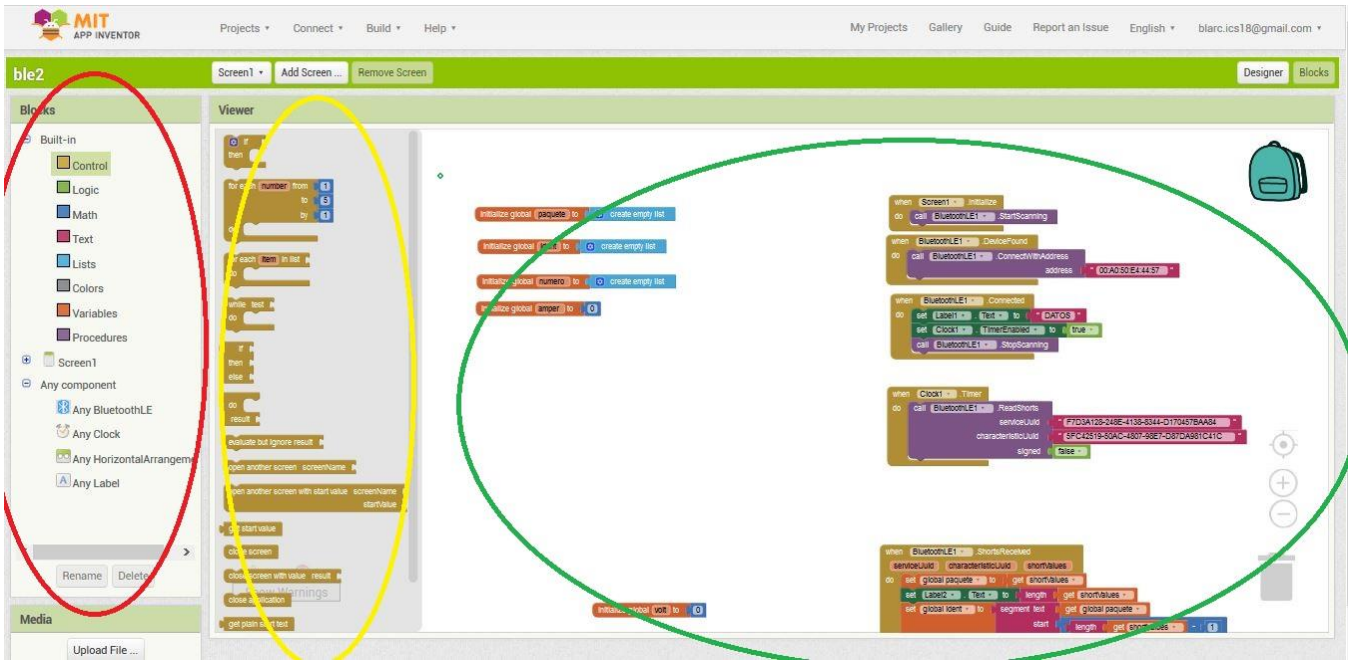


Figura 4.2 Pestaña "Blocks" de App Inventor 2

Ahora en la imagen 4.2 mostramos la parte “Blocks” dividida en tres partes:

En el área roja seleccionamos de la lista de componentes y recursos los bloques que deseamos manipular, en este caso “control”. En el área amarilla observamos los bloques asociados a control. Por último, en el área verde, tenemos nuestro espacio de trabajo, y es aquí donde arrastramos los bloques deseados y los enlazamos para que ejecuten las instrucciones requeridas.

4.2 Diagrama de flujo

Mostraremos en este subcapítulo el diagrama de flujo programado en la plataforma App inventor.

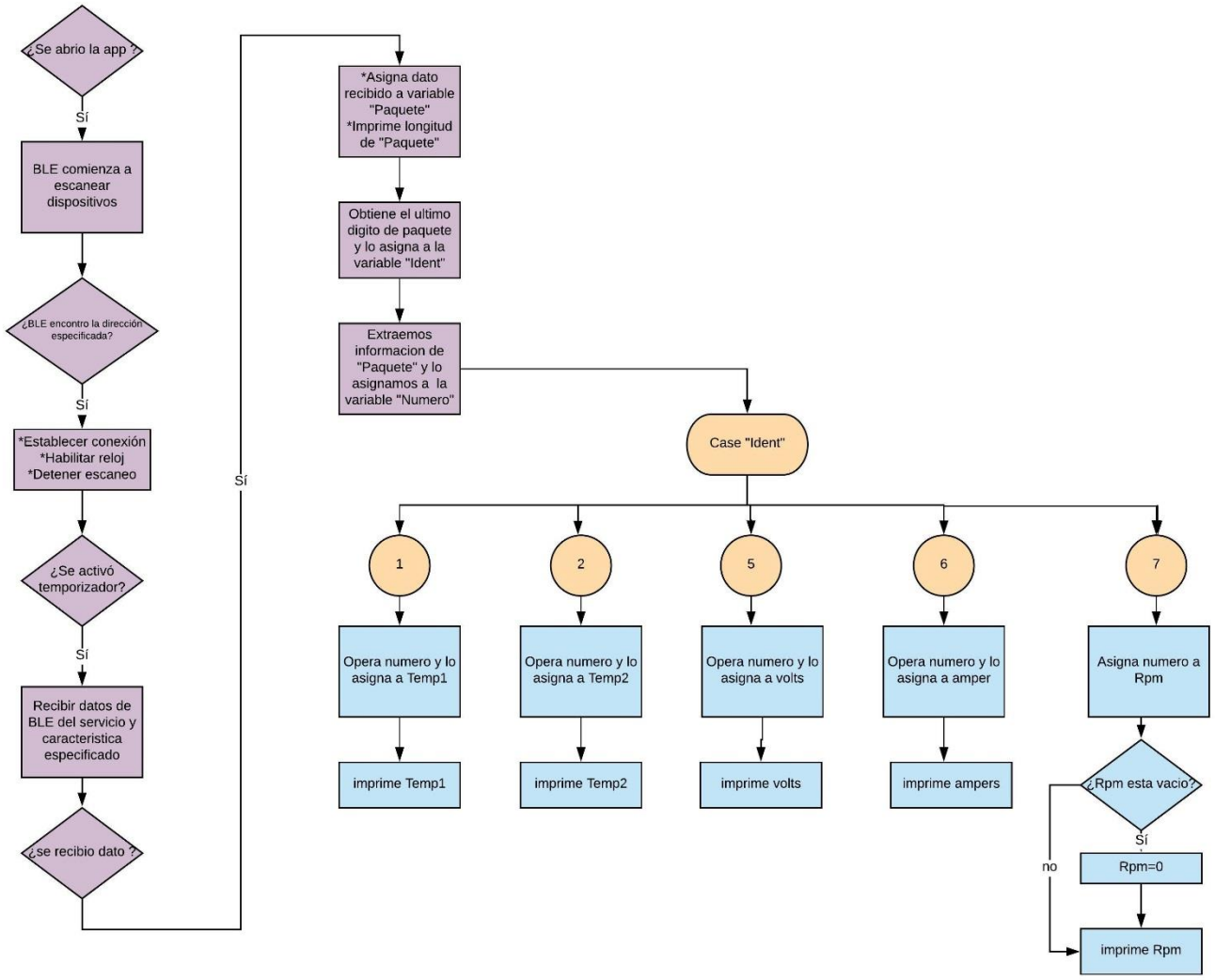


Figura 4.3 Diagrama de flujo del software para la aplicación

En la figura 4.3 solo se muestra dos temperaturas en vez de 4 y un dato de rpm en vez de dos por cuestiones de espacio en el diagrama.

El diagrama de flujo se explica de la siguiente forma: al comenzar a ejecutarse la aplicación, inmediatamente comienza a escanear dispositivos Bluetooth disponibles para conexión, pero solo se podrá conectar en la dirección previamente especificada, la cual es la dirección de nuestro microcontrolador. De esta manera nosotros como usuario solo abrimos la aplicación y los dispositivos se conectan en automático. Una

vez hecho esto se desactiva el escaneo y se habilita un reloj, el cual servirá de temporizador para estar recibiendo datos nuevos cada que el temporizador se dispare. Esto último fue necesario implementar, dado que la librería BLE de la plataforma no actualizaba de manera automática al recibir dato nuevo, lo cual debería suceder. El siguiente paso es configurar los ID del servicio y característica del cual queremos recibir información y preparar la lectura de esta.

Posteriormente guardamos el dato recibido en la variable "Paquete", la longitud de tal variable es importante, pues la necesitamos saber para el proceso donde se obtiene el ultimo dígito de "paquete", el cual es el identificador y se guarda en la variable "Ident". El resto del paquete es la información enviada por los sensores y se guarda en la variable "Numero".

4.3 Procesamiento de datos

Teniendo todo esto se pasa a un proceso de selección el cual depende del identificador que se esté recibiendo. Se procesa el número recibido para poder desplegarlo en pantalla como información legible para el usuario.

4.3.1 Temperaturas

Para las temperaturas es más sencillo pues el dato que se recibe solo se divide entre 100 para obtener un valor real en grados Celsius. Por ejemplo: se recibe un 2491, el ultimo dígito "1" nos indica que es temperatura del aire, entonces nos queda 249 que al dividirlo entre 10 nos da 24.9 [°C], aquí decidimos cuantos decimales queremos mostrar. Para los rangos de temperatura que manejaremos (2[°C]-150[°C]) creo conveniente no usar decimales, pues no serán relevantes en la lectura. Este proceso se realiza para los identificadores 1,2,3 y 4 que son los asignados a las temperaturas de aire, motor, driver y batería respectivamente.

4.3.2 Voltaje

En cuanto al voltaje de la batería, si el dato que se recibe es por ejemplo 15605, el ultimo dígito "5" es su identificador y nos dice que es lectura de voltaje. El 1560 es la lectura entregada en milivolts por el ADC, pero recodemos que tenemos un divisor de voltaje de 10 a 1, por lo tanto, en vez de ser unos 1.560 volts el valor del voltaje en la batería es de 15.60 volts. Entonces en resumen el valor recibido se divide entre 100 para mostrar el valor real de la batería, en este campo creo conveniente usar un decimal para más precisión en el nivel de carga.

4.3.3 Corriente

Para corriente es algo similar, se recibe la información en milivolts con cuatro dígitos más el identificador "6". Con la peculiaridad de que 2500 milivolts es igual a cero amperes por razones propias del sensor que ya explicamos en el capítulo dos. Por lo tanto, al dato recibido se le restaran 2500 y el resultado se dividirá entre 66 que es la resolución del mismo. Ejemplo: se recibe 29856, el ultimo dígito "6" indica que es dato de corriente. A 2985 le restamos 2500 y nos queda 485 que al dividirlo entre 66 nos da 7.36. Este último valor es el correspondiente a amperes.

4.3.4 RPM Rueda y Pedales

En el caso de las RPM, cuando estas son igual a cero, solo se envía el identificador, es por eso que se pregunta si la variable número está vacía, de ser así, se le asigna a la variable Rpm el valor de cero.

El valor de Rpm se recibe tal cual fue censado de la rueda porque se puede desplegar sin procesamiento previo, es decir, si se recibe el 15897, el "7" indica que es dato de Rpm rueda, y el 1589 sobrante es tal cual el dato de revoluciones.

4.4 Posprocesamieto

Cuando hablamos de posprocesamiento nos referimos a las acciones que haremos con los datos ya recolectados para poder mostrar información adicional. Como, por ejemplo: velocidad, distancia recorrida, potencia instantánea, calorías quemadas, energía consumida, autonomía de batería, etc.

4.4.1 Velocidad y distancia

La rueda de la bicicleta tiene un diámetro de 26 pulgadas ó 66.04 centímetros, por lo tanto, por cada revolución de la rueda y sin deslizamientos hay un desplazamiento lineal de 2.07 metros.

$$D = \pi(2r) = \pi\ (66.04\ [m]) = 2.07[m]$$

Ecuación 8 Avance lineal por giro de la rueda

Donde D es el diámetro de la rueda o desplazamiento lineal y r el radio de la misma.

De esta manera solo tenemos que multiplicar las revoluciones de la rueda por 2.07 para obtener la distancia recorrida. Por lo tanto, si habilitamos un contador que este acumulando el total de revoluciones de la rueda y después las convierta a distancia multiplicando por el diámetro de la rueda, tenemos una manera sencilla de saber la distancia total recorrida.

Para saber la velocidad instantánea hace falta obtener un factor, con el cual al multiplicarlo por las Rpm nos de velocidad en kilómetros por hora.

Entonces el factor queda así.

$$\left(\frac{1\,rev}{1\,min}\right)\left(\frac{1\,min}{60\,seg}\right)\left(\frac{2.07\,m}{1\,rev}\right)=\left(\frac{2.07\,m}{60\,seg}\right)\left(\frac{2.07\left(10^{-3}\right)km}{2.07\,m}\right)\left(\frac{3600\,seg}{1\,h}\right)=0.1242\,km/h$$

Ecuación 9 Calculo del factor de conversión de RPM a Km/h

Por lo tanto, las revoluciones por minuto las multiplicamos por (0.1242) para obtener los kilómetros por hora.

4.4.2 Potencia instantánea

Este parámetro es sumamente sencillo pues basta con multiplicar los datos de amperes y voltaje que ya tenemos. De esta manera desplegamos tal información en watts.

$$P=V*I$$

Ecuación 10 Potencia instantánea

4.5 Despliegue de datos en pantalla

Aquí nos ocuparemos de explicar y mostrar cómo se diseñó la pantalla de nuestra aplicación, sobre la cual se despliegan los datos en el teléfono móvil.

La pantalla principal de la aplicación se ve de la siguiente manera. (ver figura 4.4)

Figura 4.4 Pantalla de usuario en la aplicación

En la parte alta y como primer texto en blanco, se mostrará la leyenda "conectado", cuando la conexión haya sido exitosa, en el siguiente texto inmediato inferior se muestra el número de dígitos que conforman al dato recibido, esto es únicamente para saber que se está actualizando constantemente la información recibida, pues cada dato varía en longitud.

Como se puede observar, en los espacios en rosa se mostrarán las temperaturas referentes asociadas, en azul las revoluciones, en amarillo la corriente que circula de la batería hacia la carga, en verde el voltaje de la batería, en rojo la potencia instantánea entregada a la carga y por último en azul marino la velocidad instantánea calculada.

Esto se realizó con el uso consecutivo de *HorizontalArragement* (Disposición horizontal) y *Labels* (Etiquetas), Disponibles en la sección de recursos de la plataforma. Para cada dato se utiliza un *HorizontalArragement* y dos *Labels* dentro de este, el

primero mostrara la información recibida y el segundo informa de que dato se trata y es fijo.

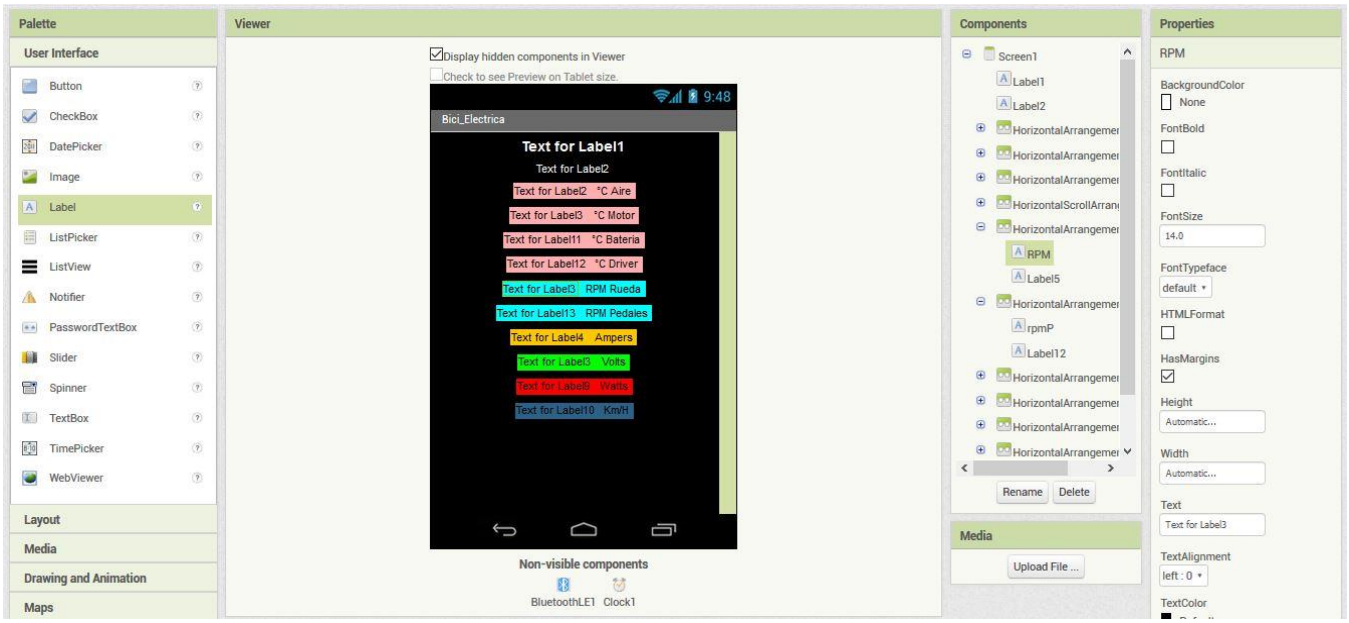


Figura 4.5 Espacio de desarrollo de la parte grafica de la aplicación

En la figura 4.5, se puede observar en la sección de componentes, como está organizada la disposición de los *HorizontalArragement* y *Labels* citadas anteriormente y en propiedades los ajustes de estas mismas. Por último, en la parte baja de la imagen se muestran los "Non-visible components", también seleccionados en la parte de recursos, los cuales únicamente son un reloj y la librería BLE de la plataforma.

4.6 Conclusiones del capitulo

Aunque la plataforma app inventor ofrece poco poder de programación, esta resulta suficiente para llevar a cabo nuestros requerimientos principales, cosa que no se puede decir en lo que respecta a la parte gráfica de nuestra aplicación, pues esta queda absolutamente limitada, pero cumple con su función. Es en este último punto y en lo que involucra el posprocesamieto y almacenamiento de los datos es donde se pueden hacer muchas mejoras en trabajos futuros para este prototipo.

Por razones de tiempo y de confidencialidad del prototipo para un eventual producto comercial terminado, queda pendiente el desarrollo de los siguientes puntos: los avisos de temperaturas críticas, la información referente a calorías quemadas por el usuario, la distancia total recorrida, la energía consumida y actual disponible de la batería.

CAPITULO 5 PRUEBAS DEL PROTOTIPO

En esta sección abordaremos la realización de las pruebas de precisión y funcionamiento de cada uno de los sensores instalados. Dichas pruebas se centrarán en observar los datos arrojados por el sistema directamente en el dispositivo móvil, es decir, en la interfaz con el usuario y compararlos con dispositivos de medición comerciales tales como pueden ser: GPS, estroboscopio, multímetro, termómetro, etc.

5.1 Diseño de pruebas

Como primera prueba, nos centraremos en el medidor de velocidad o rpm del motor, ya que los demás parámetros tales como: voltaje, corriente, potencia, etc. Las analizaremos con respecto a este parámetro. Por lo tanto, es de suma importancia que dicho dato sea preciso pues de aquí parten las demás pruebas del sistema.

Para dicha prueba es necesario comparar los datos que arroje el teléfono móvil contra otro u otros dispositivos comerciales cuya precisión sea confiable y demostrada. Para tales efectos haremos uso de una cámara deportiva de alta gama de la marca Garmin Virb, la cual incluye GPS y acelerómetro, y es capaz de grabar a 1080p y 30 [fps]. Con este dispositivo grabaremos en video y al mismo tiempo los datos que despliega tanto el teléfono como el display genérico de la bicicleta. En un análisis posterior del video y a una velocidad de un cuadro por segundo (1[fps]), tendremos tres datos de velocidad (GPS, display y prototipo) por cuadro de video que podremos graficar y comparar. Esta prueba será la única en realizarse de manera dinámica, es decir con la bicicleta en movimiento, las siguientes serán de manera estática, con el vehículo montado en un soporte central que permita el libre rodamiento de la rueda trasera donde va instalado el motor. Esto para simplificar la toma de datos, pues de igual manera, se compararán los datos arrojados por el prototipo contra un multímetro digital (para el caso de voltaje y corriente) y grabados en video de la misma manera que en el caso de velocidad.

5.2 Pruebas de velocidad.

Como ya mencionamos antes, esta prueba será en movimiento, por lo tanto, usaremos un arnés para fijar la cámara deportiva al pecho del usuario y desde ese ángulo poder grabar los datos desplegados por el prototipo y el display de la bicicleta. (ver figura 5.1)

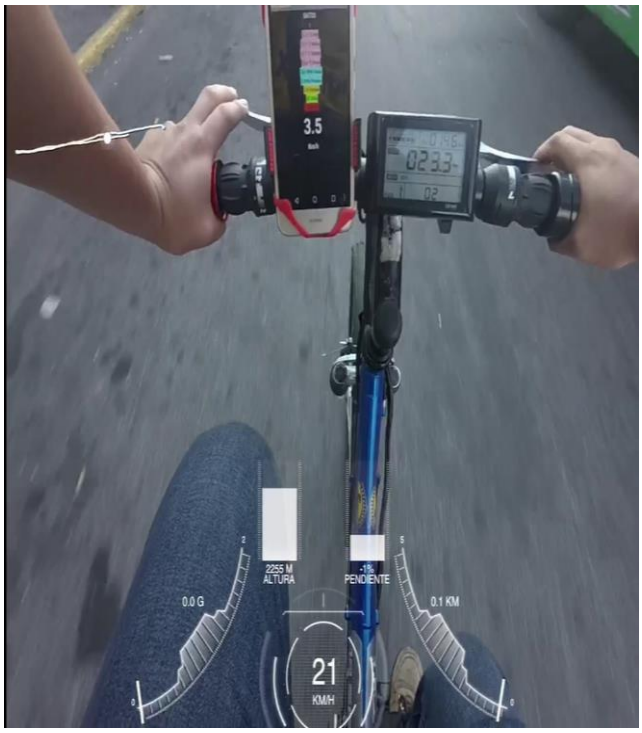


Figura 5.1 Prueba dinámica del prototipo

La primera corrección que se hizo al realizar esta prueba fue la de hacer más grande y contrastante la información de velocidad desplegada en el celular, pues de la manera en que se mostraba en un principio, no se alcanzaba a distinguir los dígitos.

Esta prueba se realizó en una superficie plana con aceleraciones y desaceleraciones durante alrededor de dos minutos.

Los datos obtenidos después de analizar el video a un cuadro por segundo se muestran en la figura 5.2.

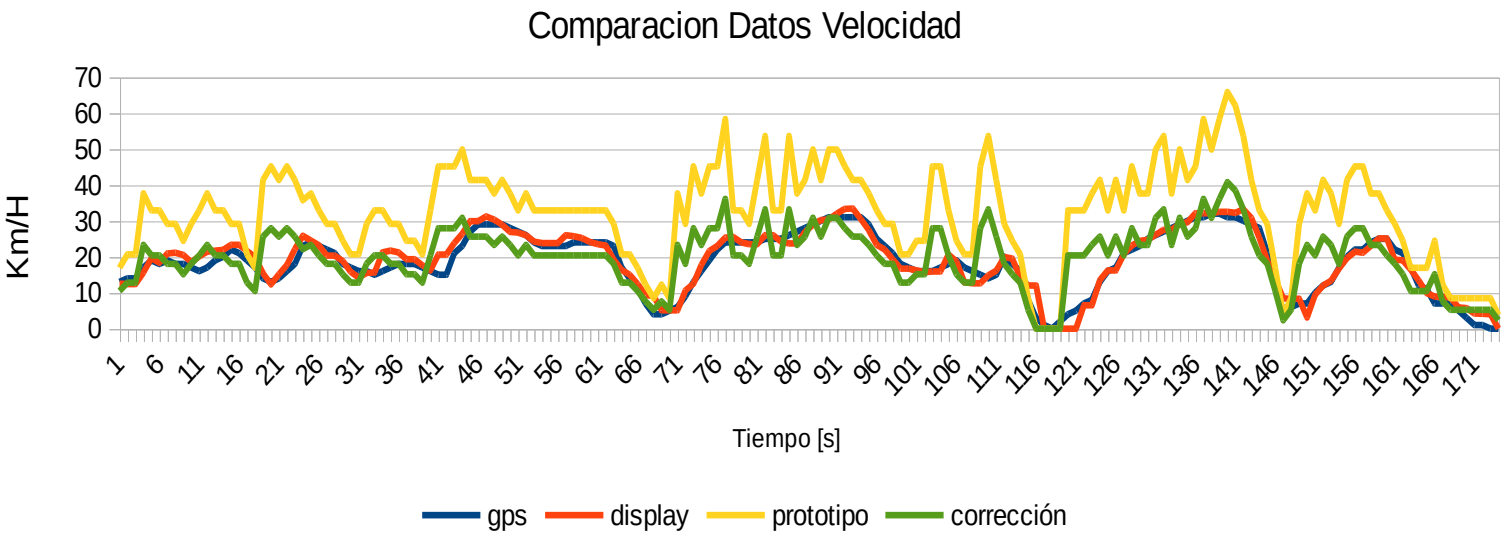


Figura 5.2 Grafica de velocidades

De la figura 5.2 se pueden deducir las siguientes conclusiones:

1.- Los datos de GPS y display son prácticamente iguales, con una diferencia de tan solo 4.3%. Por lo tanto, para efectos de esta y pruebas posteriores será considerado como el valor de referencia.

2.- En las aceleraciones y desaceleraciones se muestra una respuesta más rápida del prototipo, incluso de alrededor de dos segundos en comparación al tiempo que tardan las señales en ser recibidas y envidas por los satélites de la red GPS.

Es importante mencionar que para esta prueba las ocho entradas de adquisición, procesamiento y despliegue de datos estaban operando completamente, por lo que es de resaltar la velocidad del sistema funcionando al 100%.

3.- La gráfica en amarillo muestra los datos desplegados tal cual fueron entregados por el prototipo, la cual se encuentra "escalada" en comparación con los datos GPS. Estos datos arrojan una diferencia de +41.1% respecto a las referencias

En verde se muestra la respectiva corrección hecha con base en el promedio de error. Es decir, al escalar los datos al 58.9% (quitando el +41.1% de error arrojado por la prueba).

Este error lo atribuimos a dos cosas, 1) El código de adquisición de datos presenta una alteración en el conteo de veces que atraviesa el disco perforado a través del codificador y por lo tanto las rpm están incorrectas, así como también los rebotes de la señal de entrada pueden estar marcando falsos pulsos positivos. 2) El cálculo para convertir [rpm] en [km/h] es equivocado, esto podría deberse a que no se consideraron las medidas reales de la rueda.

4.- Los datos del prototipo muestran demasiadas perturbaciones o picos, lo cual se debe a que los saltos entre cada uno son de 2.6 [Km/H]. Esto lo sabemos por el hecho de que por cada ventana que detecta el codificador colocado en la rueda cada 100[mS], traducido en [RPM] son 33 o 2.6 [Km/H]. Esto se explica con la siguiente ecuación.

$$RPM=(Conteo)\ (600)\ /\ (18)$$

Ecuación 11 Calculo de RPM con base en conteo de pasos por ventana

Donde Conteo es igual a el número de ventanas detectadas por el sistema en un lapso de 100[mS], después al multiplicarlo por 600 obtenemos el número de pasos en un minuto, por último, dividimos entre 18, pues es el número de ventanas que representa una revolución de nuestra rueda. Al sustituir el Conteo por "1" obtenemos 33 [RPM], es decir el salto entre lecturas siempre será de 33 RPM, que relacionado con el diámetro de la rueda nos queda 2.6 [Km/H]. Por lo anterior, es necesario reducir estos saltos para obtener una lectura más lineal.

Después de analizar todos estos puntos considero aplicar las correcciones necesarias a dicho parámetro para poder continuar con las demás pruebas, ya que las RPM serán la base de comparación para los otros sensores.

5.2.1 Análisis de datos y correcciones.

Se aplicaron tres correcciones a lo que respecta la adquisición de datos de RPM. La primera fue cambiar el umbral de voltaje de entrada para el puerto digital. Ya que manejábamos valores de tecnología CMOS, cambiamos a valores de TTL, pues son

estos niveles los que entrega el sensor. Con este cambio logramos una respuesta con menos perturbaciones pues manejamos una configuración "pull up" a la entrada, es decir se activa con "ceros", por lo tanto, se hace más pequeño el umbral para detectar el "0" lógico, de (0-1.5[V] CMOS a 0-0.8[V] TTL). También se comprobó que el dispositivo solo se activara con el flanco de subida de la señal, pues llegamos a creer que podría estar contando ambos flancos, tanto el de subida como el de bajada. De igual manera se corroboro que los cálculos de conversión y medida de la rueda fueran correctos. Pese a todo esto el error de escala seguía presentándose, quedando como único motivo que la señal del codificador presenta ruido de magnitud y frecuencia constante, por ello se optó por hacer el ajuste directamente en la aplicación, es decir el dato de RPM recibido lo multiplicamos por nuestro factor de corrección obtenido experimentalmente.

Por último y como principal corrección, se rediseño el disco del codificador, pasando de 18 ventanas a 36, es decir pasamos de 20° a 10° como ángulo de detección del giro. El tiempo de detección paso de 100 [mS] a 125 [mS], todo esto para asegurar que los datos en [Km/H] (ya con el factor de corrección) tengan saltos de 1[Km/h] entre lecturas. Esto queda demostrado en las siguientes ecuaciones.

$$RPM=(Conteo)\ (480)\ /36$$

Ecuación 12 Calculo de RPM optimizado

al sustituir "Conteo por 1" en la ecs. 12:

$$RPM=(1)(480)/(36)=13.3$$

Calculamos los cambios de velocidad mínimos registrados con las modificaciones efectuadas

$$Km/H=\ RPM*(Factor\ de\ conversión\ corregido)$$

$$Km/H=\ 13.3*(.0803)\ =\ 1.07$$

Ecuación 13 Velocidad mínima registrada

Después de todo esto, los datos arrojados por la segunda prueba de velocidad son los mostrados en la siguiente gráfica.

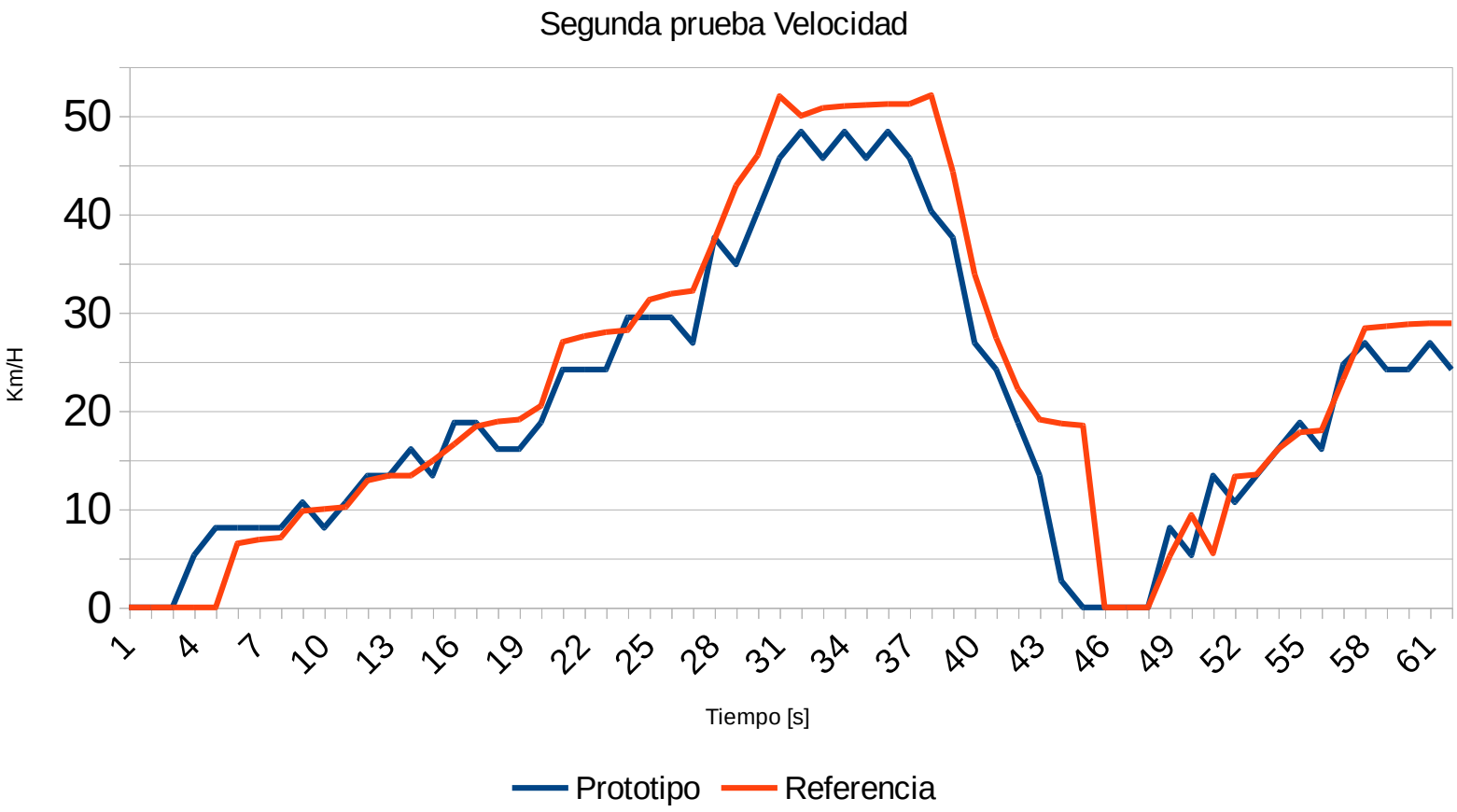


Figura 5.3 Grafica de velocidades (prueba 2)

Con la realización de esta prueba es evidente que nuestro prototipo se ajusta de manera muy aceptable al valor de referencia con los cambios efectuados anteriormente. El porcentaje de error promedio ahora es de -5.1%. Por lo tanto, seguiremos adelante con las demás pruebas y daremos esta por concluida exitosamente.

5.3 Pruebas de Corriente directa instantánea

De la misma manera que hicimos para la prueba de velocidad, tomaremos nota de los datos arrojados por el prototipo y los compararemos contra un amperímetro digital de la marca Uni-T modelo UT33C para después graficarlos y hacer los ajustes necesarios. Esta primera prueba se hará con la bicicleta montada en un soporte central que permite el libre giro del motor, es decir, sin carga.

Los datos obtenidos de la primera prueba de corriente sin carga al motor se muestran a continuación. (ver figura 5.4)

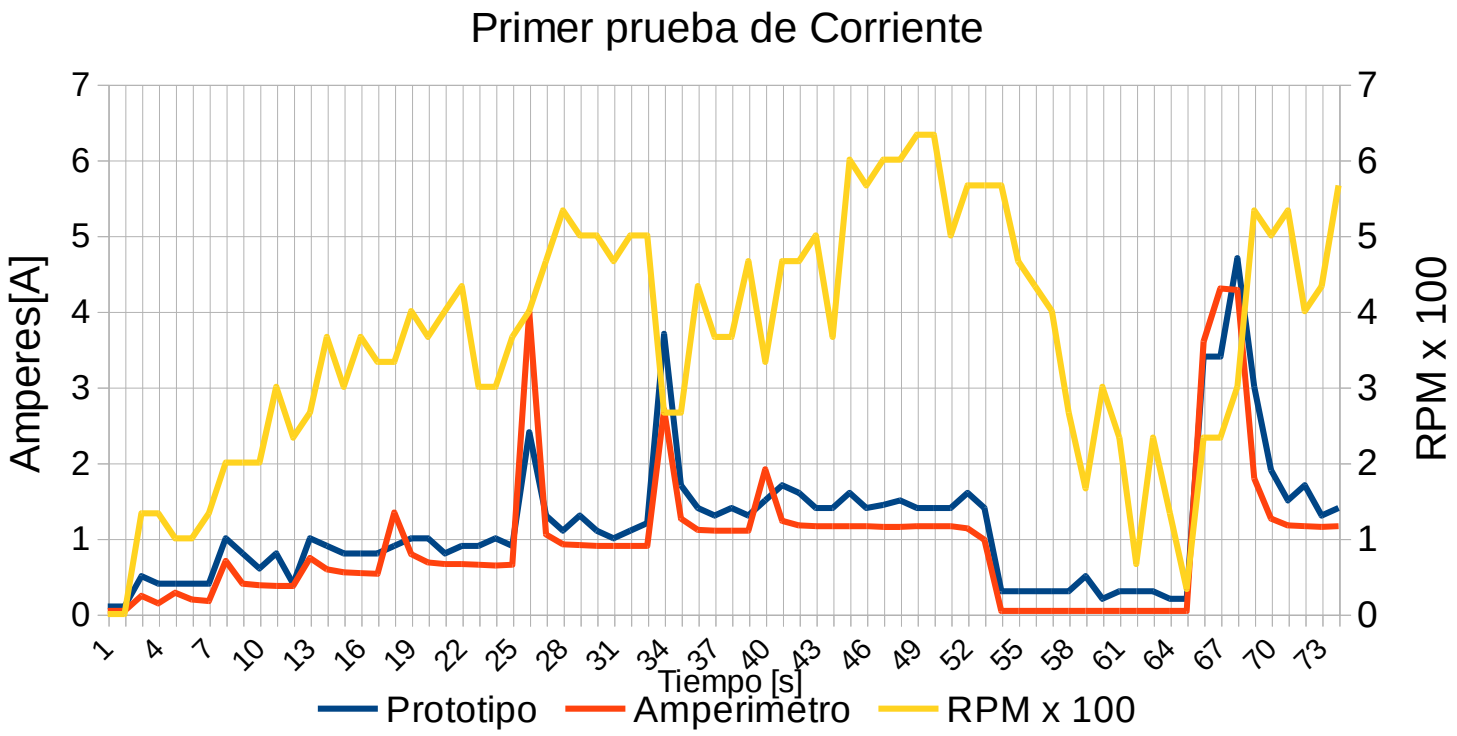


Figura 5.4 Grafica de corriente instantánea

En la figura 5.4 se presenta tanto los datos que deseamos comparar del amperímetro y del prototipo como las revoluciones por minuto escaladas 1:100 para que pueda ser visualizado en la misma gráfica.

5.3.1 Análisis de los datos y correcciones.

Valor promedio de corriente: 0.92 [A]

Diferencia promedio porcentual entre Prototipo y amperímetro = 33%

La prueba se hizo en aceleración constante salvo en 3 puntos, en donde se acciono el acelerador al máximo (en el segundo 25, 34 y 67), en este último se obtuvo el pico máximo de corriente pues el motor estaba en completo reposo.

De esta prueba destacamos dos cosas principalmente.

1) La velocidad de despliegue de datos prácticamente es igual, un hecho muy bueno pues estamos comparando contra un dispositivo comercial que solo se dedica a censar un parámetro.

2) La diferencia entre ambas lecturas es de alrededor de 270 [mA] (30% de error de acuerdo al valor medio de corriente) durante toda la prueba. Lo cual es perfectamente atribuido a una cosa: recordemos que el sensor tiene la peculiaridad de entregar [Vcc/2] a la salida cuando la corriente es igual a cero y una resolución de 66 [mV/A]. Si Vcc es igual a 5[V] y la corriente que circula es igual a 1[A], a la salida nos entrega 2566 [mV]. La línea de software que calcula este parámetro hace el siguiente calculo: (Lectura- 2500 / 66 = Corriente), es decir no contempla cambios en el Voltaje de alimentación Vcc. Por lo tanto, si Vcc varia o tiene voltaje de rizo de alrededor de ± 20[mV] se verá directamente reflejado en el valor de corriente entregado en ± 300[mA]. Es decir que mientras más alta sea la corriente que circula, el error será menor, pues el voltaje de alimentación Vcc que produce el error es independiente. Y para las corrientes nominales de operación del vehículo de 0 a 25 [A], 250[mA] resultan casi despreciables.

En conclusión, esta prueba fue totalmente satisfactoria con lo que respecta a precisión y velocidad en la lectura y no necesita ajustes mayores el sistema, salvo una alimentación Vcc al prototipo lo más justa y estable posible a 5 [V].

Segunda prueba

La segunda prueba de corriente será realizada de manera dinámica, es decir condiciones normales de uso, donde afectaran el peso total del vehículo y usuario, pendiente de camino, etc. Dado que la corriente máxima del amperímetro digital con el que se realizó la primera prueba es de 10 [A] y las corrientes que se esperan registrar son de alrededor de 20 [A], esta prueba solo se realizara con el prototipo.

Los resultados de la prueba dinámica de corriente son los mostrados en la figura 5.5.

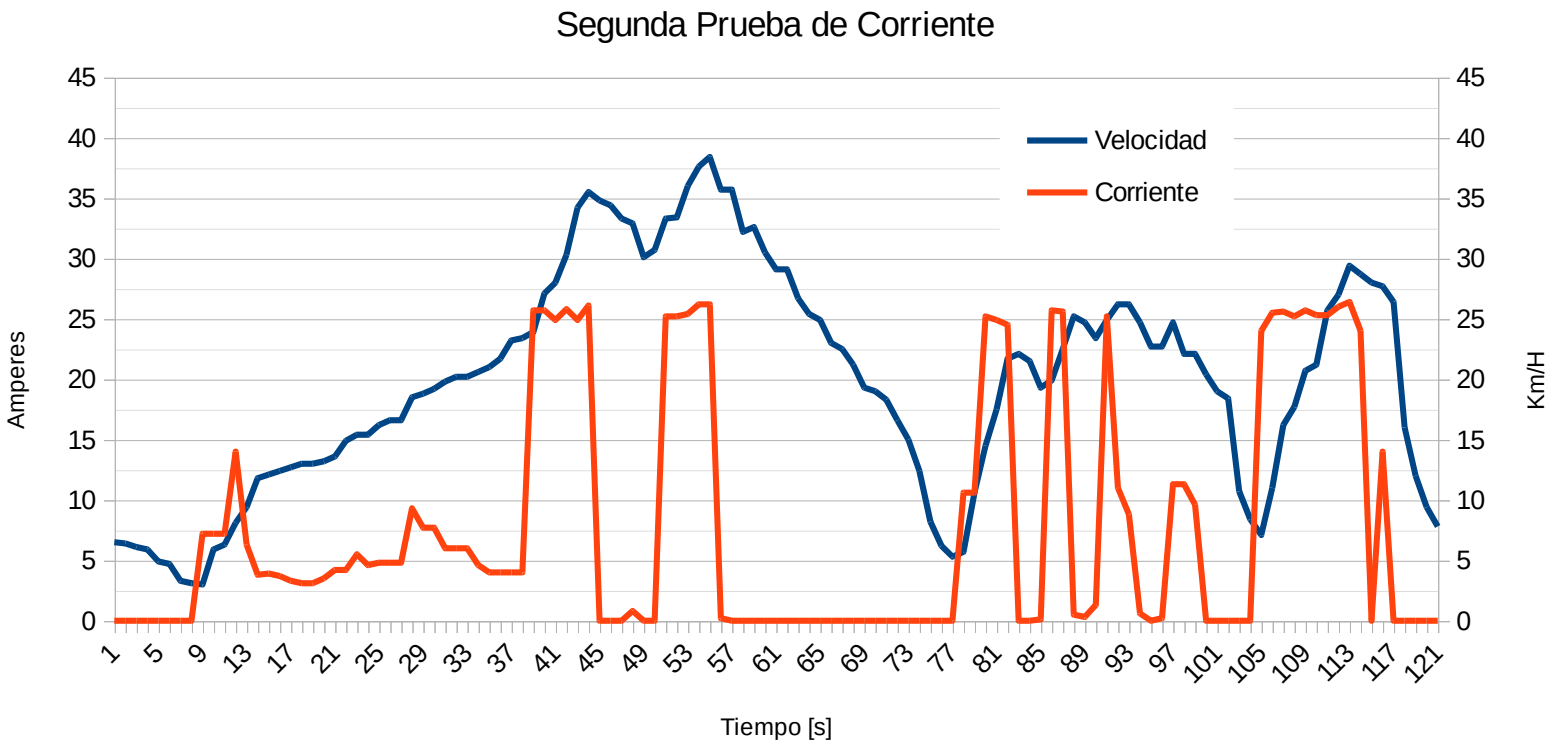


Figura 5.5 Grafica de corriente (prueba 2)

Esta prueba se realizó en una superficie plana, con un peso de usuario de 70 [Kg], y como se observa la corriente máxima es de alrededor de 25 [A] y una velocidad aproximada de 38 [Km/H]. Ambos datos son tomados del prototipo. Y como se aprecia la respuesta es limpia, rápida y sin perturbaciones. Por lo tanto, concluimos hasta este punto como exitosos los sistemas de adquisición de datos de Velocidad (Rpm rueda), y corriente instantánea.

5.4 Pruebas de voltaje en batería.

Para esta primera prueba se utilizó el mismo multímetro de la prueba pasada. Y se comparó en paralelo lo datos del mismo con los del prototipo. La bicicleta se montó sobre el banco de pruebas, es decir sin carga. La batería se encontraba cien por ciento cargada. (ver figura 5.6)

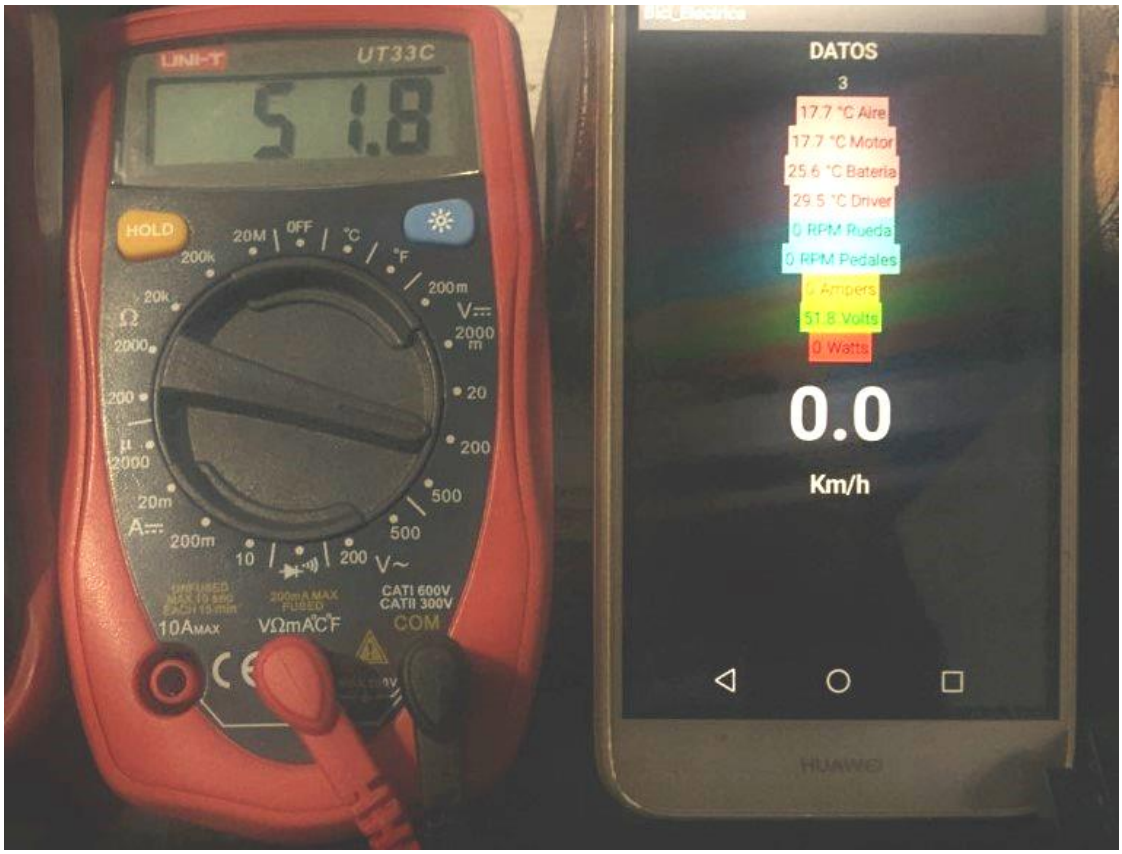


Figura 5.6 Comparación entre multímetro y prototipo

En la figura 5.6 se muestra cómo se tomaron los datos de comparación entre los dispositivos.

Dichos datos se muestran en la figura 5.7.

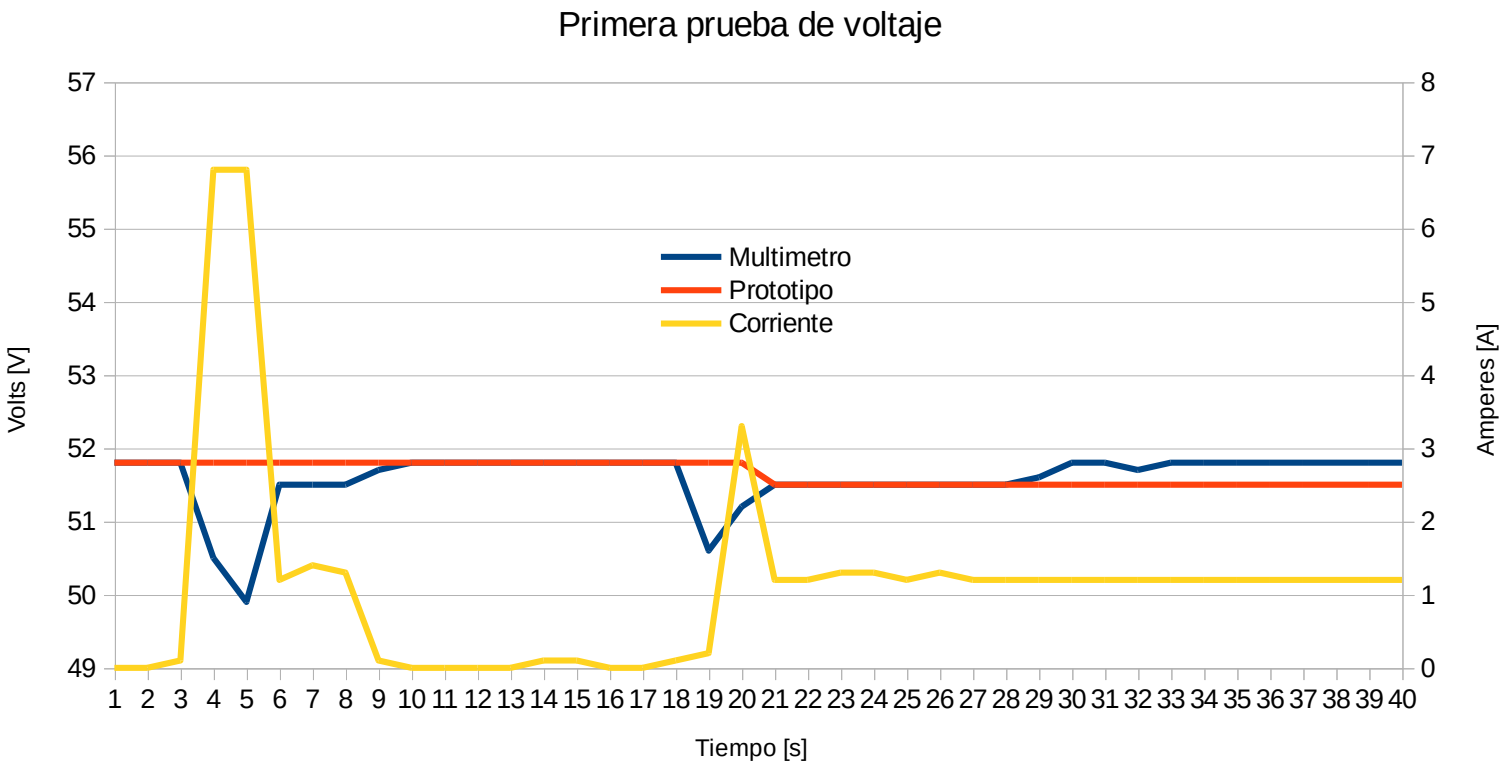


Figura 5.7 Grafica de voltajes

5.4.1 Análisis de datos y correcciones

De tales resultados, podemos concluir lo siguiente: los datos arrojados por el prototipo se ajustan de manera muy aceptable a los entregados por el multímetro. Ya que promediando la diferencia entre ambas lecturas obtenemos un 0.13% de diferencia. En los picos de corriente que es donde el voltaje baja durante unos instantes, el prototipo actúa como filtro pues estas oscilaciones no las muestra, esto es debido a la configuración de las etapas de divisor de voltaje y seguidor, dado que por cada 100[mV] que oscile la batería el divisor lo hace en 5[mV], después estos pequeños cambios el software los redondea y es por eso que no se observan. Durante la segunda prueba, que se realizara con carga, las variaciones de voltaje serán más evidentes

Segunda prueba.

Esta prueba se realizó comparando los datos de corriente y de voltaje únicamente del prototipo, solo por la dificultad que supone llevar el multímetro conectado en movimiento, además de que consideramos evidente que nuestros datos son lo suficientemente apegados a lo real, como quedó demostrado en las pruebas anteriores. Los resultados se muestran en la figura 5.8.

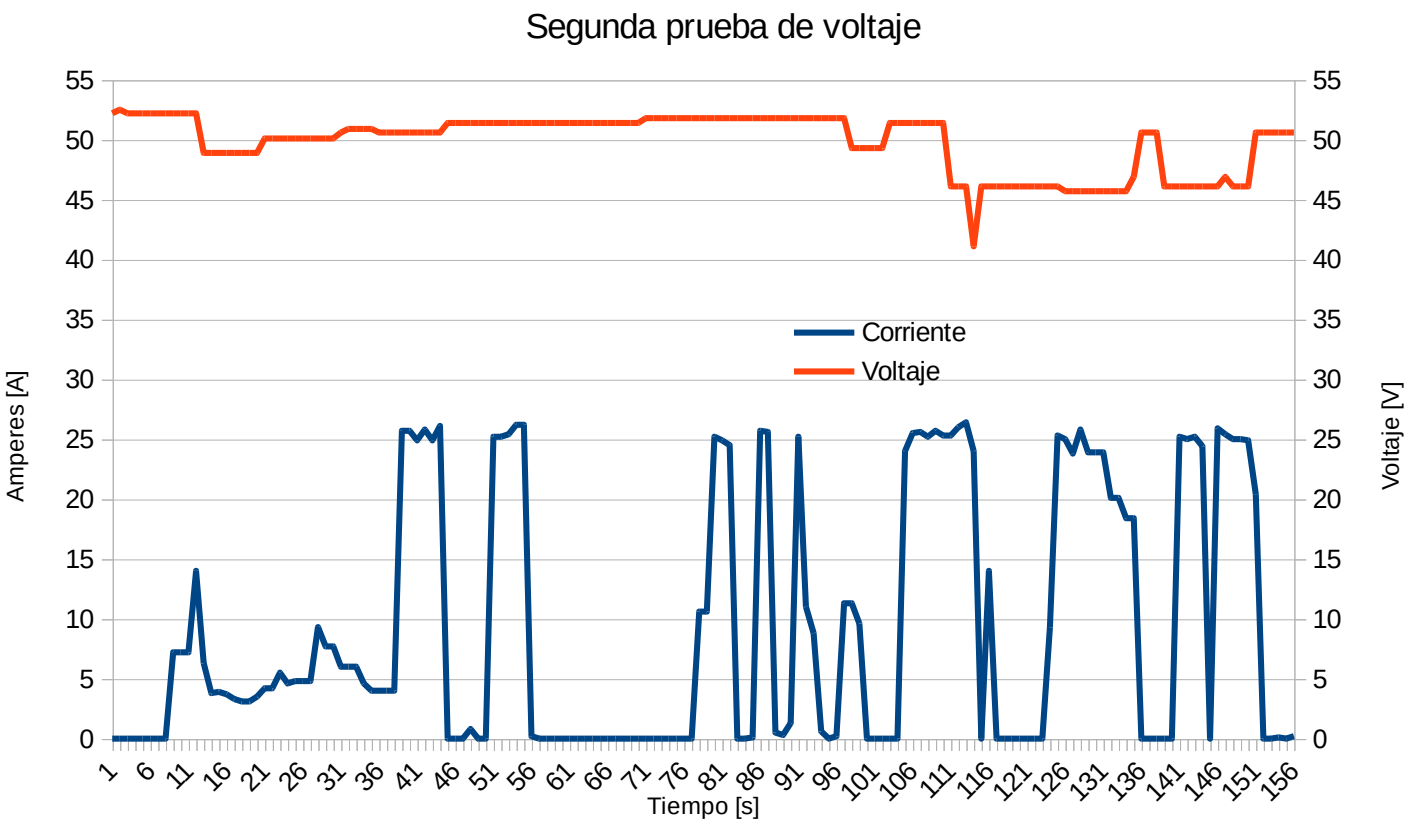


Figura 5.8 Grafica de voltaje (prueba 2)

Como se observa las variaciones en el voltaje de la batería ya son más evidentes, debido a que la corriente de consumo es de hasta 25[A]. Se aprecia que entre más tiempo se mantenga ese consumo de corriente alto, el voltaje disminuye más.

En conclusión, al llegar a este punto ya podemos calcular la potencia instantánea del sistema, debido a que los datos de corriente y voltaje adquiridos se pueden asumir como correctos.

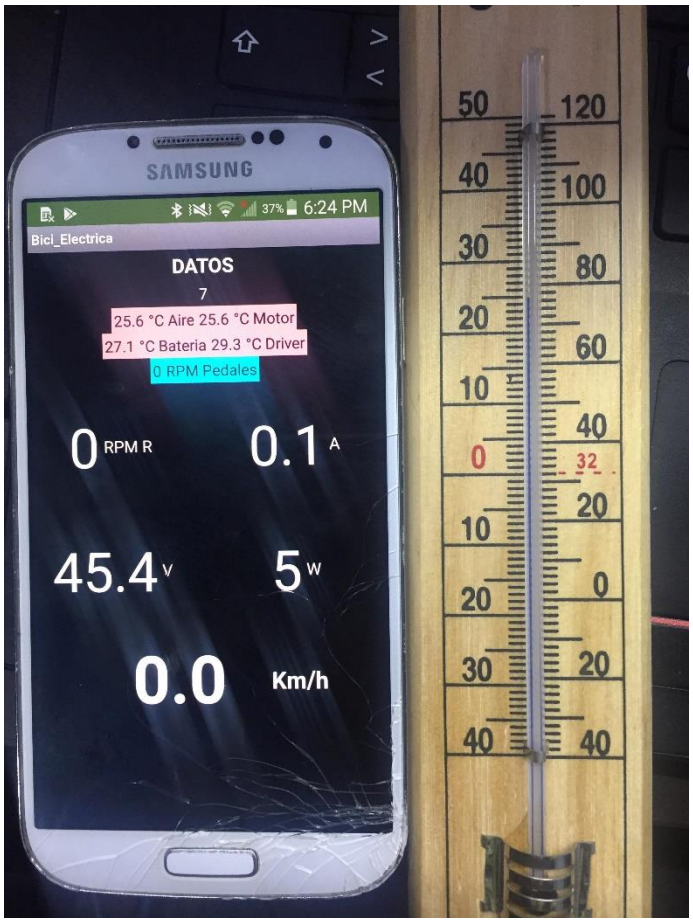
5.5 Pruebas de temperaturas

Según el fabricante de los integrados LM35, en teoría estos sensores no necesitan calibración, en la práctica nos damos cuenta de que por diferentes motivos como pueden ser *offsets* y/o ruido en la línea de datos por ser esta demasiado larga, así como

problemas mismos de la placa donde van montados, estos pueden entregar datos un tanto diferentes a los reales.

Nuestro sistema no es la excepción a estos casos, y es por eso por lo que es imprescindible una calibración a los 4 sensores del prototipo.

Como primer paso tomaremos nota de los datos arrojados por los cuatro sensores a temperatura ambiente en diferentes ocasiones. Esta temperatura de referencia será tomada con un termómetro analógico de mercurio. De esta manera observaremos el desfase que presenta nuestro prototipo y haremos los ajustes necesarios. (ver siguiente figura)



Toma de datos en las pruebas de temperatura (interfaz gráfica rediseñada)

Sensor	Temp. Registrada[°C]	Temp. referencia[°C]	Desfase[°C]	Desfase promedio[°C]
Aire	17.6	21.6	-4	-2.9
	17.6	21.0	-3.4	
	17.6	20.0	-2.4	
	17.6	22.0	-4.4	
	15.6	22.0	-6.4	
	17.7	20.0	-2.3	
	17.7	18.5	-0.8	
	17.7	19.0	-1.3	
	17.7	19.0	-1.3	
Motor	17.6	21.6	-4	-3.5
	15.6	21.0	-5.3	
	17.6	20.0	-2.4	
	15.6	22.0	-6.4	
	13.7	22.0	-8.3	
	17.7	20.0	-2.3	
	17.7	18.5	-0.8	
	17.7	19.0	-1.3	
	17.7	19.0	-1.3	

Tabla 5 Pruebas de Temperaturas 1

	23	21.6	+2.4	
	23.5	21.0	+2.5	
	19.6	20.0	-0.6	
	23	22.0	+1	

Batería	23.5	22.0	+1.5	+3.1
	27.6	20.0	+7.6	
	23	18.5	+4.5	
	25.6	19.0	+6.6	
	21.7	19.0	+2.7	
Driver	29.4	21.6	+7.8	+8.8
	29.4	21.0	+8.4	
	29.4	20.0	+9.4	
	29.4	22.0	+7.4	
	25.4	22.0	+3.4	
	31.5	20.0	+11.5	
	29.2	18.5	+11.7	
	31.5	19.0	+12.5	
	27.6	19.0	+8.6	

Tabla 6 Pruebas de temperatura 2

Los datos obtenidos de las diferentes pruebas de temperatura se muestran en la figura 5.9 y tabla 6.

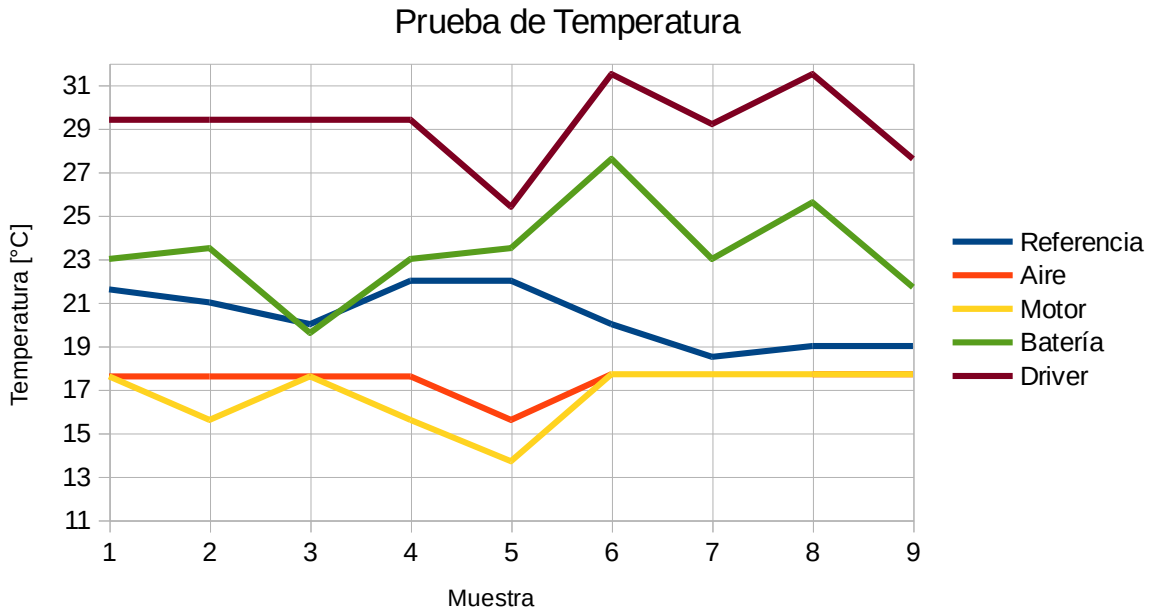


Figura 5.9 Grafica de temperaturas

5.5.1 Análisis de datos y correcciones.

Analizando los datos y promediando los desfases entre cada lectura, se puede llegar a las siguientes conclusiones.

El desfase promedio de la lectura de cada uno de los sensores es:

Temperatura Aire: -2.9 [°C]

Temperatura Motor: -3.5 [°C]

Temperatura Batería: +3.1 [°C]

Temperatura Driver: +8.8 [°C]

Si recordamos que el sensor tiene una resolución de 10[mV/°C], esto quiere decir que tenemos en el sistema un offset y/o ruido de hasta 88 [mV].

Por lo tanto, para resolver dicho desfase, resolvimos ajustar el valor de cada parámetro directamente en la aplicación del móvil, es decir, cuando se recibe cada dato de temperatura, a este se le suma o se le resta el desfase correspondiente. De esta manera, acercamos de manera satisfactoria los datos adquiridos al valor de referencia.

5.6 Conclusiones del capitulo

El punto clave de este capítulo es el diseño de las pruebas. El hecho de que se hayan comparado los datos del prototipo contra los de dispositivos comerciales mediante el uso de una videocámara y el posterior análisis del video proporciono la ventaja de obtener los datos desde la perspectiva que tendría el usuario final, en lo que se refiere a velocidad de despliegue y exactitud de los datos. La única prueba que no se realizó mediante este procedimiento, es la de temperaturas. Por el hecho de que estas no varían con las mismas velocidades que los otros parámetros, por lo tanto, la toma de lecturas de estas se realizó en diferentes horas del día y sin necesidad de video.

Por otra parte, en este capítulo no mostramos las pruebas de velocidad de pedaleo, dado que es el mismo sistema que el que mide las revoluciones del motor. Así evitamos ser redundantes en el texto.

Para finalizar, debo mencionar que la adquisición y despliegue de todos los datos en la interfaz del usuario funciona mucho mejor que lo que teníamos previsto.

APENDICE

A continuación, mostraremos las líneas de código del cual se compone el firmware de nuestro prototipo de adquisición de datos en el PsoC 4, el código del módulo BLE 4.2 y los bloques que componen el código de la aplicación en App Inventor 2

Código del PsoC 4

```
#include "stdio.h"
#include "project.h"
volatile int16 cuenta1=0,cuenta2=0;//Entradas de los encoder
uint16 i2cBuffer[4]; //Buffer de datos para el protocolo I2C
uint16 Taire, Tmotor , Tdriver, Tbateria, i=0,j=0,lectura,
rpmRueda=0,prom,volts,rpmPedal;//variables globales
char str1[32], str2[32],str3[32],str4[32],str5[5]; // Arreglos de caracteres para
poder imprimir en LCD

 CY_ISR(isrRueda){       //Interrupción para el encoder de la rueda, cada que esta
es activada cuenta1 se incrementa en una unidad, al final se limpia la bandera de
interrupción.
  cuenta1++;
  EncoderRueda_ClearInterrupt();
}

CY_ISR(isrPedal){       //Interrupción para el encoder de los pedales
    cuenta2++;
    EncoderPedal_ClearInterrupt();
}
void DameRPMRueda(void); //Declaración de las funciones usadas en este software
void DameRPMPedal(void);
void DameTaire(void);
void DameTmotor(void);
void DameTdriver(void);
void DameTbateria(void);
void DameAmpers(void);
void DameVolts(void);
void EnviaDatosI2C(void);
void ImprimeLCD(void);

int main(void)
{

  CyGlobalIntEnable; //Habilita interrupciones internas
  ADC_Start();      //Inicializa ADC
  LCD_Start();      //Inicializa LCD
  EZI2C_Start();    //Inicializa I2C
  Clock_Start();    //Inicializa Reloj
 //Declaramos la variable que sera usada como buffer en la comunicación I2C
  EZI2C_EzI2CSetBuffer1(sizeof(i2cBuffer),sizeof(i2cBuffer),(uint8 *)&i2cBuffer);
  PWM_Start();     //Inicializa PWM
    for(;;)
    {
        DameRPMRueda();  //Se ejecutan cada una de las funciones de manera ciclica
y consecutiva
        DameVolts();
        DameAmpers();
        DameTaire();
        DameTmotor();
        DameTdriver();
        DameTbateria();
        EnviaDatosI2C();
        DameRPMPedal();
        ImprimeLCD();
    }
 }
void DameRPMRueda(void){
```

```
        isrRueda_StartEx(isrRueda); //Se Inizializa la interrupción
        CyDelay(125);           // Delay para hacer el conteo de ventanas del
encoder
        isrRueda_Stop();        // Se detiene la interrupción

        if(cuenta1==0){  //Se verifica que cuenta no sea igual a cero, de ser asi
se le asigna el valor de su identificador
            rpmRueda=7;
           }else{
        rpmRueda=(cuenta1*480)/36; // Se convierte la cuenta de pasos del encoder a
RPM
        rpmRueda=(rpmRueda*10)+7; // Se agrega identificador
        cuenta1=0;}
}
void DameRPMPedal(void){ // Esta función es idéntica a DameRPMRueda()

        isrPedal_StartEx(isrPedal);
        CyDelay(100);
        isrPedal_Stop();

        if(cuenta2==0){
            rpmPedal=8;
           }else{
        rpmPedal=(cuenta2*600)/18;
        rpmPedal=(rpmPedal*10)+8;
        cuenta2=0;}
}
void DameTaire(void){  //Función para obtener temperatura del aire, idéntica a para
Tdriver,Tbateria,Tmotor.
        Taire= ADC_ReadResult_mVolts(0);// Se asigna el resultado del ADC en el
canal cero a Taire
        Taire=((Taire*100)+1); // Se asigna digito identificador

}
void DameTdriver(void){
        Tdriver= ADC_ReadResult_mVolts(4);
        Tdriver=((Tdriver*100)+3);

}
void DameTbateria(void){
        Tbateria= ADC_ReadResult_mVolts(5);
        Tbateria=((Tbateria*100)+4);

}
void DameTmotor(void){
        Tmotor= ADC_ReadResult_mVolts(1);
        Tmotor=((Tmotor*100)+2);
}
void DameAmpers(void){ //Función para obtener corriente instantánea promedio.
 for(j=0;j<=9;j++){ //Iniciamos un ciclo de 10 repeticiones
    lectura = lectura + ADC_ReadResult_mVolts(2); //Sumamos las 10 lecturas de
corriente
        CyDelay(3);// retardo para mejorar la respuesta del filtro
}
        prom=lectura/10; //obtenemos el promedio de la lectura
        prom=(prom*10)+6;//asignamos digito identificador
        lectura=0; // igualamos lectura  a cero para volver a usarla
    }
void DameVolts(void){ //Función para obtener voltaje en bateria
```

```
    volts= ADC_ReadResult_mVolts(3);//Asignamos el valor entregado por el ADC en el
canal 3 a Volts
    volts=((volts)*10)+5; //Agregamos digito identificador

}
//Esta función agrega al buffer de I2C cada dato a la vez para poder ser enviado
por BLE, se hace un delay para darle tiempo a las etapas posteriores del sistema y
evitar perder información.
void EnviaDatosI2C(void){
     for(i=0;i<=7;i++){ // Se inicia un ciclo de 8 repeticiones
        if(i==0){
             I2cBuffer[0]=Taire; //Enviá Taire
            CyDelay(10);
            }
        if(i==1){
             I2cBuffer[0]=Tmotor; //Enviá Tmotor
            CyDelay(10);
        }
         if(i==2){
             I2cBuffer[0]=rpmRueda; //Enviá rpmRueda
            CyDelay(100);
        }
         if(i==3){
             I2cBuffer[0]=prom; //Enviá prom (Corriente instantánea promedio)
            CyDelay(100);
        }
          if(i==4){
             I2cBuffer[0]=volts;//Enviá volts
            CyDelay(10);
        }
         if(i==5){
             I2cBuffer[0]=Tbateria; //Enviá Tbateria
            CyDelay(10);
            }
         if(i==6){
             I2cBuffer[0]=Tdriver; //Enviá Tdriver
            CyDelay(10);
            }
         if(i==7){
             I2cBuffer[0]=rpmPedal;// Enviá rpmPedal
            CyDelay(100);
        }
    }

}
//La siguiente función imprime cada dato en la pantalla LCD, dicha función se uso
en la fase de desarrollo anterior al envio y despliegue de datos en el smarthphone,
por lo tanto puede ser suprimida para la versión final y optimizar tiempo y
recursos de ejecución.
void ImprimeLCD(void){
        sprintf(str1,"%d",Tbateria);//Se convierten los datos a cadenas de
caracteres
        sprintf(str2,"%d",Tdriver);
        sprintf(str3,"%i",prom);
        sprintf(str4,"%i",cuenta1);
        sprintf(str5,"%i",volts);


        LCD_Position(0,4); // Se posiciona el cursor y se imprimen datos
```

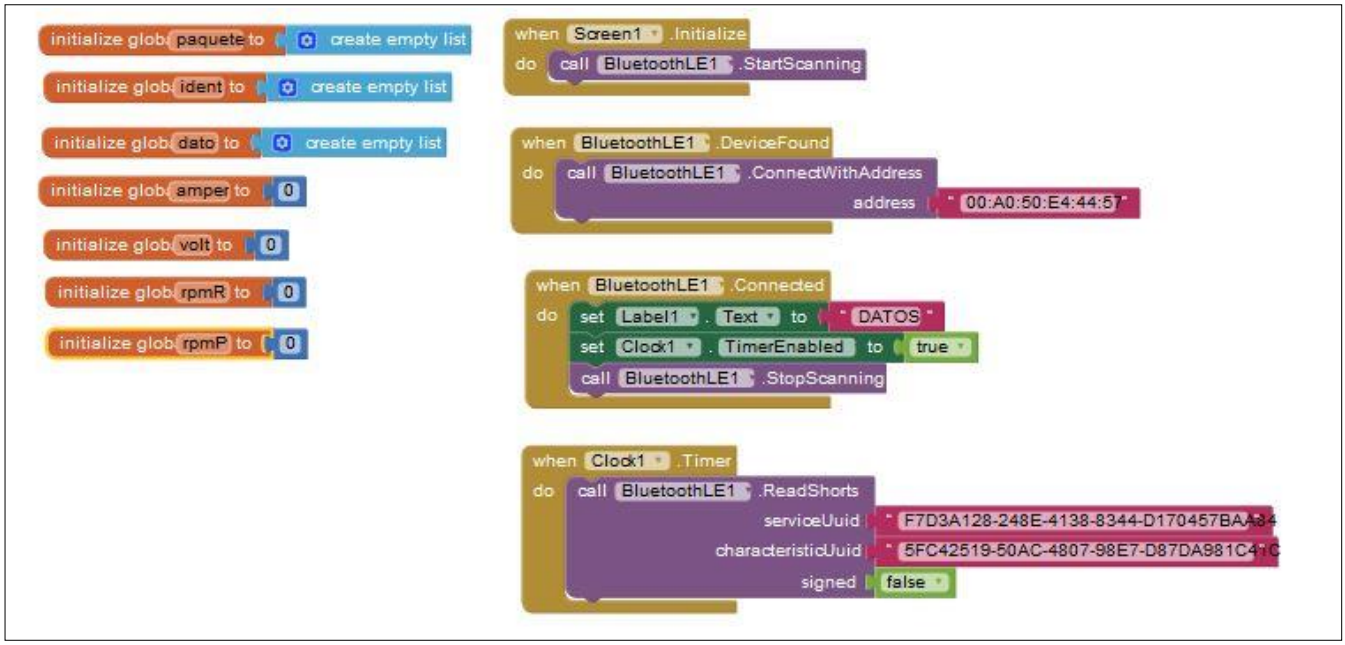
```
            LCD_PrintString(str5);
            LCD_Position(1,0);
            LCD_PrintString(str1);
            LCD_Position(1,4);
            LCD_PutChar(LCD_CUSTOM_0);
            LCD_Position(1,5);
            LCD_PrintString("C");
            LCD_Position(1,8);
            LCD_PrintString(str2);
            LCD_Position(1,12);
            LCD_PutChar(LCD_CUSTOM_0);
            LCD_Position(1,13);
            LCD_PrintString("C");
            LCD_Position(0,11);
            LCD_PrintString(str3);
            LCD_Position(0,15);
            LCD_PrintString("A");
            LCD_Position(0,0);
            LCD_PrintString(str4);
}
```

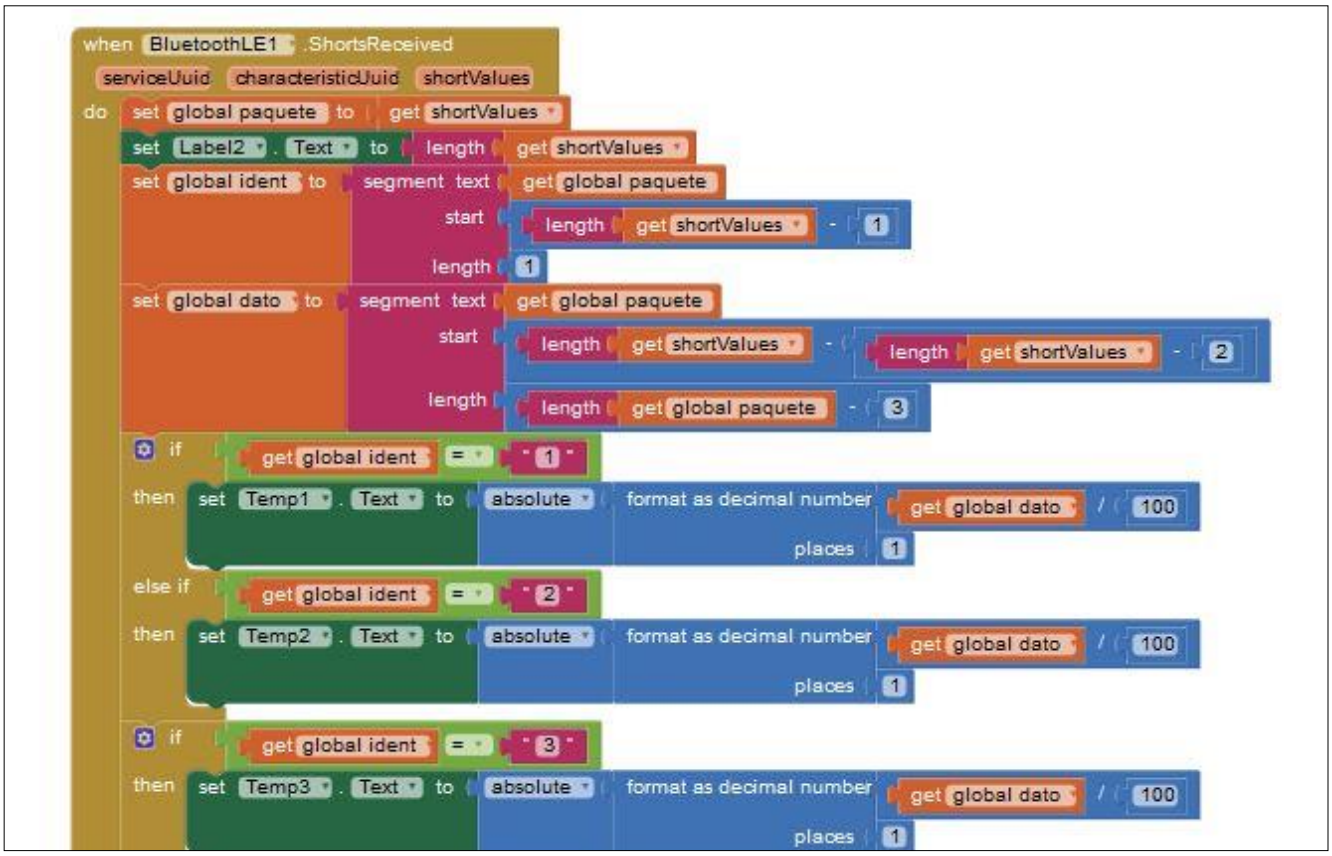
Código del módulo BLE

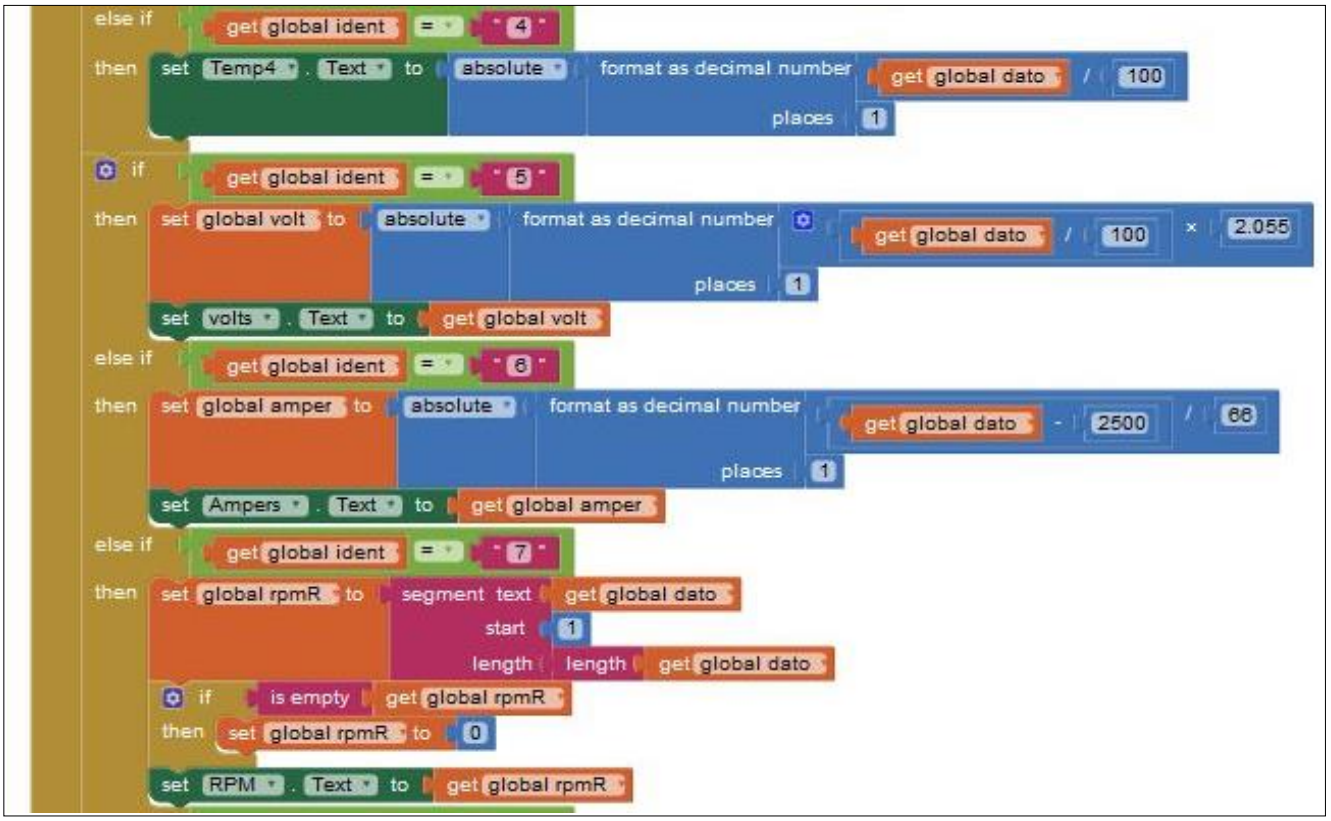
```
#include <project.h>
uint16 i2cBuffer[1];//Declaramos buffer para recibir datos de I2C
uint16 Dato; //Variable que guardara el dato recibido por I2C

void updateDato() //Función para enviar el dato por BLE
{
    //Verificamos estado de conexión.
    if(CyBle_GetState() != CYBLE_STATE_CONNECTED)
        return;
    //Se crea estructura temHandle para el manejo del dato a enviar
      CYBLE_GATTS_HANDLE_VALUE_NTF_T      tempHandle;
    tempHandle.attrHandle = CYBLE_BICIELEC_DATO_CHAR_HANDLE; //Se asigna el nombre
de la caracteristica a usar.
      tempHandle.value.val = (uint8 *)&Dato;// Se asigna dato a enviar.
    tempHandle.value.len = 2; //Numero de bytes del dato a enviar

CyBle_GattsWriteAttributeValue(&tempHandle,0,&cyBle_connHandle,CYBLE_GATT_DB_LOCALL
Y_INITIATED );  //Se envia dato
    CyBle_GattsNotification(cyBle_connHandle,&tempHandle); //Se notifica al cliente
que un dato ha sido enviado
}

//Función para administrar los estados de operación del BLE
void BleCallBack(uint32 event)
{

    switch(event)
    {
        //En el caso de que la pila halla pasado de un estado de Reset a On y el
dispositivo se encuentre desconectado, se activa el modo de listo para conexión  y
se apaga el PWM del LED
        case CYBLE_EVT_STACK_ON:
        case CYBLE_EVT_GAP_DEVICE_DISCONNECTED:
          CyBle_GappStartAdvertisement(CYBLE_ADVERTISING_FAST);
          PWM_Stop();
            break;

       //Cuando se establezca una conexión, se llama a la función updateDato() y se
activa el PWM del LED
        case CYBLE_EVT_GATT_CONNECT_IND:
            updateDato();
            PWM_Start();

            break;
        default:
            break;
    }
}


int main (void)
{

    CyGlobalIntEnable;  //Se habilitan interrupciones globales
    Clock_Start(); //Se inicializa reloj
    I2C_Start(); //Se inicializa  I2C
    CyBle_Start(BleCallBack); //Se inicializa BLE controlado por la función
BleCallBack
    for(;;)
```

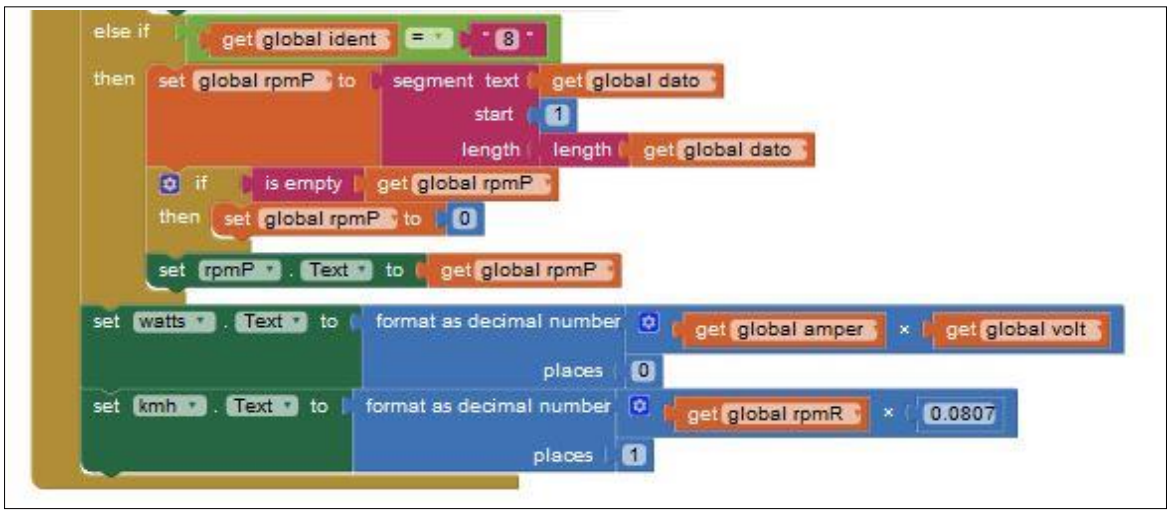
```
    {
        //Leemos el buffer de I2C
      I2C_I2CMasterReadBuf(0x08,(uint8 *)&i2cBuffer,2u,I2C_I2C_MODE_COMPLETE_XFER);
     if(I2C_I2CMasterStatus()== I2C_I2C_MSTAT_RD_CMPLT)//Se verifica que la lectura
del dato se haya completado
    {

        Dato=i2cBuffer[0]; //Asignamos el valor leido del buffer a la variable Dato
        updateDato(); //Lamamos a la función que envià dato por BLE


    }else
    CyBle_ProcessEvents(); //Función nativa de PSoC que controla los procesos del
BLE

    }
}
```

Código de Bloques de App Inventor 2




when BluetoothLE1 .ShortsReceived
serviceUuid characteristicUuid shortValues
do set global paquete to get shortValues
set Label2 . Text to length get shortValues
set global ident to segment text get global paquete
start length get shortValues - 1
length 1
set global dato to segment text get global paquete
start length get shortValues - length get shortValues - 2
length length get global paquete - 3
if get global ident = "1"
then set Temp1 . Text to absolute format as decimal number get global dato / 100
places 1
else if get global ident = "2"
then set Temp2 . Text to absolute format as decimal number get global dato / 100
places 1
if get global ident = "3"
then set Temp3 . Text to absolute format as decimal number get global dato / 100
places 1


else if get global ident = "4"
then set Temp4 . Text to absolute format as decimal number get global dato / 100
places 1
if get global ident = "5"
then set global volt to absolute format as decimal number get global dato / 100 × 2.055
places 1
set volts . Text to get global volt
else if get global ident = "6"
then set global amper to absolute format as decimal number get global dato - 2500 / 66
places 1
set Ampers . Text to get global amper
else if get global ident = "7"
then set global rpmR to segment text get global dato
start 1
length length get global dato
if is empty get global rpmR
then set global rpmR to 0
set RPM . Text to get global rpmR


else if
get global ident
=
8
then
set global rpmP to
segment text
get global dato
start
1
length
length
get global dato
if
is empty
get global rpmP
then
set global rpmP to
0
set rpmP . Text to
get global rpmP
set watts . Text to
format as decimal number
get global amper
×
get global volt
places
0
set kmh . Text to
format as decimal number
get global rpmR
×
0.0807
places
1

CONCLUSIONES

En el aspecto profesional, la realización de esta tesis ha aportado gran valor en mis conocimientos. Antes de este trabajo, mi vocación y gusto se inclinaban por el lado de la electrónica analógica y de potencia. Pero dado los requerimientos del prototipo tuve que desarrollar y mejorar mis aptitudes en cuanto al uso de microcontroladores y escritura de software se refiere. En este último punto es importante mencionar, que las plataformas de desarrollo que usamos, tanto PSoC Creator como App inventor, simplifican de una manera notable el desarrollo de software tanto para el ambiente académico como en el industrial.

Recordando las metas fijadas al inicio del trabajo, el objetivo principal se cumplió completamente. Es decir, se diseñó y construyó la instrumentación de adquisición de datos del vehículo, dichos datos se envían perfectamente de manera inalámbrica y por último el despliegue de estos parámetros mediante el uso de un teléfono inteligente funciona correctamente como lo demuestran las pruebas hechas. En cuanto a los objetivos específicos, hay ciertos puntos que faltaron por concluir. Nos referimos al posprocesamiento de los datos para obtener valores como son: calorías quemadas, registros de distancias recorridas, avisos de temperaturas críticas, etc. esto debido al tiempo que supone desarrollar estas características de la aplicación y la limitación del mismo. Por lo tanto, dejamos como trabajo a futuro el desarrollo de estos puntos, así como también el mejoramiento gráfico de la interfaz. Nos referimos a que esta sea mucho más atractiva visualmente, característica que queda limitada por el bajo potencial de gráficos de la plataforma app inventor. Otro punto importante para el trabajo a futuro es la implementación de una base de datos directamente en la aplicación, capaz de almacenar todos los datos para que el usuario pueda hacer uso de estos en un análisis posterior al uso del vehículo. De esta manera podrá analizar su rendimiento físico y del vehículo después de la rutina y el uso de este.

Dado que la base de todo el sistema, es decir la adquisición y envió de datos está perfectamente acoplados. Solo queda trabajo en las características de posprocesamiento ya mencionadas y muchas otras más que se pueden ir agregando, como lo es una base de datos.

APORTACIONES

Por otra parte, si reparamos un poco en lo que se logró, podemos darnos una idea de los diversos campos donde se puede aplicar, es decir, tenemos un sistema que adquiere datos mediante diferentes sensores, los envía a un dispositivo externo, y este ultimo los despliega en tiempo real. Una de las áreas más evidentes aparte del mundo automotriz de competición, donde tiene un potencial claro, es el biomédico. Si

cambiamos la bicicleta por un paciente, los sensores de rpm, corriente, voltaje, etc., por diferentes sensores médicos en el cuerpo de la persona. Podríamos darle tanto a los familiares como a los médicos encargados, la posibilidad de estar vigilando el estado del paciente en tiempo real desde cualquier lugar que se encuentren, esto pensando en cambiar el modo en que se envían los datos actualmente (BLE, que solo tiene alcance de unos cuantos metros) a Ethernet.

Por último, tenemos que mencionar que logramos implementar el sistema a muy bajo costo. Con un precio final en materiales que no supera los 50 USD tenemos la capacidad de competir en el mercado con un producto que ofrece características no vistas comúnmente en este sector.

Una de las grandes aportaciones dentro de este proyecto es todo el contenido (códigos y diseños) que se desarrolló para el uso de la tecnología PsoC, ya que actualmente es muy limitada y/o escasa en el idioma español.

REFERENCIAS

[i] **Ivan Evtimov, University of Rusia, "A study on electric bicycle energy efficiency",DOI:10.2137/tp-2015-041**

[ii] **https://greentransportation.info/energy-transportation/kwh-evcars-gizmos.html**

[iii] **[L.G. O'Connell, "Electric Vehicles: A clean, energy-efficient urban transportation alternative", EPRI Rep., Vol. 30, no. 5, sep/oct 1995**

[iv] **Yu-xiang Zhao, IEEE Access 10.1109/access.2017.2740419 19 septiembre 2017**

[v] **https://buy.garmin.com/es-MX/MX/p/510943**

[vi] **https://www.emotionbikesmexico.com/collections/all**

[vii] **https://www.prodecotech.com.mx/collections/bicicletas-electricas/products**

[viii] **http://electrobike.com.mx/mx/**

[ix] **http://www.electropedaleo.com.mx/**

[x] **Lista elaborada con datos de las empresas comercializadoras**

[xi] **Philips Semiconductors. Power Semiconductor Applications. Chapter 3: Motor Control, March 2010.**

[xii] **http://www.redwings.cl/la-diferencia-del-motor-central.html**

[xiii] **Robert Cong, Electrical bicycle system, California Polytechnic State University, San Luis Obispo June 2010**

[xiv] **"Charging lithium-ion batteries." Battery University.com. Isidor Buchmann, March 2006.**

[xv] **https://www.mouser.mx/ProductDetail/Texas-Instruments/MSP-EXP430G2?qs=sGAEpiMZZMv1ORdfpzTN%252bMwZ3%252b5KGk2B**

[xvi] **https://www.mouser.mx/ProductDetail/Texas-Instruments/MSP-EXP430G2?qs=sGAEpiMZZMv1ORdfpzTN%252bMwZ3%252b5KGk2B**

[xvii] **http://www.cypress.com/products/psoc-creator-integrated-design-environment-ide**

[xviii] **https://www.sparkfun.com/datasheets/BreakoutBoards/0712.pdf**