Trabajo Fin de Grado

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Sistema de telemetría y representación de datos para un monoplaza de competición.

Autor: José Luis Silva Valero

Tutor: Alfredo Pérez Vega-Leal

Dep. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Sistema de telemetría y representación de datos para un monoplaza de competición.

Autor:
José Luis Silva Valero

Tutor:
Alfredo Pérez Vega-Leal
Profesor Contratado Doctor

Dep. de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo Fin de Grado: Sistema de telemetría y representación de datos para un monoplaza de competición.

Autor: José Luis Silva Valero

Tutor: Alfredo Pérez Vega-Leal

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Sevilla, 2017

A mi familia

A mis maestros

Agradecimientos

Después de un intenso periodo de estudios, escribo esta nota para agradecer a mis padres haberlo hecho posible. Ha sido un largo camino de aprendizaje, pero aquí estoy, dando las últimas pinceladas a lo que será el trabajo que finalizará el grado en la Universidad. Por apoyarme enormemente en esta fase, reconozco a mi familia en esta labor y a todos mis compañeros, que han estado conmigo a lo largo de todo el camino. Por supuesto, no olvidar a mis más fieles tutores y su gran forma de impartir conocimiento.

También dar mi más sincero agradecimiento al equipo ARUS Andalucía Racing, por haberme dado la gran oportunidad de participar con ellos en unas competiciones con un monoplaza diseñado y construido con nuestras propias manos que, sin ninguna duda, dejará una huella muy importante en mi vida. La convivencia, la experiencia, los conocimientos adquiridos y una larga lista de aportaciones que no serían posible de otra forma.

Muchas gracias a todos una vez más, es todo un honor recibir la confianza necesaria, no os defraudaré.

José Luis Silva Valero

Diseñador electrónico en ARUS Andalucía Racing Team.

Sevilla, 2017

Resumen

En el presente documento se detallará la realización de un proyecto en el que se pretende dotar al coche de combustión del equipo ARUS Andalucía Racing Team de un sistema capaz de enviar los datos más relevantes de forma remota para tener control sobre la pista. Dicho vehículo participará en la competición fórmula SAE (Society of Automotive Engineers), donde cada equipo formado por estudiantes de universidades de todo el mundo debe diseñar, fabricar y validar un prototipo de vehículo capaz de cumplir las numerosas normas de seguridad y restricciones de la competición, todas expuestas en la normativa. Con la participación en estas pruebas se pretende que los estudiantes tengan una oportunidad para adquirir conocimientos y prácticas más allá de los enseñados en la universidad.

El sistema a implementar tiene la función principal de comunicarse con los diferentes subsistemas del vehículo y tras interpretar los datos, será capaz de enviar las mediciones de los distintos sensores por radiofrecuencia. El receptor, con ayuda de un programa en LabVIEW, será capaz de visualizarlo o incluso grabarlo para su posterior análisis.

La implementación está integrada en la placa de circuito impreso que forma el salpicadero del vehículo. Aquí se encuentra un microcontrolador de la familia MSP430 de Texas Instruments que, con ayuda de un controlador de CAN bus (MCP2515), estará comunicado con todas las placas que componen la electrónica. Los datos se obtienen tanto de la ECU (Engine Control Unit) como de la tarjeta de adquisición de datos diseñada por un miembro del equipo ARUS.

Una vez interpretados los datos de interés, se enviarán a través de un módulo de radiofrecuencia a 433MHz muy usado en aeromodelismo. En el receptor se encuentra otro módulo habilitado para conectar directamente a un ordenador por USB, de forma que se trabajará como un puerto serie (COM).

En el apartado software se ha utilizado el compilador Code Composer Studio disponible en la página web de Texas Instruments. Con él se ha programado en lenguaje C el código necesario para que el microcontrolador pueda interpretar y enviar los datos. En la parte receptora del sistema, se ha programado un código en LabVIEW que incluye una interfaz visual donde se podrá interaccionar con los resultados obtenidos. Como anexo, se ha realizado un pequeño script en Matlab que permite representar gráficamente los datos obtenidos.

Abstract

This document details the realization of a project which is intended to equip the combustion car of the ARUS Andalucía Racing Team with a system capable of sending the most relevant data remotely to have control over the race. This vehicle will participate in the Formula SAE (Society of Automotive Engineers) competition, where each team made up of students from universities all over the world must design, build, and validate a vehicle prototype able to pass safety regulations and restrictions of the competition. Students who participate in these tests have an opportunity to acquire knowledge and practices beyond those taught in the University.

The system to be implemented has the main function of communicating with other subsystems of the vehicle and after interpreting the data, it will be able to send the measurements of sensors by radiofrequency. The receiver will be able to visualize this information or even record it for later analysis with the help of a LabVIEW program.

The implementation is integrated in a printed circuit board that forms the dashboard of the vehicle. Here there is a microcontroller from the MSP430 family of Texas Instruments that works with a CAN bus controller (MCP2515) to connect dashboard to all PCB which make up the whole electronics. Data is obtained from both the ECU (Engine Control Unit) and the data acquisition card designed by a member of the ARUS team.

Once data of interest is interpreted properly, it will be sent through a 433MHz radiofrequency module, commonly used in aeromodelling. The receiver has another module able to connect directly to a computer via USB, so that it will work as a serial port (COM).

For software, it is used the Code Composer Studio compiler, that it is available from Texas Instrument website. Necessary code for microcontroller to read and send data is programmed in C language with this tool. In the receiving part of the system, a code has been programmed in LabVIEW that includes a visual interface where it can interact with obtained results. As an attachment, it was made a small script in Matlab that allows graphing the data.

Índice

Agradecimientos viii

Resumen x

Abstract xii

Índice xiii

Índice de Tablas xv

Índice de Figuras xvii

Notación xx

1 La competición 1
1.1. Sobre Formula Student 1
1.2. Evolución del ART-17 2
1.3. Importancia de un sistema de telemetría 2

2 Requisitos de diseño 3
2.1. Cumplimiento de la normativa 3
2.2. Necesidades del equipo 4
2.2.1. Ahorro de costes y espacio 4
2.2.2. Representación de los datos a tiempo real 5
2.2.3. Captación de resultados 5
2.2.4. Alcance efectivo 5
2.3. Estudio previo 5
2.3.1. Alcance necesario 5
2.3.2. Estimación de la tasa de datos 6
2.4. Ensayos y validaciones 10
2.4.1. Módulos de telemetría 10
2.4.2. Elección de la antena para el radioenlace 12
2.4.3. Microcontrolador 14
2.4.4. Tasa de datos y errores 16

3 Desarrollo del hardware 11
3.1. Esquemático de la electrónica 11
3.1.1. Microcontrolador MSP430G2553 12
3.1.2. Controlador MCP2515 y transceptor CAN 14
3.1.3. Shift Register y display de 16 segmentos. 16
3.1.4. Módulos de telemetría 17
3.2. Diseño de la PCB 18

4 Software y programas **22**
4.1. Code Composer Studio 6.2.0 *22*
4.1.1. Librería para el controlador del bus CAN (MCP2515) 22
4.1.2. Librería para la utilización de la UART 25
4.1.3. Librería para la utilización del Shift Register (74HC595) 27
4.1.4. Explicación del Código principal 28
4.2. LabVIEW 2014 *36*
4.2.1. Lectura de datos de entrada 36
4.2.2. Escritura de datos en fichero 37
4.2.3. Interpretación de los datos 38
4.2.4. Otras estrategias 39
4.3. 3DR Radio Config 1.3.2 *42*

5 Problemas y soluciones **45**
5.1. Organización de las tramas *45*
5.2. Compatibilidad del bus CAN *45*
5.3. Alimentación de las PCB *46*

6 Resultados y conclusiones **48**
6.1. Funcionamiento general *48*
6.2. Grabación de datos *50*

7 Presupuesto **52**

Referencias **54**

Bibliografía **55**

ÍNDICE DE TABLAS

Tabla 1. Cálculos de la ocupación del bus CAN. 8

Tabla 2. Comparación de módulos para telemetría. 10

Tabla 3. Comparación entre distintos tipos de antenas. 14

Tabla 4. Datos y errores recibidos según la distancia. 20

Tabla 5. Funciones disponibles en la librería del MCP2515. 23

Tabla 6. Valores de temporización más comunes para el bus CAN. 25

Tabla 7. Funciones disponibles en la librería de la UART. 26

Tabla 8. Funciones disponibles en la librería para el registro de desplazamientos. 27

Tabla 9. Variables pantalla general. 51

Tabla 10. Variables pantalla dinámica. 51

Tabla 11. Variables pantalla suspensión. 51

ÍNDICE DE FIGURAS

Ilustración 1. Disposición del salpicadero en CATIA. 4
Ilustración 2. Distancia máxima entre antenas para el circuito de Austria (izquierda) y Alemania (derecha). 6
Ilustración 3. Esquema general del flujo de datos. 6
Ilustración 4. Tramas enviadas por cada sistema mediante CAN bus. 7
Ilustración 5. Cobertura en mapa real de RedBull Ring en Spielberg, Austria. 9
Ilustración 6. Cobertura en mapa real de Hockenheimring en Hockenheim, Alemania. 9
Ilustración 7. Módulos de telemetría. 10
Ilustración 8. Funcionamiento de la modulación FSK 11
Ilustración 9. Suavizado mediante filtro gaussiano. 11
Ilustración 10. Comprobación del alcance utilizando antenas Wi-Fi de reducido tamaño. 13
Ilustración 11. Comprobación del alcance con antenas superiores. 13
Ilustración 12. LaunchPad de TI con microcontrolador MSP430. 15
Ilustración 13. Esquema general de la conexión de los sistemas mediante el bus CAN. 16
Ilustración 14. Distancia máxima probada con las nuevas antenas. 17
Ilustración 15. Interfaz gráfica de la prueba con LabVIEW. 18
Ilustración 16. Diagrama de flujo de la prueba con LabVIEW. 18
Ilustración 17. Gráfica de datos recibidos frente a la distancia entre antenas. 20
Ilustración 18. Esquema general del hardware para el sistema de telemetría. 12
Ilustración 19. Conexión del microcontrolador MSP430G2553. 13
Ilustración 20. Pinout del microcontrolador en la placa de prueba. 14
Ilustración 21. Conexión general de los integrados necesarios para controlar el bus CAN. 14
Ilustración 22. Conexión del integrado controlador del bus CAN y transceptor. 15
Ilustración 23. Bus CAN de alta velocidad y de baja velocidad tolerante a fallos. 16
Ilustración 24. Conexión del display de 16 segmentos con registros de desplazamiento. 16
Ilustración 25. Conexiones para módulos de telemetría y programación. 17
Ilustración 26. Esquema genérico de programación on-board. 18
Ilustración 27. Explosionado de los elementos del salpicadero. 18
Ilustración 28. Diseño de la PCB multicapa. 19
Ilustración 29. Diseño de la PCB con plantilla. 19

Ilustración 30. Vista frontal del diseño 3D de la PCB. 20
Ilustración 31. Vista trasera del diseño 3D de la PCB. 20
Ilustración 32. Diseño final de PCB a dos capas. 21
Ilustración 33. Fotografía de la PCB con componentes soldados. 21
Ilustración 34. Comprobación de la librería en MSP430 con módulo de bus CAN. 24
Ilustración 35. Comprobación de pulsos generados con la librería de datos y reloj. 27
Ilustración 36. Relación entre segmentos y pines del registro. 28
Ilustración 37. Diagrama de flujo del código principal. 29
Ilustración 38. Pestaña general de la telemetría. 40
Ilustración 39. Interfaz gráfica de telemetría para suspensión. 41
Ilustración 40. Interfaz gráfica de telemetría para dinámica. 41
Ilustración 41. Interfaz del programa utilizado para configurar los módulos. 42
Ilustración 42. Configuración utilizada en los módulos de telemetría. 43
Ilustración 43. Trama estándar de bus CAN vs. trama extendida. 46
Ilustración 44. Convertidor DC/DC utilizado. 46
Ilustración 45. Comprobación de la tensión de alimentación con resistencia de carga. 47
Ilustración 46. Funcionamiento de los datos generales de la telemetría. 49
Ilustración 47. Funcionamiento de datos para suspensión. 50

Notación

FSAE	Formula Student (Society of Automotive Engineers, SAE)
kBAUD	Kilobaudios
kbps	Kilobyte por segundo
VCC	Tensión de alimentación
GND	Conexión a referencia
GPIO	General Purpose Input/Output
UART	Universal Asynchronous Receiver-Transmitter
SPI	Serial Peripheral Interface
MISO	Master Input Slave Output
MOSI	Master Output Slave Input
SCLK	System Clock
µC	Microcontrolador
LED	Light-Emiting Diode
CCS	Code Composer Studio
TI	Texas Instruments
IDE	Integrated Development Enviroment
PCB	Printed Circuit Board
RSSI	Received Signal Strength Indication
dBm	Decibelio-milivatio
ECC	Error-Correcting Code
CRC	Código de Redundancia Cíclica
ID	Número de identificación
LBT	Listen Before Talk
AFA	Adaptive Frequency Agility

1 LA COMPETICIÓN

Llegar juntos es el principio. Mantenerse juntos, es el progreso. Trabajar juntos es el éxito.

- Henry Ford -

1.1. Sobre Formula Student

Formula Student es el nombre con el que se denomina a la competición de motorsport más conocida de Europa. También conocida como Formula SAE, tiene como objetivo desarrollar jóvenes ingenieros innovadores y emprendedores por todo el mundo. El formato del evento proporciona una oportunidad única para los estudiantes de poder demostrar y comprobar las capacidades necesarias para desarrollar un producto tan complejo como un bólido de carreras, en un ambiente de una competición de motorsport. El feedback proporcionado en este tipo de eventos es de lo más importante ya que permite poder seguir mejorando año tras año, como si de una empresa de competición se tratara.

Los inicios de la competición se remontan a 1979, pero no es hasta 1981 cuando en la Universidad de Texas se celebra la primera edición de la Formula SAE, con 6 equipos participantes y 40 alumnos.

Actualmente se celebran competiciones en numerosos países de todo el mundo, llegando a albergar a 120 equipos y más de 2000 estudiantes. Todas siguen una normativa base original de la Formula SAE, permitiendo así a un equipo adaptarse fácilmente a las competiciones en el caso de participar en varias durante una misma temporada, como es el caso de ARUS Andalucía Racing Team.

1.2. Evolución del ART-17

El equipo ARUS añade con esta temporada 2017 su cuarto monoplaza de combustión fabricado hasta la fecha. Han sido cuatro años de evolución y corrección de errores para progresar en el equilibrio que debe existir entre fiabilidad y rendimiento, cuyo objetivo principal será el de mejorar los resultados en la competición.

El primer diseño que fue llevado a la realidad es el ART-14, donde se priorizó la fiabilidad y estabilidad frente a los buenos resultados. El vehículo lucía sólido y consistente, preparado para afrontar por primera vez las duras pruebas. Para conseguirlo, se sobredimensionaron numerosas partes debido a la inexperiencia del equipo en estos aspectos. En el ámbito de la electrónica tenía muchas carencias, y surgía la necesidad de implementar un sistema capaz de monitorizar el estado del monoplaza en carrera.

Por ello, en cada nuevo diseño se ha ido evolucionando en la electrónica de forma conjunta al resto de departamentos. Adquiriendo así una flexibilidad que permitirá adaptarse a cualquier necesidad que surja en otros aspectos. Gracias a este trabajo se conseguía tener un fiel pilar en el que apoyarse al validar los modelos obtenidos, debido al seguimiento que se puede hacer de los valores de los sensores necesarios para dicho propósito.

A continuación, se expondrán algunos de los problemas principales que más preocupaban en la electrónica tras analizar los resultados obtenidos en el ART-16 (monoplaza anterior a la temporada actual). Uno de los más conflictivos fue el acoplamiento de ruido entre la parte de señal y la de potencia en las placas e incluso en el cableado. Otro problema encontrado que causaba mal funcionamiento en el display del volante fue la mala fiabilidad del conector utilizado, que no proporcionaba un buen contacto entre dispositivos. También se encontraron limitaciones al utilizar un sistema electrónico embebido con Arduino como microcontrolador debido a su forma de trabajar con el bootloader característico de la marca.

Todos estos problemas fueron estudiados y resueltos en el nuevo diseño del ART-17, pero aún se tenía la necesidad de anticiparse a los errores para solventarlos con la mayor brevedad posible. Así surgió la idea de retomar el concepto de un sistema de telemetría, implementado por primera vez en la temporada 2015, cuyos resultados no convencieron. Con una metodología completamente diferente y buses de comunicación más avanzados, se implementará un nuevo sistema en el monoplaza de la temporada 2017.

1.3. Importancia de un sistema de telemetría

Para evitar reproducir los problemas de años anteriores, se tendrán en cuenta las correcciones oportunas de los fallos actuales. Como novedad con respecto al ART-16, el nuevo monoplaza implementará un sistema electrónico capaz de predecir roturas o daños notificando en tiempo real el estado de los numerosos sensores disponibles en el vehículo.

Por esa razón, el bueno diseño de un sistema de telemetría puede llegar a ser muy importante. La capacidad de monitorizar una gran cantidad de datos sobre el funcionamiento del vehículo, acompañada de un procesamiento adecuado, podrá conseguir una evolución significativa en el desarrollo del mismo. Se podrá utilizar como apoyo para adaptar el set-up del monoplaza a las características únicas de cada prueba, o incluso analizar le estilo de conducción de cada piloto para ajustar los parámetros oportunos.

Además, al implementar un sistema completo de telemetría y tratamiento de datos en un vehículo de Formula Student, se adquiere gran importancia a la hora de presentar los diseños en las distintas pruebas estáticas de la competición. Encontrando la manera adecuada de plasmar el diseño único de la electrónica ante los jueces, se podrá conseguir unos puntos muy significativos, ya que contarán de la misma forma que la más dura de las pruebas dinámicas.

2 REQUISITOS DE DISEÑO

La formulación de un problema, es más importante que su solución.

- Albert Einstein -

Para todo buen diseño se debe hacer una fase previa en la que se estudien los requisitos necesarios y las limitaciones existentes por la normativa, para cerciorarse de estar siempre cumpliendo dichas reglas. Al tratarse de un sistema por radiofrecuencia, podrá alterar comportamientos de otros dispositivos cercanos, por lo que también existirán unas normas externas a la competición sobre el uso de las bandas de frecuencia. Todo esto se explicará a continuación con detalle.

2.1. Cumplimiento de la normativa

Es requisito indispensable tener en cuenta la extensa normativa [1] que pone a disposición la competición para poder participar en las pruebas. El incumplimiento de alguna de ellas puede dejar al monoplaza fuera de la clasificación. Las normativas de las diversas competiciones deben ser parecidas, ya que está permitido que un equipo participe en pruebas de distintos países en una misma temporada. Los leves cambios que difieren unas de otras hacen que la competición de Alemania tenga una normativa algo más estricta, por lo que para el diseño se tomará en cuenta la de FS Germany.

Para verificar el cumplimiento de la misma, el personal encargado de la organización de evento realiza unas pruebas de validación del monoplaza previas a los eventos dinámicos. En este examen se revisan y miden de forma exhaustiva numerosos aspectos de aerodinámica, chasis, suspensión y seguridad en general. Todo el proceso de fabricación del vehículo debe quedar perfectamente recogido en un documento que se deberá entregar a los jueces.

La normativa centrada en la parte electrónica no es muy extensa, y principalmente va destinada a la correcta elección de una batería y a los elementos de seguridad del vehículo, como serán la seta de emergencia o los contactos.

Concretamente, para los sistemas de telemetría no hay ninguna restricción, por lo que se diseñará libremente siempre atendiendo a las bandas de frecuencias del espectro radioeléctrico que estén permitidas para su libre uso en territorio europeo. El diseño de la placa electrónica se adaptará a la ergonomía del salpicadero, y se procurará utilizar una antena cuyo diseño no perjudique al flujo de aire en la aerodinámica delantera del vehículo.

Bandas libres de frecuencia

Las bandas de frecuencia son intervalos del espectro electromagnético divididos según el uso que tengan en las radiocomunicaciones. Como ejemplo, radiodifusión, telefonía móvil o radionavegación se colocan en rangos de frecuencia no solapados. Su uso está regulado por la Unión Internacional de Telecomunicaciones (UIT). Las bandas UIT de radio se establecieron en las Regulaciones de Radio [2] en el Artículo 2, provisión No. 2.1.

Se tienen dos posibilidades bien diferenciadas para utilizar radiofrecuencia en la telemetría bajo las bandas de frecuencias que no requieren licencia (license-free). La primera sería optar por bandas en 2.4GHz como frecuencia de libre utilización. Las normas que deben cumplir en Europa tienen que ver principalmente con la potencia máxima y con las interferencias. Para aplicaciones de interior se limita la potencia a 200mW, mientras que para exteriores se amplía de 1W a 4W la potencia permitida. Para evitar las interferencias, el sistema debe tener un mecanismo de salto automático de canal cuando se detecte que está saturado o pueda ser perjudicial para otras aplicaciones.

Como segunda opción se puede optar por una zona de frecuencias menores. En la región europea se tiene el intervalo de 433.05 – 434.79 MHz, la conocida banda LDP433. Este intervalo está reservado internacionalmente para aplicaciones de uso no comercial, y la máxima potencia radiada aparente permitida es de 100mW. Esta banda ISM (Industrial, Scientific and Medical) formará parte de la banda para radioaficionados UHF de 70cm, que concretamente en España está comprendida entre los 430 MHz y los 440 MHz.

2.2. Necesidades del equipo

A continuación, se analizarán los diversos aspectos que se tendrán en cuenta para diseñar y elegir los dispositivos necesarios para el sistema. Estas características basadas en las necesidades del monoplaza anterior definirán las nuevas funcionalidades.

2.2.1. Ahorro de costes y espacio

El sistema estará integrado en la placa del salpicadero, que deberá tener un tamaño suficiente para albergar todos los componentes. El diseño de esta placa debe ser preciso, ya que la posición de todos los indicadores y actuadores irá en conjunto con el Dpto. de Interior. Se destinará un espacio para el salpicadero de no más de 35cm de ancho, en la parte superior del volante, donde se deberá albergar la electrónica necesaria y los botones de emergencia.

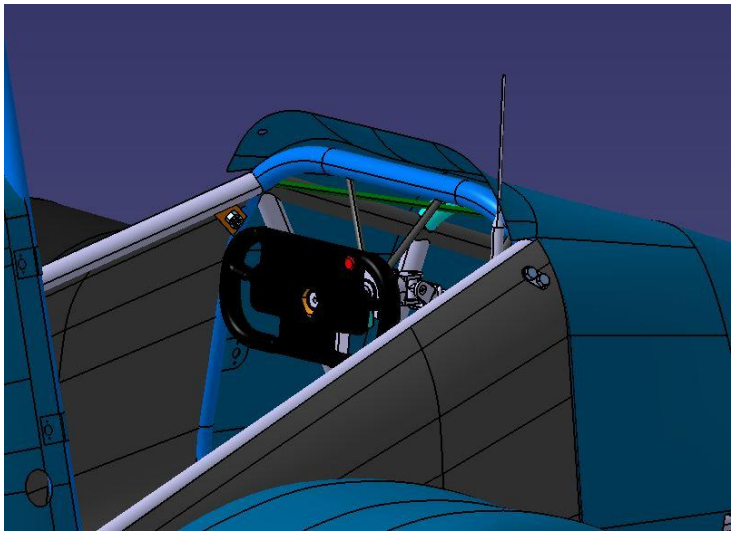


Ilustración 1. Disposición del salpicadero en CATIA.

El diseño de la aerodinámica cambiará, incluyendo una visera para acotar el salpicadero. Esto restringe a un tamaño de la placa electrónica de 30x10cm. La disposición de la antena vendrá dada por los resultados de las simulaciones realizadas por el Dpto. de Aerodinámica. Los lugares más apropiados serán en los laterales, de forma vertical, y lo más cercano al final del morro del vehículo.

2.2.2. Representación de los datos a tiempo real

El vehículo dispone de una gran cantidad de sensores que ayudan a verificar los modelos analizados en la fase de diseño. La telemetría deberá trasmitir valores de 22 sensores localizados por todo el chasis, con el fin de comprobar el correcto funcionamiento del coche en las pruebas realizadas antes de la competición. No se pretende ser una segunda tarjeta de adquisición de datos, por lo que los registros no deberán tener una alta tasa de muestras por segundo ni se tendrán que sacar conclusiones precisas de los resultados. Gracias al uso del bus CAN, todos los sistemas estarán comunicados entre sí, facilitando mucho el intercambio de información. Por ello, será la ADQ la que envíe los datos de sensores, mientras seguirá muestreando y grabando la información a alta velocidad para un análisis más detallado.

Habrá que clasificar los datos según la tasa de actualización deseada para adecuar el volumen de datos de la comunicación. Se establecerán tres velocidades distintas: la más rápida incluirá datos como las revoluciones que necesitan una fluidez adecuada, la velocidad media comprenderá sensores como la presión de frenada y, por último, los sensores de temperatura serán los valores más lentos en ser actualizados.

2.2.3. Captación de resultados

Debe tener la posibilidad de grabar en un fichero de texto todos los datos obtenidos a modo de historial, para poder procesar e interpretar los resultados de forma más detenida. Esto deberá ser posible para todas las pruebas realizadas, sin ningún límite de tiempo. El resultado será un archivo de texto plano cuyos valores se puedan analizar con un programa de visualización de gráficas. Los valores deberán tener una resolución apropiada, similar a la leída por la tarjeta de adquisición de datos. Si fuera necesario, se utilizarán dos bytes de la trama en lugar de uno al enviar un dato para aumentar el rango disponible.

2.2.4. Alcance efectivo

El sistema tendrá que ser capaz de enviar datos en todo momento. Para ello se ha tomado la prueba de resistencia (la más larga) como referencia y se han estudiado las distancias máximas del circuito donde se disputa la prueba. Se establecerá un alcance objetivo de 500 metros para la fase de diseño, acorde con las distancias máximas visibles en la Ilustración 2.

2.3. Estudio previo

En este apartado se hablará sobre la estimación de la calidad de señal, el alcance máximo que debe tener el radioenlace, y se evaluará una primera versión del protocolo que seguirá la electrónica del monoplaza para tener una idea de la tasa de datos necesaria.

2.3.1. Alcance necesario

Se utilizará la herramienta de Google Maps para realizar un primer vistazo rápido a la zona de pruebas de ambos circuitos, donde se podrán observar los puntos más lejanos con respecto a la zona habilitada para la antena receptora. La distancia más crítica se cumple en el circuito de Red Bull Ring en Spielberg, Austria. Dicha distancia no es superior a 300 metros, por lo que, para garantizar la buena calidad de la señal, se diseñará y probará un sistema que logre alcanzar los 500 metros efectivos.

Ilustración 2. Distancia máxima entre antenas para el circuito de Austria (izquierda) y Alemania (derecha).

2.3.2. Estimación de la tasa de datos

Existen dos conexiones cuya tasa de datos tendrá que ser configurada. La primera corresponde a la velocidad en baudios por segundo que tendrá la comunicación serie entre el microcontrolador y el módulo de telemetría, y será necesario conocer primeramente una aproximación de la cantidad de tramas que se analizarán. Mientras que la segunda corresponde a la velocidad en kilobits por segundo entre los propios módulos.

La idea principal se muestra en el diagrama a continuación, donde el microcontrolador se encargará de procesar las tramas correspondientes en cada caso y enviará solo los datos considerados necesarios en cada momento.

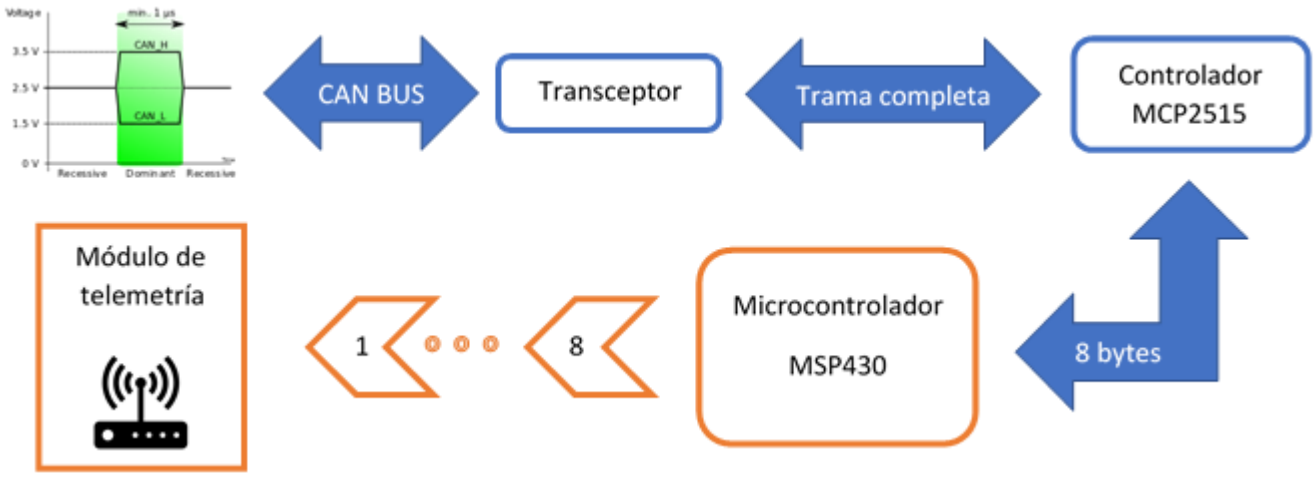


Ilustración 3. Esquema general del flujo de datos.

El bus dispone de dos cables, cuyos datos se envían con tensiones diferenciales. Para transmitir los valores lógicos existen dos estados; dominante, cuando existe una diferencia de tensión entre ambos cables, y recesivo, cuando la tensión es la misma. El transceptor se encargará de manejar estos niveles y los convertirá a niveles TTL, valores digitales de 0 a 3.3V. Posteriormente el controlador interpretará los datos y los almacenará en un buffer, analizando la ID de cada mensaje y los valores. El microcontrolador se comunicará por SPI con el controlador y solicitará la ID y los 8 bytes de información. Siguiendo la operación inversa, se podrá enviar una trama a través del bus. Finalmente, el microcontrolador interpretará los bytes necesarios según el identificador y los pasará uno a uno al módulo de telemetría, que modulará la señal para enviar por radiofrecuencia.

Flujo de datos en el bus CAN

Haciendo un pequeño repaso a las tramas que puede enviar el protocolo de CAN bus, se puede observar como el máximo está limitado a 8 bytes de datos. Se estimó el volumen de datos necesario para enviar por telemetría y otros datos relevantes para mostrar al piloto mediante la pantalla del salpicadero. Para ello, se establecieron una serie de tramas asignado a cada sistema, para que el microcontrolador fuera capaz de interpretar los datos en las posiciones correctas. Cada una debe tener indicada una prioridad para el correcto funcionamiento del protocolo, siendo la de menor frecuencia la más prioritaria, para evitar así verse solapada por tramas mucho más rápidas.

De esta forma, se consiguió compactar los datos en 6 tramas, eligiendo la ID apropiada para cada una según la prioridad deseada. Cabe destacar que, como limitación de programación en otros sistemas, se tuvieron que repetir algunos datos en tramas más lentas.

También se pensó en reservar ciertas ID (las más prioritarias) para unas tramas de errores que se pudieran enviar y recibir desde todos los sistemas conectados, mostrando el sistema que envía el error y un código de identificación del mal funcionamiento. El protocolo estimado se muestra a continuación.

Sistema	ID	1º Byte	2º Byte	3º Byte	4º Byte	5º Byte	6º Byte	7º Byte	8º Byte	Freq. Envio
Ecu	70	Revoluciones		ECT	Lambda		TP	Presión de aceite		50Hz
	40	IAT	MAP	ECT	Lambda		Marcha			10Hz
Potencia	50	Int 1	Int 2	Int 3	Carga	Voltaje				50Hz
ADQ	90	WS FR		WS FL		WS RR		WS RL		50 Hz
	80	Giro Volante	Acele. X	Acele. Y	Acele. Z	Ext. FR	Ext. FL	Ext. RR	Ext. RL	50 Hz
	20	RECORD	Temp. Aceite	Temp. Cockpit	Temp. Rad.	Presión Frenada_R	Presión Frenada_F	Carga		1Hz
Errores	10	Sistema	Mensaje							1Hz

Ilustración 4. Tramas enviadas por cada sistema mediante CAN bus.

Como se puede observar en la imagen anterior, las revoluciones se envían con una tasa de 50 muestras por segundo, garantizando así la fluidez necesaria para la correcta visualización. También existen tramas más lentas donde se enviarán datos como la temperatura, que no necesita ser actualizada tan rápido. Para calcular la carga estimada del bus, se utilizó una aplicación en hoja de Excel proporcionada por la empresa OptimumG, donde introduciendo las características del bus y la longitud de las tramas a enviar, devuelve el valor de carga de dicho bus.

Como la organización de las tramas estaba siempre en constante cambio, se tomaron en cuenta las siguientes hipótesis:

- Las tramas tendrán siempre una longitud de 8 bytes, puesto que será el peor de los casos.
- Se utilizará la versión de protocolo que incluye la ID extendida por compatibilidad con el microcontrolador de la serie Tiva (con protocolo CAN ya incorporado).
- El volumen de los mensajes será de 100 tramas por segundo.

Con todo esto, y una velocidad supuesta de 250 kBAUD para el bus, la aplicación define una carga estimada de la línea del 42.3%. Esta estimación se puede considerar válida, ya que este dato debe mantenerse en valores inferiores al 60-70% para garantizar el correcto funcionamiento del protocolo.

Inputs		
Data rate	250	kBAUD (1 kBAUD = 1 kbit/s)
Protocol	CAN 2.0B	2.0A = 11 bit ID, 2.0B = 29 bit ID
Frequency	100	Hz

Description	Data Length	Total Message Length
Message 1	64	151
Message 2	64	151
Message 3	64	151
Message 4	64	151
Message 5	64	151
Message 6	64	151
Message 7	64	151
Message 8	0	0
Message 9	0	0
Message 10	0	0
Message 11	0	0
Message 12	0	0
Total data/cycle		1057

Outputs		
Baud	250	kBAUD
Address Bits	32	12 or 32
Start/Stop/Etc Bits	32	
Frequency	100	Hz
Other Message Bits		
Start of Frame		1
Arbitration Field		32
Control Field		6
CRC Field		16
Acknowledge Field		2
End of Frame		7
Bits before stuff bit		5
%Max BUS load		42,3%

Tabla 1. Cálculos de la ocupación del bus CAN.

Velocidad de transmisión aérea

Una vez fijada la velocidad de transmisión de datos, se podrá estimar una velocidad apropiada en los módulos de telemetría, que se comunicarán a través de la UART del microcontrolador. Habrá que establecer una primera velocidad en baudios acorde con los datos a enviar, y una segunda velocidad en kbps para la transmisión aérea entre los módulos. La forma en la que se enviarán los datos desde el microcontrolador se explicará detalladamente más adelante, simplemente comentar que el valor será enviado junto con un caracter identificativo al principio. Un ejemplo de trama podría ser W601 para indicar que la temperatura del agua de la refrigeración (a la que se le asigna W), está a 60.1 °C.

El número de datos que viajará por el bus CAN e irá destinado al apartado de telemetría se estima en 500 por segundo, puesto que son 22 sensores, pero solo algunos tendrán una alta tasa de muestreo. Según el protocolo con el que se enviará, cada dato se compondrá de 5 caracteres (una letra y cuatro números).

$$500\ \frac{datos}{s} * 5\ \frac{símbolos}{dato} * 8\ \frac{bits}{símbolo} = 20.000\ bits/s$$

El software de telemetría indica que, si se activa el algoritmo de corrección de errores, se enviará información adicional que llegará a ocupar hasta el doble del ancho de banda disponible. Este mecanismo es recomendable para largas distancias, por lo que se supondrá activado.

También cabe destacar que los propios módulos envían una identificación con cada mensaje, por lo que, a esos 20kb, habrá que añadir un 20% más, quedándose en 24kb de volumen de datos. Para activar la corrección de errores sin tener problemas de ancho de banda, se establecerá en una velocidad de 64 kbps. La tasa de errores disminuirá mientras más baja sea la velocidad, por eso es importante no establecer un ancho de banda en exceso.

Con estos datos y la ayuda del programa Radio Mobile Online, se simuló la cobertura para la tasa de transferencia estimada, con las características de los módulos elegidos (véase apartado 2.4.1) y antenas tipo monopolo de cuarto de onda para 433MHz.

Ilustración 5. Cobertura en mapa real de RedBull Ring en Spielberg, Austria.

Ilustración 6. Cobertura en mapa real de Hockenheimring en Hockenheim, Alemania.

2.4. Ensayos y validaciones

En este apartado se expondrán los motivos por los cuales se ha tomado la decisión de utilizar cada dispositivo y se tratará de comprobar de forma empírica las suposiciones antes realizadas, desde la elección de las antenas hasta la programación del microcontrolador.

2.4.1. Módulos de telemetría

De las dos bandas de frecuencias más notables para utilizar sin licencia, se tomó la decisión de utilizar la banda de menor frecuencia. Aquí se tienen módulos comerciales preparados para acoplar a drones y poseen un alcance muy bueno. Existen normalmente dos variantes, una de 433MHz y otra de 915MHz. Esta última es la banda sin licencia que se utiliza en E.E.U.U., por lo que para su uso en la UE se deberán escoger los módulos de 433MHz.

La principal ventaja que tiene frente a la frecuencia de 2.4GHz es que puede cubrir una mayor distancia instalando módulos de menor potencia, debido a la mayor longitud de onda. Al no ser necesario un radioenlace de alta velocidad, se descartó el uso de frecuencias Wi-Fi. También, debido a experiencias anteriores, se dejaron de utilizar dispositivos como Bluetooth o ZigBee al no proporcionar el alcance adecuado.

Frecuencia	433 MHz	915 MHz	2.4 GHz	5 GHz
Potencia	100 mW (20dBm)	100 mW (20 dBm)	2 W	6 W
Sensibilidad	-117 dBm	-121 dBm	-95 dBm	-80 dBm
Rango	1 km	1 km	> 2 km	> 4km
Velocidad	Hasta 250 kbps	Hasta 250 kbps	10 Mbps	6-54 Mbps

Tabla 2. Comparación de módulos para telemetría.

Los módulos elegidos tienen las siguientes características:

- Banda de frecuencia: 433MHz
- Potencia: 100mW ajustable
- Interfaz: Estándar TTL UART
- Sensibilidad receptora: -117dBm
- Velocidad de datos: hasta 250kbps
- Rango aproximado de 1 milla
- Verificación de errores (ECC)
- Soporte para LBT y AFA
- Duty Cycle configurable
- Control de flujo de datos

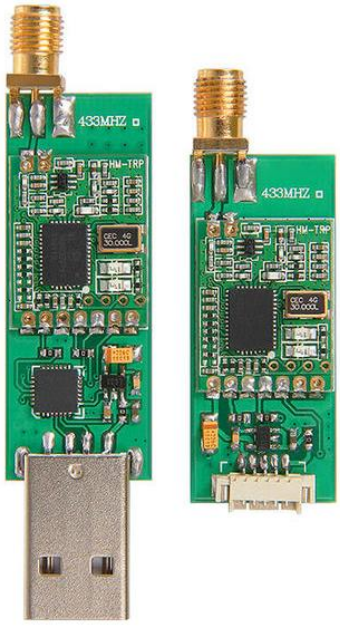


Ilustración 7. Módulos de telemetría.

Corrección de errores

Como se mencionaba en las características, si se activa la opción ECC, el módulo soportará un código de corrección de Gray 12/24. Esto significa que, por cada 12 bits de datos, el módulo enviará 24 bits, y se computará utilizando la tabla de referencia de código Gray [3]. El proceso de recepción de un paquete permite restaurar 3 bits de cada 12 bits transmitidos, lo que equivale a reducir los errores en un 25%. La desventaja de activar esta opción es que reducirá a la mitad el ancho de banda disponible, pero es altamente recomendado en la mayoría de situaciones, ya que permite una conexión más fiable para grandes distancias.

Cambio automático de frecuencia

Estos módulos disponen de la tecnología AFA (Adaptive Frequency Agility), una técnica utilizada por los transmisres para evitar interferencias con canales ocupados. El transmisor monitoriza periódicamente el entorno local y analiza los canales ocupados. Basado en esta información, el módulo selecciona una frecuencia de operación que no se esté utilizando. Esta posibilidad puede ser muy útil cuando la banda de frecuencia está compartida entre un gran número de usuarios, como puede ser el entorno de una competición de FSAE.

Esta tecnología se suele combinar con LBT (Listen Before Talk), que permite al transmisor escuchar el canal antes de comenzar a transmitir, para así cerciorarse de estar en un canal operativo libre.

Modulación

Esos módulos utilizan una modulación por desplazamiento de frecuencia gausiana (Gaussian Frequency Shift Keying, GFSK). Aquí se representará un 1 lógico mediante la desviación positiva de la frecuencia de la onda portadora, y un 0 lógico mediante una desviación negativa.

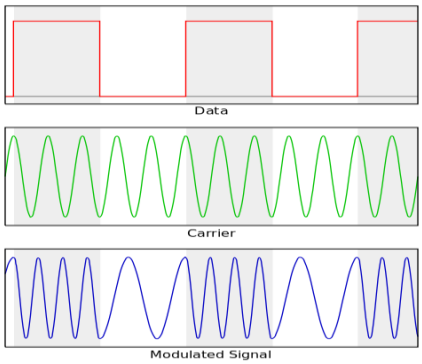


Ilustración 8. Funcionamiento de la modulación FSK

En la modulación GFSK, a diferencia de FSK, los pulsos digitales pasan a través de un filtro gaussiano antes de la modulación. Esto hace que la señal sea más suave (elimina los cambios bruscos de la señal digital) y, por lo tanto, limita el ancho del espectro modulado.

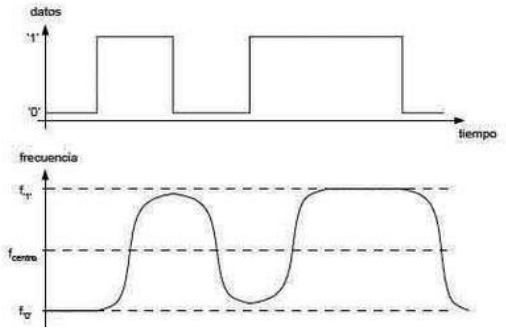


Ilustración 9. Suavizado mediante filtro gaussiano.

Configuración de los módulos

La comunicación con el PC o con el microcontrolador se realiza por puerto serie, y se puede configurar fácilmente mediante comandos AT. Para ello se dispone de un software preparado para leer el estado y configurar los parámetros adecuadamente a la aplicación que se realizará. Las configuraciones disponibles se describirán a continuación:

- Baud: ratio con el que el programa o vehículo se comunicará con la radio local (por defecto está a 57, que indica 57600 baudios)
- Air Speed: ratio con el que se comunicará el radioenlace. La distancia depende de la velocidad, por lo que es importante utilizar la velocidad óptima, pudiendo ganar alcance y calidad de señal.
- ECC: cuando está habilitado, enviará un CRC de 16 bits con los datos para detectar posibles fallos. Esto reduce notablemente la tasa de datos, pero se recomienda mantener activado, especialmente en ambientes ruidosos.
- MAVLink: modo optimizado para paquetes MAVLink. Se debe utilizar el modo Low Latency cuando se pretende controlar un vehículo con un tiempo de respuesta mínimo.
- Tx Power: la potencia de transmisión del módulo. Se establecerá a máxima potencia (100mW), cuyo valor está permitido en las normas reguladoras del espectro radioeléctrico.
- Duty Cycle: porcentaje máximo de tiempo en el que el módulo transmitirá paquetes. En algunas regiones, está permitido subir la potencia o cambiar la frecuencia si se disminuye este valor.
- Max Window: asegura el envío de un paquete al vehículo cada tiempo en milisegundos especificado.
- LBT Rssi: mantiene el umbral utilizado para "listen-before-talk" que permite cumplir con los requisitos reglamentarios de algunas regiones. Este parámetro mantiene la intensidad de la señal receptora por debajo de las cuales son consideradas ondas silenciosas.
- RTS CTS: control de flujo de datos por hardware.

2.4.2. Elección de la antena para el radioenlace

Los módulos de telemetría venían acompañados de unas antenas para frecuencias Wi-Fi de 2.4GHz apropiadas para acoplar en drones, debido a su tamaño reducido. Puesto que aún no se disponían de las antenas definitivas, se decidió empezar a experimentar con las actuales. Se probó la eficiencia de estas antenas, y de otras de similares características, pero de mayor ganancia, y los resultados no fueron apropiados para la aplicación buscada, como se suponía de antemano. Para ello se utilizó la misma aplicación de configuración, que posee un modo de comprobación mostrando el RSSI en todo momento.

Según el manual de usuario [4] el modo depuración de RSSI muestra un registro (R0) de 8 bits, lo que devuelve un número entre 0 y 255 indicando el nivel de potencia de la señal. Los rangos típicos de RSSI en un enlace son de 40-60 para mantener la conexión, más de 60 para enviar datos correctamente, y un valor igual a 217 como el máximo reportable en el integrado utilizado. Estos datos se pueden contrastar con la información de las gráficas en la Ilustración 10 y en la Ilustración 11.

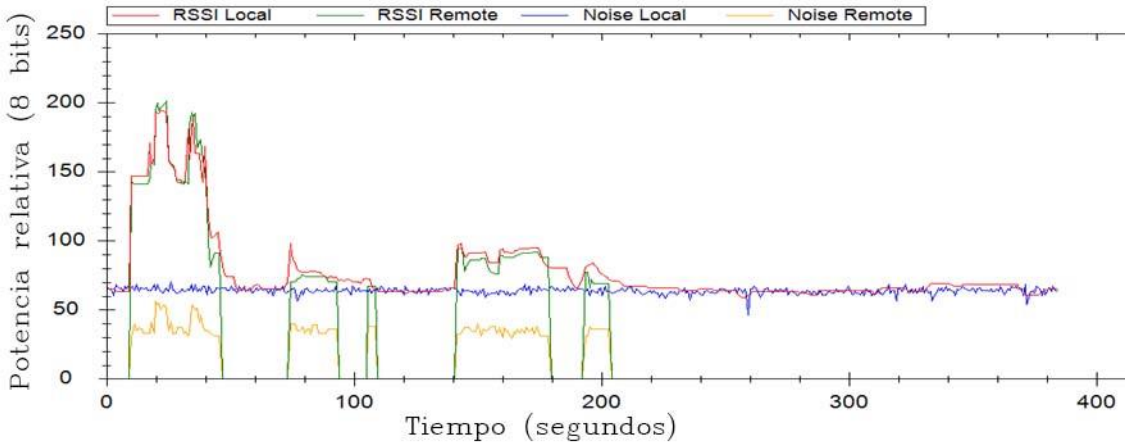


Ilustración 10. Comprobación del alcance utilizando antenas Wi-Fi de reducido tamaño.

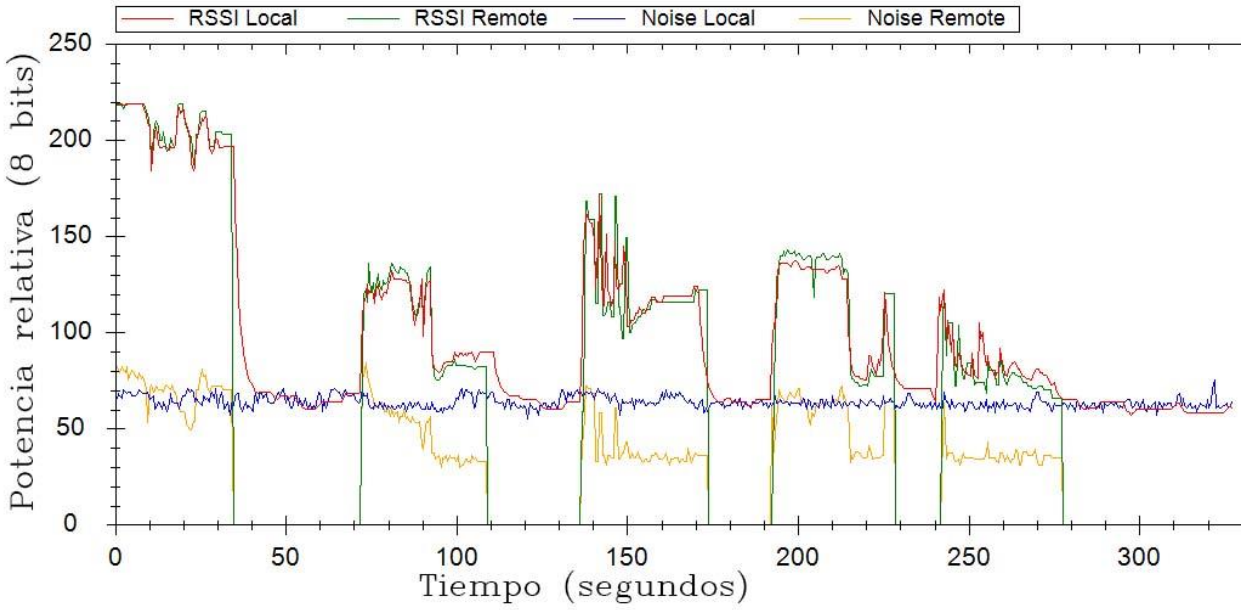


Ilustración 11. Comprobación del alcance con antenas superiores.

Las pruebas se realizaron con el test proporcionado por el software en distintos puntos de una zona urbana cercana a la ETSI. Cada valle de los gráficos corresponde a la desconexión del módulo. Esto se hizo así para comprobar la reconexión de los módulos a distintas distancias. La metodología seguida fue la de sostener la antena a una distancia de 20 metros del receptor, apagar el módulo y volver a encenderlo tras alejarse otros 20 metros. Se repitió dicho experimento hasta comprobar que los módulos no fueran capaces de reconectarse entre ellos.

Las gráficas muestran el RSSI de la señal (intensidad recibida) y el ruido de los módulos. Contrastando ambos resultados se puede calcular la calidad de la señal. También se comprobó que, como muestra la ayuda del software utilizado, mayor velocidad en la comunicación implica un menor alcance del radioenlace.

Se puede apreciar una notable mejoría al utilizar las antenas de mayor tamaño, pero aun así los resultados no fueron buenos, ya que la distancia más lejana era de apenas 70 metros, donde ya la señal recibida se confundía con el ruido local, perdiendo así la conexión entre módulos.

Para mejorar los resultados se estudiaron distintos tipos de antenas. Al estar el vehículo en constante movimiento en el circuito, se decidió elegir un tipo de antena omnidireccional frente a una directiva. Esto fue así ya que éstas radian la potencia en todas direcciones del plano horizontal, siendo las directivas inviables debido a los cambios de orientación del monoplaza durante la carrera.

	TX		RX	
Tipo	Monopolo	Monopolo	Dipolo	Panel direccional
Frecuencia	430-450 MHz	5 GHz	433 MHz	5 GHz
Ganancia	0 dB	3.5 dBi	2.15 dBi	23 dBi
Conector	SMA	TNC	SMA	N
Impedancia	50 Ohm	50 Ohm	50 Ohm	50 Ohm

Tabla 3. Comparación entre distintos tipos de antenas.

La antena será un monopolo vertical, con una longitud de cuarto de onda, conector SMA y sin ganancia propia. Al ser la frecuencia de los módulos de 433MHz, la longitud de la onda emitida será de 0.69 metros, lo que dejará una longitud de antena de aproximadamente 17 cm ($\lambda/4$). Se procuró utilizar un modelo lo más flexible posible para evitar rozamiento con el aire en el vehículo.

Para comprobar la validez de esta nueva antena se preparó una prueba pertinente similar a las condiciones de trabajo reales en las que se implementará en el monoplaza. Los resultados fueron satisfactorios, y posteriormente se explicará en el apartado 2.4.4 la metodología seguida y la configuración utilizada.

2.4.3. Microcontrolador

El microcontrolador utilizado es el MSP430G2553, que ofrece las capacidades necesarias para la aplicación y su uso es motivado por la escuela. En años anteriores se utilizó Arduino para la electrónica, ocasionando graves problemas de rendimiento debido a las limitaciones en su programación y complicando el diseño, ya que se utilizaba a modo de shield, donde la placa encajaba en una serie de pines y se programaba directamente como si de un prototipo se tratase.

La temporada anterior no se utilizó ningún sistema de telemetría, pero aún así se describirán los problemas ocasionados con Arduino por los que se decidió no utilizar. El sistema de Arduino es un microcontrolador programable de la empresa Atmel, adquirida recientemente por Microchip. La ventaja recae en que tiene integrado un bootloader que se encargará de gestionar los periféricos del integrado y todos sus puertos, aumentando así el nivel de abstracción de la programación. Tan solo es necesario indicar qué periférico se querrá utilizar y bajo qué características, y será el bootloader el encargado de gestionar su configuración y utilización. La ventaja es que un mismo código podrá ser compatible para distintos sistemas, tan solo cargando el bootloader adecuado. El problema es que no se tiene actuación sobre esa capa, por lo que tiene ciertas limitaciones de programación. A diferencia de los microcontroladores Arduino, el MSP430 se puede programar configurando todos los registros disponibles, y se podrá utilizar todos los periféricos sin restricciones.

El microcontrolador de Texas Instruments posee las siguientes características:

- Frecuencia hasta 16 MHz
- Memoria no volátil de 16 kilobytes
- Memoria RAM de 512 bytes
- 24 pines de entrada/salida
- I2C, SPI, UART
- ADC de 10 bits con 8 canales
- 2 Timers de 16 bits
- VCC max 3.6v
- VCC min 1.8v
- Temperatura de operación de -40 a 85 °C
- Encapsulado PDIP, TSSOP, VQFN

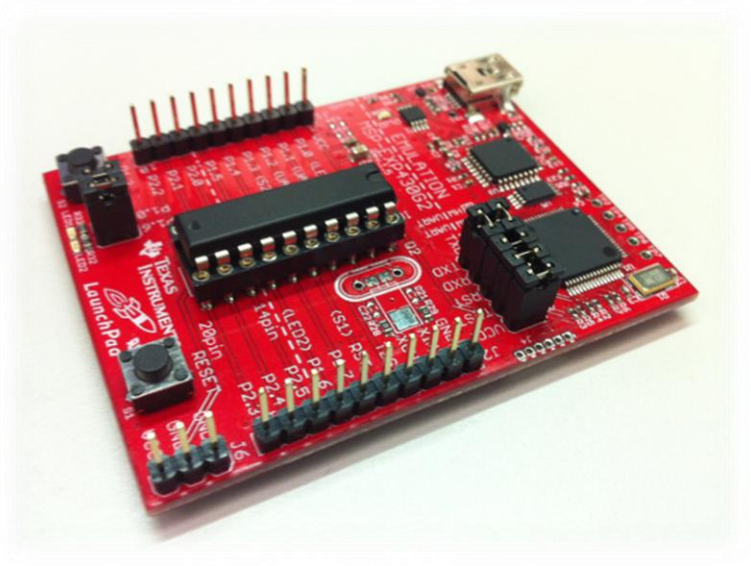


Ilustración 12. LaunchPad de TI con microcontrolador MSP430.

Las características necesarias para la aplicación de telemetría son principalmente el uso de UART para comunicación, pines digitales de carácter general (GPIO), y el uso de temporizadores para coordinar un ejecutivo cíclico que dividiera las tareas a realizar en un intervalo de tiempo.

Este microcontrolador también se utilizó en los demás subsistemas, a excepción de la tarjeta de adquisición de datos que necesitaba mayor velocidad y el salpicadero, que se decidió utilizar Arduino por la facilidad de programación (existencia de librerías para la pantalla). Todos estos sistemas estarán comunicados mediante el bus CAN, y con la ayuda del controlador MCP2515 que se comunica con el microcontrolador mediante SPI, se podrán enviar y recibir tramas estableciendo un orden de prioridades.

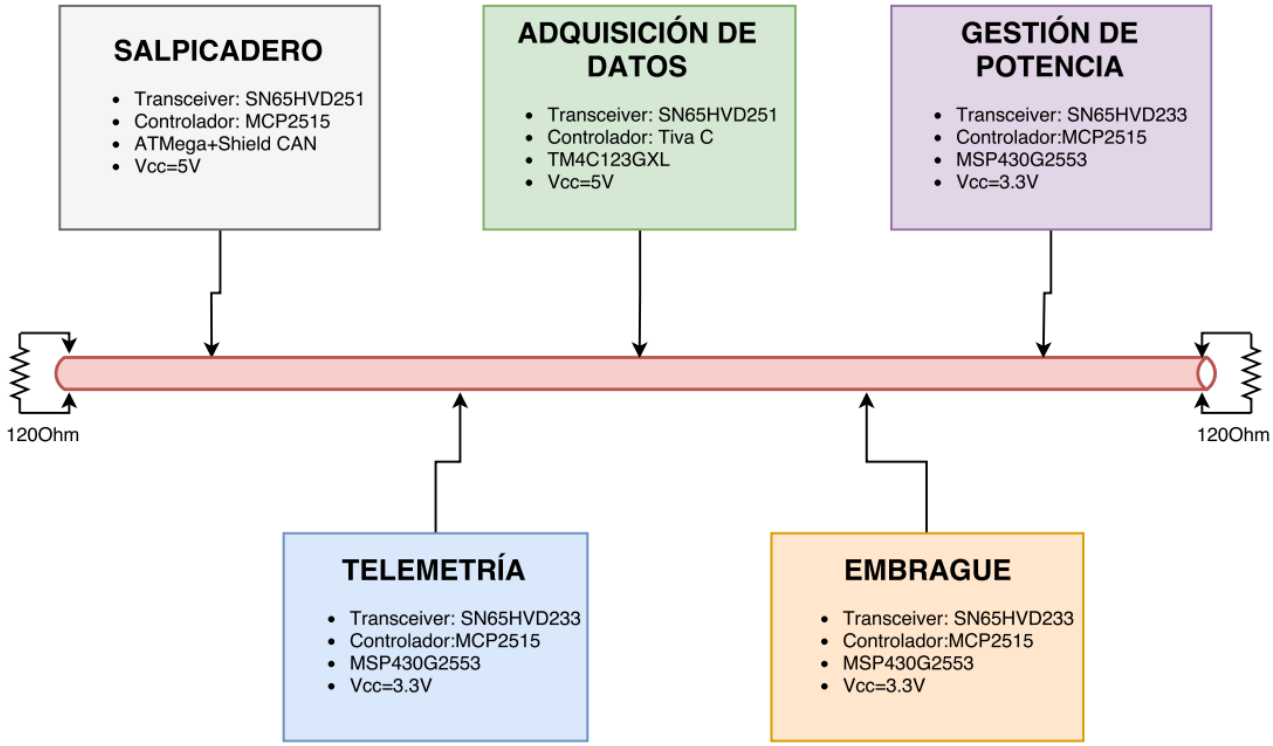


Ilustración 13. Esquema general de la conexión de los sistemas mediante el bus CAN.

Dicho microcontrolador se puede programar con Code Composer Studio (CCS) utilizando un lenguaje adaptado de C. Una característica muy importante que se ha conseguido frente al uso de Arduino es la capacidad de utilizar un modo de depuración de fallos mientras se ejecuta la aplicación programada, permitiendo pausar la ejecución y comprobar los valores de todos los registros y variables. Esto se hace posible conectando los pines de programación del microcontrolador al PC a través de la placa de desarrollo ofrecida por Texas Instrument, de la misma forma que se programa en memoria, se puede utilizar este método de depuración. Se puede observar un diagrama de la conexión en la Ilustración 26. Esquema genérico de programación on-board.

2.4.4. Tasa de datos y errores

Finalmente se validaron los dispositivos utilizados en una prueba que simula de manera fiel la funcionalidad real que tendrá en el monoplaza. Con un LaunchPad de Texas Instruments (MSP430) conectado al módulo de aire de telemetría (simulación del vehículo) y el módulo de tierra conectado a un PC, se procesaron los datos enviados por el microcontrolador en la computadora con un sencillo programa de LabVIEW.

Dicho experimento consistía en enviar durante 10 segundos una trama similar a la que se tendrá en el sistema final a razón de 100 datos por segundo y comparar esa trama con el programa de PC.

Si el dato llega correctamente, se incrementará el número de datos correctos. Si por lo contrario la trama no coincide, se guardará en un fichero de texto los caracteres leídos en ese momento para analizarlos posteriormente. De esa forma se validará el número de datos correctos frente a los enviados y, además, se podrá analizar el tipo de fallo que pueda aparecer para evitar falsear la medida con datos erróneos.

Las pruebas se realizaron en un descampado situado en zona urbana, donde los obstáculos que se podían encontrar eran pocos árboles y unos cables de alta tensión, los cuales podrían tener efectos negativos para las pruebas, pero que a priori se veían con buenos resultados.

Ilustración 14. Distancia máxima probada con las nuevas antenas.

Para la realización de las pruebas, primeramente, se empleó el programa utilizado para configurar las antenas, el cual proporciona datos a tiempo real sobre la calidad de la señal. La configuración de las antenas fue la siguiente:

- Potencia máxima de emisión (100mW)
- Velocidad de 128kbps (se bajará a 64kbps para el sistema del vehículo, ya que el volumen estimado de datos a enviar será de aproximadamente 24kbps).
- Frecuencia principal de 430MHz y canal 15, siendo posible cambiarlo si se detectara algún tipo de interferencia.
- Código de corrección de errores (ECC) activado, siendo capaz de restaurar hasta un 25% de los datos corruptos, asumiendo el coste de doblar la cantidad de bits enviados y restar así ancho de banda.

Una primera prueba consistió en medir la calidad de señal mediante los datos proporcionados por el programa de configuración de los módulos, aumentando la distancia entre antenas hasta lograr perder la señal. Seguidamente se utilizó un programa realizado en LabVIEW junto con el MPS430 para una segunda prueba en la que se enviaba una cadena de datos y se verificaba posteriormente.

Primera prueba: medición del RSSI

Como se comentó anteriormente, se utilizó el programa 3DR Radio para contrastar los resultados de esta prueba. Debido a la difícil exportación de los datos y a la lentitud de dichas pruebas, solo se comentarán las conclusiones obtenidas en el experimento.

Primordialmente, las antenas se comportaron de la forma esperada, disminuyendo el ratio de potencia de señal/ruido con la distancia. Se observó que dicha relación variaba de forma prácticamente aleatoria a velocidades de transmisión altas, siendo mucho más estable cuando se disminuía la misma. También se observó que en ciertos puntos de la zona se podía observar un descenso pronunciado en la gráfica, que recuperaba cambiando la posición, aun estando más lejos que la anterior. Esto demuestra que la transmisión de los datos dependerá mucho del entorno, siendo muy difícil predecir a priori el alcance total de las antenas.

La distancia total probada donde se podía mantener conexión con la antena fue de aproximadamente 450 metros, siendo esta fácilmente ampliable elevando la antena de tierra unos metros por encima del suelo. Para una altura "h" de instalación de un monopolo de L/4 en las bandas VHF y UHF, trabajando a una longitud de onda L, se tiene que la ganancia máxima se conseguirá para $h = 3n * L$, siendo n un número entero; y la ganancia mínima se conseguirá con $h = 6n * L$. Estos cálculos se tendrán en cuenta para la fase de fabricación, pero debido a la falta de materiales, no se pudo comprobar dicho efecto en toda la prueba, ya que se deberá fabricar un cable prolongador para asegurar la antena a un soporte mientras sigue conectada a un puerto del ordenador.

Segunda prueba: envio de datos y medición de errores

Para esta segunda prueba se realizó un pequeño programa para el microcontrolador, consistente en enviar una trama larga de 26 caracteres cada 100ms. Dicha trama se recogerá en LabVIEW y se analizará como se muestra a continuación: el panel incluye la configuración básica del puerto serie, donde se indicará el puerto correspondiente, y una velocidad de 57600 baudios. Contiene indicadores para mostrar el tiempo inicial de la recogida de datos, el buffer de lectura para corroborar el buen funcionamiento de la trama y la cuenta de los datos y errores obtenidos.

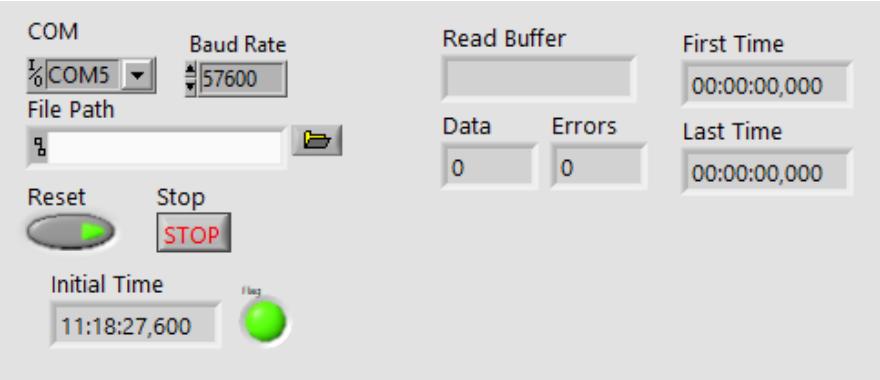


Ilustración 15. Interfaz gráfica de la prueba con LabVIEW.

Se configurará un tiempo de recogida de datos y una ruta de archivo. Tras pulsar el botón reset comenzará la prueba desde cero. El programa estará durante el tiempo indicado analizando las tramas que ha llegado y, al finalizar, grabará en un archivo de texto las tramas erróneas que ha recibido y el número total de datos.

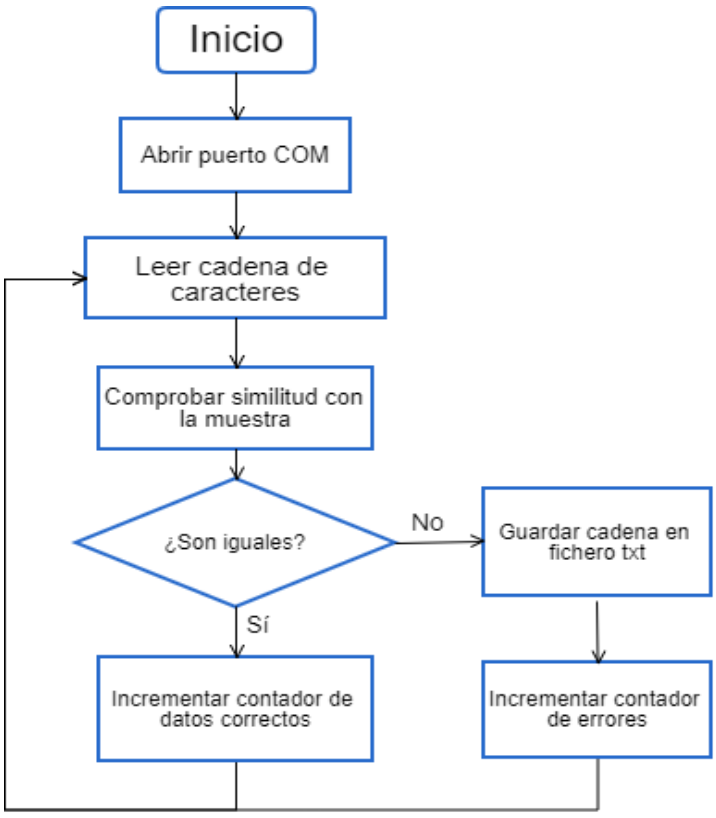


Ilustración 16. Diagrama de flujo de la prueba con LabVIEW.

A continuación, se muestra un extracto del archivo de texto generado, donde estarán indicados los errores detectados.

```
PQRSTUVWXYZ    016
ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWX    133
YZ    234
ABCDEFGHIJKLMNOABCDEFGHIJKLMNOPQRSTUVWXYZ    343
XYZ    445
FIN TEST OK 98 DATOS
```

Cada trama errónea comparada con la original se escribe en el fichero, acompañada de una tabulación y un número que indica el dato correspondiente. Analizando la primera línea se tiene lo siguiente: la trama llegada es *PQRSTUVWXYZ*, que comparando con la original *ABCDEFGHIJKLMNOPQRSTUVWXYZ*, se puede observar como se ha producido un corte al principio, y el número a su lado es 016, lo que quiere decir que antes llegaron 15 datos correctos, siendo este el número 16.

Se puede observar como los errores obtenidos son mezclas de tramas, donde falta algún tipo de dato para ser completado correctamente. Como ocurre en la segunda línea, el caracter perdido en este caso es el salto de línea, por lo que se concatenó con la trama llegada posteriormente. En ningún momento se ha localizado un fallo donde haya aparecido algún caracter corrupto o inesperado, siempre son fallos en los que se pierde información.

En las pruebas se tomó un tiempo de recogida de datos de 10 segundos, y para distancias largas se repitió varias veces. Los datos obtenidos se representan a continuación, donde se puede apreciar el número de mensajes recibidos y los errores encontrados. Cabe destacar que, debido al tiempo que tarda el microcontrolador en añadir al buffer y enviar los datos, el tiempo tomado por el módulo para la corrección de errores y el procesado de la información por LabVIEW, los datos esperados serán menos de 100.

Esto se traduce en que, al aumentar las distancias, ocurrirán más errores o pérdidas de datos, por lo que el sistema de detección de errores que poseen los módulos tendrá que intentar recuperarlos. Este tiempo de procesamiento hará que la siguiente trama permanezca en el buffer un tiempo demasiado largo, ocasionando la pérdida de los siguientes datos enviados.

Los errores analizados se pueden solventar fácilmente a la hora de interpretar la cadena de caracteres, simplemente haciendo que lea un formato específico. Si por algún motivo la cadena se solapa con la siguiente, se detectará que no cumple el formato y se podrá desechar el dato.

Metros	Datos	Errores
0	98	0
50	99	0
100	98	0
150	96	0
200	92	1
250	93	0
300	85	1
320	95	0
340	77	3
360	68	1
380	59	2
400	77	1
410	44	1
420	48	0

Tabla 4. Datos y errores recibidos según la distancia.

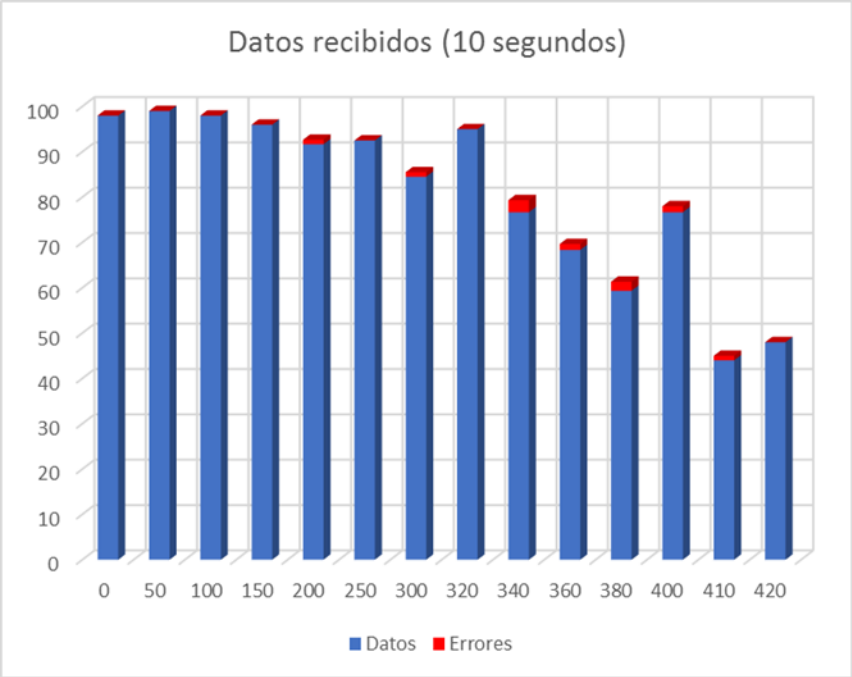


Ilustración 17. Gráfica de datos recibidos frente a la distancia entre antenas.

Una vez más, se encuentra un comportamiento lógico donde empeora la tasa con la distancia, salvo en algunos puntos donde debido a difracciones de las ondas se podrán obtener mejores o peores resultados. En general, para la distancia obtenida se han obtenido resultados bastante buenos, ya que la condición del envío de datos es mucho más desfavorable de la que se utilizará en la implementación del sistema para el ART-17. Pero no se podrá comprobar dicho comportamiento en su totalidad hasta que el sistema no esté en funcionamiento bajo condiciones reales.

3 DESARROLLO DEL HARDWARE

Si una persona es pereseverante, aunque sea dura de entendimiento, se hará inteligente; y aunque sea débil se transformará en fuerte.

- Leonardo Da Vinci -

En este apartado se prende explicar en profundidad el funcionamiento de toda la electrónica implicada. Mostrar el diseño y todos los pasos realizados para lograr el resultado esperado.

Para realizar el diseño de la placa se ha utilizado el software Proteus 8. Este programa, como muchos otros dedicados a dicho fin, se compone de dos partes. La primera se compone de una aplicación de diseño asistido por computadora (CAD) en 2D, donde se podrá dibujar todo tipo de componentes y conexiones. La segunda parte toma el esquemático realizado y trata de facilitar lo máximo posible el diseño de la PCB indicando el conexionado restante.

También se han utilizado programas de diseño en 3D para realizar los soportes necesarios y cajas protectoras, cuya fabricación se realizará en impresión 3D.

3.1. Esquemático de la electrónica

En la placa que forma el salpicadero se engloban dos sistemas completamente integrados y comunicados por el bus CAN. Los microcontroladores disponibles son un ATMEGA328p, programado como Arduino y un MSP430 de Texas Instrument. El primero se encargará de mostrar los datos relevantes al piloto mediante una pantalla y otros indicadores LEDs. También tiene la posibilidad de controlar ciertas funciones con switches. Por otra parte, el MSP430 será el que se tratará para este proyecto, encargado principalmente de la telemetría, y de una función auxiliar para controlar un display de 16 segmentos.

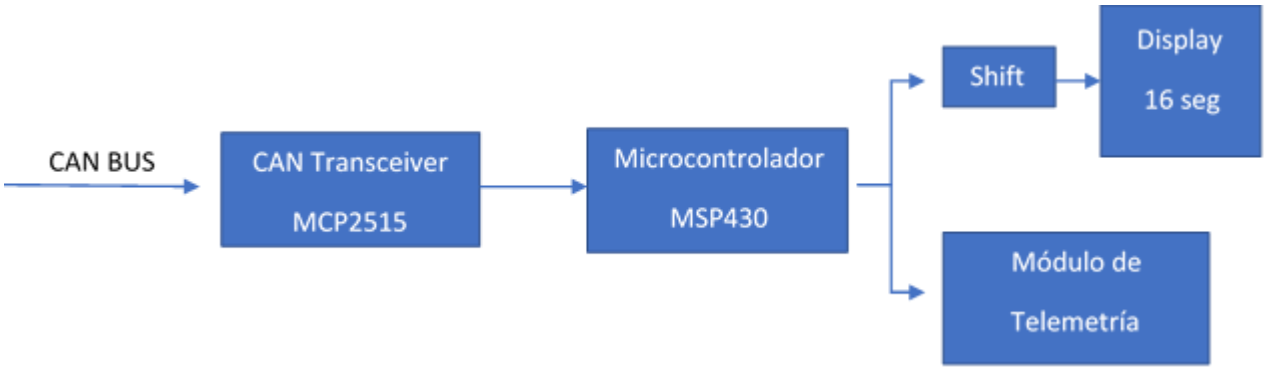


Ilustración 18. Esquema general del hardware para el sistema de telemetría.

Existen tres partes bien diferenciadas en el esquemático. La primera y más importante es la destinada al microcontrolador, el cerebro que se encargará de leer y administrar los datos necesarios. También se encuentra la parte que se encarga de captar los datos llegados por el bus CAN, el cual comunica con el resto de los sistemas. Y, por último, están los dos acondicionamientos para el display que mostrará la marcha actual y el módulo para la telemetría, que se encontrará externo a dicha placa.

3.1.1. Microcontrolador MSP430G2553

El microcontrolador posee un condensador de 100nF (C22) lo más próximo a la alimentación posible, como se recomienda en todos los encapsulados de la placa. También tiene una configuración de pull-up en el pin de reset para mantener el correcto funcionamiento y un botón a gnd para poder reiniciarlo manualmente. El condensador C33 sirve para retrasar la alimentación en dicho pin, de forma que el microcontrolador permanecerá un instante reseteado en el inicio. Esto se hace como medida de seguridad para garantizar el correcto funcionamiento del código.

Para mostrar visualmente la actividad del dispositivo, se ha añadido un LED en uno de los pines digitales de salida, con una resistencia apropiada para controlar la intensidad que circula, calculada de la siguiente forma:

$$R = \frac{V_{ref} - V_{LED}}{I_{LED}}$$

Donde se utilizó un led verde cuya caída de tensión es de 2.1V, y la intensidad admisible es de 10mA.

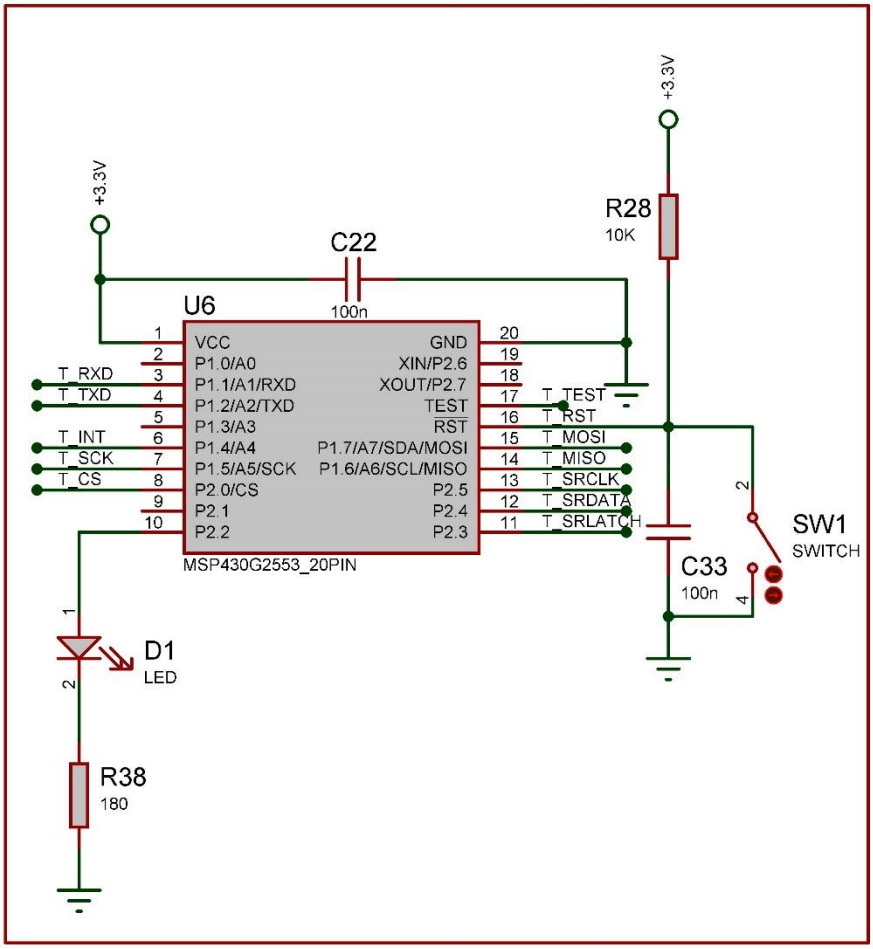


Ilustración 19. Conexión del microcontrolador MSP430G2553.

Los pines utilizados son los siguientes:

- Pines 1 y 20: alimentación del dispositivo a 3.3V
- Pines 3 y 4: conexión con la UART para la comunicación serie con el módulo de telemetría.
- Pin 6: interrupción del MCP2515 encargado del protocolo CAN.
- Pines 7 y 8: reloj y chip select para la comunicación SPI con el MCP2515.
- Pin 10: salida digital para encender o apagar el LED.
- Pines 11, 12 y 13: latch, dato y reloj para el shift register, encargado de pasar serie a paralelo y controlar el display de 16 segmentos.
- Pines 14 y 15: entradas y salidas para la comunicación SPI.
- Pin 16: reset físico del microcontrolador, necesario también para programación.
- Pin 17: pin de test para programar el microcontrolador

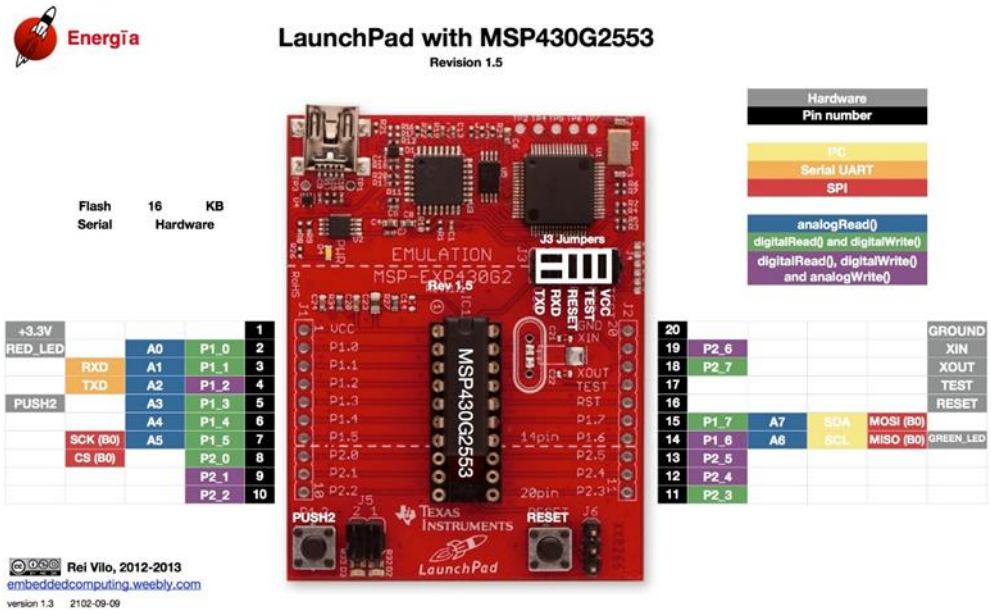


Ilustración 20. Pinout del microcontrolador en la placa de prueba.

3.1.2. Controlador MCP2515 y transceptor CAN

Esta parte tiene como objetivo comunicar el sistema de telemetría con el resto de sistemas del coche mediante el uso del bus CAN. Debe estar configurado correctamente para poder leer y enviar tramas cumpliendo el protocolo correspondiente a la velocidad adecuada.

Se ha diseñado el circuito con base a un shield de arduino que realiza el mismo fin y utiliza los mismos integrados. El MCP2515 se comunica con el microcontrolador mediante comunicación SPI, y también hace uso de un pin de interrupción para indicar que hay una trama lista para ser leída en el buffer.

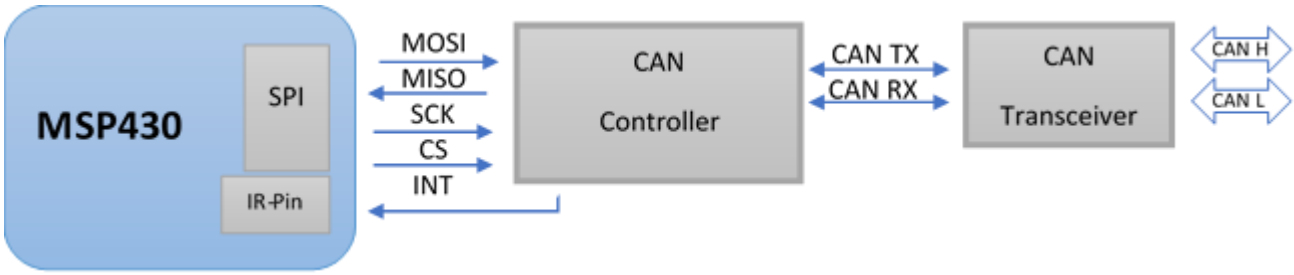


Ilustración 21. Conexión general de los integrados necesarios para controlar el bus CAN.

Se ha añadido externamente un reloj de 16 MHz necesario para su funcionamiento, con condensadores para estabilizar la señal, y el transceptor utilizado de Texas Instruments a 3.3V para traducir los niveles de tensión TTL a la tensión diferencial utilizada por el bus CAN. Ambos integrados tienen condensadores de alimentación y resistencias pull-up en los pines anteriormente mencionados.

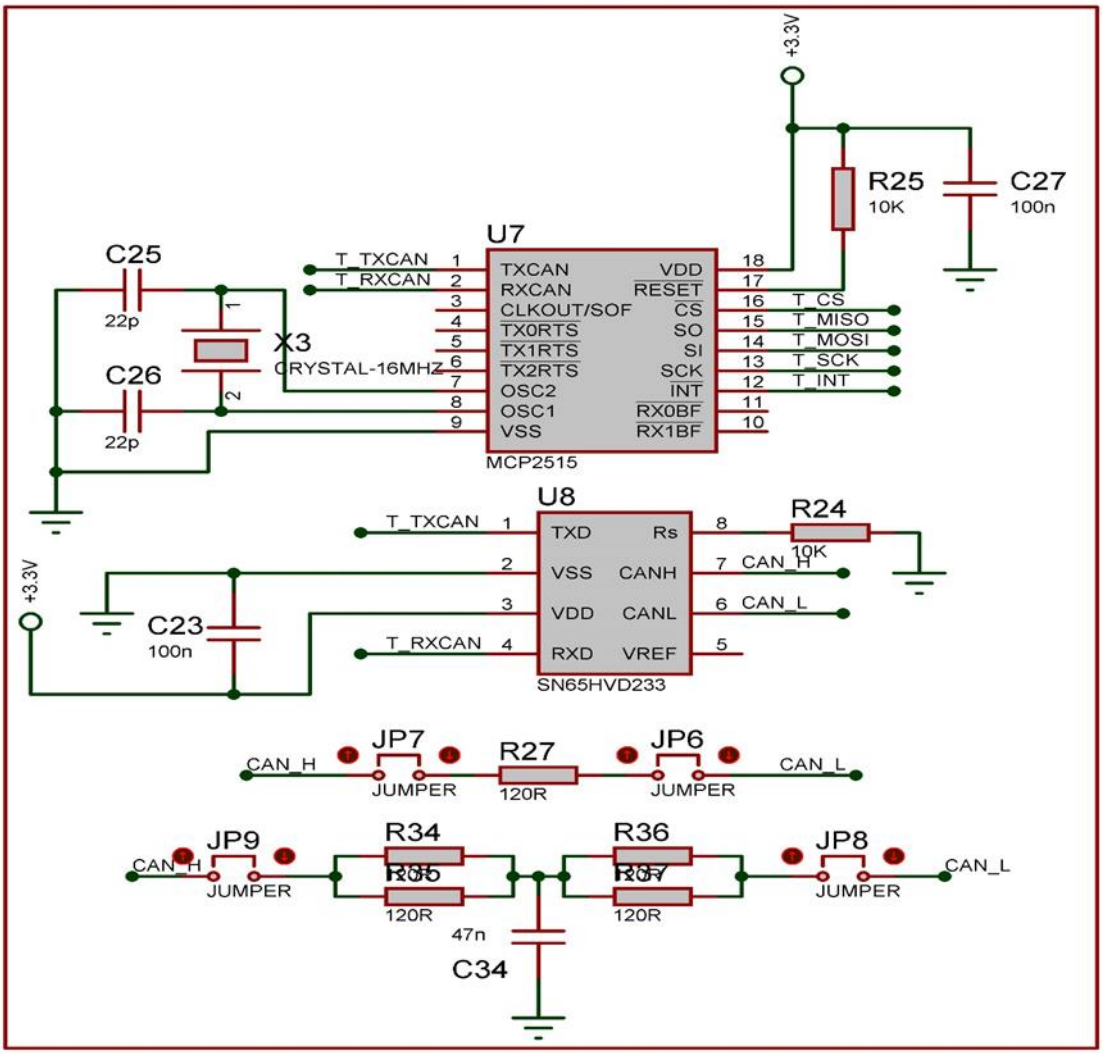


Ilustración 22. Conexión del integrado controlador del bus CAN y transceptor.

Cabe destacar que se ha añadido la resistencia terminal del bus de forma que pueda adquirir distintas configuraciones en un futuro. Gracias a la colocación de los jumpers se puede conectar tanto como configuración terminal como en estrella, o incluso con condensador recomendado para largas distancias, para así preservar los mensajes enviados.

La configuración de CAN de alta velocidad utiliza un único bus lineal con dos resistencias terminales de 120Ω, una en cada extremo, para evitar reflexiones en la línea. Mientras que la configuración de baja velocidad puede utilizar un bus lineal, un bus en extrella o mixto. En cada nodo existirá una resistencia terminal equivalente a la fracción correspondiente, para que esta sea de valor total próximo a 100Ω.

Es por ello por lo que se diseñó la PCB con estas posibilidades y combinaciones, ya que no estaba cerrada la configuración del bus CAN.

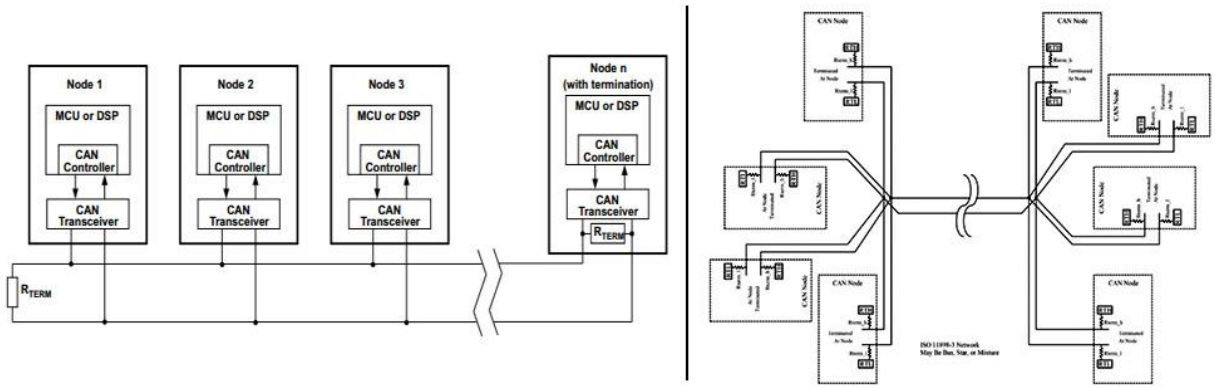


Ilustración 23. Bus CAN de alta velocidad y de baja velocidad tolerante a fallos.

3.1.3. Shift Register y display de 16 segmentos.

Este es un elemento indispensable del salpicadero. En un principio se pensó implementar en otro sistema ajeno a este, pero por falta de recursos en el otro microcotrolador del salpicadero se decidió implementar en el microcontrolador MSP430. Puesto que la telemetría ya se encargaba de enviar las marchas, esto no supondría un mayor problema que el de diseñar el circuito para ser alimentado a 3.3V.

El display de 16 segmentos estará controlado por dos registros de desplazamiento que convertirán los datos de serie a paralelo para mantener encendido los leds en el orden apropiado. Ambos integrados estarán comunicados al microcontrolador en configuración de cascada (uno detrás del otro) siendo el primero el que se comunica directamente con 3 pines (datos, reloj, enable). Como todos los integrados también necesitará un condensador de alimentación cercano para una mejor estabilidad y pines conectados a alimentación o tierra para configurar los enables de los biestables internos.

Internamente posee una serie de biestables tipo D que almacenan el dato y lo irán pasando al siguiente a medida que lee uno nuevo por la señal del reloj. Una vez completada la trama, se deberá activar un latch que pasará los datos en paralelo a otra serie de biestables conectados a la salida con una puerta triestado. De ese modo se consigue no alterar la salida mientras llegan los nuevos datos, para actualizar todas las salidas al mismo tiempo.

El display es de ánodo común, lo cual significa que el común estará conectado a alimentación mientras que los registros conectarán los pines de cada LED a GND para encender dicho led. Las resistencias están diseñadas de tal forma que limitan a la salida de cada led y pensadas para que, en el peor de los casos, cuando los 16 leds estén encendidos, la intensidad total que pase por los registros de desplazamiento no supere el límite establecido por el fabricante [5] de 70mA. Aun pudiendo dar más intensidad luminosa el display, se ha comprobado que la visibilidad de los números es más que suficiente, y como segundo efecto se conseguirá

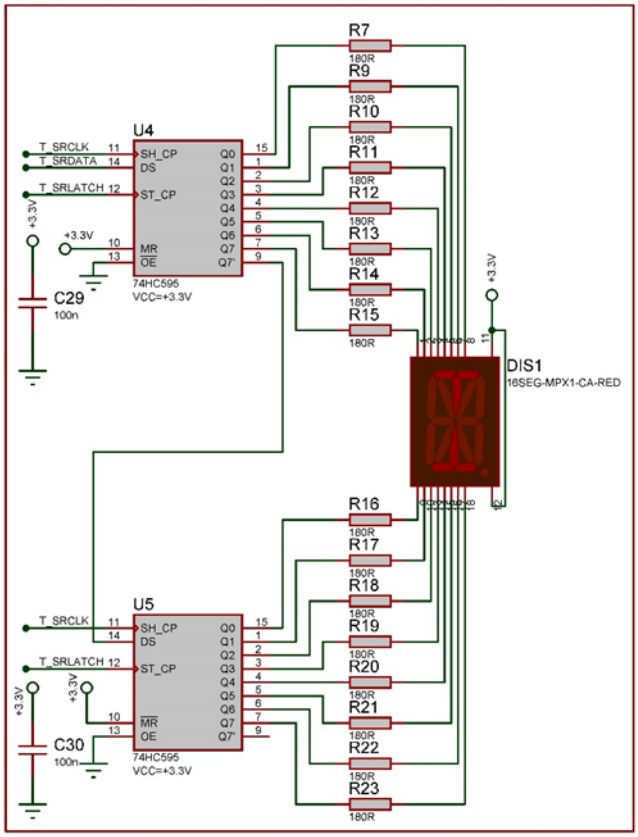


Ilustración 24. Conexión del display de 16 segmentos con registros de desplazamiento.

ahorrar batería. Este factor es importante, ya que el monoplaza no dispone de alternador, y la capacidad de la batería se tendrá que dimensionar de forma que no haya problemas para acabar la prueba de resistencia.

Cabe destacar que la información de la marcha actual es recibida por la ECU, calculando la relación entre las revoluciones del motor y la velocidad. Dicho cálculo no se hace inmediatamente y, en carrera, se aprecia un pequeño retraso en el cambio. Aun así, este indicador es muy importante para el piloto, que tendrá la certeza de que el cambio de marcha se ha realizado exitosamente, puesto que al ser un mecanismo automático puede fallar en alguna ocasión.

3.1.4. Módulos de telemetría

La comunicación entre el microcontrolador y el módulo de telemetría (emisor) es una simple conexión serie, donde dicho módulo necesita 2 pines para alimentación y otros 2 para enviar/recibir datos. Cabe destacar que en el mismo conector se ha añadido la referencia de la placa del volante y una señal para activar o desactivar la emisión de audio del Walkie Talkie mediante un botón alojado en los controles del piloto.

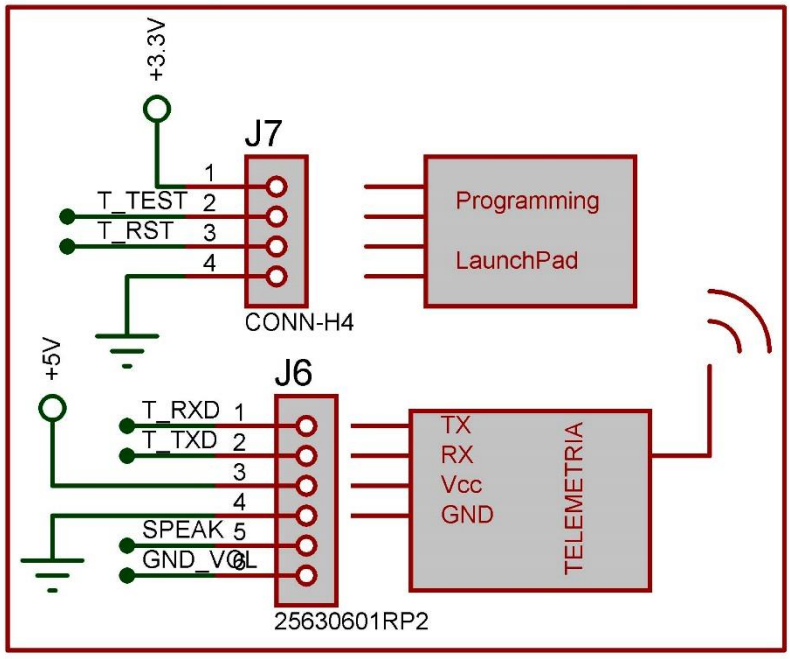


Ilustración 25. Conexiones para módulos de telemetría y programación.

Un factor a tener en cuenta es que, como en todas las conexiones de UART, el canal emisor (TX) de un dispositivo tendrá que ir conectado al canal receptor (RX) del otro dispositivo a comunicar. Es un error muy frecuente pensar que se tendrán que conectar ambos pines TX juntos, y de la misma forma los pines RX.

También se han añadido a la placa 4 pines a modo de conector para facilitar el acceso a los pines de programación del microcontrolador. De esta forma se podrá programar el chip on-board con ayuda de la parte de simulación del LaunchPad con la siguiente conexión:

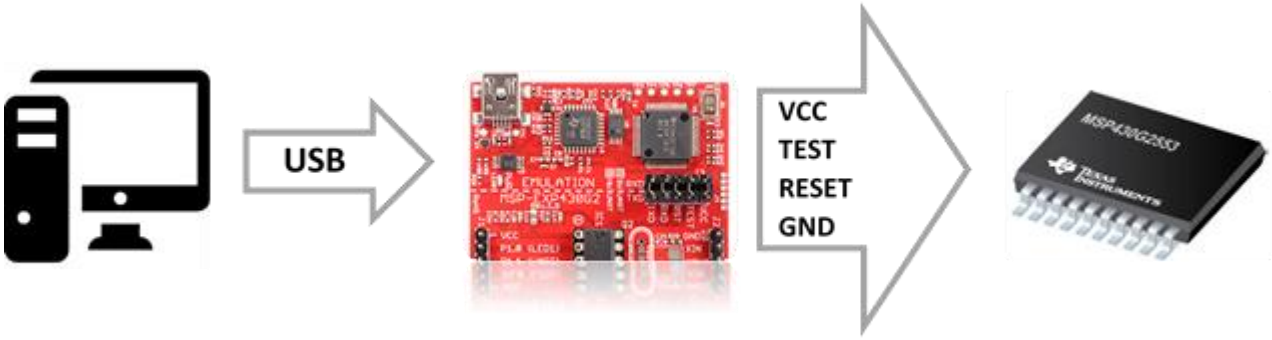


Ilustración 26. Esquema genérico de programación on-board.

3.2. Diseño de la PCB

En el salpicadero se deben posicionar diversos elementos para obtener una buena ergonomía. Por ello se ha de trabajar con el departamento de interior para adaptar el diseño a la disposición adecuada de pantalla, contactos, seta de emergencia, indicadores y actuadores. Esta labor la llevarían a cabo otros miembros del equipo, pero como se explicó anteriormente, también estará involucrado el microcontrolador de la telemetría debido al control del display de las marchas.

Una vez obtenida la colocación de dichos elementos, se procedió a exportar un plano con la silueta del salpicadero. Posteriormente, se comenzó a trabajar en Proteus partiendo de eso. Primero se colocaron todos los elementos es los huecos apropiados y, puesto que la placa debe abarcar un gran espacio del salpicadero, no se tienen estrictas limitaciones de espacio para el diseño.

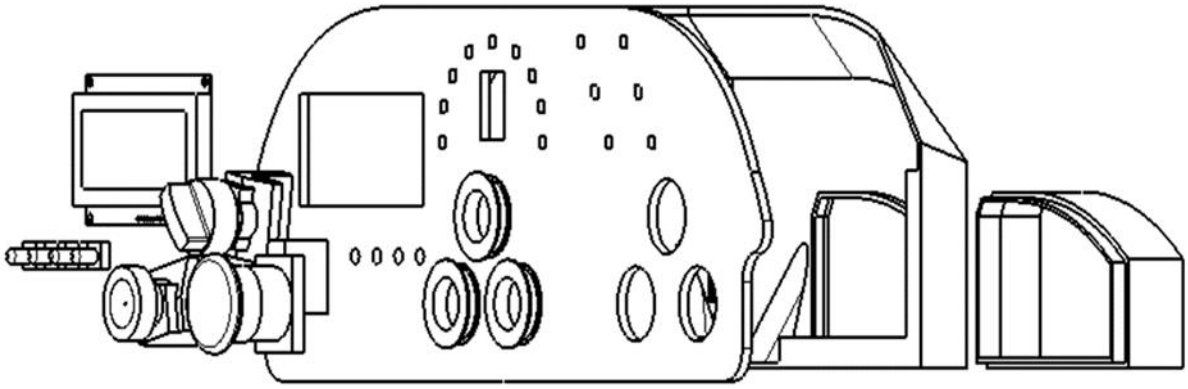


Ilustración 27. Explosionado de los elementos del salpicadero.

La realización del diseño se hizo con la ayuda de un compañero de equipo, quien estaba responsabilizado del sistema integrado en el salpicadero (pantalla, LEDs de aviso, …). En este documento se explicarán las estrategias seguidas que involucran únicamente el enrutado de los esquemáticos explicados anteriormente en el apartado 3.1. Las ilustraciones incluidas en este documento muestran el resultado final del trabajo realizado por ambos miembros del equipo, puesto que indicar tan solo una parte se antoja difícil.

En un principio se decidió realizar las placas del equipo en 4 capas, con planos de alimentación y tierra en el centro. De esta forma se conseguía evitar las interferencias electromagnéticas que pudieran ocasionarse debido al movimiento del motor y el vehículo en general. En siguiente ilustración se puede apreciar un sombreado sobre la zona de microcontroladores, correspondiente a los planos antes mencionados.

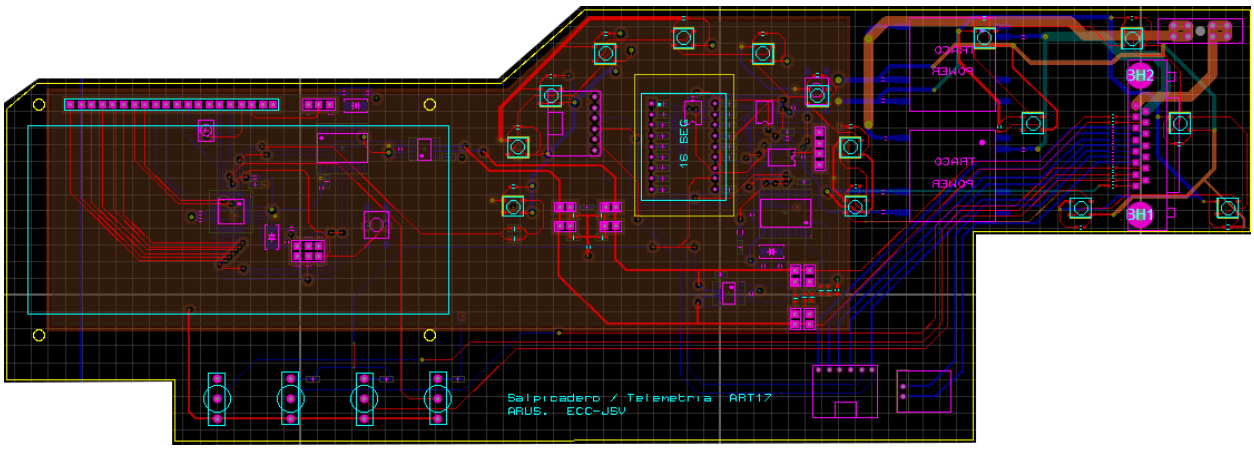


Ilustración 28. Diseño de la PCB multicapa.

El resultado obtenido cumplía perfectamente las especificaciones de diseño, ya que primero se colocaron los dispositivos visuales indicados en la plantilla, y posteriormente se añadieron los componentes y pistas. Al ser una placa cuyas limitaciones de tamaño venían dadas por la disposición de los elementos activos, se pudo realizar un diseño electrónico sin problemas de espacio. También se pensó en realizar varias tarjetas separadas, pero por experiencia del equipo se conocía de antemano que traería problemas con las sujeciones y los contactos de los elementos.

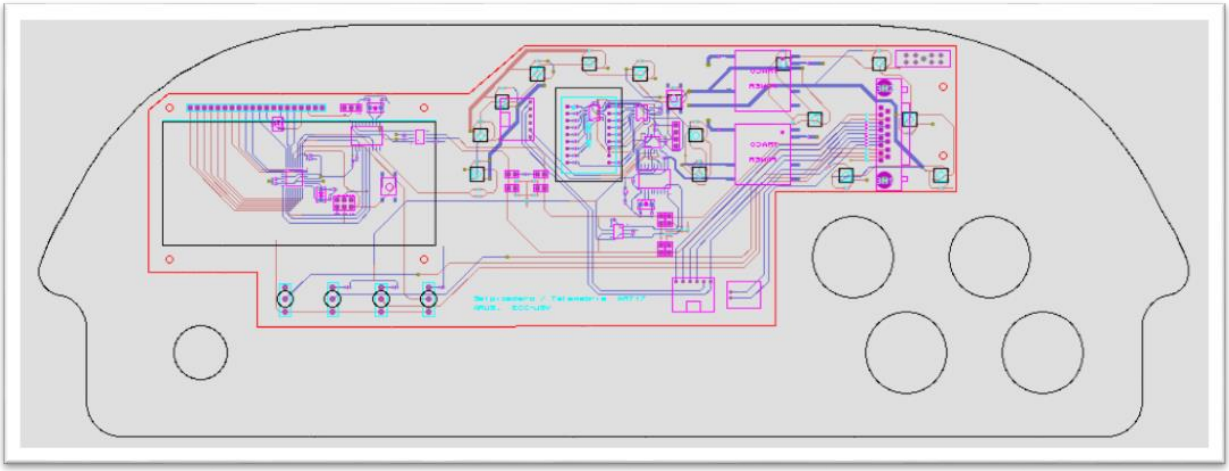


Ilustración 29. Diseño de la PCB con plantilla.

Tras realizar el diseño, el siguiente paso fue exportar a los formatos Gerber [6] correctos las capas para entregárselo a la empresa encargada de revelarla. Una vez se proporcionó los documentos necesarios, resultó en un coste muy elevado de fabricación debido al tamaño de la placa (26cm de largo) y ser de 4 capas. Por ello, se tomó la decisión de reducir el modelo a dos capas, una complicación que se solventó en poco tiempo.

Se tuvieron que tomar en cuenta ciertas restricciones al cambiar el modelo a dos capas. Al realizar el nuevo enrutado, las pistas de alimentación deben ser considerablemente mayores a las pistas de señal. Dicho conflicto no existía en el diseño anterior ya que, al tener un plano entero destinado a alimentación, no se daba la necesidad de enrutar pistas de alimentación, simplemente eran sustituidas por vías al interior del plano. El resultado se muestra a continuación.

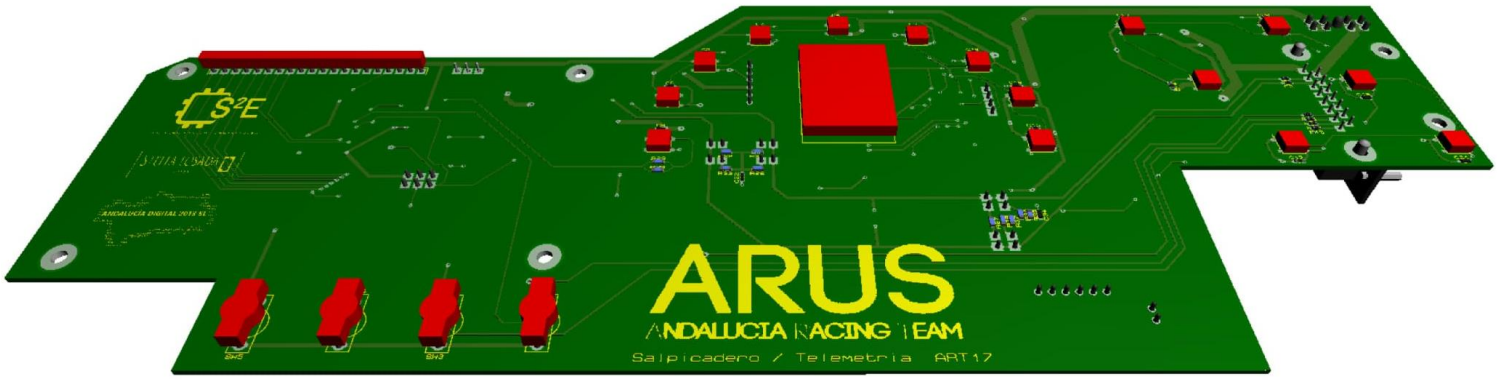


Ilustración 30. Vista frontal del diseño 3D de la PCB.

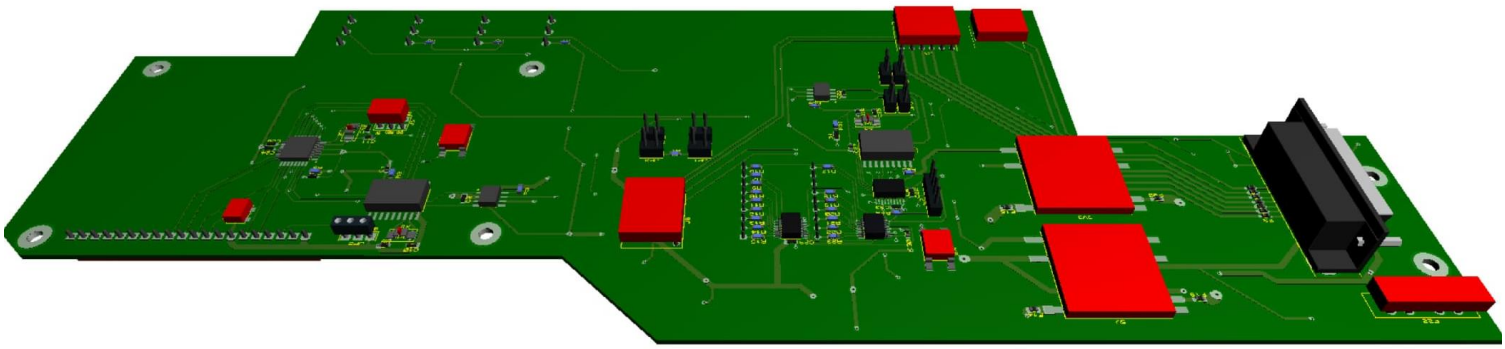


Ilustración 31. Vista trasera del diseño 3D de la PCB.

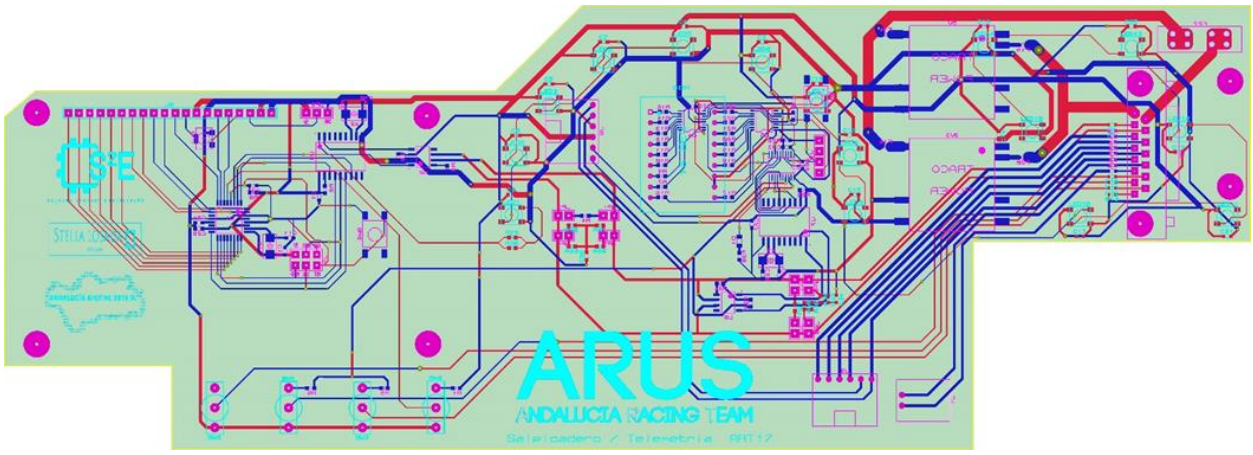


Ilustración 32. Diseño final de PCB a dos capas.

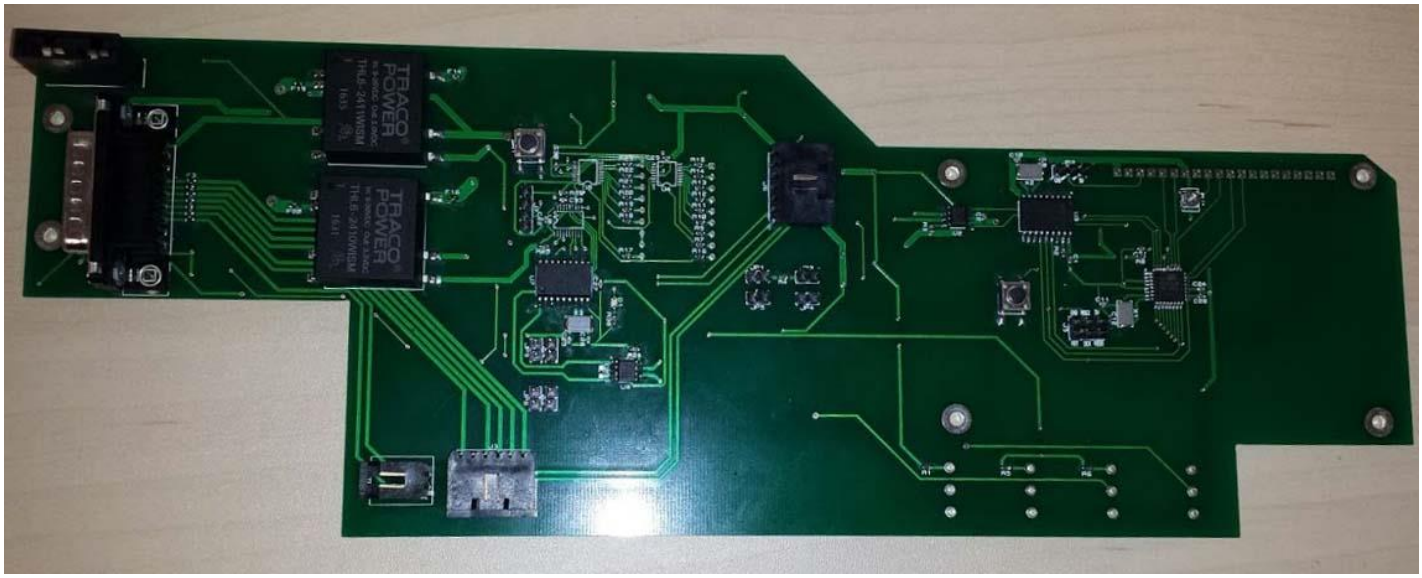


Ilustración 33. Fotografía de la PCB con componentes soldados.

4 SOFTWARE Y PROGRAMAS

El auténtico genio consiste en la capacidad para evaluar información incierta, aleatoria y contradictoria.

- Winston Churchill -

En este apartado se pretende analizar en profundidad los programas utilizados, la configuración de programación y los códigos utilizados. Para ello se explicarán las librerías utilizadas, se expondrán extractos de los códigos y ejemplos de las funcionalidades.

4.1. Code Composer Studio 6.2.0

Code Composer Studio (CCS) es un entorno de desarrollo integrado (IDE) que admite la serie de microcontroladores y procesadores embebidos de Texas Instruments. Incluye un conjunto de herramientas utilizadas para desarrollar y depurar aplicaciones embebidas. Posee un compilador C / C++ optimizado, un editor de código fuente, entorno de creación de proyectos, depurador, generador de perfiles y muchas otras características. Code Composer Studio combina las ventajas del software Eclipse con capacidades avanzadas de depuración integradas de TI, resultando en un entorno de desarrollo atractivo.

Este programa se ha compilado utilizando la versión de compilador TI v15.12.3.LTS, que se podrá elegir al crear un nuevo proyecto. Se seleccionará el microcontrolador MSP430G2553 y un proyecto vacío con el archivo main.c, puesto que así creará una pequeña plantilla para comenzar.

Se describirán las librerías necesarias para el funcionamiento del código principal, una librería para el MCP2515 creada por K. Evangelos para www.electrodummies.net[1], una librería para la UART creada por el profesor D. Manolo Perales utilizada en prácticas y por último una para el registro de desplazamiento escrita para facilitar el código.

4.1.1. Librería para el controlador del bus CAN (MCP2515)

Esta librería se compone de dos archivos, 'mcp2515.h' y 'mcp2515.c', que juntos componen las funciones necesarias para manejar el integrado MCP2515, responsable de la comunicación e interpretación de los datos del bus.

[1] Página web que ofrece tutoriales y librerías relacionadas con la utilización del LaunchPad MSP430. La librería de CAN utilizada se puede encontrar en el siguiente enlace: http://www.electrodummies.net/en/msp430-2/msp430-can-interface/

El creador de la librería posee una entrada en la página www.electrodummies.net, donde explica detalladamente el diseño de un shield de CAN para el LaunchPad MSP430. Aquí se detallarán las funciones que resultarán útiles para el proyecto de telemetría, y las configuraciones necesarias para cada tipo de microcontrolador y velocidades.

Listado de funciones disponibles

MCP2515_SPI_init()	Inicializa la interfaz SPI del microcontrolador.
MCP2515_SPI_transmit()	Envía y recibe mensajes a través de la interfaz SPI.
MCP2515_spi_test()	Comprueba si la comunicación entre microcontrolador y MCP2515 es correcta. Si no lo es devuelve 1, de lo contrario devuelve 0.
MCP2515_reset()	Reset por software del MCP2515. El pin manual de reset estará siempre en alto, por lo que esta es la única forma de reiniciar el controlador.
MCP2515_CanVariable_init()	Inicializa las variables de estructura can_t con valores iniciales.
MCP2515_init()	Inicializa el MCP2515 con datos como velocidad, filtros y canales de recepción y envío de datos.
MCP2515_bit_modify()	Sólo se configuran ciertos bits con esta función. El bit correspondiente depende de la máscara, los bits en alto de la máscara serán modificados.
MCP2515_write()	Escribe vía SPI en un registro simple del MCP2515.
MCP2515_write_many_registers()	Para escribir múltiples registros en el MCP2515
MCP2515_write_id()	Describe la ID de la trama CAN que será enviada.
MCP2515_read()	Lee un registro simple.
MCP2515_read_many_registers()	Lee múltiples registros del MCP2515
MCP2515_read_id()	Lee la ID de la trama CAN que ha sido recibida.
MCP2515_can_tx0()	Para enviar una trama CAN (ID, RTR y datos) en el canal 0 de transmisión.
MCP2515_can_tx1()	Para enviar una trama CAN (ID, RTR y datos) en el canal 1 de transmisión.
MCP2515_can_tx2()	Para enviar una trama CAN (ID, RTR y datos) en el canal 2 de transmisión.
MCP2515_can_rx0()	Para recibir una trama CAN (ID, RTR y datos) en el canal 0 de recepción.
MCP2515_can_rx1()	Para recibir una trama CAN (ID, RTR y datos) en el canal 0 de recepción.

Tabla 5. Funciones disponibles en la librería del MCP2515.

Configuración necesaria

Primeramente, se debe configurar la función MCP2515_SPI_init() que, inicialmente, está preparada para el microcontrolador MSP430G2553. Si se utilizara otro, habría que cambiar los pines físicos del puerto SPI, las direcciones de lectura y de escritura. Posteriormente se deberá cambiar las definiciones de MCP2515_CS_LOW y MCP2515_CS_HIGH acorde con el diseño que se haya realizado. En este caso, el Chip Select del MCP2515 se ha conectado al pin P2.0, por lo que MCP2515_CS_LOW será <P2OUT &=~ BIT0> y MCP2515_CS_HIGH será <P2OUT |= BIT0>. Estas definiciones se encontrarán en la cabecera mcp2515.h.

Una vez configuradas estas opciones, la librería estará preparada para funcionar con el controlador MCP2515 con la siguiente configuración por defecto:

- Velocidad del bus: 250 kbit/s
- Canales de recepción RX0 y RX1 activados
- Envío disponible solo por el canal TX0
- Deshabilitar RXnBF (previene que se active una interrupción no deseada)
- Deshabilitados pines de RTS (previene que se active una interrupción no deseada)
- Permitir interrupciones del pin INT.

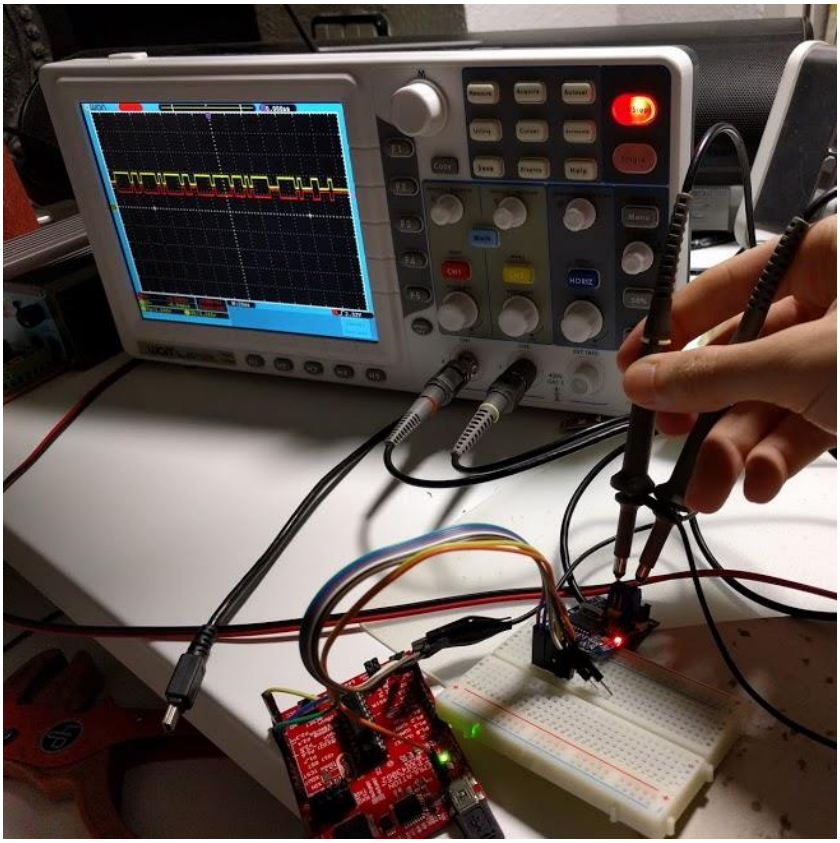


Ilustración 34. Comprobación de la librería en MSP430 con módulo de bus CAN.

Cambiar la velocidad del bus no es algo trivial, el controlador MCP2515 posee tres registros de tiempos donde se tendrá que establecer la velocidad adecuada según la velocidad establecida por el reloj externo y la velocidad de datos deseada.

Existen numerosas herramientas que calculan dichos tiempos, y al no ser tiempos exactos (existen tolerancias), según la aplicación se pueden conseguir valores diferentes pero compatibles.

La herramienta Microchip CAN Bit Timing Calculator está destinada a este fin. Se puede establecer para una frecuencia del reloj y velocidad del bus, el retraso de propagación, los segmentos necesarios para la fase 1 (antes de leer dato) y fase 2 (después de leer dato) y el ancho de sincronización en el muestreo. Posteriormente generará un informe e indicará los valores de los registros CNF1, CNF2 y CNF3 del MCP2515.

Se intentaron numerosas configuraciones sin éxito, e incluso se compararon resultados con otras herramientas. Debido a los valores tan dispares que se obtenían, se optó por utilizar una tabla predefinida en una librería preparada para Arduino, donde según la velocidad deseada existían unas definiciones para los valores de los tres registros. A continuación, se muestran los valores más comunes:

	MCP_8MHz	**MCP_16MHz**
CAN 1000 kBPS	CNF1 (0x00) CNF2 (0x80) CNF3 (0x00)	CNF1 (0x00) CNF2 (0xD0) CNF3 (0x82)
CAN 500 kBPS	CNF1 (0x00) CNF2 (0x90) CNF3 (0x02)	CNF1 (0x00) CNF2 (0xF0) CNF3 (0x86)
CAN 250 kBPS	CNF1 (0x00) CNF2 (0XB1) CNF3 (0x05)	CNF1 (0x41) CNF2 (0XF1) CNF3 (0x85)
CAN 100 kBPS	CNF1 (0x01) CNF2 (0xB4) CNF3 (0x06)	CNF1 (0x03) CNF2 (0xFA) CNF3 (0x87)

Tabla 6. Valores de temporización más comunes para el bus CAN.

Todos los valores anteriormente indicados para un oscilador de 16MHz han sido comprobados y funciona perfectamente. Se han probado tanto con dispositivos equivalentes utilizando la misma librería como con el protocolo integrado del microcontrolador Tiva o incluso con el shield preparado para Arduino. Para cambiarlo, habrá que dirigirse a la función MCP2515_init() en mcp2515.c. Aquí aparecerán tres líneas de escritura (MCP2515_write(MCP2515_CNF1,0x00)) donde se indica el registro y el valor.

4.1.2. Librería para la utilización de la UART

Esta librería, como se mencionó anteriormente, se ha utilizado en prácticas de microcontroladores en la universidad. A continuación, se detallarán las funciones de mayor utilidad para el proyecto. Puesto que solo se utilizará para enviar información, se prescindirá de la parte dedicada a recibir datos por complicar el código.

Tabla de funciones

UARTinit()	Configuración de la UART para pines P1.1 y P1.2.
UARTprintc()	Enviará un caracter cuando el canal TX esté libre.
UARTprint()	Enviará un conjunto de caracteres utilizando la función anterior.
UARTprintCR()	Enviará un conjunto de caracteres añadiendo un retorno de carro al final.
UARTnum()	Traducirá un número entero en caracteres y los enviará.

Tabla 7. Funciones disponibles en la librería de la UART.

Configuración necesaria

En esta librería el único parámetro relevante que se tendrá que configurar es la velocidad de baudios de la UART. Para ello, el microcontrolador posee dos registros, UCA0BR0 y UCA0BR1, que controlarán la tasa de baudios de la USCI A0.

Para realizar la modificación habrá que editar la función UARTinit() dentro del archivo UART.c. La configuración de los baudios depende de la velocidad a la que se haya establecido el reloj interno del microcontrolador, y se calcula de la siguiente forma:

$$\frac{Freq.SCLK\ (Hz)}{(BR1 * 256 + BR0)} = Baud\ Rate$$

De esta forma, si se tiene una frecuencia principal de 1MHz y se quiere obtener una velocidad de 9600 baudios, habrá que realizar la siguiente operación:

$$\frac{1000000}{9600} = (BR1 * 256 + BR0) \sim (0 * 256 + 104)$$

$$UCA0BR0 \rightarrow 104$$

$$UCA0BR1 \rightarrow 0$$

$$\frac{1000000}{0 * 256 + 104} = 9615.38\ baudios$$

Cabe destacar que la temporización no será exacta, puesto que en el ejemplo anterior se obtendría una velocidad de 9612.38 baudios. Si el receptor está configurado para recibir a 9600 baudios, el pequeño error que se produce se irá acumulando y desfasando el dato, ocasionando una posible pérdida en cierto momento.

Para la comunicación con los módulos se utilizó una velocidad de 19200 baudios, que corresponde con los valores 52 y 0 para los registros UCA0BR0 y UCA0BR1 respectivamente.

4.1.3. Librería para la utilización del Shift Register (74HC595)

Esta librería ha sido creada para conseguir reducir el código principal y hacer así más legible y fácil de explicar su función. En la cabecera shift.h se tienen las definiciones de los pines latch, dato y reloj, necesarios para el encapsulado. En shift.c se encuentran las funciones que se describirán a continuación y una tabla donde se traducen los números a los distintos segmentos que deben ser activados para su representación en el display.

Tabla de funciones

shiftInit()	Inicializa los pines necesarios como salida.
shiftWrite()	Escribe el número indicado en el display.
pinWrite2()	Escribe un valor en el pin indicado.
pulseClock2()	Realiza un pulso en el pin del reloj.
pulseLatch2()	Realiza un pulso en el pin de latch.
shiftOut2()	Desplaza los 8 bits indicados desde el más significativo hasta el menos.

Tabla 8. Funciones disponibles en la librería para el registro de desplazamientos.

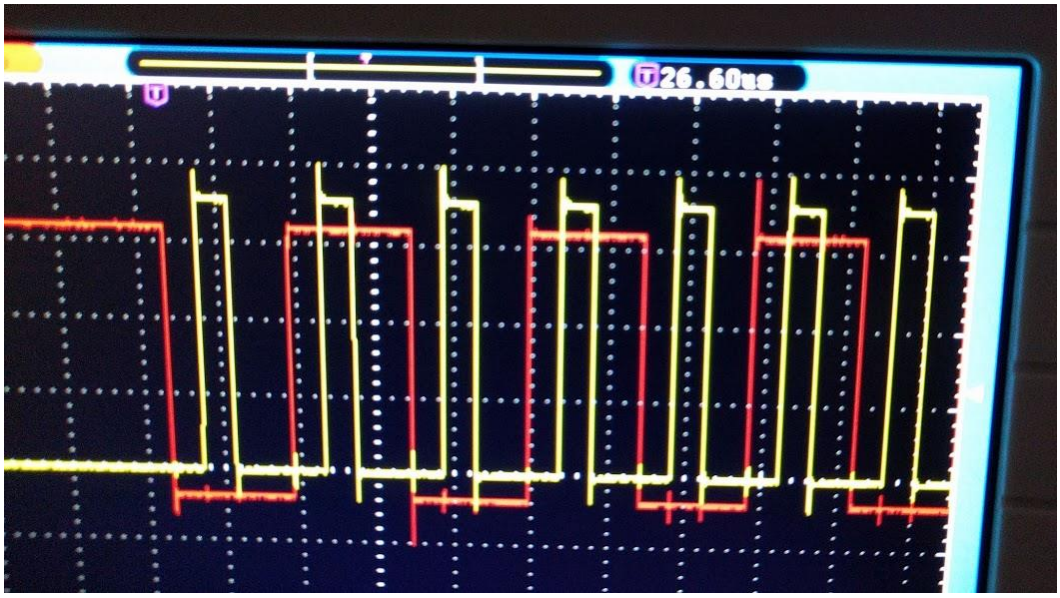


Ilustración 35. Comprobación de pulsos generados con la librería de datos y reloj.

Configuración necesaria

Tan solo se tendrán que modificar las definiciones de los pines de dato, latch y reloj según la configuración optada. Dichas definiciones se encuentran en shift.h. También se deberá cambiar la tabla gear_vector[] de shift.c para activar los distintos segmentos según se quieran añadir nuevas representaciones. Cabe destacar que, al ser el display de ánodo común, se activarán poniendo el pin a 0.

Al ser un display de 16 segmentos, se utilizarán 2 registros de desplazamiento de 8 bits cada uno, siendo el caracter N (punto muerto) el siguiente mostrado, acorde con la relación entre los segmentos y los pines conectados.

```
char gear_vector[] = { 0b11010011, 0b11001110, …};
                       Q15 …                 … Q0
```

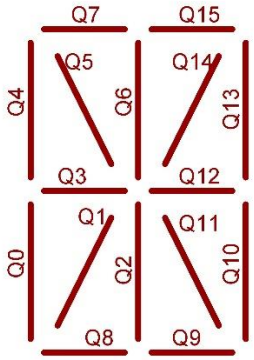


Ilustración 36. Relación entre segmentos y pines del registro.

4.1.4. Explicación del Código principal

El código consta de tres partes bien diferenciadas. La primera trata de configurar todos los periféricos necesarios que se listarán en el apartado 4.1.4.2. Posteriormente se leerán tramas del bus CAN siempre que estén disponibles. Existirá un pin de interrupción que habilita el controlador y, cuando el microcontrolador detecte dicha interrupción, cambiará la bandera para indicar que existe un nuevo mensaje. Por último, se enviarán los datos por UART hacia el módulo de telemetría, siguiendo un protocolo simple con distintas temporizaciones para separar los datos con más tasas de muestras que otros.

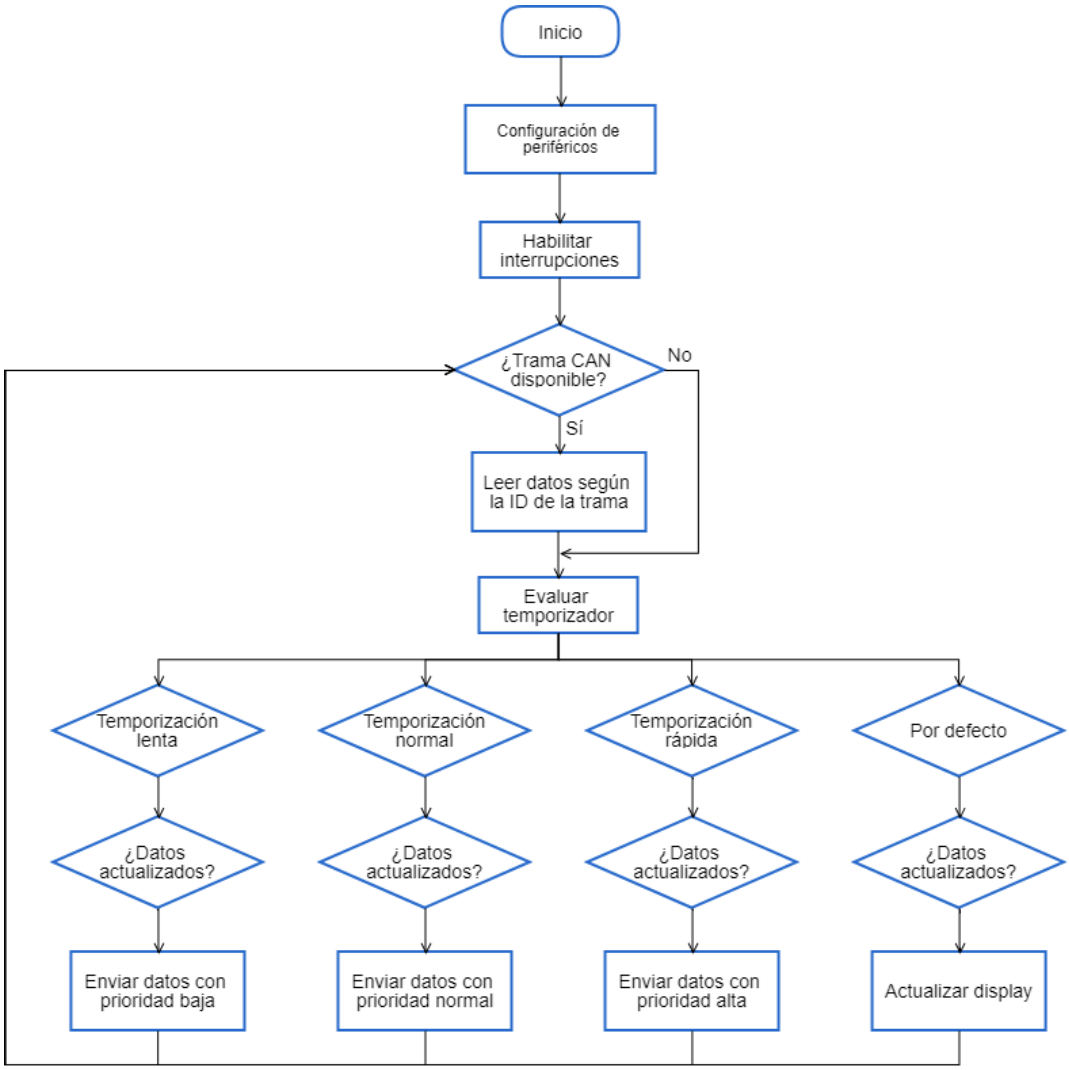


Ilustración 37. Diagrama de flujo del código principal.

4.1.4.1. Variables y definiciones

En este apartado se indicarán las declaraciones necesarias incluidas en el código. Incluye definiciones de las ID de los distintos sistemas enlazados y los índices de los vectores. Las variables necesarias se declararon según la necesidad por tamaño; si la variable necesita 2 bytes de la trama, se guardará en un entero, si por el contrario es de 1 byte, se utilizará una variable tipo char.

```
// --- Definitions ---
#define ERROR       10     // ID for CAN
#define ECU1        70     // ID for CAN
#define ECU2        40     // ID for CAN
#define POWER       50     // ID for CAN
#define ADQ1        30     // ID for CAN
#define ADQ1_RE     20     // ID for CAN
#define ADQ2        80     // ID for CAN
#define ADQ3        90     // ID for CAN
#define DBOARD      0xA0   // ID for CAN
#define CLUTCH      0xA1   // ID for CAN
```

```
#define _ECU1       0      // Index for vector
#define _ECU2       3      // Index for vector
#define _POWER      6      // Index for vector
#define _ADQ1       9      // Index for vector
#define _ADQ2       12     // Index for vector
#define _ADQ3       15     // Index for vector
#define _DBOARD     18     // Index for vector
#define _CLUTCH     21     // Index for vector
#define FAST 0
#define MED         1
#define SLOW 2

#define N_SYSTEMS   8      // Number of systems for the array

// --- Declare variables ---
char new_can_data=0;       // Flag to indicate a new unprocessed frame
int discarded_id=0;        // Frame discarded
char data_updated[N_SYSTEMS*3]; // Vector of flags to see if data is updated
int timer=0;         // Increases in 100ms
char i;
char fast_send=0,          // Flags to send data with different speeds
     medium_send=0,
     slow_send=0;

int revs=0;          // Revolutions from ECU
int    speed=0;      // Speed from ECU
char gear=7;         // Gear from ECU
int lambda=0;        // Lambda from ECU
int tp=0;            // TP from ECU
char ect=0;          // Cooler temperature from ECU2
int map=0;           // Manifold absolute pressure from ECU
char iat=0;          // Intake air temp. from ECU2
char int1=0;         // from Power
char int2=0;         // from Power
char int3=0;         // from Power
char load=0;         // from Power
char voltage=0;      // from Power

int  oil_press=0;    // from ADQ1
char oil_temp=0;     // from ADQ1
char water_temp=0;   // from ADQ1
char cockpit=0;      // from ADQ1
char brake_front=0;  // from ADQ1
char brake_rear=0;   // from ADQ1
char st_wheel=0;     // from ADQ2
char accelX=0;       // from ADQ2
char accelY=0;       // from ADQ2
char accelZ=0;       // from ADQ2
char extFR=0;        // from ADQ2
char extFL=0;        // from ADQ2
char extRR=0;        // from ADQ2

int wsFR=0;          // from ADQ3
int wsFL=0;          // from ADQ3
int wsRR=0;          // from ADQ3
int wsRL=0;          // from ADQ3

can_t can_tx;              // CAN Send variable
can_t can_rx;              // CAN Receive variable
```

4.1.4.2. Configuración de los periféricos

Watchdog Timer

Lo primero que se debe hacer antes de empezar a configurar el microcontrolador es parar el watchdog timer (WDT). Esto es un temporizador que tiene el dispositivo por seguridad. El propósito es producir un reinicio del dispositivo cada cierto periodo de tiempo, para así evitar que entre en un lazo infinito o en una espera muy prolongada. Esto puede evitarse reiniciando la cuenta antes del final de su periodo o extendiendo el tiempo con preescaladores. La desactivación se realiza con el siguiente registro:

```
WDTCTL = WDTPW + WDTHOLD; // Stop Watchdog-Timer
```

Cabe destacar que, si se utilizaran programas como Energía, esta acción no es posible, ya que el software utiliza dicho temporizador para establecer funciones que tengan que ver con el tiempo.

Reloj del sistema

El reloj del sistema se establecerá en 1MHz, pudiendo aumentar hasta 16MHz siempre que se reconfiguren los tiempos de la UART y de posibles temporizadores que dependan del mismo.

```
if (CALBC1_1MHZ==0xFF) while(1);
DCOCTL = 0;  // Take the lowest frequency, DCOx and MODx settings
BCSCTL1 = CALBC1_1MHZ;
DCOCTL = CALDCO_1MHZ;
```

Primeramente, se borra la configuración actual del reloj, y seguidamente se establece la velocidad en los registros BCSCTL1 y DCOCTL. Las máscaras están incluidas en la librería que proporciona el compilador al seleccionar el microcontrolador adecuado.

Configuración de puertos

En este apartado se explicará la configuración de pines llevada a cabo, los periféricos activados y las interrupciones configuradas. El pin del LED se establecerá como salida y en alto, para encenderse inmediatamente después de iniciar el microcontrolador. Se apagará el indicador tras finalizar la configuración.

```
P2DIR |= BIT2;      // Set P2.2 (led) to output
P2OUT |= BIT2;      // set P2.2 (led) to On

(...)

P2OUT &= ~BIT2;     // set P2.2 (led) to Off
```

Para configurar los pines del registro de desplazamiento se utilizará la función de librería shiftInit() mencionada anteriormente.

```
// Configure shift pins
void shiftInit(void)
{
      P2DIR |= (DATA + CLOCK + LATCH);// Set pins P2.3/4/5 as output
}
```

Se utilizará un temporizador con una interrupción habilitada cada 100 Hz utilizando una división de 8 en el reloj del sistema y con cuenta ascendente. La configuración e interrupción se muestra a continuación:

```
CCTL0 = CCIE;                           // CCR0 interrupt enabled
TACTL = TASSEL_2 + MC_1 + ID_3;         // SMCLK/8, upmode
CCR0 =  1250;                           // 100 Hz
```

Con una frecuencia de 100 Hz se ejecutará la interrupción, que consiste en activar unas banderas para realizar operaciones en el bucle principal. Se ha escalado el temporizador de forma que las banderas se activen con frecuencias distintas.

```
// Timer A0 interrupt service routine
#pragma vector=TIMER0_A0_VECTOR
__interrupt void Timer_A (void)
{
      timer++;
      if (timer%2==0)     fast_send=1;        // 100Hz/2 = 50Hz
      if (timer%10==0)    medium_send=1;      // 100Hz/10 = 10Hz
      if (timer%100==0){  slow_send=1;        // 100Hz/100 = 1Hz
            timer=0;
      }
}
```

La configuración de la UART se realizará con otra función de librería, donde estará establecido a una velocidad aproximada de 19200 baudios, en los pines P1.1 (RXD) y P1.2 (TXD).

```
void UARTinit(void){
       P1SEL       |= BIT1 | BIT2;     //P1.1, P1.2 para la UART
       P1SEL2      |= BIT1 | BIT2;

       UCA0CTL1 |= UCSWRST;            // Reset uart para configurar
       UCA0CTL1 = UCSSEL_2 | UCSWRST; // Conf. como asincrono
     UCA0MCTL = UCBRF_0 | UCBRS_1;     // Modulacion
    if( BCSCTL1 == CALBC1_16MHZ )
      {
      UCA0BR0 = 65;       // 16MHz/(3*256+65)=19207bps
      UCA0BR1 = 3;
      }
      else
      {
      UCA0BR0 = 52;       // 1MHz/(3*0+52)=19230bps
      UCA0BR1 = 0;
      }
     UCA0CTL1 &= ~UCSWRST;             // Quitar Reset
     IFG2 &= ~(UCA0RXIFG);             // Borrar flag
     IE2 &=~ (UCA0RXIE+UCA0TXIE);      // Deshabilitar las ints.
}
```

La inicialización del controlador del bus CAN consta de 4 partes. Primeramente, se configurará el pon P1.4 como interrupción en el microcontrolador. Posteriormente se iniciará la comunicación por SPI con el controlador y, una vez establecida correctamente, se procederá a iniciar el MCP2515 con la configuración adecuada. Por último, se inicializarán las variables que recogerán las tramas del bus.

```
// --- Configure pin P1.4 for can interrupt --- //
P1DIR &= ~BIT4;     // P1.4 input
P1REN |= BIT4;      // P1.4 enable resistor
P1OUT |= BIT4;      // P1.4 pull-up
P1IE  |= BIT4;      // P1.4 Activate interrupt (only for this pin)
P1IES |= BIT4;      // P1.4 Hi/lo Edge
P1IFG &= ~BIT4;     // P1.4 IFG Cleared

// --- Init SPI, MCP2515 and CAN-Variables --- //

MCP2515_SPI_init();       // Initialize SPI
MCP2515_init();           // Initialize MCP2515

MCP2515_CanVariable_init (&can_tx);    // Initialize Send variable
MCP2515_CanVariable_init (&can_rx);    // Initialize Receive variable

__enable_interrupt();           // Enable interrupts

// Port 1: Interrupt-Service-Routine
#pragma vector=PORT1_VECTOR
__interrupt void Port_1(void)
{
      if (P1IFG & BIT4){
            P1IFG &= ~BIT4;
            new_can_data = 1;
      }
}
```

4.1.4.3. Bucle principal

El bucle principal tiene la misión de leer e interpretar una trama del bus CAN siempre que esté disponible (indicada por la bandera de la interrupción), que leerá la ID para saber de quién procede y, acorde con el protocolo preestablecido para el resto de sistemas, se interpretarán de una forma u otra los 8 bytes de datos. También se activará el aviso correspondiente para indicar que los datos de un sistema han sido actualizados.

```
// Read new frame of canbus //
if (new_can_data){        // Check if new data arrived
      new_can_data = 0;               // Put flag down
      MCP2515_can_rx0(&can_rx);       // Read buffer frame
      P2OUT ^= BIT2;                  // Toggle P2.2 (led)
      switch (can_rx.COB_ID){         // Compare CAN ID
      case ECU1: // 70
            revs   = can_rx.data[0] * 256 + can_rx.data[1];
            ect    = can_rx.data[2];
            lambda = can_rx.data[3] * 256 + can_rx.data[4];
            tp           = can_rx.data[5];
            oil_press    = can_rx.data[6] * 256 + can_rx.data[7];
            data_updated[_ECU1+FAST] = 1;
            data_updated[_ECU1+MED] = 1;
            data_updated[_ECU1+SLOW] = 1;
                  break;

      (...)

}
```

Una vez guardados los datos en la memoria RAM del microcontrolador, se enviarán a través de la UART. Para ello, en el código se ejecutará la comprobación de si es el momento de enviar los datos o no (con las banderas del temporizador). Existen tres frecuencias distintas, baja, media y alta. La más baja envía un mensaje cada segundo, y estará destinada a datos que no necesiten tanta velocidad, como las temperaturas o la carga de la batería. La velocidad media envía 10 mensajes por segundo, y la alta asciende a 50.

Si se corresponde con la división del temporizador, se comprobará seguidamente si existe un dato nuevo procedente de alguna ID. De ser así, enviará el dato correspondiente al sistema con la ID comprobada y eliminará la bandera para no repetir el mismo mensaje sin actualizar.

El protocolo de envío de mensajes consiste en anteponer un carácter alfanumérico antes del valor, para así poder reconocer su procedencia cuando se tenga que interpretar en LabVIEW.

```
// Send data to telemetry module //

if (slow_send){

      if (data_updated[_POWER+SLOW]){
            sendData('B',load);
            data_updated[_POWER+SLOW] = 0;
      }
      if (data_updated[_ADQ1+SLOW]){

            sendData('O',oil_temp);

            (...)

      }

      (...)

}
```

En este ejemplo se comprueba si está activada la bandera para enviar un mensaje lento, y posteriormente se comprueba si el dato de la batería que envía el sistema de potencia se ha actualizado. Si ambas condiciones se cumplen, se enviará a través de los módulos un mensaje similar a B90, para indicar un 90% de batería. Dicho mensaje se enviará cada segundo siempre que el sistema de potencia envíe una trama con el valor actualizado.

Cabe destacar que existe la posibilidad de que un sistema tenga datos que se tengan que enviar rápidamente y otros que no, por lo que se ha optado por añadir una bandera para cada velocidad en todos los sistemas.

Por último, el microcontrolador también tiene la obligación de actualizar el display de 16 segmentos de la siguiente forma:

```
if (data_updated[_ECU2]){
      shiftWrite(gear);  // Function to write in shift register from 0
}
```

Donde la marcha actual es recibida por la segunda trama de la ECU (Engine Control Unit). La función shiftWrite(int) se encuentra en la librería creada para este propósito.

```
// Writes the gear in display
void shiftWrite(unsigned char gear)
{
      shiftOut2(gear_vector[2*gear]);        // First part of digit
      shiftOut2(gear_vector[2*gear+1]);      // Second part of digit

      pulseLatch2();
}
```

Leerá del vector los 16 bits que tendrá que desplazar y seguidamente dará un pulso en latch para que los biestables de salida se activen y se pueda visualizar el número.

4.2. LabVIEW 2014

Para el desarrollo de este proyecto se ha utilizado la versión de LabVIEW 2014 con el módulo VISA (Virtual Instrument Software Architecture) instalado. Esto es un estándar para configurar, programar y solucionar problemas de sistemas que comprenden GPIB, VXI, PXI, Serial, Ethernet e interfaces USB. El módulo ofrece una interfaz de programación entre el dispositivo hardware y el entorno de desarrollo de LabVIEW. NI-VISA, la implementación de National Instrumens, incluye librerías, utilidades interactivas y configuraciones para los desarrollos.

Con esto se permitirá establecer una comunicación rápida y de fácil configuración entre el puerto serie reconocido por el sistema operativo Windows y el programa creado en el entorno de LabVIEW. Este complemento se puede descargar desde la página web oficial de forma gratuita.

También se podrá optar por descargar temas personalizados para LabVIEW, y así se podrá conseguir un entorno visual mucho más amigable que el que traerá por defecto el software. Una vez instalado todo lo necesario, se procederá a crear un proyecto simple con un archivo .vi que incluirá un panel frontal donde se podrá interactuar con el usuario y un diagrama de bloques, donde estará todo el programa en lenguaje visual gráfico.

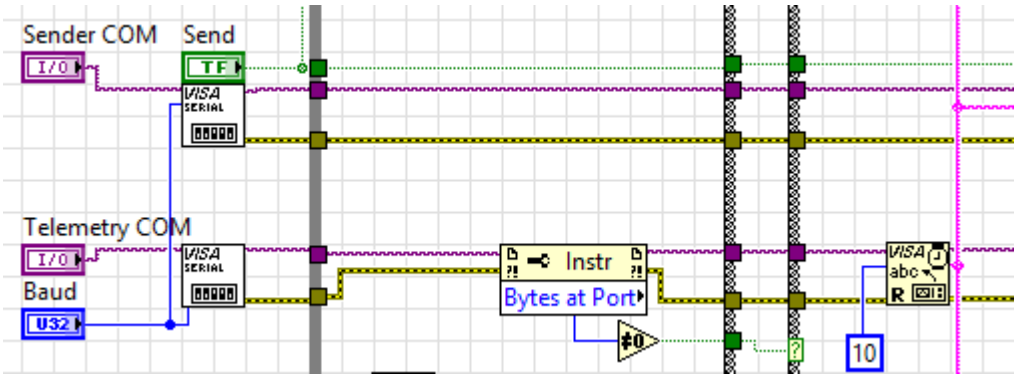
El programa consta de las siguientes funcionalidades:

- Elección del puerto de entrada y velocidad.
- Posibilidad de redireccionar la información leída a un puerto de salida.
- Grabación de los datos en un archivo de texto plano.
- Visualización de las variables y representación gráfica en el tiempo.
- Distintas pestañas para organizar el entorno.
- Indicador de datos desactualizados.

4.2.1. Lectura de datos de entrada

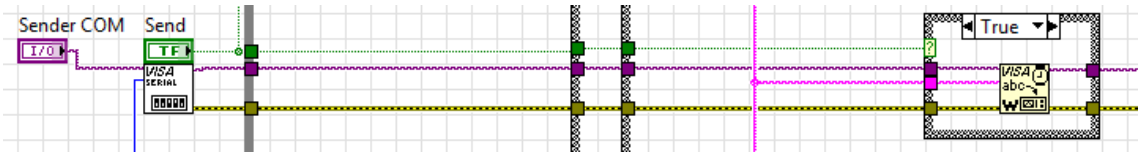
En la realización de esta parte del código se ha utilizado el módulo VISA que se mencionó anteriormente. Este es capaz de proporcionar una comunicación directa con el puerto serie que creará Windows al reconocer el dispositivo USB.

El módulo VISA recibirá como parámetros el puerto seleccionado y la velocidad en baudios. Tras ejecutar el programa establecerá la conexión y quedará abierta como recurso. Se ha utilizado un “Property Node” para consultar el número de bytes disponibles en el puerto, de forma que en su salida se comparará si es distinto de cero y se habilitará el siguiente paso. De esta forma, se evita que la computadora esté continuamente leyendo el puerto, siendo ejecutado sólo cuando se detecten datos en la comunicación. Seguidamente se utilizará un bloque de lectura, donde se seleccionará que siempre lea 10 bytes de datos, ya que nunca sobrepasarán este límite los mensajes enviados. Los caracteres leídos se conseguirán en forma de cadena por la conexión de color rosa que se puede apreciar en la imagen.



Cabe destacar que en un principio se hizo de forma que se leyeran tantos bytes de datos como hubiera detectado el nodo de propiedades. Los resultados no fueron los esperados, ya que a veces no se conseguía leer la totalidad del mensaje, y es por ello por lo que se decidió dejar un valor fijo de 10 bytes.

Para utilizar la opción de enviar los mismos datos por un puerto distinto, se utilizará el módulo VISA Write conectado a la línea donde se abrirá el puerto de salida. Dicha función estará dentro de una función 'case', que solo se activará si la señal Send está activada en el momento de ejecutar el programa. En el entorno gráfico esto se traduce en un selector de puertos y un switch de dos posiciones.



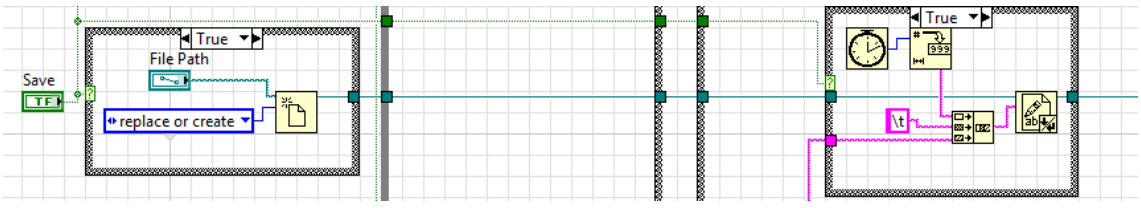
El módulo VISA Write recibe el identificador de la conexión, la línea de errores por si los hubiera y los datos ya leídos (línea rosa). Si no estuviera activada la variable Send, el cuadro True no tendría efecto.

Hay que tener cuidado con las conexiones abiertas entre LabVIEW y los puertos de Windows, teniendo que cerrarlas debidamente al finalizar el programa. Si por algún casual esto no pasara, se podría tener problemas al volver a conectar con el dispositivo, teniendo que reiniciar el sistema por completo.

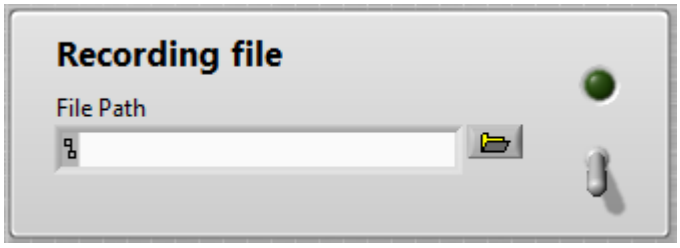
4.2.2. Escritura de datos en fichero

El entorno de LabVIEW tiene un espacio reservado para funciones que manejan ficheros de texto. Primeramente, se creará un archivo en la ruta que se deberá especificar en el entorno gráfico. Esto creará un ID de identificación de archivo (línea azul) que conectará con el módulo de escritura en el archivo.

Una vez más se tendrá que habilitar esta función con un switch en el entorno gráfico, que estará conectado con la variable Save. Si está activada y el fichero se ha creado correctamente, guardará en el archivo cada mensaje llegado del puerto de entrada (línea rosa) junto con el tiempo actual del sistema. La trama será creada con el módulo Concatenate Strings, que permitirá dar un pequeño formato a los datos guardados.



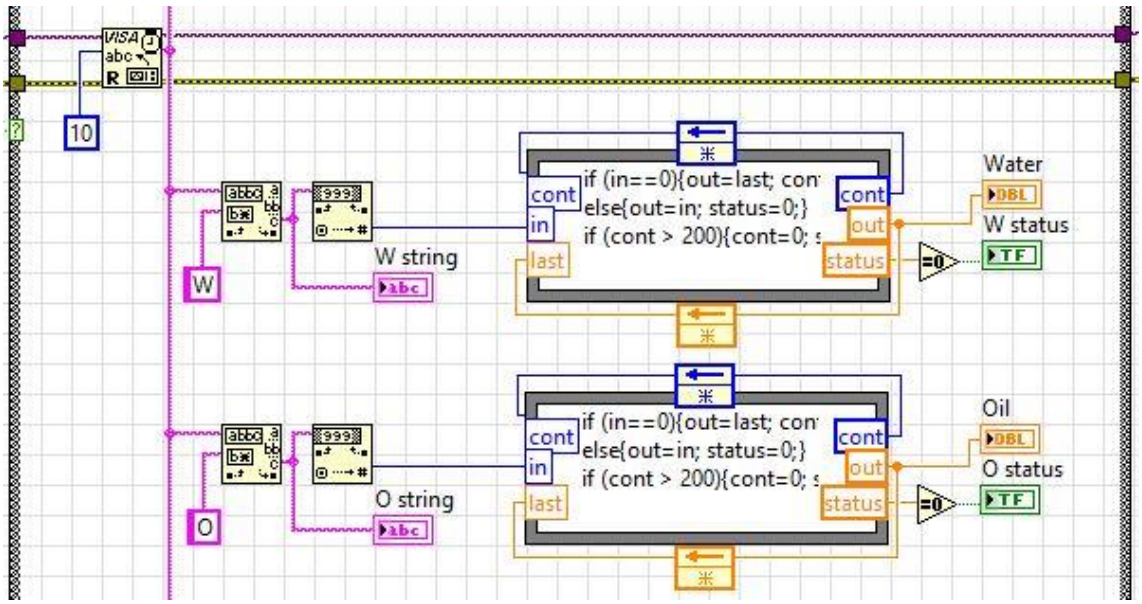
Como se tenía que hacer en la conexión del puerto serie, habrá que procurar cerrar debidamente el archivo al finalizar el programa. Esto se realizará con el módulo Close File. Si no se cerrara correctamente, podría quedar ilegible.



4.2.3. Interpretación de los datos

Como se explicó anteriormente, los datos van precedidos por unas letras indentificativas. Por ello, nada más leer la trama por el puerto serie, se comparará en un bloque Match Pattern. El funcionamiento es el siguiente: este bloque comparará la trama introducida con una expresión, si coincide, podrá devolver los caracteres anteriores al patrón, el propio patrón o los caracteres siguientes. En esta aplicación se utilizará el último caso. De forma que, si el dato es R143 y la expresión para comparar es R, devolverá los caracteres 143.

Si no encontrara coincidencias, devolverá una trama vacía. Posteriormente se utilizará un bloque Decimal String To Number para transformar el valor a número entero. Aquí puede surgir un problema, ya que, al convertir una cadena vacía a entero, el valor mostrado es cero. Esto puede llegar a causar confusión en la representación porque podrá falsear las medidas. La solución a lo anterior se antoja fácil, y se explicará a continuación.



Se utilizará una caja de Formula Node, que permite ejecutar un código secuencial mientras que el programa sigue ejecutándose de forma paralela. Aquí se implementarán las conversiones oportunas de las unidades de forma sencilla y el método para detectar valores no actualizados. Dicho método consiste en añadir un contador que, si lee durante 200 ciclos el valor cero (no hay dato), activará la señal status, conectada a un indicador en la parte visual que cambiará a rojo. Para utilizar dicho código, fueron necesarios dos nodos de realimentación para mantener las variables, ya que se reiniciarían cada vez que se ejecutara de no tenerlos.

Si todo funciona correctamente, la salida será la entrada con los cálculos necesarios para convertir las unidades. Si por el contrario no se reciben datos, gracias a este bloque se mantendrá el valor anterior.

Cabe acentuar que el cero al significar que no hay dato, puede entrar en conflicto cuando realmente quiera enviarse un cero, ya que se ignorará y entrará en marcha el contador, indicando un valor desactualizado cuando realmente es cero. Esto pasaba en el indicador de marchas cuando el coche estaba en punto muerto. Para solventarlo, se sumó una unidad al valor en el microcontrolador para posteriormente restarlo en LabVIEW para su representación.

En el microcontrolador se enviaban los datos de la siguiente forma:

`sendData('G',gear+1);` y la salida en LabVIEW se tomaba como `out=in-1.`

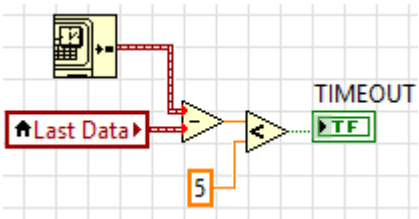
Esta misma estrategia, sencilla pero eficaz, también se utilizó para enviar decimales, multiplicando previamente para después dividir el número.

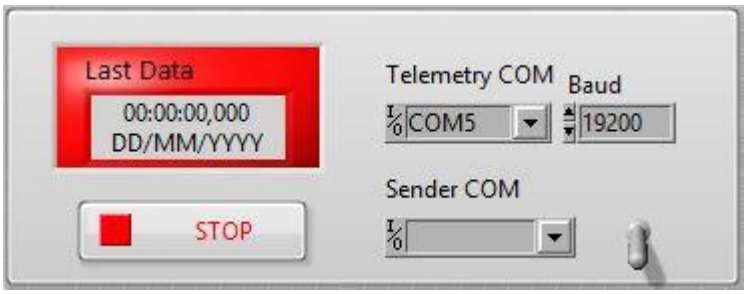
4.2.4. Otras estrategias

Se implementó a modo de indicador de correcto funcionamiento un reloj que se actualizara cada vez que llegara un dato. Comparando la última hora con la hora actual se podía conocer el tiempo que estuviera sin encontrar señal, activando un indicador si fuera mayor de 5 segundos.

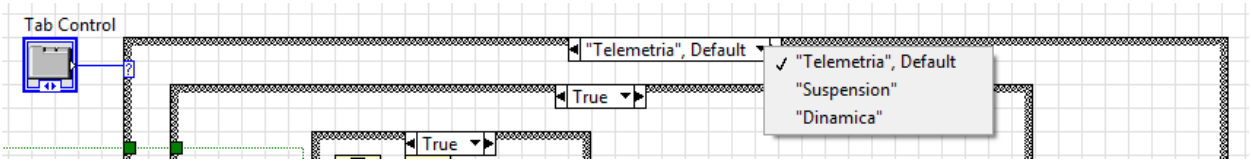
Para ello se creó una variable de tiempo Last Data que, con el bloque Get Date/Time In Seconds en el bloque case donde se leen los valores, almacenaba la hora actual en formato de Tiempo Unix. De esta forma, si no llegaban datos, no llegaba a ejecutarse dicha asignación permaneciendo la última hora registrada.

Se contrastará el valor con la hora actual para determinar si se ha perdido la conexión o no. En el panel de control se verá reflejada la hora del último mensaje llegado, mostrando el cuadro de color rojo si no se recibieran mensajes en 5 segundos, o en verde si todos los valores estuvieran actualizados.





Para saciar las necesidades de los distintos departamentos, se tuvieron que representar los datos más relevantes de las distintas partes del coche. Para ello, se decidió utilizar un bloque Tab Control en el programa que permite añadir pestañas en una misma ventana. En el diagrama de bloques se ve reflejado en una estructura Case donde cada caso tendrá asociada una representación.



Primeramente, se posee una pestaña general donde se mostrarán los datos más relevantes del monoplaza durante la carrera, con datos como las revoluciones o las temperaturas generales.

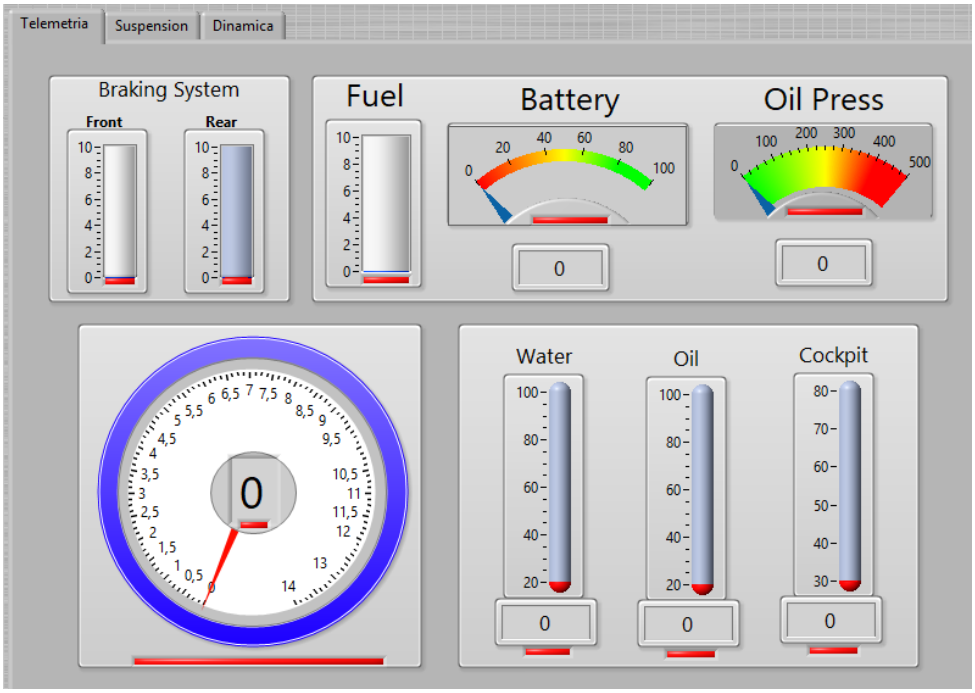


Ilustración 38. Pestaña general de la telemetría.

Se tienen dos pestañas más específicas, una para suspensión y otra para corroborar modelos de dinámica. En esta primera se muestran datos como velocidades de las ruedas o gráficas de los extensómetros situados en los amortiguadores y posición del volante. Así se podría detectar cualquier tipo de subviraje o sobreviraje en la dirección.

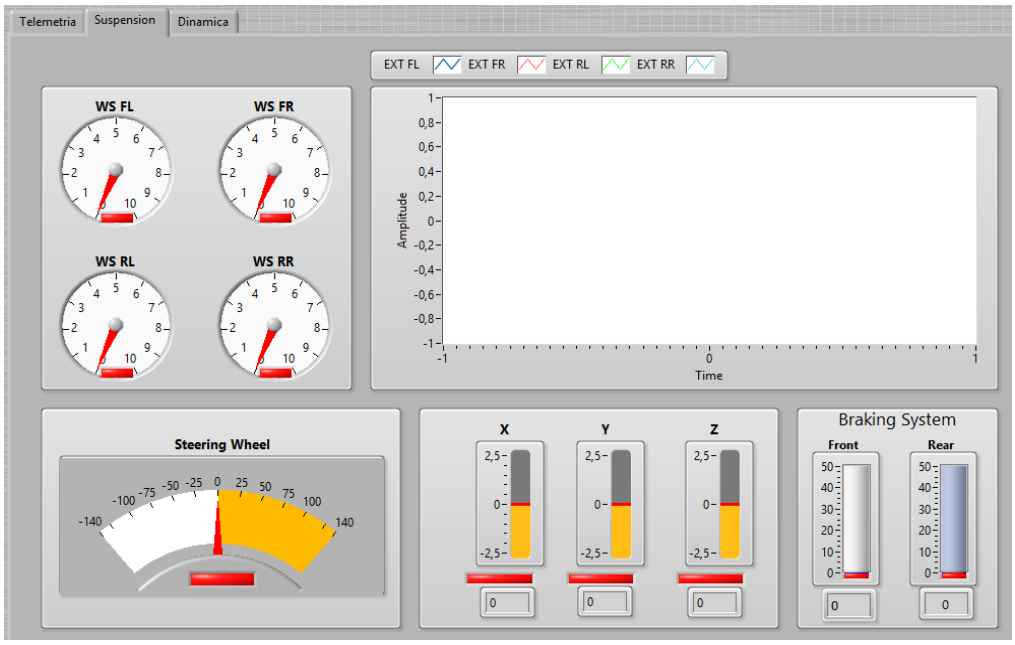


Ilustración 39. Interfaz gráfica de telemetría para suspensión.

En la pestaña de dinámica aparecerán gráficas de todos los datos relevantes para la dinámica del vehículo, como son los siguientes: presión de frenada, aceleraciones, revoluciones y velocidad, medidas del acelerómetro en los distintos ejes. Comparando todos estos datos se podrá verificar el correcto comportamiento del monoplaza en general.

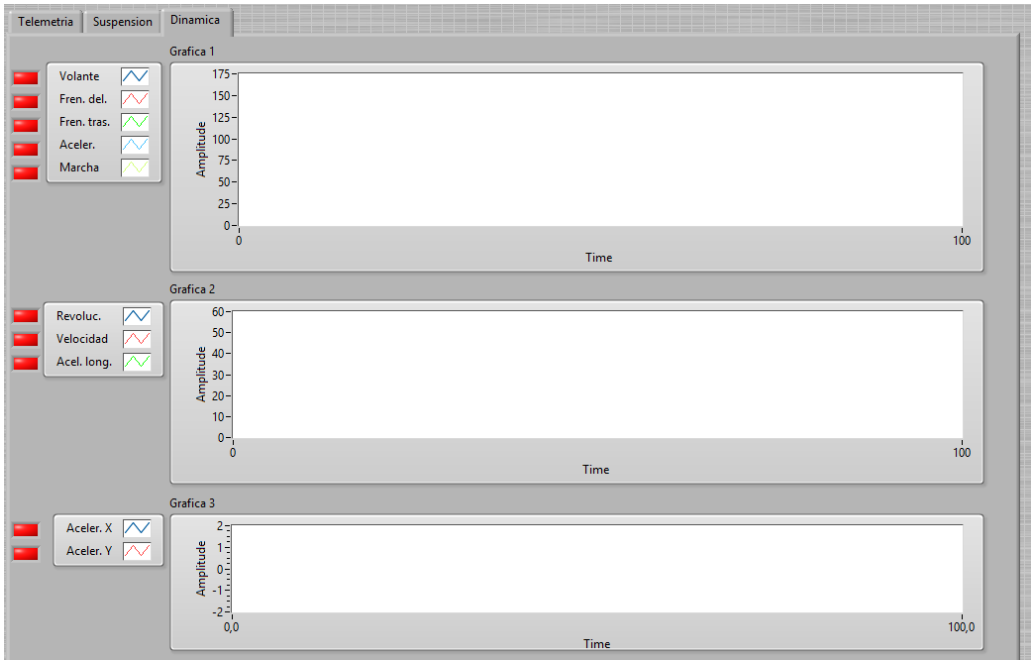


Ilustración 40. Interfaz gráfica de telemetría para dinámica.

4.3. 3DR Radio Config 1.3.2

Por último, se intentará abordar en detalles la configuración de los módulos de telemetría. Para ello, lo primero que se tendrá que conseguir es el software especializado. En este caso se utilizará 3DRRadio Config 1.3.2 – by Michael Oborne.

Es muy común encontrar dicho software integrado en la plataforma de Ardupilot para drones, ya que estos módulos son frecuentados para tal uso. SiK Radio se puede encontrar en Mission Planner.

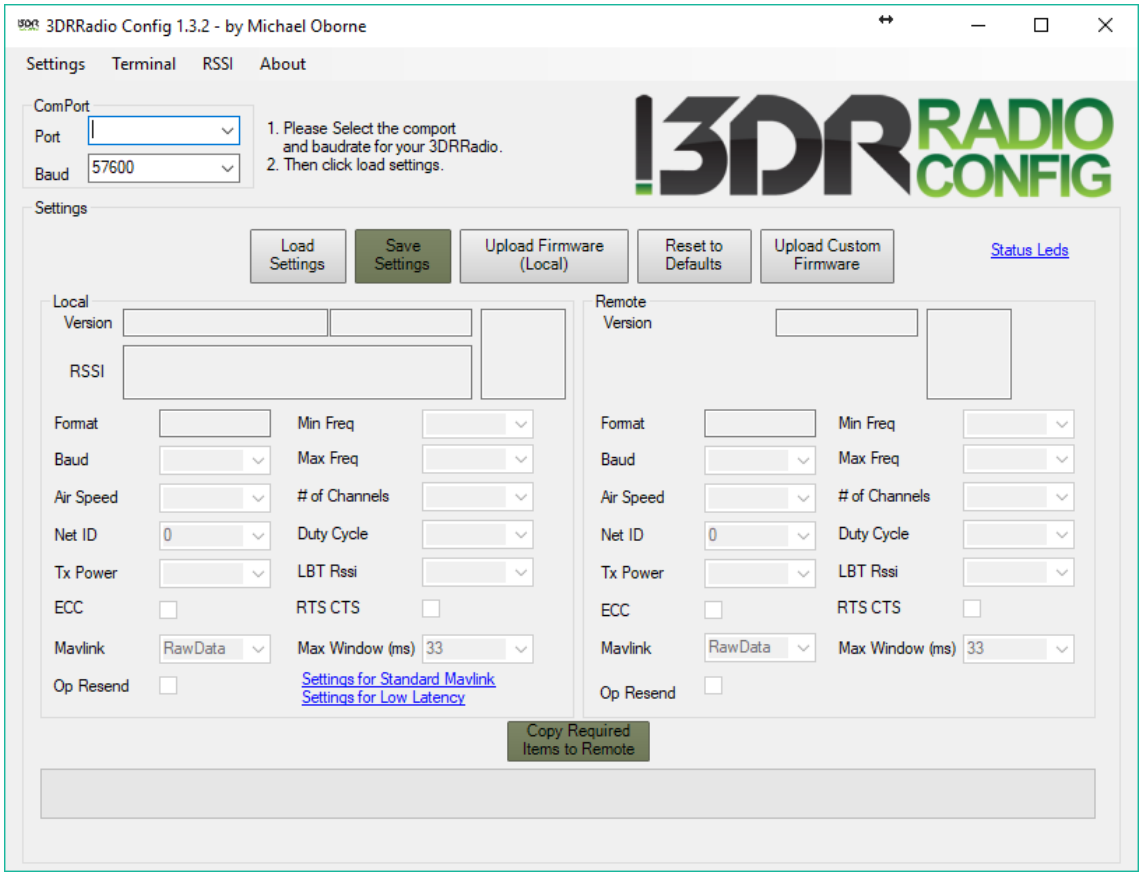


Ilustración 41. Interfaz del programa utilizado para configurar los módulos.

La interfaz es sencilla de utilizar, y muestra información de cada configuración. Lo primero que se tendrá que hacer es seleccionar el puerto del módulo de tierra y los baudios a los que está programado. Una vez conectado se podrán cargar la configuración actual del módulo. Si se tiene alimentado y emparejado el módulo de aire, también se podrán cargar las configuraciones de dicho módulo. En caso de no poder conectarse debido a un error en la configuración que ocasione disparidad entre ellos, se podrá conectar directamente al computador mediante un convertidor USB a TTL (puerto COM virtual).

A continuación, se explicarán las configuraciones utilizadas en la carrera, una pequeña explicación y el motivo de la elección.

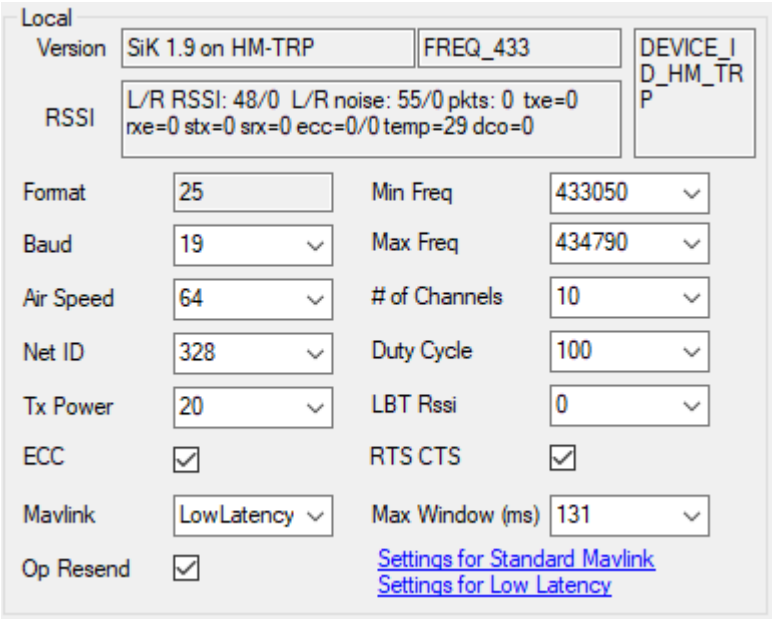


Ilustración 42. Configuración utilizada en los módulos de telemetría.

- **Versión**: La versión instalada es la SiK 1.9, esta no es la que venía de fábrica. Con una pequeña búsqueda en internet se consiguió la última versión y se actualizaron los módulos sin problemas. Se ganó en estabilidad de la señal con este cambio. El programa indica que los módulos son de 433MHz, esto es debido a que existe una misma versión de los módulos para 915MHz, que es la frecuencia libre recomendada en E.E.U.U.

- **Min y Max Freq**: Se puede establecer un rango de frecuencias de trabajo para los módulos, donde posteriormente se dividirán en canales. Esta configuración se dejó por defecto, ya que en la competición no existían muchas interferencias y el funcionamiento era el correcto.

- **# of Channels**: Indica el número de canales (divisiones en la frecuencia) que podrá leer el módulo. Debe estar acorde con el módulo de aire. Se dejó en 10 por defecto, ya que no dio problemas.

- **Duty Cycle**: Indica el porcentaje de tiempo que está permitido transmitir. En algunas regiones está limitado dicho número. También el uso de algunas frecuencias estará permitido solo si este porcentaje no supera cierto umbral. En este caso, no se ha tenido problemas con las normas por trabajar en una banda completamente libre y se dejó al 100%.

- **Baud**: Indica la velocidad de la comunicación por el puerto serie, por defecto venía en 57600 baudios, pero por numerosas pruebas y estimaciones de ancho de banda se decidió bajar a 19200 para obtener menos errores. La forma que tiene el programa de indicarlo es en kBAUD, de forma que 9 implica 9600, 19 serán 19200 y 57 serán 57600.

- **Air Speed**: La velocidad de transmisión de datos entre módulos, se expresa en kbps. No se recomienda un valor inferior a 16 kbps por el propio funcionamiento de la modulación. Se ha experimentado mayor alcance con menor velocidad de transmisión. Es por ello por lo que se ha decidido mantener en 64 kbps, que no es un valor muy elevado y más que suficiente.

- **Net ID**: Es un identificador de red de 16 bits para identificar paquetes de diversos módulos. Ambos deben tener la misma ID para poder identificar los paquetes.

- **Tx Power**: Potencia de transmisión de los módulos. Se ha establecido en la máxima de 20 dBm, que corresponde con 100mW. Es la máxima permitida en esta banda de frecuencia, por lo que es perfectamente válida.

- **LBT Rssi**: Umbral de Listen Before Talk. Es un valor que sirve para indicar si una señal se considera buena o simplemente ruido. Con el valor por defecto ha dado buenos resultados.

- **ECC**: Algoritmo de detección de errores. Enviará más bytes en la transmisión, pero será capaz de recuperar 3 de cada 12 bits corruptos. Las pruebas sin esta opción activadas resultaron en un incremento de errores del 20%, inadmisible en muchos casos.

- **RTS CTS**: Control de flujo de datos por hardware. Se obtuvieron buenos resultados con esta opción por defecto.

- **Op Resend**: Enviará algunos datos dos veces, y será descartado por el receptor si el primero llegó correctamente. Crea una gran diferencia de calidad en la conexión entre ambos con esta opción activada.

Una vez configurados todos los parámetros, los módulos estarán listos para utilizar. Si todo está correcto, los módulos se sincronizarán automáticamente cuando estos se alimenten adecuadamente, dejando así una luz verde fija. Si por algún casual se perdiera la conexión, dicha luz verde pasaría a un estado de intermitencia.

5 PROBLEMAS Y SOLUCIONES

Lo que llamamos progreso es el cambio de un inconveniente por otro.

- Henry Havelock Ellis -

La implementación del bus CAN es la primera vez que se extiende al uso de todo los sitemas del monoplaza. Debido a la inexperiencia del equipo en este tema, surgieron numerosos problemas que afortunadamente se consiguieron solventar.

5.1. Organización de las tramas

Según artículos leídos, el uso normal del bus es dstinar cada ID a un único dato de misma natraleza, pero la implementación en el ART-17 difiere mucho de esta idea.

Primeramente, se utilizará una ID para cada sistema, donde se alojarán los datos necesarios en los 8 bytes máximos de cada trama, ahorrando así ancho de banda en el bus y adquriendo más facilidad de procesamiento. Si un sistema requiriera más, se le destinarían varias ID según las tramas necesarias. Surgió también la necesidad de enviar datos con diferentes temporizaciones, por lo que está aún más justificada la división de varias tramas por sistema.

Nace así la necesidad de pactar las posiciones de los datos y las distintas identificaciones de forma común para todos los sistemas. La organización de dicha tarea se llevó a cabo en una hoja de cálculos de Google Drive, donde todos los miembros tenían acceso a ella y se podía actualizar en cualquier momento.

5.2. Compatibilidad del bus CAN

El protocolo del bus CAN fue lanzado por primera vez en 1986 en el congreso de la Sociedad de Ingenieros Automotrices (SAE). Desde entonces, la empresa Bosch publicó varias versiones de la especificación CAN, siendo la última de ellas la especificación CAN 2.0.

Esta especificación consta de dos partes; la parte A para el formato estándar y la parte B para el formato extendido. Esto originalmente produjo ciertos problemas de incompatibilidad, ya que un dispositivo CAN con formato estándar utiliza un identificador de 11 bits, mientras que el de la trama extendida utilizará un identificador de 29 bits.

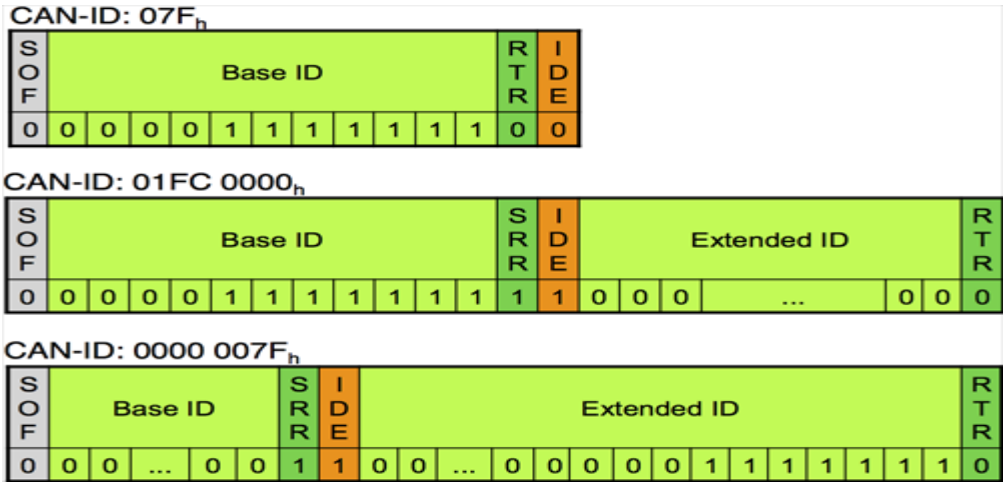


Ilustración 43. Trama estándar de bus CAN vs. trama extendida.

La trama que se vaya a utilizar se puede elegir poniendo el bit RTR a 0 o 1, para así compatibilizar en su toalidad la nueva trama con la versión anterior. Los problemas surgen cuando no hay concordancia entre los dispositiovos y se intenta interpretr una trama extendida como si fuera estándar.

Los ejemplos con los shield de Arduino funcionaban perfectamente. En cambio, algunos dispositivos no conseguían leer la trama debido a esta configuración. Resultaba muy extraño ya que las conexiones estaban bien, y el envío y la recepción eran correctas, ya que funcionaba entre los arduinos de forma externa.

El problema radica en que el controlador utilizado de Microchip, el MCP2515, utiliza la versión de CAN 2.0B Active, lo cual interpretará siempre la trama extendida. Por suerte, la librería utilizada estaba preparada para ambas versiones, cambiando como se mencionó anteriormente, el bit RTR a 1.

5.3. Alimentación de las PCB

Otro inconveniente muy problemático surgido al comienzo de las pruebas fue la alimentación de las mismas. Se utilizó un Convertidor DC/DC de la conocida marca TracoPower, de modelo THL 6-240WISM. La razón de elegir un convertidor y no un simple regulador de 3.3V es sencilla, los convertidores tienen una eficiencia del 76% según el fabricante, que será mucho mayor que utilizar un componente pasivo para transformar los 12V de la batería en la tensión necesaria para el microcontrolador. Además, aporta aislamiento galvánico entre las alimentaciones.

Ilustración 44. Convertidor DC/DC utilizado.

La forma de trabajar de estos convertidores inteligentes es al de conmutar la tensión de entrada y, en base a un PWM y con un filtro a la salida, se consigue bajar la tensión a la adecuada, siempre que la de entrada sea mayor. El problema surge cuando la salida del convertidor no tiene ningún tipo de carga, lo que hace muy difícil o imposible de regular la salida a la tensión fija de 3.3V, provocando una continua conexión y desconexión de emergencia al verse elevada la tensión en vacío. Esto provocaba una señal triangular a la salida de valor RMS menor a 3.3V.

Este suceso se ocasionaba de forma aleatoria al dar contacto en el vehículo, y la explicación se antoja enrevesada. El consumo total del salpicadero está calculado de forma que, con todos los dispositivos conectados, se podrá alimentar con dos convertidores, uno de 3.3V (para Texas Instruments) y otro de 5V (para Arduino y pantalla). Inicialmente, el consumo del convertidor de 3.3V se resume en el display de 16 segmentos y los encapsulados conectados, incluido el MSP430. En el código del microcontrolador, el display estará inicializado con varios segmentos encendidos, pero el problema surge cuando no hay alimentación suficiente en el primer transitorio, donde los segmentos del display podrán estar encendidos o apagados de forma aleatoria. Si resulta estar apagados, no existirá un consumo mínimo a la salida del convertidor y este no podrá estabilizar la tensión, provocando continuos reinicios en el microcontrolador, imposibilitando inicializar el display. Si por el contrario algunos segmentos toman valores encendidos en un principio, existirá consumo mínimo, por lo que se estabilizará la alimentación y el microcontrolador podrá ejecutar sin problemas la inicialización del display al caracter deseado y todo funcionará sin problemas.

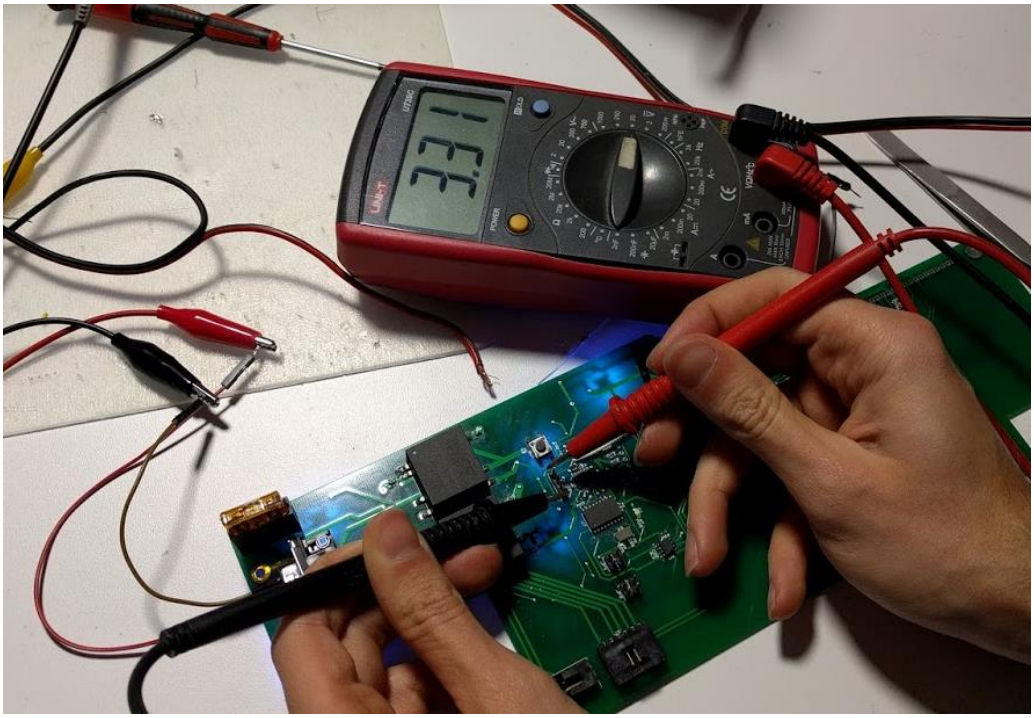


Ilustración 45. Comprobación de la tensión de alimentación con resistencia de carga.

Como no se podía permitir este factor de aleatoriedad, la solución tomada fue poner una resistencia a modo de consumo mínimo para lograr una buena estabilidad en la tensión. El valor de dicha resistencia deberá ser el mayor posible, para que pase la menor intensidad y consuma lo menos posible, siempre que cumpla el mínimo requerido para el buen funcionamiento. Se tomó de forma empírica un valor de 120Ω, ya que con una resistencia de 220Ω, el consumo seguía siendo tan bajo que no conseguía estabilizar la tensión del convertidor conmutado.

Por esta resistencia auxiliar pasarán 27mA según la ley de Ohm, lo que supone un consumo de aproximadamente 90mW. Sabiendo que el convertidor es capaz de entregar una potencia máxima de 4W, este consumo necesario pero desperdiciado equivale a un 2% del total, lo cual se considerará despreciable para la aplicación general.

6 RESULTADOS Y CONCLUSIONES

Si estamos juntos no hay nada imposible. Si estamos divididos todo fallará.

- Winston Churchill -

En este apartado se procurará hacer un análisis de las funcionalidades aportadas al monoplaza, lo que verdaderamente ha beneficiado y las cosas que no se han conseguido implementar.

Primeramente, habrá que destacar que, por culpa de una rotura en unas cogidas de la suspensión, se dañaron algunos extensómetros, por lo que no se disponen de datos reales del funcionamiento de los mismos. Otro problema muy común es que, como es necesario el correcto funcionamiento del bus CAN para transmitir los datos, si esta falla no se podrán obtener los valores. Esto se traduce en que, aunque el sistema funcionara correctamente, se depende de la labor de los demás subsistemas, incluido el bus. Cualquier error en algún punto podría dejar incomunicada la telemetría.

6.1. Funcionamiento general

Finalmente, ya que las pestañas de suspensión y dinámica tenían como misión principal la de obtener los datos necesarios para validar el modelo y realizar ajustes en el monoplaza, durante la carrera solo se prestó especial atención a los datos generales. Aquí se controlaron las revoluciones y marchas, para comprobar el buen funcionamiento de la ECU, pero sobre todo, se procuró vigilar exhaustivamente las temperaturas del motor y cockpit (la cabina que se encuentra bajo el respaldo del piloto) para detectar una posible situación de emergencia. O simplemente para comunicar al piloto que los valores son adecuados y podría cambiar la forma de conducción a una más agresiva.

En la imagen se pueden observar los rangos de valores reales. Originalmente se preparó para tener un sensor de nivel en el depósito de gasolina, pero por diversos problemas de capacidad se decidió suprimir la funcionalidad. Como se puede apreciar en la imagen, el indicador correspondiente a dicho valor estará en color rojo, lo que significa que el valor no está actualizado (debido a que no se recibe ningún valor del mismo).

El sensor de temperatura del agua situado a la salida de la refrigeración del motor es uno de los datos más relevantes. En el momento que supere los 100 grados centígrados, significará que la refrigeración no es la adecuada, y se deberá parar el vehículo inmediatamente para evitar problemas graves.

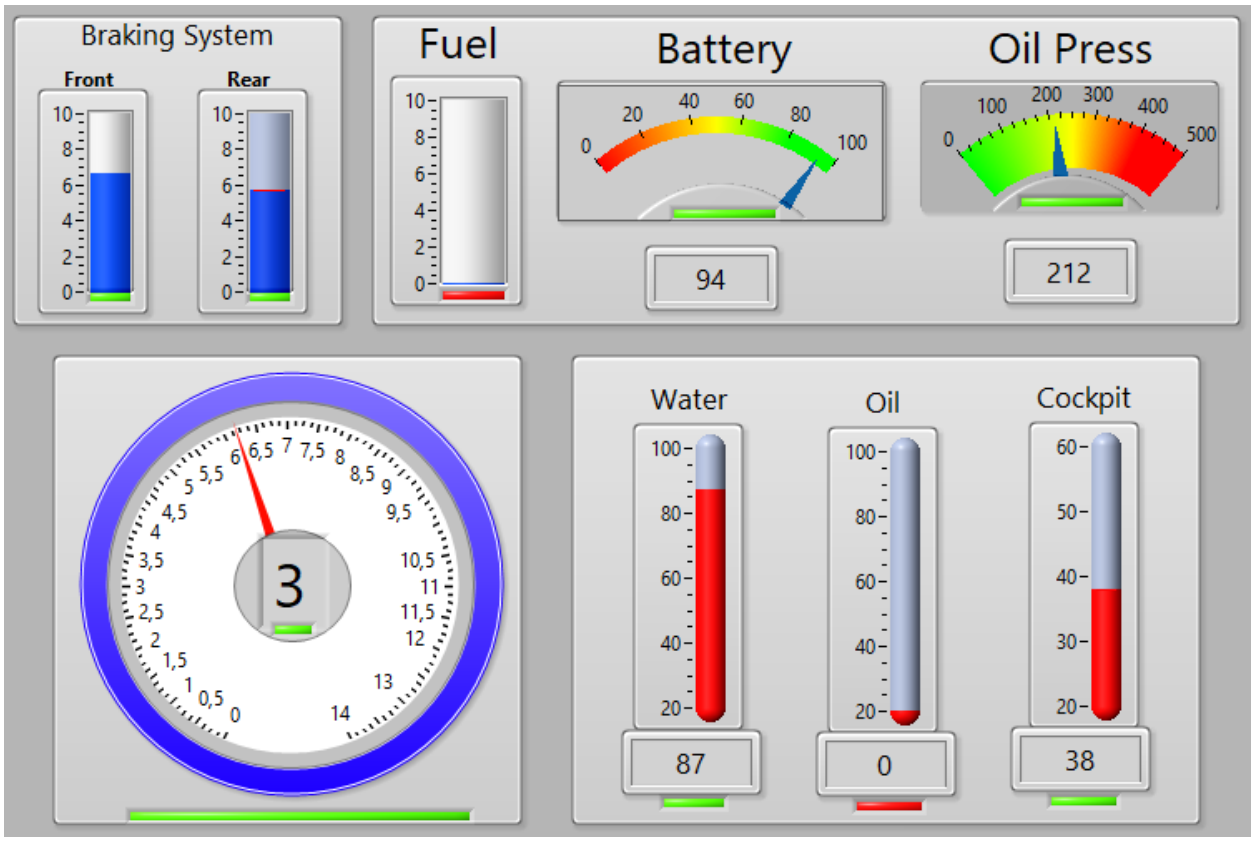


Ilustración 46. Funcionamiento de los datos generales de la telemetría.

La temperatura del cockpit también es muy importante que esté controlada en todo momento, ya que es donde se situa la batería y gran parte de la electrónica. Es una cabina que está delimitada por los colectores del motor que, a pesar de estar debidamente aislado, es una gran fuente de calor que sufre más al estar el vehículo parado, porque no habrá flujo de aire hacia atrás. Al ser una cabina cerrada, todo el calor por conducción que se transmita será difícil de refrigerar. El fabricante de la batería asegura una temperatura máxima de funcionamiento de 60 ºC, por lo que, si se llega a una temperatura próxima a esta, se desalojará el monoplaza y se abrirá el compartimento lo antes posible, ya que las baterías que contienen litio pueden llegar a ser muy inestables.

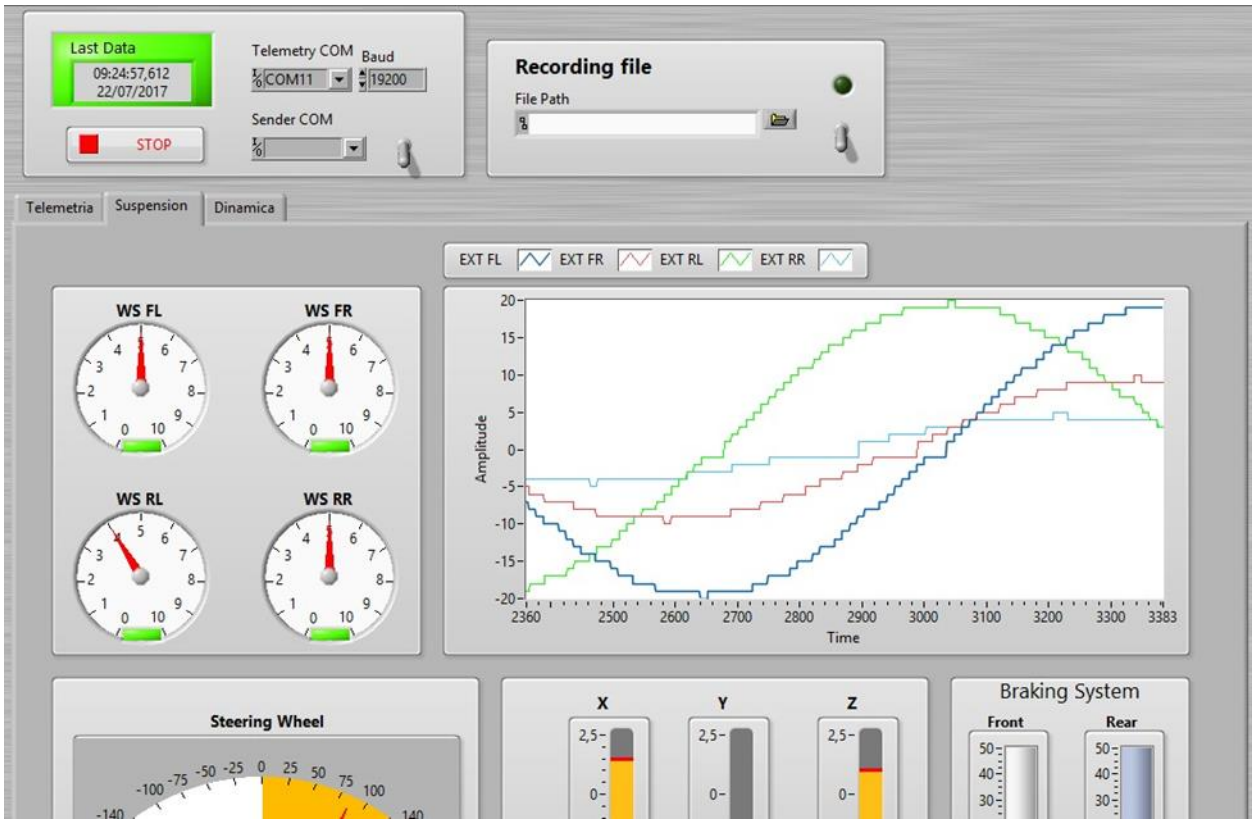


Ilustración 47. Funcionamiento de datos para suspensión.

La pantalla de suspensión luce de la siguiente forma. Se puede estimar a tiempo real el estado de los amortiguadores de suspensión, pero no se podrá obtener datos necesarios para conseguir un modelo preciso del comportamiento, ya que la tasa de refresco es baja. Para ello ya existe un log en la tarjeta de adquisición de datos, por lo que lo mostrado en telemetría sirve para diagnosticar el estado del vehículo, no para obtener información precisa.

Una vez más, se tienen indicadores en cada sensor para comprobar que los datos esten actualizados en todo momento. Esto adquiere más importancia en medidas que no estén en continuo cambio, ya que de no existir dicho indicador no se podría diferenciar un dato antiguo de uno actual si fuese el mismo.

6.2. Grabación de datos

Como se mencionó en el apartado de software, cabe la posibilidad de grabar los datos obtenidos en un fichero de texto. Aquí se mostrará un extracto de dicho fichero, explicando cómo se graba y cómo se podría interpretar para su futuro estudio. Para ello se utilizará un sencillo programa en Matlab.

Haciendo un breve repaso al protocolo de comunicación, los datos se enviarán en una trama de caracteres alfanuméricos, donde cada valor según la procedencia irá precedido por una letra mayúscula. LabVIEW se

encargará de clasificarlo según la letra y convertirá la cadena en un número decimal para representarlo.

El fichero de texto almacena la cadena de carcteres tal y como llega al puerto serie del computador, junto con el tiempo local proporcionada por el bloque Tick Count, que proporciona los milisegundos desde que se comenzó a ejecutar el programa.

A continuación, se expondrá un extracto del fichero de texto para su análisis:

538854512	T145
538854523	S110
538854524	V191
538854525	U184
538854625	A37
538854626	X0
538854627	Y0
538854628	Z10
538854631	S107

La primera columna corresponde a los microsegundos registrados en el temporizador de LabVIEW, que empezará a contar una vez se inicie el programa. Justo al lado de cada tiempo, se guardará la trama leída por el puerto serie. En este caso, los valores acompañados de las letras S, T, U y V corresponden a los valores de los cuatro extensómetros, mientras que las letras X, Y y Z corresponden con los tres ejes del acelerómetro. Este último indicará la fuerza G con una precisión de una cifra decimal, por lo que se tendrá que dividir por 10 al momento de representar o procesar el dato.

Tabla 11. Variables pantalla suspensión.

Nombre	Variable	Letra
WS Front Left	wsFL	I
WS Front Right	wsFR	J
WS Rear Left	wsRL	K
WS Rear Right	wsRR	L
Steering Wheel	st_wheel	A
Ext Front Left	extFL	S
Ext Front Right	extFR	T
Ext Rear Left	extRL	U
Ext Rear Right	extRR	V
Acel. Eje X	accelX	X
Acel. Eje Y	accelY	Y
Acel. Eje Z	accelZ	Z
Frenada delantera	brake_front	D
Frenada trasera	Brake_rear	E

Tabla 10. Variables pantalla dinámica.

Nombre	Variable	Letra
Steering Wheel	st_wheel	A
Frenada delantera	brake_front	D
Frenada trasera	brake_left	E
Pos. Acelerador	tp	H
Marcha	gear	G
Revoluciones	revs	R
Velocidad	---------	
Acel. Eje X	accelX	X
Acel. Eje Y	accelY	Y

Tabla 9. Variables pantalla general.

Nombre	Variable	Letra
Frenada delantera	brake_front	D
Frenada trasera	brake_rear	E
Revoluciones	revs	R
Marcha	gear	G
Batería	load	B
Presión de aceite	oil_press	P
Temp. del agua	ect	W
Temp. del aceite	oil_temp	O
Temp. del cockpit	cockpit	C

7 PRESUPUESTO

La riqueza consiste mucho más en el disfrute que en la posesión.

- Aristóteles -

A continuación, se elaborará una lista que incluye todos los componentes necesarios para la elaboración del proyecto. Para el presupuesto final se deberán incluir otros accesorios como los conectores, cableado y cajas externas. Aquí solo se indicarán los componentes electrónicos y dispositivos utilizados. Se consiguió la fabricación de la PCB por patrocinio, por lo que no se incluye en la tabla.

Componente	Unidades	Precio unitario	Pack	Precio total
Módulos Telemetría	1	23,940 €	1	23,940 €
Antena de 433MHz	2	13,500 €	1	27,000 €
Display 16 segmentos	1	2,650 €	1	2,650 €
Traco Power de 3.3V	1	36,790 €	1	36,790 €
Condensador 100nF	6	0,015 €	25	0,375 €
Condensador 22pF	2	0,022 €	25	0,550 €
Condensador 47nF	1	0,154 €	10	1,540 €
Resistencia 180Ω	17	0,370 €	20	7,400 €
Resistencia 120Ω	5	0,150 €	5	0,750 €
Resistencia 10kΩ	3	0,021 €	50	1,050 €
74HC595	2	0,165 €	1	0,330 €
MSP430G2553	1	2,172 €	1	2,172 €

MCP2515	1	1,700 €	1	1,700 €
SN65HVD233	1	2,870 €	1	2,870 €
Oscilador 16MHz	1	0,500 €	1	0,500 €
Diodo LED	1	0,235 €	1	0,235 €
Ferrita	13	0,060 €	100	6,000 €
Portafusibles	1	3,350 €	1	3,350 €
Postes 4 pines	3	0,740 €	1	3,700 €
Conector 6 pines	1	1,440 €	1	1,440 €
Pulsador	1	0,124 €	20	2,480 €

TOTAL 126,822 €

El presupuesto inicial para el sistema de telemetría fue de 200€, por lo que se ha conseguido ahorrar costes para invertir en mejoras del cableado, algo que normalmente se desprecia, pero es muy importante. El sistema en general ha funcionado como es debido, realmente ha resultado útil, y ha causado pocos problemas en las competiciones. Puesto que ha tenido resultados satisfactorios, se seguirá utilizando el mismo sistema para la siguiente temporada, posiblemente cambiando parte de la interfaz gráfica, pero con la misma metodología.

REFERENCIAS

[1] FSAE, «2016 Formula SAE® Rules,» *SAE International,* p. 12, 2017.

[2] ITU, Reglamento de Radiocomunicaciones, Ginebra, 2012.

[3] F. Gray, Pulse code communication, U.S., 1953.

[4] R. P. Ltd, «SiK Manual,» 2014. [En línea]. Available: http://files.rfdesign.com.au/Files/documents/Software%20manual.pdf. [Último acceso: 2017].

[5] Texas Instruments, «SN74HC595 - Texas Instruments,» 2015. [En línea]. Available: http://www.ti.com/lit/ds/symlink/sn74hc595.pdf.

[6] Ucamco, Gerber File Format Specification.pdf, 2013.

BIBLIOGRAFÍA

Se han utilizado los siguientes programas para la edición:

- Proteus Design Suite 8.0 para el diseño de PCB.
- Code Composer Studio 6.2.0 para la programación del microcontrolador.
- NI LabVIEW 2014 para la programación gráfica e interfaz de usuario.
- 3DRRadio Config 1.3.2 para la configuración de los módulos de telemetría.
- Microsoft Office Word 2016 para la edición de texto.

Se han visitado las siguientes páginas web:

- RF Wireless World: Vendors and resources. <http://www.rfwireless-world.com>
- ArduPilot Open Source. <http://ardupilot.org>
- Geeetech Wiki. <http://www.geeetech.com/wiki>
- Antenas: Más distancia para radiocontrol. <http://www.neoteo.com/antenas-mas-distancia-para-radiocontrol>
- MSP430 CAN interface – electrodummies. <http://www.electrodummies.net/en/msp430-2/msp430-can-interface/>
- Cacoo: Online Diagram Software. < https://cacoo.com/>