



Universidad
Carlos III de Madrid

Universidad Carlos III de Madrid
Escuela Politécnica Superior
Departamento de Ingeniería Mecánica

TRABAJO FIN DE GRADO:

**DISEÑO DE LA TELEMETRÍA PARA UNA
MOTOCICLETA DE COMPETICIÓN II**

Autor:

Fernando Alonso Sacedo

INGENIERÍA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Tutor: Juan Carlos García Prada

Marzo de 2015



Título: Diseño de la telemetría para una motocicleta de competición II

Autor: Fernando Alonso Sacedo

Tutor: Juan Carlos García Prada

EL TRIBUNAL

Presidente:

Vocal:

Secretario:

Realizado el acto de defensa y lectura del Trabajo Fin de Grado el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de:

VOCAL

SECRETARIO

PRESIDENTE



RESUMEN

En el siguiente proyecto se cuenta con el desarrollo de una telemetría para una motocicleta de competición, que servirá para comprender y evaluar el comportamiento y estado de la misma.

Para conseguir este propósito, se realiza y desarrolla la lectura y tratamiento de datos de un conjunto de sensores acoplados a la motocicleta de competición que, posteriormente, se monitorizan y se pueden analizar e interpretar a través de una interfaz gráfica.

A continuación, partiendo de una idea inicial sobre la lectura de datos, se avanza siguiendo las pautas establecidas por el diseño a través de la programación de una serie de funciones y un espacio visual que cumpla con todos y cada uno de los requisitos y objetivos que necesita este trabajo.

Por último, una vez logrado todo lo anterior, se lleva a cabo una serie de pruebas visuales donde se interactúa con una aplicación para observar el buen funcionamiento de esta. En consecuencia de ello, permite tener en cuenta unas posibles ideas futuras con la finalidad y el deseo de una mejora del diseño del proyecto.

Palabras clave: MotoStudent, motocicleta, telemetría, lectura de datos, interfaz gráfica.



ABSTRACT

The next project is the development of the telemetry for motorcycle racing, which will serve to understand and evaluate the performance and status of it .

To achieve this purpose, is carried out and developed the reading and processing data of a set of sensors coupled to the racing motorcycle that subsequently can be monitored and analyzed and interpreted through a graphical interface.

Then, starting from an initial idea of reading data, advances following the guidelines established by the design through a series of programming functions and a visual space that meets every one of the requirements and objectives need this job.

Finally, once achieved all this, is holding a series of visual tests where it interacts with an application to monitor the functioning of this. In consequence, allows taking into account a possible future ideas with the aim and desire for improvement of the project design.

Keywords: MotoStudent, motorbyke, telemetry, data reading, graphical interface.



AGRADECIMIENTOS

Doy mil gracias a mis padres, los cuales con su enorme esfuerzo diario por dar lo mejor a sus hijos, me han brindado la oportunidad de tener una educación, una formación, una madurez y en definitiva un futuro. Sin ellos no hubiese sido posible.



ÍNDICE GENERAL

Capítulo 1: INTRODUCCIÓN	1
1.1. INTRODUCCIÓN AL PROYECTO	1
1.2. MOTIVACIÓN Y OBJETIVOS	3
1.2.1. MOTIVACIÓN	3
1.2.2. OBJETIVOS.....	3
1.2.2.1. Objetivos particulares	4
1.3. ESTRUCTURACIÓN DEL PROYECTO	5
Capítulo 2: HISTORIA Y ESTADO DEL ARTE	7
2.1. HISTORIA DE LA TELEMETRÍA	7
2.2. ESTADO DEL ARTE	11
2.2.1. TELEMETRÍA DE COMPETICIÓN (AUTOMOVILISMO)	11
2.2.2. TELEMETRÍA DE COMPETICIÓN (MOTOCICLISMO).....	14
2.2.3. TELEMETRÍA EN LA ACTUALIDAD.....	17
Capítulo 3: ESTRUCTURA DEL SISTEMA DE TELEMETRÍA	23
3.1. INTRODUCCIÓN	23
3.2. HARDWARE DEL SISTEMA	23
3.2.1. HARDWARE DE ADQUISICIÓN	24
3.2.2. HARDWARE DE TRATAMIENTO	25
3.2.2.1. Sistemas embebidos.....	25
3.2.2.2. Arduino.....	25
3.2.2.3. Placa electrónica Arduino Mega 2560 ADK (R3)	30
3.2.2.4. SD card shield.....	36
3.2.2.5. Micro tarjeta SD	37
3.2.2.6. Adaptador de micro tarjeta SD	38
3.2.2.7. Cable USB	38
3.3. SOFTWARE DEL SISTEMA.....	39
3.3.1. SOFTWARE DE ADQUISICIÓN	39
3.3.2. SOFTWARE DE TRATAMIENTO.....	40
3.3.2.1. Entorno de desarrollo integrado	40
3.3.2.2. Entorno de desarrollo integrado de la plataforma Arduino	40
3.3.2.3. Entorno de desarrollo integrado de la plataforma MATLAB	45
Capítulo 4: ACONDICIONAMIENTO Y DISEÑO DEL SISTEMA DE TELEMETRÍA	53
4.1. INTRODUCCIÓN	53
4.2. CONEXIONADO	53
4.3. DISEÑO DEL SOFTWARE DE DESARROLLO DEL SISTEMA DE LECTURA DE DATOS	55
4.3.1. ESTRUCTURA BÁSICA DE PROGRAMACIÓN ARDUINO	55
4.3.2. PROGRAMA <i>LecturaTarjetaSD</i>	56
4.4. DISEÑO DEL SOFTWARE DE DESARROLLO DEL SISTEMA DE PROCESAMIENTO DE DATOS ...	58
4.4.1. FUNCIÓN PRINCIPAL <i>datosSesion()</i>	58
4.4.2. FUNCIÓN <i>datosTemperatura()</i>	62
4.4.3. FUNCIÓN <i>datosSuspDelantera()</i>	64
4.4.4. FUNCIÓN <i>datosSuspTrasera()</i>	65
4.4.5. FUNCIÓN <i>datosAcelerometro()</i>	66
4.4.6. FUNCIÓN <i>datosGps()</i>	68
4.5. DISEÑO DEL SOFTWARE DE DESARROLLO DEL SISTEMA DE MONITORIZACIÓN DE DATOS ..	69
4.5.1. FUNCIÓN DE ENTORNO <i>guidePrincipal()</i>	70
4.5.2. FUNCIÓN DE ENTORNO <i>guideSesion()</i>	71
4.5.2.1. Instrucción guardar sesión	71



4.5.3. FUNCIONES DE ENTORNO DE APERTURA DE SESIONES	72
4.5.4. FUNCIONES DE ENTORNO DE ESTUDIO PERSONALIZADO DE SENSORES	72
4.5.5. FUNCIÓN DE ENTORNO <i>guideScompararsesion()</i>	72
4.5.6. FUNCIONES DE ENTORNO DE COMPARACIÓN DE SENSORES	72
4.5.7. FUNCIONES DE ENTORNO DE COMPARACIÓN DE VARIABLES	73
4.6. DISEÑO DE LA INTERFAZ GRÁFICA DE USUARIO DEL SISTEMA DE ANÁLISIS DE DATOS	73
4.6.1. INTERFAZ GRÁFICA TELEMETRÍA	74
4.6.2. INTERFAZ GRÁFICA TELEMETRÍA-SESIÓN	75
4.6.3. INTERFACES GRÁFICAS DE SENSORES	76
4.6.4. INTERFAZ GRÁFICA TELEMETRÍA-COMPARAR SESIÓN	80
4.6.5. INTERFACES GRÁFICAS DE COMPARACIÓN ENTRE VARIABLES DE UNA MISMA SESIÓN	81
4.6.6. INTERFACES GRÁFICAS DE COMPARACIÓN ENTRE SENSORES DE DISTINTA SESIÓN	82
Capítulo 5: FUNCIONAMIENTO DEL SISTEMA DE TELEMETRÍA	83
5.1. INTRODUCCIÓN	83
5.2. FUNCIONAMIENTO	83
5.2.1. FLUJOGRAMA FUNCIONAL DE LA INTERFAZ TELEMÉTRICA	84
5.3. APLICACIÓN PARA EL SISTEMA DE TELEMETRÍA	89
Capítulo 6: VISUALIZACIÓN DE DATOS EN GOOGLE EARTH.....	93
6.1. INTRODUCCIÓN	93
6.2. PROCESO DE VISUALIZACIÓN DE DATOS	93
Capítulo 7: ENSAYOS EXPERIMENTALES Y RESULTADOS REALES	96
7.1. INTRODUCCIÓN	96
7.2. DESARROLLO DE LOS ENSAYOS EXPERIMENTALES	96
7.3. RESULTADOS REALES	97
Capítulo 8: CONCLUSIONES Y TRABAJOS FUTUROS.....	112
8.1. CONCLUSIONES	112
8.2. TRABAJOS FUTUROS.....	113
BIBLIOGRAFÍA.....	115
MATERIAL DE CONSULTA.....	115
ENLACES.....	115
ANEXOS	116
ANEXO 1: MICROPROCESADOR ATMEL8U2	116
ANEXO 2: MICROPROCESADOR ATMEGA2560.....	117
ANEXO 3: CÓDIGO ARDUINO.....	121
ANEXO 4: CÓDIGO MATLAB.....	123



ÍNDICE DE FIGURAS

Figura 1.1. Logo de la competición MotoStudent.....	2
Figura 1.2. Influencia de la telemetría en las nuevas tecnologías.....	3
Figura 1.3. Herramientas de programación utilizadas (ARDUINO y MATLAB).....	4
Figura 2.1. Regulador "flyball governor" aplicado a motor de vapor.....	7
Figura 2.2. El "Winter Palace" (izquierda) y el Edificio del Estado Mayor de San Petersburgo, cuartel general del ejército ruso en 1845, (derecha)	7
Figura 2.3. Observatorio Vallot en el Mont Blanc	8
Figura 2.4. "Selsyn" y su esquemático	8
Figura 2.5. Astrógrafo del Observatorio Púlkovo	8
Figura 2.6. Funcionamiento general del Canal de Panamá.....	9
Figura 2.7. Cohete balístico V2.....	9
Figura 2.8. Puesta a punto del satélite Sputnik.....	10
Figura 2.9. Puesta a punto de la sonda Mariner IV	10
Figura 2.10. Ingeniero Karl Kempf y vehículo Tyrell 010.....	11
Figura 2.11. Monoplaza Benetton con "Opción 13" incorporada.....	12
Figura 2.12. Esquemático ECU (tratamiento de datos).....	12
Figura 2.13. Telemetría en la F1 (ondas de radio y microondas)	13
Figura 2.14. Telemetría en la F1 (infrarrojos).....	13
Figura 2.15. Antonio Cobas y su telemétrica puesta a punto	14
Figura 2.16. Antonio Cobas (segundo por la derecha) y Ramón Aurín (segundo por la izquierda) en el equipo Ducados Honda Pons	15
Figura 2.17. Repsol Honda HRC 2006 parametrizada por Ramón Aurín	16
Figura 2.18. Ramón Aurín y Dani Pedrosa estudiando su telemetría en el box de HRC.....	16
Figura 2.19. Telemetría desarrollada por McLaren Electronics para ayudar a un bebé de 3 meses con problemas coronarios (izquierda) y nuevos marcapasos guiados a través de órdenes telemétricas (derecha)	17
Figura 2.20. Esquema del funcionamiento de la tobillera para delincuentes.....	17
Figura 2.21. Sala de control de una plataforma petrolera (izquierda) y sala de control de una central nuclear (derecha)	18
Figura 2.22. Estación metereológica inalámbrica (izquierda) y monitorización de aguas (derecha).....	18
Figura 2.23. Implantación de microchip electrónico (izquierda) en una rana (derecha).....	18
Figura 2.24. Interfaz de análisis Diablo Super Biker	19
Figura 2.25. Interfaz de análisis TrackMaster	19
Figura 2.26. Interfaz de análisis Race Sense	19
Figura 2.27. Display y módulo GPS Alfano ADM-BX4GPS	20
Figura 2.28. Dispositivo GPT con laptimer.....	20
Figura 2.29. Display Percul Lapcom S Lite 100	20
Figura 2.30. Data-logger BMW HP Racer	20
Figura 2.31. Instalación Telemétrica Profesional Bottpower	21
Figura 2.32. ECU Magneti Marelli	21
Figura 2.33. Software Atlas (izquierda) y software Wintax4 (derecha)	21
Figura 2.34. Sistema completo de telemetría 2D-datarecording.....	22
Figura 2.35. Honda CBR1000RR Fireblade SP y su display de competición	22
Figura 3.1. Sensor LM35DZ.....	24
Figura 3.2. Acelerómetro SparkFun ADXL335	24
Figura 3.3. GPS SparkFun Venus	24
Figura 3.4. Sensores para suspensiones 2D.....	24
Figura 3.5. Arduino Due	29
Figura 3.6. Arduino Mega 2560.....	29
Figura 3.7. Arduino Leonardo.....	29
Figura 3.8. Arduino Duemilanove.....	29
Figura 3.9. Arduino Uno	29
Figura 3.10. Arduino Fio.....	29
Figura 3.11. Arduino Red board.....	29
Figura 3.12. Arduino Pro	29
Figura 3.13. Arduino Mini	30
Figura 3.14. Arduino Bluetooth	30



Figura 3.15. Arduino LilyPad	30
Figura 3.16. Arduino Mega 2560 ADK (R3) (parte delantera)	30
Figura 3.17. Arduino Mega 2560 ADK(R3) (parte trasera)	30
Figura 3.18. Fuentes de alimentación externa (conexionado)	32
Figura 3.19. USB como fuente de alimentación (conexionado).....	32
Figura 3.20. Diagrama externo de Arduino Mega ADK (R3)	33
Figura 3.21. SD card shield V3.1	36
Figura 3.22. Diagrama externo SD card shield V3.1.....	37
Figura 3.23. MicroSD SanDisk	38
Figura 3.24. Adaptador de micro tarjeta SD.....	38
Figura 3.25. Cable USB	38
Figura 3.26. Elección de la placa Arduino	41
Figura 3.27. Elección del puerto serial.....	41
Figura 3.28. Entorno de desarrollo integrado Arduino.....	42
Figura 3.29. Barra de herramientas	42
Figura 3.30. Barra de menús	43
Figura 3.31. Entorno de ventanas del entorno de desarrollo integrado.....	44
Figura 3.32. Entorno de desarrollo integrado de MATLAB.....	46
Figura 3.33. Barra de herramientas	46
Figura 3.34. Barra de menús	47
Figura 3.35. Entorno de ventanas.....	48
Figura 3.36. Complemento de elementos de la barra de herramientas de MATLAB	48
Figura 3.37. Menú Start / Toolboxes.....	49
Figura 3.38. Subentorno Simulink	50
Figura 3.39. Subentorno GUIDE.....	50
Figura 3.40. Barra de herramientas	51
Figura 3.41. Barra de menús	51
Figura 4.1. Paso 1.....	53
Figura 4.2. Paso 2.....	54
Figura 4.3. Diagrama del conexionado de pines entre SD card shield y Arduino (según "Tabla 4.1.")	54
Figura 4.4. Paso 3.....	55
Figura 4.5. Estructura básica de programación de Arduino	55
Figura 4.6. Flujograma LecturaTarjetaSD.....	57
Figura 4.7. Flujograma función principal datosSesion()	61
Figura 4.8. Flujograma función datosTemperatura().....	63
Figura 4.9. Flujograma función datosSuspDelantera().....	64
Figura 4.10. Flujograma función datosSuspTrasera()	65
Figura 4.11. Flujograma función datosAcelerometro()	67
Figura 4.12. Flujograma función datosGps()	69
Figura 4.13. Interfaz gráfica TELEMETRÍA del sistema de telemetría.....	74
Figura 4.14. Interfaz gráfica TELEMETRÍA-SESIÓN del sistema de telemetría.....	75
Figura 4.15. Interfaz gráfica TELEMETRÍA-TEMPERATURA AMBIENTE del sistema de telemetría	77
Figura 4.16. Interfaz gráfica TELEMETRÍA-SUSPENSIÓN DELANTERA del sistema de telemetría	77
Figura 4.17. Interfaz gráfica TELEMETRÍA-SUSPENSIÓN TRASERA del sistema de telemetría.....	78
Figura 4.18. Interfaz gráfica TELEMETRÍA-ACELERÓMETRO del sistema de telemetría.....	78
Figura 4.19. Interfaz gráfica TELEMETRÍA-GPS del sistema de telemetría	79
Figura 4.20. Interfaz gráfica TELEMETRÍA-CLIMATOLOGÍA del sistema de telemetría	79
Figura 4.21. Interfaz gráfica TELEMETRÍA-COMPARAR SESIÓN del sistema de telemetría.....	80
Figura 4.22. Interfaz gráfica TELEMETRÍA-COMPARAR VARIABLES del sistema de telemetría	81
Figura 4.23. Interfaz gráfica TELEMETRÍA-COMPARAR SENSORES del sistema de telemetría.....	82
Figura 5.1. Flujograma funcional de la interfaz gráfica del sistema de telemetría	89
Figura 5.2. Paso 1.....	90
Figura 5.3. Paso 2.....	90
Figura 5.4. Paso 3.....	91
Figura 5.5. Paso 4.....	91
Figura 5.6. Paso 5. Apertura del sistema de telemetría desde la aplicación creada	92
Figura 6.1. Aviso de internet al navegador de MATLAB sobre su compatibilidad al acceder a Google Earth.....	93
Figura 6.2. Proceso 1: inclusión de Google Earth en MATLAB.....	94
Figura 6.3. Ejemplo de archivo de datos para convertir a KML.....	94



Figura 6.4. Conversión del archivo a KML (Primera imagen), Creación del archivo KML(segunda imagen) y Visualización de los datos (KML) del ejemplo "prueba GEW" mediante Google Earth (tercera imagen).....	95
Figura 7.1. Introducción de datos para el ensayo Universidad Carlos III de Madrid	97
Figura 7.2. "MENÚ SESIÓN" con el tablón de datos y el recorrido del ensayo Universidad Carlos III de Madrid	97
Figura 7.3. Estudio de los resultados del sensor temperatura del ensayo Universidad Carlos III de Madrid	98
Figura 7.4. Estudio de los resultados del sensor suspensión delantera del ensayo Universidad Carlos III de Madrid .	99
Figura 7.5. Estudio de los resultados del sensor suspensión trasera del ensayo Universidad Carlos III de Madrid ...	100
Figura 7.6. Estudio de los resultados del sensor acelerómetro del ensayo Universidad Carlos III de Madrid.....	101
Figura 7.7. Estudio de los resultados del sensor GPS del ensayo Universidad Carlos III de Madrid	102
Figura 7.8. "MENÚ COMPARAR" variables de una misma sesión del ensayo Universidad Carlos III de Madrid ..	103
Figura 7.9. Estudio de la relación entre las variables altitud y temperatura del ensayo Universidad Carlos III de Madrid.....	103
Figura 7.10. Estudio de la relación entre las variables suspensión delantera y suspensión trasera del ensayo Universidad Carlos III de Madrid	104
Figura 7.11. Estudio de la relación entre las variables suspensión delantera y altitud del ensayo Universidad Carlos III de Madrid.....	104
Figura 7.12. Estudio de la relación entre las variables suspensión trasera y altitud del ensayo Universidad Carlos III de Madrid.....	105
Figura 7.13. Estudio de la relación entre las variables velocidad y altitud del ensayo Universidad Carlos III de Madrid	105
Figura 7.14. Estudio de la relación entre las variables velocidad y suspensión delantera del ensayo Universidad Carlos III de Madrid.....	106
Figura 7.15. Estudio de la relación entre las variables velocidad y suspensión trasera del ensayo Universidad Carlos III de Madrid.....	106
Figura 7.16. Estudio de la relación entre las variables velocidad y temperatura del ensayo Universidad Carlos III de Madrid.....	107
Figura 7.17. "MENÚ COMPARAR" ensayo Universidad Carlos III de Madrid con ensayo Vallecas	107
Figura 7.18. Comparación del sensor acelerómetro entre el ensayo Universidad Carlos III de Madrid y el ensayo Vallecas.....	108
Figura 7.19. Comparación del sensor GPS entre el ensayo Universidad Carlos III de Madrid y el ensayo Vallecas .	108
Figura 7.20. Comparación del sensor suspensión delantera entre el ensayo Universidad Carlos III de Madrid y el ensayo Vallecas.....	109
Figura 7.21. Comparación del sensor suspensión trasera entre el ensayo Universidad Carlos III de Madrid y el ensayo Vallecas.....	109
Figura 7.22. Comparación del sensor temperatura entre el ensayo Universidad Carlos III de Madrid y el ensayo Vallecas.....	110
Figura 7.23. Recorrido del ensayo Universidad Carlos III de Madrid.....	110
Figura 7.24. Recorrido del ensayo Vallecas.....	111
Figura 1.Anexo 1. Esquemático Microprocesador Atmel8U2	116
Figura 1.Anexo 2. Esquemático microprocesador ATMEGA2560.....	117



ÍNDICE DE TABLAS

Tabla 3.1. Estructura del hardware del sistema de telemetría	23
Tabla 3.2. Modelos de Arduino y sus especificaciones	26
Tabla 3.3. Especificación	37
Tabla 3.4. Estructura del software del sistema de telemetría.....	39
Tabla 3.5. Elementos GUIDE	52
Tabla 4.1. Conexiones de la función de transferencia	54
Tabla 4.2. Funciones de entorno de comparación de variables	73



Capítulo 1: INTRODUCCIÓN

1.1. INTRODUCCIÓN AL PROYECTO

"Scientia potentia est.", en castellano, *"El conocimiento es el poder"* es una cita atribuida al filósofo anglosajón Francis Bacon (22 de enero de 1561 - 9 de abril de 1626), aunque la única constancia que afirma dicha frase célebre, por primera vez se encuentra en la obra *Leviatán* (1651) del también filósofo inglés Thomas Hobbes (5 de abril de 1588 - 4 de diciembre de 1679). Esta cita es conocida hoy en día y coloquialmente como *"La información es poder"* y es el rasgo más significativo de los propósitos de la telemetría.

Desde el Grupo de investigación Maqlab y el equipo MotoStudent pertenecientes al departamento de Ingeniería Mecánica de la Universidad Carlos III de Madrid, se contempla una gran concienciación y motivación para conseguir mediante esfuerzo y trabajo una evolución diaria. En consecuencia, se plantea, por primera vez en la universidad, la creación de un sistema propio de telemetría para una motocicleta de competición. Se pretende integrar una tecnología capaz de leer datos, obtenidos a bordo de un vehículo, para su posterior estudio y análisis. Para lograrlo es necesario, desde el punto de vista que nos ofrece la ingeniería, un intenso estudio de su funcionamiento y de sus componentes.

La telemetría es el principal elemento que forma el origen de este estudio. Por definición, este concepto se entiende como la tecnología que permite la adquisición de información o medición de magnitudes físicas que, a través de sensores, se transforman en tensiones eléctricas y se digitalizan de tal modo que se puedan enviar y procesar en un ordenador e interpretar en programas informáticos a partir de sus correspondientes interfaces gráficas. Se limita a examinar las señales vitales del elemento en cuestión, comprueba la efectividad y el funcionamiento del mismo, o refuerza y contrasta teorías y experiencias llevadas a cabo en la investigación.

Es por ello que todo el contenido de esta definición es aplicado en el presente proyecto y en la motocicleta de competición. Este medio de transporte, el cual vio la luz en el siglo XIX, es uno de los vehículos más utilizados alrededor del mundo. Su aceptación es tal, que su importancia y utilización se manifiestan de la misma forma en su aspecto más ocioso, el cual tiene su culminación en la competición. Para competir bien se necesita ser y tener lo mejor, y la telemetría es la principal herramienta para ayudar a conseguir ambos requisitos. Este análisis, en colaboración con el piloto, te permite elegir los mejores reglajes para la motocicleta y proporcionar la mejor información para sacar el máximo rendimiento al pilotaje y a la seguridad del piloto.

Se ha podido observar que la captación de información es primordial, no obstante, la optimización del proyecto se produce cuando estos "paquetes de datos" son enviados o trasladados al operador de nuestro sistema telemétrico con el fin de ser estudiados y evaluados. Hay muchos medios para enviar la información. Uno de ellos es la comunicación inalámbrica, que es la forma más rápida para transmitir la información recogida. Otro medio, igual de eficiente, es la comunicación guiada que nos pueden ofrecer la centralita y la tarjeta SD, la cual utilizamos en este proyecto, donde la información almacenada espera a ser conectada al operador telemétrico, en este caso el ordenador, para poder ser enviada y analizada.



En la asociación, en la cual van de la mano la motocicleta y la telemetría en representación del mundo de la competición y de la evolución tecnológica, se halla la competición internacional de MotoStudent. Esta prueba de las dos ruedas enfrenta a un gran número de equipos que pertenecen a las distintas universidades de todo el mundo. Su cometido es poner a prueba los conocimientos y habilidades de los integrantes de cada equipo para investigar, diseñar, desarrollar y fabricar un prototipo de motocicleta de pequeña cilindrada. Todo ello se realiza durante unas jornadas que se celebran en las instalaciones de Motorland Aragón, con el fin de superar los retos tecnológicos que nos plantea la motocicleta.



Figura 1.1. Logo de la competición MotoStudent

El proceso evolutivo del presente proyecto ha brindado la posibilidad de experimentar las distintas fases de desarrollo que contiene, las cuales se inician en la investigación de las tecnologías empleadas en esta materia y se culminan en la obtención de resultados reales. Es por ello que se realizaron pruebas en diferentes puntos de Madrid, como en Vallecas, Móstoles y Leganés para posteriormente analizar e interpretar todos los datos adquiridos en cada una de ellas.

1.2. MOTIVACIÓN Y OBJETIVOS

1.2.1. MOTIVACIÓN

El ser humano, por naturaleza, se ha caracterizado por evolucionar constantemente. Ha conseguido, a través de la indispensable motivación, alcanzar los objetivos marcados que logran mejorar nuestra calidad de vida. La telemetría es un campo que trabaja con la finalidad de ser parte de esta mejora.

La telemetría es, por muchos motivos, un gran avance. Expone, por sí misma, una multitud de características y usos que hacen tenerla muy en cuenta en el mundo que nos rodea. Tiene una gran presencia o influencia en todas o en la mayoría de las nuevas tecnologías, ofrece una multitud de posibilidades, está en continua evolución, dota de una gran funcionalidad adicional al sistema donde se instala (en el caso de nuestro estudio, en nuestra moto de competición) y con el paso del tiempo su eficiencia crece de manera exponencial. Se destaca también la popularidad y repercusión social que están teniendo últimamente todos los dispositivos telemétricos, consiguiendo asociar este tipo de tecnologías a un ámbito más amateur. Información y datos recogidos por equipos más profesionales se están consiguiendo integrar en dispositivos al alcance de cualquier usuario con una telemetría más sencilla, pero que cumple satisfactoriamente las expectativas de sus cometidos. Debido a todo ello, la telemetría ha sido una gran motivación para tomar la decisión de realizar este trabajo de fin de grado.



Figura 1.2. Influencia de la telemetría en las nuevas tecnologías

1.2.2. OBJETIVOS

Todo lo expuesto con anterioridad influye de una manera clara en los objetivos de nuestro proyecto, el cual, tiene como finalidad la adquisición, monitorización y análisis de datos experimentales. Es por ello que, con la elaboración de este trabajo, buscamos crear una herramienta que, a través de los resultados obtenidos, sea capaz de conseguir mejorar el rendimiento y la seguridad del piloto, pudiéndole de este modo, aconsejar sobre las maniobras de su pilotaje visualizando los resultados conseguidos en pista. También, de la misma forma, contiene la capacidad de utilizar la información adquirida para optimizar el diseño de la motocicleta, esto nos permite conseguir la mejor puesta a punto y construir modelos más competitivos. De este modo, nos proporciona la posibilidad de sentar las bases para futuros estudios o mejoras que se quieran realizar de este proyecto.

1.2.2.1. Objetivos particulares

Para cumplir con todos los objetivos principales mencionados en el anterior punto, es necesario plantear el desarrollo de otra serie de objetivos más particulares que van ligados directamente a este proyecto:

- Plantear una metodología de diseño que se ajuste a los medios económicos, logísticos y humanos que estén a nuestro alcance, teniendo muy en cuenta, las condiciones exigidas por una normativa de competición de alto nivel. Para ello, ha sido necesario realizar un estudio minucioso de la normativa de la competición.
- Seleccionar el conjunto de sensores para llevar a cabo la adquisición de datos.
- Seleccionar la plataforma y el dispositivo electrónico capaz de leer los datos obtenidos y comunicarlos, a través de un puerto USB, al ordenador donde se realiza la monitorización de los mismos. Para conseguirlo, se ha elegido la herramienta Arduino Mega ADK y la plataforma de programación Arduino.
- Seleccionar el medio electrónico para transportar la información obtenida de los sensores a la zona de lectura del objetivo número tres. Se logró, eligiendo una tarjeta SD de 2GB.
- Seleccionar la plataforma de programación para el tratamiento, visualización y análisis de los datos conseguidos. Para ello, se ha escogido la herramienta MATLAB 7.10.0 (R2010a).
- Desarrollar un programa, utilizando la plataforma de programación Arduino seleccionada, que consiga leer los datos almacenados en nuestra tarjeta SD para su posterior verificación y muestra por pantalla.
- Diseñar un programa y una interfaz gráfica, ambos con múltiples funciones y utilizando la plataforma de programación MATLAB, que logre todo el tratamiento y visualización de los datos telemétricos para su perfecto y preciso entendimiento.



Figura 1.3. Herramientas de programación utilizadas (ARDUINO y MATLAB)



1.3. ESTRUCTURACIÓN DEL PROYECTO

Durante la elaboración del presente documento, se ha pretendido que su organización siguiera el orden coherente de la construcción del mismo. A partir de sus ocho distintos capítulos se observa el desarrollo de la telemetría para una motocicleta de competición:

- **Capítulo 1 - "Introducción"**: Es el capítulo actual. Se desarrolla una introducción del proyecto explicando de forma breve el tema a tratar, se redactan las motivaciones por las que se ha realizado el presente estudio, se explican los objetivos fundamentales que se pretenden conseguir, se describen los objetivos más particulares que hacen posibles los anteriores y se lleva a cabo la estructura del actual informe.
- **Capítulo 2 - "Historia y estado del arte"**: En esta parte del proyecto se realiza un "viaje" a través de la historia de la telemetría, desarrollando un estudio minucioso desde sus orígenes hasta la actualidad. Se observa, en detalle, la evolución de la mencionada tecnología en el mundo de la competición a partir del estado del arte que experimenta nuestro trabajo de fin de grado.
- **Capítulo 3 - "Estructura del sistema de telemetría"**: En el presente apartado se explica la composición que forma la estructura del sistema de telemetría. Se divide en dos partes, una en hardware y otra en software. En cada una de estas partes se estudian los distintos tipos que derivan de ellas, se describen y analizan los elementos que las componen.
- **Capítulo 4 - "Acondicionamiento y diseño del sistema de telemetría"**: En el siguiente capítulo se analizan los ensamblajes e interconexiones del hardware de tratamiento. Se desarrolla el diseño de todo el software de tratamiento, se analizan estructuras de código, las funciones que forman el sistema de telemetría y el diseño de la interfaz gráfica.
- **Capítulo 5 - "Funcionamiento del sistema de telemetría"**: Se trata de un capítulo didáctico, en el cual se explica al usuario el funcionamiento y manejo de la interfaz gráfica del sistema de telemetría a través de la explicación de su flujograma. También, se enseña a generar una aplicación del sistema telemétrico para el ordenador sin necesidad de utilizar MATLAB.
- **Capítulo 6 - "Visualización de datos en Google Earth"**: En este capítulo se desarrolla la idea de complementar al sistema de telemetría con el visualizado de los datos a través de Google Earth.
- **Capítulo 7 - "Ensayos experimentales y resultado reales"**: Para esta parte del proyecto se describe como se han realizado los diferentes ensayos experimentales y se explica con todo detalle los resultados obtenidos para los ensayos propuestos. Se puede observar la capacidad de análisis que tiene el sistema de telemetría.
- **Capítulo 8 - "Conclusiones y trabajos futuros"**: Se redactan las conclusiones obtenidas a lo largo de la realización de cada una de las partes que dan forma a este proyecto. También, se describen posibles alternativas futuras con el fin y el deseo de una mejora del diseño de este sistema de telemetría.



- **Bibliografía:** Se registran y se citan todos los elementos de consulta que han sido utilizados para la investigación sobre este trabajo de fin de grado.

- **Anexos:** Lo componen cuatro tipos. En el anexo 1 y 2 se muestran los esquemáticos de los microcontroladores utilizados por Arduino. El anexo 3 está constituido por el código generado para Arduino. Por último, en el anexo 4 se encuentra todo el código programado que se encarga del funcionamiento del sistema y de la interfaz telemétrica.

Capítulo 2: HISTORIA Y ESTADO DEL ARTE

2.1. HISTORIA DE LA TELEMETRÍA

La palabra telemetría, contiene su procedencia en las raíces griegas $\tau\eta\lambda\epsilon$ (tele), que quiere decir a distancia, y $\mu\epsilon\tau\rho\upsilon\nu$ (metron), que quiere decir medida.

Esta tecnología, apoyándose en su definición más primitiva de recoger datos en un lugar inaccesible, ubica sus orígenes en el siglo XIII. En plena Revolución Industrial, los sistemas impulsados a vapor se consideraron los precursores de los procesos industriales controlados de forma fiable y determinista. En concreto, citaremos al ingeniero mecánico e inventor de origen

escocés James Watt, que desarrolló en 1788 el primer regulador centrífugo que incorporaba el sensor de medida y el actuador en un único ingenio. Este avance, conocido como el "flyball governor", funcionaba sin la necesidad de disponer de un amplificador de potencia que aislara al sensor del actuador. Se utilizó para controlar el proceso interno de un motor de vapor a distancia donde su medida tenía un dificultoso acceso. Se convirtió, de esta manera, en el primer aparato telemétrico capaz de regular una maquina a partir de los datos que obtenían sus medidas.

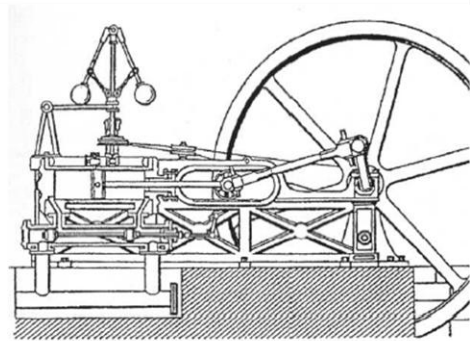
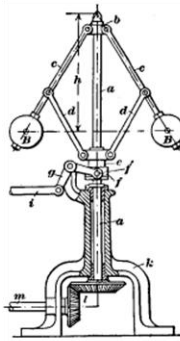


Figura 2.1. Regulador "flyball governor" aplicado a motor de vapor

El progreso telemétrico continuó en el siglo XIX con la transmisión de información por cable a través de líneas eléctricas. En 1845 la Federación de Rusia desarrolló uno de los primeros circuitos de transmisión de datos para mantener el contacto entre el Palacio Imperial de Invierno del Zar, denominado "Winter Palace", y el cuartel general del ejército ruso, actualmente conocido como el Edificio del Estado Mayor de la ciudad de San Petersburgo.



Figura 2.2. El "Winter Palace" (izquierda) y el Edificio del Estado Mayor de San Petersburgo, cuartel general del ejército ruso en 1845, (derecha)

En el año 1874, un grupo de ingenieros franceses crearon un sistema compuesto por un conjunto de sensores con la capacidad de adquirir datos meteorológicos e información para conocer la



Figura 2.3. Observatorio Vallot en el Mont Blanc

profundidad a la que se encuentra la nieve. Todo el equipo fue instalado y utilizado en la montaña Mont Blanc para transmitir, en tiempo real, toda esa información a París. Más tarde en 1890, el meteorólogo y mecenas francés Joseph Vallot construyó un pequeño laboratorio en el mismo Mont Blanc, siendo el primero en alta montaña. Pronto fue conocido como el observatorio Vallot y se utilizó para obtener y estudiar datos de todo tipo de disciplinas científicas.

El desarrollo dedicado a esta herramienta prosiguió en los inicios del siglo XX. En 1901 el inventor C. Michalke, de procedencia estadounidense, patenta "Selsyn", un circuito que produce un envío sincronizado de parámetros de rotación a distancia.

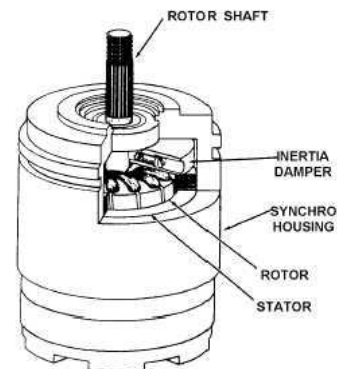


Figura 2.4. "Selsyn" y su esquemático

Por otro lado en 1906, en las instalaciones rusas del Observatorio Púlkovo, construyen una sucesión de estaciones sísmicas dotadas con una telemetría encargada de obtener información a cerca de terremotos y movimientos sísmicos. Este observatorio, por aquel entonces, también contaba con el refractor más grande del mundo y con un astrógrafo encargado de medir con precisión la posición de los astros.

Más tarde, en 1912, la compañía eléctrica Commonwealth Edison diseña un sistema de telemetría que consiguió monitorizar las cargas eléctricas, de la red eléctrica, de las calles de la ciudad de Chicago en el estado de Illinois.

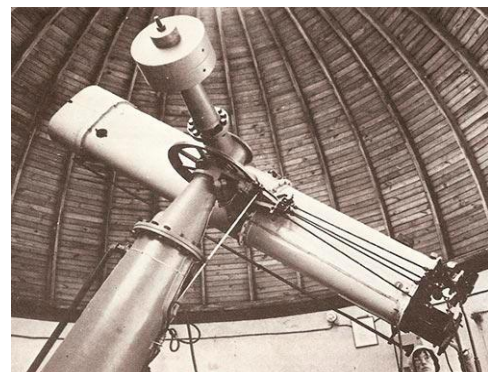


Figura 2.5. Astrógrafo del Observatorio Púlkovo

En 1913 se finaliza la construcción del Canal de Panamá, el cual, cuenta con la instalación de un dispositivo telemétrico para controlar y analizar las esclusas y los niveles de agua.

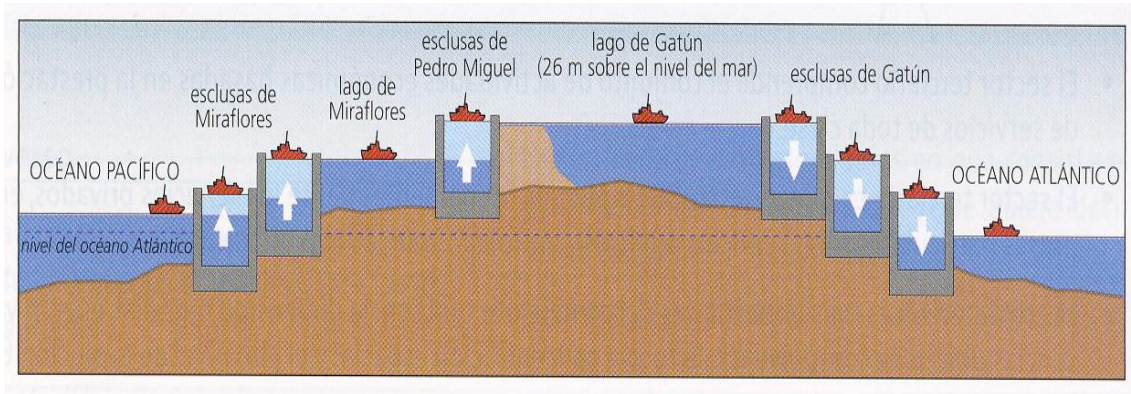


Figura 2.6. Funcionamiento general del Canal de Panamá

La telemetría, con el transcurso del tiempo, ha experimentado una evolución de tal forma que, gracias a la continua investigación, nacieron los primeros sistemas telemétricos sin la necesidad de utilizar cables. Los artífices de que este avance fuera posible son Robert Bureau, desde Francia, y Pavel Molchanov, desde Rusia, que en 1930 consiguieron desarrollar aplicaciones en radio sondas. En concreto, el estudio de Molchanov propone la transmisión por código Morse y vía "wireless" para un sistema encargado de modular medidas de presión y temperatura.

La década de 1930 serviría también para comenzar a vincular el uso de la telemetría con la aeronáutica. Se consiguió que un globo aerostático, dotado con equipo telemétrico, fuera capaz de recopilar datos sobre las condiciones atmosféricas.

En la década de 1940, en plena Segunda Guerra Mundial, se dio un gran salto tecnológico en los sistemas de telemetría. El ingeniero aeroespacial de raíces germanas Wernher von Braun implementó el primer cohete balístico, conocido como "V2", que hizo un vuelo suborbital. Para ello, el misil constaba de un dispositivo telemétrico, llamado "Messina", que estaba equipado con un multiplexado de señales de radio y dispositivos como giróscopos y acelerómetros, donde estos últimos, controlaban la dirección y la velocidad del motor respectivamente.

Los soviéticos implementaron los primeros equipos de telemetría en el espacio, los cuales, utilizaban la modulación por posición de pulso (PPM) o la modulación por duración de pulso (PDM).



Figura 2.7. Cohete balístico V2

En 1957 la URSS, en pleno comienzo de la carrera espacial contra Estados Unidos, lanzó con éxito el Sputnik demostrando la innovación en cuanto a adquisición y análisis de datos.



Figura 2.8. Puesta a punto del satélite Sputnik

Por otro lado, los estadounidenses realizaron sus primeras investigaciones espaciales utilizando métodos científicos similares a los rusos que, posteriormente, sustituyeron por sistemas de modulación por código de pulso (PCM). El empleo de estos sistemas se dejaron notar en la sonda de exploración, denominada "Mariner 4", enviada a Marte en 1964, con la cual, se registraron todo tipo de datos para el posterior estudio de cada uno de ellos.



Figura 2.9. Puesta a punto de la sonda Mariner IV

Más tarde Rusia comenzó a utilizar sistemas de radio redundantes, transmitiendo por modulación PCM en una banda de decímetros y por modulación PPM en una banda de centímetros.

Por último, debido a estas grandes innovaciones, hemos conseguido prosperar de una forma tecnológica a gran escala. Todo ello sirvió para que la telemetría siguiese y siga haciendo historia a lo largo de los años. El afán del ser humano por descubrir, conocer y saber va ligado a la existencia de este gran avance que es la telemetría.



2.2. ESTADO DEL ARTE

Este estudio está vinculado de una forma muy directa al mundo del motor y de la competición. Hoy en día, la telemetría está incorporada a esta índole de un modo impresionante, con resultados cada vez más satisfactorios y es un término que se ha hecho muy popular entre los profesionales y aficionados del gremio. Campeonatos de todo tipo en el mundo del motor incorporan este avance tecnológico, entre los más importantes a nivel mundial, destacamos la Fórmula 1 y MotoGP. Debido al desarrollo ejercido sobre esta herramienta en ambos eventos deportivos, es necesario analizar la evolución telemétrica en el mundo de la competición.

2.2.1. TELEMETRÍA DE COMPETICIÓN (AUTOMOVILISMO)

En 1980 se produce el primer intento de introducir la telemetría a través de la Fórmula 1. El estadounidense Karl Kempf, doctorado en física, diseñó e implementó un sistema compuesto por un pequeño ordenador y un conjunto de sensores de captación de datos para controlar la suspensión electrónica del Tyrell 010. La idea del proyecto se abandonaría más adelante por la complejidad del mismo, causando que el Tyrell 010 jamás compitiera con dicho sistema.



Figura 2.10. Ingeniero Karl Kempf y vehículo Tyrell 010

A partir de este momento, la electrónica toma un papel primordial. Las continuas investigaciones en la mejora de esta disciplina consiguieron que en la década de 1990, las escuderías McLaren y Williams, implementaran un sistema telemétrico capaz de funcionar a bordo de un monoplaza en un entorno de máxima competición. Estos elementos electrónicos ensalzaron el gran avance tecnológico que supuso, y todos los equipos que no lo utilizaron quedaron por detrás en la competición.

Hasta ese momento, la información adquirida se realizaba de manera unidireccional, es decir, los datos conseguidos se enviaban a un ordenador, ubicado en el coche, que se encargaba de analizar y modificar las diferentes configuraciones que ofrecía el monoplaza. A continuación, la información se guardaba y era descargada en su llegada a boxes para su posterior estudio por parte de los ingenieros. Con todo este proceso, la idea de conocer los parámetros más significativos, con la finalidad de progresar tanto en el desarrollo del vehículo como en el del piloto, se pudo desarrollar. Posteriormente, la evolución de las telecomunicaciones habilitaron un abanico de posibilidades muy grandes para la telemetría, no hay que olvidar que su significado es "medición a distancia", consiguiendo gracias a ello que los ingenieros recibieran los datos en tiempo real mientras los pilotos rodaban en pista. Esta tecnología permitió obtener información como: tiempo por vuelta, comportamiento de suspensiones, constantes vitales del piloto, etc. En la actualidad, la Fórmula 1 cumple con el significado pleno de la palabra telemetría, recogiendo cualquier tipo de dato medible en tiempo real.

Continuando en la línea histórica, en 1993 se redujeron ciertas "ayudas al piloto" en la Fórmula 1. Se prohibieron todo tipo de prestaciones electrónicas que facilitaran el ámbito de la conducción. En 1994 se puso en marcha la conocida "Opción 13", creada por el ingeniero británico Tad Czapski. Fue instalada en el Benetton de Michael Schumacher y consistía en una opción oculta en un display de 10 opciones que activaba un control de salida electrónico. En consecuencia, la Federación Internacional de Automovilismo (FIA) investigó la innovadora argucia para proceder a la imposición de una posible sanción al respecto.



Figura 2.11. Monoplaza Benetton con "Opción 13" incorporada

El siguiente gran paso se dio en 2001 de la mano de TAG Electronics, introduciendo la telemetría bidireccional en la Fórmula 1. Este sistema, además de transmitir la información del coche en el pit, introduce una variante nueva que permitía a los ingenieros modificar la configuración del monoplaza desde "el muro" y en tiempo real mientras se rodaba en pista. A partir de la investigación sobre este avance, en 2002, se consiguió modificar el mapeado del motor y activar y desactivar ciertos sensores desde boxes. La FIA prohibió este sistema en la temporada 2003, su intención era que el piloto volviera a hacerse cargo de esas tareas pero debido al espectacular avance de la electrónica y a la creación de componentes minúsculos, existía la posibilidad de que ciertos equipos no cumplieren con la normativa. Para solucionar el problema, la FIA estableció, en 2008, un uso responsable de la electrónica creando para todas las escuderías, a partir del desarrollo realizado por McLaren Electronics Systems (MES), una unidad de control electrónico, denominada ECU (Engine Central Unit), y un servidor de datos, llamado ATLAS (Advanced Telemetry Linked Acquisition System), que muestra los canales de telemetría para que los ingenieros puedan interpretar y analizar los resultados de la prueba.

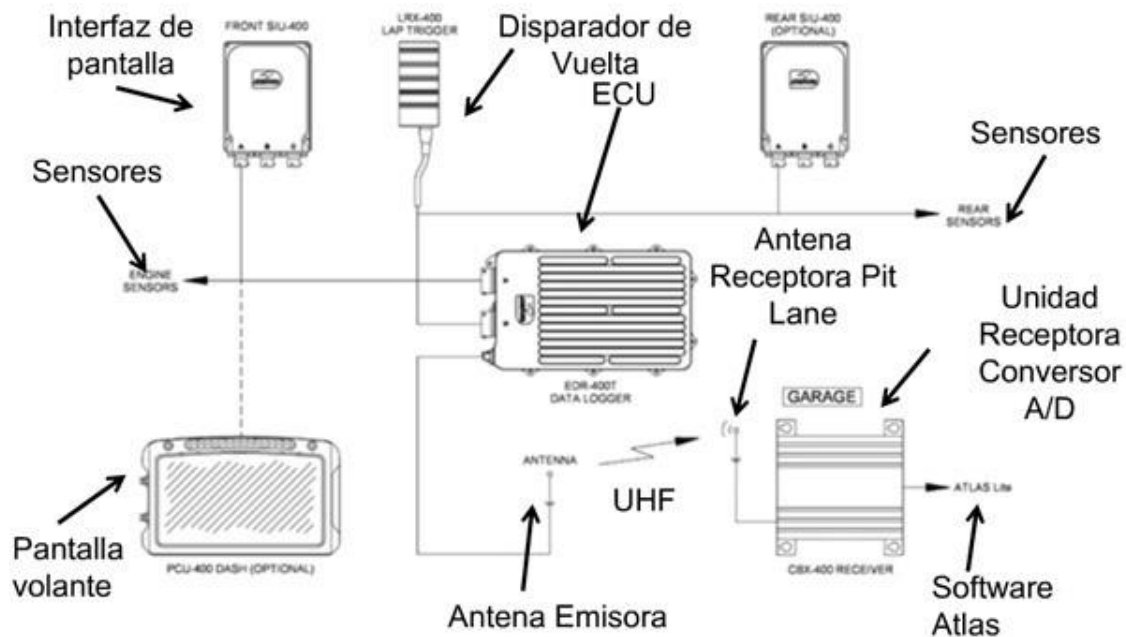


Figura 2.12. Esquemático ECU (tratamiento de datos)

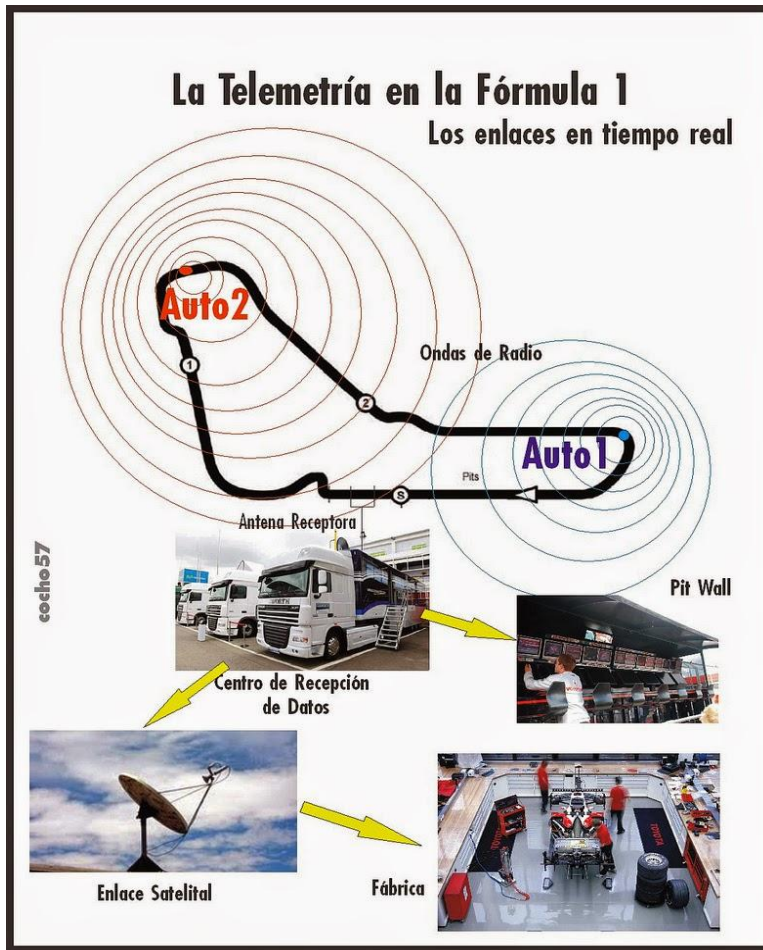


Figura 2.13. Telemetría en la F1 (ondas de radio y microondas)

Este tipo de telemetría trabaja con medios no guiados, es decir, la información recibida en el paddock se obtiene mediante una comunicación inalámbrica a través del aire. Esta transmisión de datos puede realizarse por medio de ondas de radio y microondas. Las primeras son el medio más rápido, pero también se convierten en las menos fiables ya que su ancho de banda es menor. Las segundas utilizan las bandas UHF (300MHz-300GHz) y se basan en conexiones punto a punto coche-portátil, ambas opciones para un resultado satisfactorio, deben propagarse intentando recibir las menos fluctuaciones posibles. Debido a que en cualquier circuito estas ondas deben superar obstáculos sólidos, podemos experimentar pérdidas de información ya que no siempre se podrán receptionar bien los datos. Para solucionar este problema se dispuso de un dispositivo capaz de almacenar los datos cuando las condiciones de la recepción no fueran las más favorables, cierto es, que los ingenieros no pueden analizar la información por completo en tiempo real pero, de esta manera, no se pierde ningún dato adquirido. Otro medio de transmisión para esta telemetría y que soluciona también el problema anterior es el que nos proporciona los infrarrojos. El monoplaza transmite una gran cantidad de datos (10Mb) cuando pasa por el pit lane en cada paso por vuelta.

Para enviar información a corta distancia, cada circuito del mundial dispone de antenas repetidoras que redirigen la señal, con los datos de los vehículos, al centro de recepción de datos. Desde allí se gestiona el reparto de información a los diferentes boxes y fábricas de las distintas escuderías para que así, los ingenieros puedan desarrollar y tomar decisiones de la información recibida.

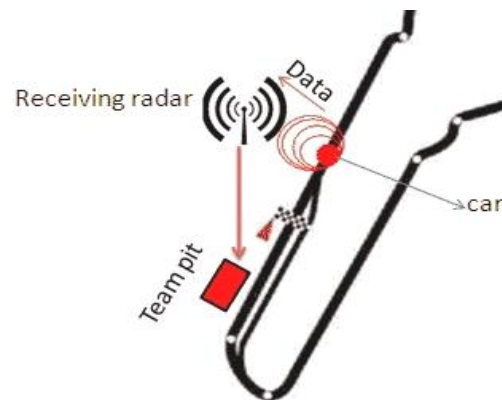


Figura 2.14. Telemetría en la F1 (infrarrojos)

2.2.2. TELEMETRÍA DE COMPETICIÓN (MOTOCICLISMO)

Los inicios del motociclismo en el mundo de la competición se remontan en el año 1894. La primera carrera oficial se produjo en Francia en 1896 y cubrió una competición de ida y vuelta entre París y Nantes. En 1904 se funda la Federación Internacional de Clubes de Motociclismo (FICM), este organismo creó en 1924 el Gran Premio de Europa. En 1949 la FICM cambia su nombre por el de la Federación Internacional de Motociclismo (FIM), organizando y anunciando en ese mismo año la primera edición del Campeonato del Mundo de Motociclismo, también conocido, de manera oficial y popular, como MotoGP.

El origen de la telemetría en el mundo de las dos ruedas fue más tardío que en el automovilismo, sus inicios se remontan en la década de 1990. Las primeras investigaciones, estudios y tomas de contacto sobre la telemetría en el motociclismo de competición fueron llevadas a cabo por Antonio Cobas y su principal pupilo, en este campo, Ramón Aurín, dos ingenieros españoles que utilizaron de forma espléndida la tecnología de los sistemas telemétricos ya existentes.

El preludeo que originó la mayor revolución en todas las facetas del mundo del motociclismo fue obra del genio español Antonio Cobas (1952-2004). Ingeniero de profesión, Antonio era reconocido como "un artista del renacimiento", fue un visionario que propuso ideas y conceptos novedosos en el mundo del motor y de la competición que en su momento la mayoría de la gente no podía entender. Sin lugar a dudas era un prodigio adelantado a su época, un técnico multidisciplinar que marco un antes y un después en la investigación, diseño y construcción de motos de carreras. Es por ello que se le consideró la persona más influyente de cuantos han



Figura 2.15. Antonio Cobas y su telemétrica puesta a punto

trabajado en el mundo del motociclismo de competición. Cobas se inició en la competición automovilística en 1975, donde en ese mismo año fundaría la escudería Teder en colaboración con Alex Soler Roig y el equipo Selex. Realizó diversos trabajos tanto a nivel nacional para mejorar equipos de la Fórmula 1430 y 1800, como a nivel internacional aportando sus conocimientos en la Fórmula Renault y en la escudería Minardi, en la cual, trabajaría durante dos años. Como apasionado de la tecnología, su entusiasmo de tal modo que, a partir de la construcción de su primera moto en 1978, se le ocurrió la gran idea de desarrollar e implementar todos esos avances telemétricos en el motociclismo de competición. Para ello, a raíz de sus trabajos en los que también diseñaba sistemas de control para centrales nucleares, pudo comprobar y darse cuenta de cómo la informática podía ayudarle a desarrollar sus planes. En consecuencia, en 1982 Antonio adquirió su primer ordenador, un 8086, y se convirtió en un experto programador. Sus líneas de código implementadas en una centralita electrónica y su equipo de sensores repartidos por toda la estructura de la motocicleta consiguieron que la investigación diera sus frutos en 1989, introduciendo de este modo la telemetría en las motos de carreras y logrando hacer campeón del mundo de 125 c.c. a Alex Crivillé con 17 años. Cobas logró perfeccionar, de un modo nunca visto, el diseño, el comportamiento y la puesta a punto de las motocicletas.

Una vez conscientes de lo que suponía la telemetría en el avance del motociclismo de competición, Antonio necesitaría, para esta nueva etapa, un colaborador que le ayudase con su propuesta sobre los innovadores sistemas telemétricos. Haría su aparición entonces Ramón Aurín y con él, la profesión de "telemétrico".

Ramón Aurín era un joven estudiante que acabó la carrera universitaria en 1990. Acto seguido, comenzó a impartir clases de física y matemáticas en un centro de formación profesional. Un día, en la prensa, se percató de un anuncio de trabajo para un equipo de competición y a pesar de que el salario era poco, se interesó y decidió presentarse a la entrevista. Una vez allí, coincidió con Antonio Cobas, el cual, necesitaba y buscaba un electrónico que supiera hacer instalaciones y manejar programas informáticos para trabajar en el desarrollo telemétrico de la adquisición de datos. Ramón sorprendió a todos y, tras superar la entrevista, se incorporó una semana más tarde como analista de telemetría junto con Antonio Cobas en el GP de Bélgica. Por aquellos entonces no había absolutamente nada de electrónica, ambos sabían que ahí estaba el futuro y se convirtieron en los pioneros de los sensores, cableados y en definitiva del desarrollo de la telemetría en el mundo de las dos ruedas.



Figura 2.16. Antonio Cobas (segundo por la derecha) y Ramón Aurín (segundo por la izquierda) en el equipo Ducados Honda Pons

Desde un principio, uno de los principales problemas que tuvieron fueron las grandes dimensiones de los equipos telemétricos, provenían del mundo automovilístico y su adaptación a la motocicleta no fue sencilla. Para solucionarlo, Antonio Cobas mantuvo conversaciones con los fabricantes para que diseñaran componentes más pequeños a pesar de ofrecer menos prestaciones. Hasta entonces, Ramón y Antonio tuvieron que arreglárselas con ingenios como el de fabricar depósitos de gasolina con la mitad de capacidad para compartir el espacio, y así, poder ubicar los aparatos dentro. A la hora de realizar las mediciones disponían de pocos canales y además debían limitar la cantidad de ellos si querían hacer una lectura de datos de buena calidad en un tiempo aceptable. Tampoco ayudaban las memorias, en un principio, no era posible instalar demasiados sensores porque las memorias no disponían de suficiente capacidad (256K) para guardar toda la información recopilada. Aún así, al final lograron los resultados propuestos, llegaron a representar gráficas de señales como la velocidad, las revoluciones, el recorrido de las suspensiones y la posición del gas entre otras pocas medidas más.

El ingeniero Ramón Aurín lleva dedicándose a la telemetría desde hace 20 años, en los cuales, llegó a ser campeón del mundo de MotoGP en 2006 con Nicky Hayden en la escudería Repsol Honda Team. Su cronología profesional como ingeniero telemétrico abarca escuderías como: JJ Cobas (1990-1991), Campsa Honda Pons (1992), Marlboro Honda Pons (1993), Ducados Honda Pons (1994), Fortuna Honda Pons (1995-1996), Movistar Honda Pons (1997-1999), Emerson Honda Pons (2000), West Honda Pons (2001-2002), Camel Honda Pons (2003-2005) y Repsol Honda Team (2006-Actualidad). Todo el trabajo desarrollado por Ramón durante todos estos años queda latente en el avance espectacular que han sufrido las motocicletas a nivel mundial. En la actualidad es el ingeniero telemétrico de Dani Pedrosa, demostrando de este modo que es un pilar fundamental e historia viva de la telemetría en el mundo del motociclismo de competición. Hoy en día la telemetría gestiona tanto la moto como la seguridad del piloto, y todo ello, usando softwares informáticos y equipos telemétricos cuyas bases siguen siendo la programación desarrollada por Antonio Cobas y los ingenios electrónicos implementados por Ramón Aurín.



Figura 2.17. Repsol Honda HRC 2006 parametrizada por Ramón Aurín



Figura 2.18. Ramón Aurín y Dani Pedrosa estudiando su telemetría en el box de HRC

En cuanto a las transmisiones utilizadas en el motociclismo de competición diferenciamos dos tipos de telemetría, una por medios guiados y otra por medios denominados "semi-guiados". Ambas recogen la información mientras el piloto está rodando, y en ambas, hay que esperar a que llegue la moto a boxes para transmitir la información adquirida de cada expedición en pista. La diferencia entre una y otra telemetría es que por medios "semi-guiados" la transmisión de datos es mediante una conexión Ethernet, vía wireless (MotoGP), y sin embargo, por medios guiados su transmisión se realiza mediante USB (Moto2 y Moto3). Una vez descargados en el ordenador es cuando los ingenieros telemétricos pueden analizar la situación de la moto y el pilotaje del piloto. A diferencia de la Fórmula1, en MotoGP está prohibida la telemetría en tiempo real.

La protección de los datos en el campo de la telemetría es una faceta también muy importante, en la cual, se esfuerzan por mejorar. Preocupa mucho el "espionaje telemétrico" en el mundo de la competición. Es por ello, que cada escudería tiene su propia red y sistema diferenciados del resto de marcas. Los datos se protegen comprimiendo los archivos adquiridos mediante un algoritmo complicado y con un tipo de archivo que sólo se puede leer con un programa que diseña el propio equipo.

2.2.3. TELEMETRÍA EN LA ACTUALIDAD

Como ya se ha comentado la telemetría es una tecnología que, con el paso del tiempo, está presente en multitud de campos, disciplinas, estudios y trabajos desarrollados por el ser humano. Debido a ello se observa que funciones desempeñan estos sistemas de adquisición de datos hoy en día. Para ello se diferencian dos telemetrías distintas, una fuera del mundo del motociclismo de competición y otra dentro de este.

Para comenzar se analiza la telemetría aplicada fuera del mundo de las dos ruedas. En este ámbito se destaca, entre muchas otras, las siguientes aplicaciones:

- ❖ **Medicina:** Es una de las aplicaciones más importantes ya que se pueden salvar vidas. Dispone de unos sistemas equipados con unos dispositivos encargados de medir, registrar y transmitir datos útiles para realizar precisos seguimientos y diagnósticos a los pacientes. Una función de alerta avisa al equipo médico de si un paciente sufre una enfermedad grave. Esta aplicación se utiliza para pacientes con riesgo a tener una actividad anormal del corazón.



Figura 2.19. Telemetría desarrollada por McLaren Electronics para ayudar a un bebé de 3 meses con problemas coronarios (izquierda) y nuevos marcapasos guiados a través de órdenes telemétricas (derecha)

- ❖ **Aplicación de las leyes:** Los sistemas telemétricos, como la tobillera o el coche cebo, ayudan a salvar vidas y facilitan el seguimiento de personas que incumplen la ley. El primero avisa a las autoridades de que un preso o un agresor supera los límites autorizados y el segundo permite detener a los ladrones de automóviles dentro de un vehículo policial habilitado como señuelo, el cual, está equipado con cámaras, control remoto para el apagado de motor y cierre de puertas y un GPS.



Figura 2.20. Esquema del funcionamiento de la tobillera para delincuentes

- ❖ **Exploración de energías, el espacio y la defensa:** Sistemas tan complejos como los que presentan las plataformas de fuentes de energía como el petróleo y el gas, lugares peligrosos o inaccesibles para el hombre como plantas químicas y centrales con radioactividad, sistemas de vuelo para aeronaves teledirigidas o naves espaciales y cohertera como misiles, están equipados con dispositivos telemétricos que controlan automáticamente el funcionamiento, las alarmas y los registros de datos necesarios para conseguir un uso eficiente y seguro de cada uno de ellos.



! *Figura 2.21. Sala de control de una plataforma petrolera (izquierda) y sala de control de una central nuclear (derecha)*

- ❖ **Agricultura:** La telemetría también está presente en una disciplina tan antigua como la agricultura. La mayoría de las actividades relacionadas con cultivos sanos y de buen rendimiento, son aquellas que disponen de mayor información meteorológica y de datos de suelo. Las estaciones meteorológicas inalámbricas se encargan de recoger toda esa información tomando decisiones en base a la radiación solar, humedad del suelo, precipitaciones, etc. Otro parámetro importante es la gestión del agua, el monitoreo de aguas subterráneas para su distribución y su buen funcionamiento permite reacciones rápidas a los acontecimientos producidos en el campo.



Figura 2.22. Estación meteorológica inalámbrica (izquierda) y monitorización de aguas (derecha)

- ❖ **Investigación y gestión de especies animales:** La telemetría también es utilizada para estudiar la vida silvestre. Los animales son equipados con unas etiquetas de instrumentación que incluyen sensores capaces de medir su temperatura, profundidad y duración de inmersión, velocidad y ubicación. Todo ello es de gran utilidad para realizar un seguimiento de cualquier especie y obtener información sobre su comportamiento y su entorno.



Figura 2.23. Implantación de microchip electrónico (izquierda) en una rana (derecha)

La segunda telemetría diferenciada es la empleada dentro del mundo del motociclismo. Se observa como el proyecto está relacionado con muchas de las funciones que se analizan en los siguientes tres tipos de sistemas de adquisición de datos:

❖ **Tipo 1 - Aplicaciones para smartphone:** Consisten en una serie de softwares que utilizan al máximo las prestaciones que les ofrece el GPS de un móvil de última generación. Son muy económicas e incluso algunas de ellas son gratuitas. Las utilizan pilotos amateur o aficionados que quieren conocer y mejorar su pilotaje. Algunos de los ejemplos más novedosos son:

➤ **Diablo Super Biker:** Software telemétrico gratuito y disponible para los sistemas operativos Android e iOS (iphone). Es un aplicación ideal para medir la posición en tiempo real, grado de inclinación en las curvas y velocidad media y máxima. Utiliza los mapas de Google Maps para trazar nuestro recorrido. Permite compartir pruebas realizadas por el piloto en las redes sociales.



Figura 2.24. Interfaz de análisis Diablo Super Biker

➤ **TrackMaster:** Software de adquisición de datos para los sistemas operativos Android. Es un programa que mide datos posicionales, aceleraciones, velocidades, altitudes y tiempos parciales y por vueltas. La aplicación representa y analiza el recorrido en Google Earth. Comparte los resultados en las redes sociales.



Figura 2.25. Interfaz de análisis TrackMaster

➤ **Race Sense:** Software disponible para los sistemas operativos Android e iOS (iphone). La aplicación, que ha sido desarrollada en colaboración con el piloto de Moto2 Anthony West, simula un sistema completo de telemetría. Representa nuestras sesiones sobre Google Maps o sobre un trazado digital similar a los utilizados en el Mundial. Destaca por medir aceleraciones laterales, grados de inclinación de la moto y por incluir un programa para instalar en nuestro ordenador para el análisis de todos los datos.



Figura 2.26. Interfaz de análisis Race Sense

❖ **Tipo 2 - Laptimer:** El siguiente sistema consta de un emisor, un receptor y la pantalla del laptimer. El emisor es situado en la línea de meta para recoger, a través del receptor instalado en la motocicleta, el tiempo por vuelta. Para obtener tiempos parciales se distribuirían más emisores a lo largo del recorrido. En la pantalla aparecen las marcas en tiempo real. Estos dispositivos suelen instalarse acompañados por un data-logger, que almacena todos los datos para analizarlos después, y por un conjunto de sensores para formar un equipo telemétrico más profesional. A continuación, se analizan algunos ejemplos actuales de este tipo de tecnología:

- **Alfano ADM-BX4GPS:** Dispositivo compuesto por un sensor de revoluciones por minuto (RPM), imanes redondos para el sensor de tiempo, Bluetooth y un módulo que incluye un GPS para 4 sensores (los laptimer no disponían de GPS, pero el avance tecnológico ha hecho posible la incorporación de este). No necesita emisor, utiliza la banda magnética de la línea de meta que hay en todos los circuitos de velocidad.



Figura 2.27. Display y módulo GPS Alfano ADM-BX4GPS

- **GPT Racing Time:** Algunos sistemas no funcionan con los emisores instalados en todos los circuitos oficiales. Esto ocurre en el caso de este dispositivo, el cual, dispone de un emisor propio para solucionar el problema. Este modelo almacena los tiempos de hasta 240 vueltas por sesión. Dispone de un GPS y está preparado para funcionar como un data-logger.



Figura 2.28. Dispositivo GPT con laptimer

- **Percul Lapcom S Lite 100:** Sistema de los más completos compuesto por sensor de revoluciones por minuto (RPM) y velocidad, sensor magnético y óptico para los tiempos por vuelta y sensor de desplazamiento y fuerzas G. Muestra la marcha engranada y almacena datos de hasta 999 vueltas. Dispone de tarjeta SD y conexión con ECU. No necesita emisor.



Figura 2.29. Display Percul Lapcom S Lite 100

- **BMW HP Race:** Este dispositivo es un data-logger completo que captura y almacena los parámetros de la moto y del pilotaje. Contiene conexión USB y su software permite analizar en detalle los recorridos junto con Google Maps. Funciona conectado a un laptimer.



Figura 2.30. Data-logger BMW HP Racer

- ❖ **Tipo 3 - Telemetría profesional:** Son aquellos sistemas de adquisición de datos que se utilizan en competiciones profesionales como MotoGP y la F1. Estos equipos son los más completos que existen, cuentan con una gran cantidad de sensores repartidos por toda la estructura del vehículo, disponen de los mejores data_logger del mundo para evaluar el comportamiento del vehículo (ya sean comerciales o diseñados por las propias escuderías), cuentan con un GPS que permite comparar trazadas superpuestas, están equipados con los mejores softwares para el análisis de datos, adquieren la información con la máxima precisión, son capaces de realizar y obtener mayores parámetros de medida que en los dos tipos de sistemas examinados con anterioridad y son los más caros. La monitorización es total, llegando a examinar hasta la dinámica del aire, acelerómetros, velocidades de giro de cada rueda, fuerzas G, distintas temperaturas, etc. Entre los sistemas profesionales se destacan los siguientes:

- **Bottpower:** Esta instalación telemétrica se emplea en la categoría de Moto2 del mundial de motociclismo. Muestra el "mazo" de cables que integran todo el sistema de adquisición de datos.



Figura 2.31. Instalación Telemétrica Profesional Bottpower

- **Magneti Marelli:** Se trata de una de las centrales de control electrónico (ECU) utilizadas en la categoría de MotoGP del mundial de motociclismo. Incorpora comunicación CAN, PWM, Ethernet, etc.



Figura 2.32. ECU Magneti Marelli

- **Software Atlas y Wintax4:** Ambos programas son de carácter profesional y desempeñan la función de analizar todos los datos telemétricos recogidos durante la carrera. El software Atlas, desarrollado por McLaren Electronics, es utilizado en la F1, mientras que el software Wintax4, desarrollado por Magneti Marelli, es el empleado en MotoGP.

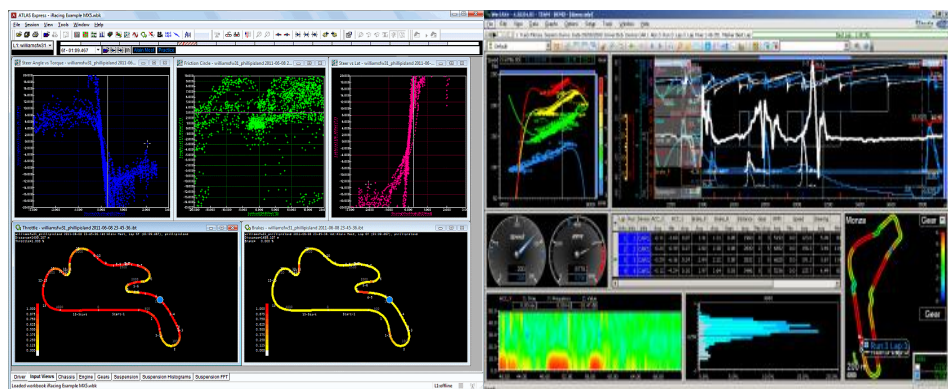


Figura 2.33. Software Atlas (izquierda) y software Wintax4 (derecha)

- **2D-datarecording:** A continuación, se muestra un equipo de telemetría profesional al completo. Este sistema ha sido utilizado en motocicletas para grandes premios del mundial de motociclismo de Moto2.

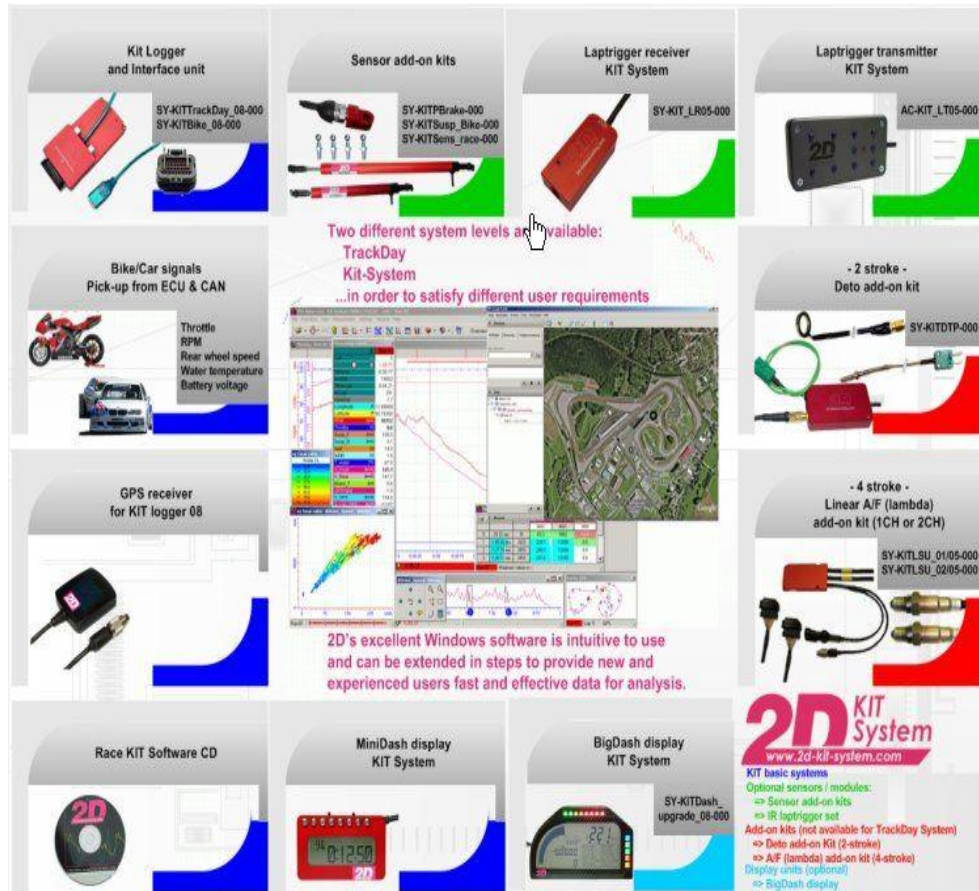


Figura 2.34. Sistema completo de telemetría 2D-datarecording

- **Instrumentación Honda CBR1000RR Fireblade SP 2014:** La firma japonesa incorpora de serie para este modelo de moto una instrumentación telemétrica, acercando a los aficionados del mundo del motor un sistema completamente profesional. Las medidas que puede registrar son: temperatura refrigerante, velocidad, tiempo por vuelta, contador para las vueltas, marcha engranada, consumo y eficiencia de la gasolina, revoluciones y distancia recorrida. Todas estas lecturas numéricas son realizadas a través de un LCD multifunción como los empleados en la máxima competición.

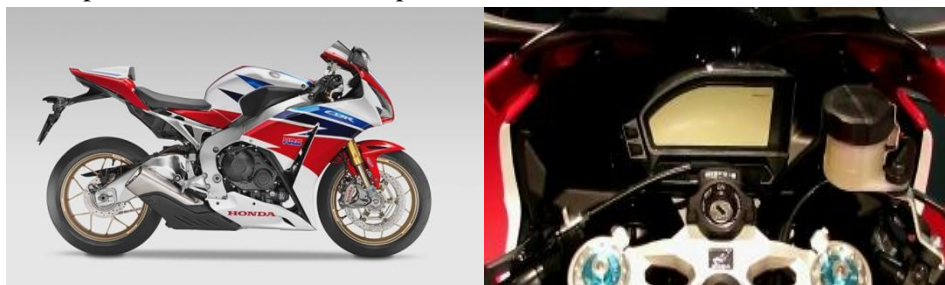


Figura 2.35. Honda CBR1000RR Fireblade SP y su display de competición



Capítulo 3: ESTRUCTURA DEL SISTEMA DE TELEMETRÍA

3.1. INTRODUCCIÓN

Con la finalidad de dar un punto de vista más claro al lector, la estructuración del proyecto se divide en dos partes. La primera parte está formada por el hardware del sistema, mientras que la segunda es constituida por el software del sistema. Tanto para el hardware como para el software se describen y analizan los componentes que lo forman.

En el presente documento cabe mencionar que a diferencia de la parte hardware, cuyo proceso se basa en el montaje y la conexión adecuada de un conjunto de dispositivos electrónicos que mencionaremos más adelante, en el software, a pesar de contar con plataformas de programación, el desarrollo para el funcionamiento del sistema parte desde cero.

3.2. HARDWARE DEL SISTEMA

La composición del hardware está formada, a su vez, por dos grupos de elementos electrónicos. El primer grupo es denominado como hardware de adquisición y se encarga de obtener y registrar todas las medidas que nos proporcionan la información necesaria sobre la motocicleta. El segundo grupo se define como hardware de tratamiento y tiene la función de leer y procesar la información recogida por el hardware de adquisición. Los elementos que forman ambos grupos se observan en la tabla siguiente:

Tabla 3.1. Estructura del hardware del sistema de telemetría

HARDWARE DE ADQUISICIÓN	HARDWARE DE TRATAMIENTO
➤ Sensores:	➤ Placa electrónica Arduino (lectura de datos)
→ Temperatura ambiente	➤ SD card shield (lectura de datos)
→ Acelerómetro	➤ Micro tarjeta SD (almacenado de datos)
→ GPS	➤ Adaptador micro tarjeta SD (ordenador)
→ Suspensión delantera	➤ Cable USB (comunicación puerto serie)
→ Suspensión trasera	
➤ Placa electrónica Arduino (escritura de datos)	
➤ SD card shield (escritura de datos)	
➤ Micro tarjeta SD (y adaptador de micro tarjeta SD)	
➤ Cable USB (comunicación puerto serie)	

3.2.1. HARDWARE DE ADQUISICIÓN

Es importante destacar que el hardware de adquisición no es desarrollado en este trabajo de fin de grado. Sin embargo, es necesario realizar una pequeña introducción explicativa sobre el mismo para posteriormente entender, de un modo correcto, el funcionamiento del hardware de tratamiento, el cual si se desarrolla en el presente documento. Para comprender mejor esta idea es preciso describir, de un modo resumido, los componentes del hardware de adquisición mencionados en la "tabla 3.1.":

- ❖ **Sensores:** Son dispositivos encargados de detectar determinadas magnitudes externas, denominadas variables de instrumentación, y transformarlas en magnitudes eléctricas que seamos capaces de cuantificar y manipular. En este proyecto se utilizan los siguientes:

- **Sensor de temperatura:** El elegido fue el modelo LM35DZ. Este sensor dispone de un rango de temperaturas entre 0 °C y 100°C y se encarga de medir la temperatura ambiente a la que se encuentra la motocicleta.

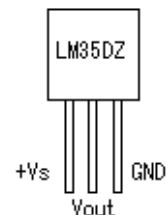


Figura 3.1. Sensor LM35DZ

- **Sensor acelerómetro:** Se trata de un acelerómetro de tres ejes. Para cada eje, el sensor devuelve una tensión proporcional a la aceleración que experimenta. De este modo se obtienen las aceleraciones que sufre la moto en referencia a su eje de coordenadas. El modelo escogido fue el SparkFun ADXL335.



Figura 3.2. Acelerómetro SparkFun ADXL335

- **Sensor GPS:** El dispositivo seleccionado es el modelo SparkFun Venus. Es un GPS con una frecuencia de actualización de 10Hz y es capaz de medir situación horaria en la que nos encontramos, longitud, latitud, altitud y velocidad de la motocicleta.

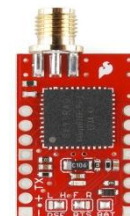


Figura 3.3. GPS SparkFun Venus

- **Sensor suspensión delantera y trasera:** Se compone de dos potenciómetros lineales de distinta longitud. El primero mide 150 mm y calcula el desplazamiento de la suspensión delantera, mientras el segundo mide 75 mm y registra el desplazamiento de la suspensión trasera. El modelo utilizado ha sido el 2D-datarecording.



Figura 3.4. Sensores para suspensiones 2D

- ❖ **Componentes de escritura:** Se trata de los elementos electrónicos restantes que forman el hardware de adquisición, los cuales son: placa electrónica Arduino, SD card shield, MicroSD con adaptador para el ordenador y cable USB. Todo este conjunto es programado y ensamblado con fines de escritura. De este modo se almacenan los datos obtenidos y toda la información queda habilitada para su posterior lectura y tratamiento.



3.2.2. HARDWARE DE TRATAMIENTO

El hardware de tratamiento, a pesar de constituir uno de los dos grupos que dan forma al esqueleto del hardware del sistema de telemetría, es el único que, en realidad, cumple con los cometidos de este trabajo de fin de grado. Se acondiciona y acopla con el fin de leer y transformar, en datos numéricos, las medidas electrónicas recibidas. Dichas medidas se quedan preparadas para un posterior estudio en el ordenador. En este apartado se aprecia en profundidad tanto el análisis como el montaje de los distintos dispositivos electrónicos del hardware de tratamiento expuestos en la "Tabla 3.1."

3.2.2.1. Sistemas embebidos

Un sistema embebido se define como una herramienta computacional que consiste de una electrónica programable especialmente diseñada para realizar funciones específicas sobre un sistema de tiempo real. Se trata de un ordenador más, carece de pantalla y teclado pero está dotado por una placa compuesta por el conjunto de módulos necesarios para desempeñar el trabajo programado. De ahí su versatilidad en cuanto al tamaño y su coste óptimo. Utilizan sistemas operativos muy potentes (Linux, Windows, MS-DOS), pudiendo usar herramientas de desarrollo de software del mismo calibre. Son sistemas que se pueden programar empleando lenguajes como el ensamblador, C, C++ y JAVA. De forma general, es una "pieza especializada" que se instala en un sistema anfitrión, al cual le soluciona de forma óptima la tarea a resolver. Entender las funciones que desempeña este sistema es vital para concienciar al lector de las múltiples alternativas con las que cuenta a la hora de elaborar ciertos proyectos.

Debido a la investigación sobre estos sistemas y a las prestaciones y capacidades que ofrecen, se toma la decisión de cimentar las bases del hardware del presente estudio con uno de los sistemas embebidos que nos ofrece Arduino. Esta plaqueta electrónica es una pieza clave y debido a su importancia es necesario un estudio minucioso de la misma.

3.2.2.2. Arduino

Arduino centra sus orígenes en Italia, comienza en la ciudad de Ivrea en un instituto dedicado a la formación y enseñanza de diseño interactivo. Uno de sus profesores, Massimo Banzi, consciente de los exorbitantes precios que tenían que pagar los alumnos por los microcontroladores que utilizaban, se planteó en el 2003 diseñar su propia placa de hardware para poner solución al problema de las dificultades económicas. De este modo, Massimo Banzi se convirtió en el fundador de este desafiante proyecto junto con el español David Cuartiles y Dave Mellis, los cuales tomaron la decisión de publicar sus avances en la red para desarrollar y conseguir, junto a otros colaboradores, una plataforma con un software y un hardware libre. En 2005 se comenzaron a fabricar las primeras plaquetas y se incorporó al equipo el profesor Tom Igoe, que ofreció sus conocimientos de computación física para inyectar al proyecto un desarrollo a gran escala y para distribuir las tarjetas en el mercado estadounidense. Arduino consiguió entonces, que su funcionamiento fuese del tipo "plug and play" y que su diseño fuese compatible con plataformas informáticas como: Windows, GNU/Linux y MacOSX. Más adelante, Google participó en la implementación del Kit Android ADK (Accessory Development Kit), una placa Arduino capaz de comunicarse con teléfonos móviles con sistema operativo Android. De este modo, Arduino se ha convertido en un fenómeno en continuo auge para todos aquellos usuarios que quieren iniciarse o especializarse en el funcionamiento de este tipo de sistema embebido.



Arduino contiene una arquitectura básica que se compone por un microcontrolador, cuyo lenguaje de programación está basado en Wiring, y por un conjunto de puertos de entrada y salida. Las entradas de carácter analógico y digital capacitan a la placa electrónica con la posibilidad de adquirir información del entorno para controlar las funciones a realizar. Estas placas, también permiten ampliarse para aumentar sus funcionalidades. Pueden dotar al microcontrolador de un soporte para tarjetas de memoria SD y de internet. Toda esta composición es diseñada para facilitar el uso de la electrónica en proyectos multidisciplinarios.

Todas las prestaciones que ofrece este tipo de sistema embebido hacen que el hardware Arduino esté presente en multitud de aplicaciones electrónicas. Entre las más destacadas se pueden citar las siguientes: osciloscopio de código abierto, temática relacionada con la domótica, aeronaves no tripuladas, simulador de terremotos, equipo científico para investigaciones, proyectos de robótica, económetro (indicador de consumo) para vehículos, materia de seguridad, telefonía móvil, etc. En nuestro caso su aplicación es destinada a un sistema de telemetría para una motocicleta de competición.

Para observar los distintos modelos de placas con las que contamos para realizar este trabajo de fin de grado, se adjunta la siguiente tabla con las especificaciones de cada una de ellas:

Tabla 3.2. Modelos de Arduino y sus especificaciones

Modelo	Microcontrolador	Volta je de entrada	Volta je del sistema	Frecuen cia de Reloj	Dig ital I/O	Entradas Analógicas	P W M	U A R T	Memoria Flash	Cargador	Interfaz de Programación
Arduino Due	AT91SAM3X8E	5-12V	3,3V	84MHz	54*	12	12	4	512Kb	Due	Nativa USB
Arduino Leonardo	ATmega32U4	7-12V	5V	16MHz	20*	12	7	1	32Kb	Leonardo	Nativa USB
Arduino Uno - R3	ATmega328	7-12V	5V	16MHz	14	6	6	1	32Kb	Optiboot	USB vía ATmega16U2
RedBoard	ATmega328	7-15V	5V	16MHz	14	6	6	1	32Kb	Optiboot	USB vía FTDI
Arduino Uno SMD (descontinuado)	ATmega328	7-12V	5V	16MHz	14	6	6	1	32Kb	Optiboot	USB vía ATmega8U2
Arduino Uno (descontinuado)	ATmega328	7-12V	5V	16MHz	14	6	6	1	32Kb	Optiboot	USB vía ATmega8U2



Arduino Duemilanove (descontinuado)	ATmega328	7-12V	5V	16MHz	14	6	6	1	32Kb	AtmegaBOOT	USB vía FTDI
Arduino Bluetooth (descontinuado)	ATmega328	1,2-5,5V	5V	16MHz	14	6	6	1	32Kb	AtmegaBOOT	SerialBluetooth
Arduino Pro 3.3V/8MHz	ATmega328	3,35 - 12V	3,3V	8MHz	14	6	6	1	32Kb	AtmegaBOOT	Cabecera compatible con FTDI
Arduino Pro 5V/16MHz	ATmega328	5 - 12V	5V	16MHz	14	6	6	1	32Kb	AtmegaBOOT	Cabecera compatible con FTDI
Ethernet Pro (descontinuado)	ATmega328	7-12V	5V	16MHz	14	6	6	1	32Kb	AtmegaBOOT	Cabecera compatible con FTDI
Arduino Mega 2560 R3	ATmega2560	7-12V	5V	16MHz	54	16	14	4	256Kb	STK500v2	USB vía ATMega16U2
Arduino Mega 2560 (descontinuado)	ATmega2560	7-12V	5V	16MHz	54	16	14	4	256Kb	STK500v2	USB vía ATMega8U2
Arduino Mega (descontinuado)	ATmega1280	7-12V	5V	16MHz	54	16	14	4	128Kb	STK500v2	USB vía FTDI
Mega Pro 3.3V	ATmega2560	3,3-12V	3,3V	8MHz	54	16	14	4	256Kb	STK500v2	Cabecera compatible con FTDI
Mega Pro 5V	ATmega2560	5-12V	5V	16MHz	54	16	14	4	256Kb	STK500v2	Cabecera compatible con FTDI



Arduino Mini 04 (descontinuado)	ATmega328	7-9V	5V	16MHz	14	6	8	1	32Kb	AtmegaBOOT	Cabecera Serial
Arduino Mini 05	ATmega328	7-9V	5V	16MHz	14	6	8	1	32Kb	AtmegaBOOT	Cabecera Serial
Arduino Pro Mini 3.3V/8MHz	ATmega328	3,35-12V	3,3V	8MHz	14	6	6	1	32Kb	AtmegaBOOT	Cabecera compatible con FTDI
Arduino Pro Mini 5V/16MHz	ATmega328	5 - 12V	5V	16MHz	14	6	6	1	32Kb	AtmegaBOOT	Cabecera compatible con FTDI
Arduino Fio	ATmega328P	3,35-12V	3,3V	8MHz	14	8	6	1	32Kb	AtmegaBOOT	Cabecera compatible con FTDI o Inalámbrica vía XBee ¹
Mega Pro Mini 3.3V	ATmega2560	3,3-12V	3,3V	8MHz	54	16	14	4	256Kb	STK500v2	Cabecera compatible con FTDI
Pro Micro 5V/16MHz	ATmega32U4	5-12V	5V	16MHz	12	4	5	1	32Kb	DiskLoader	Nativa USB
Pro Micro 3.3V/8MHz	ATmega32U4	3,35-12V	3,3V	8MHz	12	4	5	1	32Kb	DiskLoader	Nativa USB
LilyPad Arduino 328 Main Board	ATmega328	2,7-5,5V	3,3V	8MHz	14	6	6	1	32Kb	AtmegaBOOT	Cabecera compatible con FTDI
LilyPad Arduino Simple Board	ATmega328	2,7-5,5V	3,3V	8MHz	9	4	5	0 ²	32Kb	AtmegaBOOT	Cabecera compatible con FTDI

En esta tabla se puede examinar la existencia de varias versiones para un mismo modelo. Esto se debe a las continuas actualizaciones que sufren y mejoran este tipo de tecnologías.

Las siguientes figuras permiten observar los modelos que contiene la "Tabla 3.2." expuesta con anterioridad.

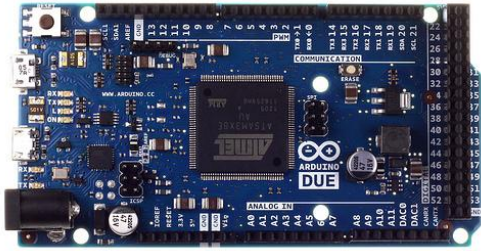


Figura 3.5. Arduino Due



Figura 3.6. Arduino Mega 2560

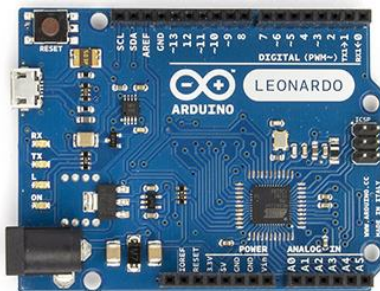


Figura 3.7. Arduino Leonardo

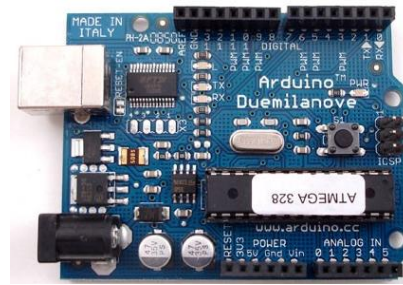


Figura 3.8. Arduino Duemilanove

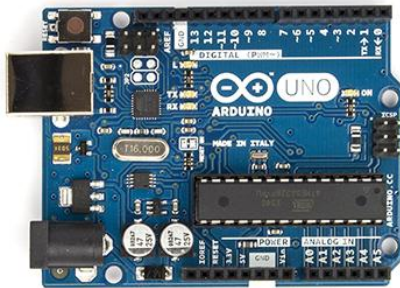


Figura 3.9. Arduino Uno

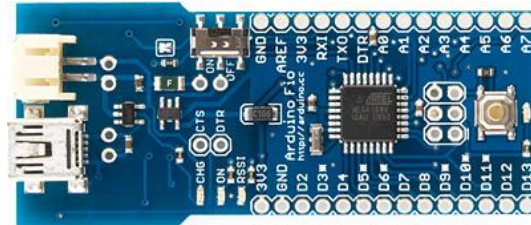


Figura 3.10. Arduino Fio

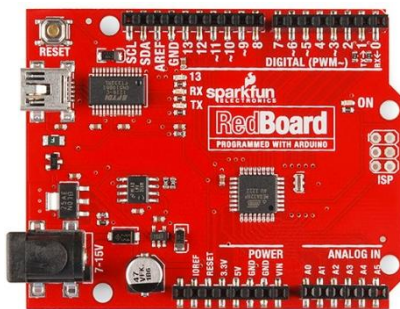


Figura 3.11. Arduino Red board

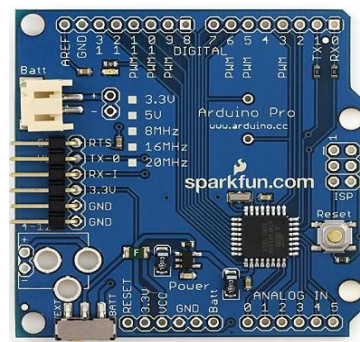


Figura 3.12. Arduino Pro

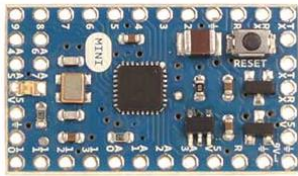


Figura 3.13. Arduino Mini

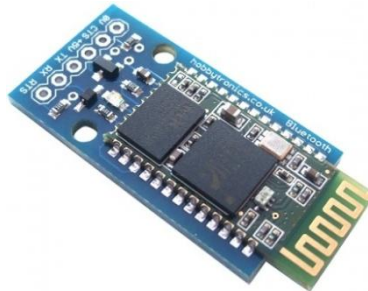


Figura 3.14. Arduino Bluetooth

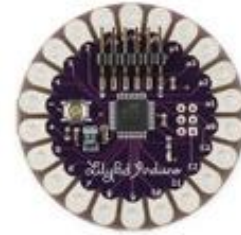


Figura 3.15. Arduino LilyPad

El módulo Arduino tiene la habilidad de ofrecernos dos opciones de desarrollo. La primera consta de la posibilidad de implementar el dispositivo para controlar objetos interactivos autónomos. La segunda opción permite, a través de un puerto USB, conectar el sistema embebido junto con un software, en un ordenador, para alcanzar el objetivo de examinar toda la información recopilada durante el funcionamiento del proceso a realizar. Este proyecto se ha declinado por utilizar la segunda opción de desarrollo, que es la que cubre con todas las necesidades reales que requiere este estudio. Consigue, de esta manera, que el sistema lea y trate todos los datos analíticos adquiridos de la motocicleta de competición durante su rodaje.

3.2.2.3. Placa electrónica Arduino Mega 2560 ADK (R3)

Para poner en práctica, en este trabajo, todos y cada uno de los cometidos que se exponen en la segunda opción de desarrollo escogida en el punto anterior, se toma la decisión de elegir uno de los modelos de placas que ofrece Arduino. En concreto, se selecciona la versión ADK (R3) del modelo Arduino Mega 2560 como dispositivo idóneo para este estudio. La elección de este módulo se justifica por los motivos expuestos a continuación: Debido a la cantidad y a la variedad en cuanto al tipo de dato recibido y a la gran carga de funciones que se tiene que realizar, era necesario que el dispositivo contase con el microcontrolador más potente de Arduino, consiguiendo, de este modo, trabajar con total fluidez y rapidez. Este dispositivo es de los más modernos ya que está preparado para interactuar con smartphones o tablets que dispongan de un sistema operativo Android, a causa de ello permitiría, si fuese necesario, cualquier innovación respecto al proyecto en un futuro. En cuanto a los soportes que se necesitan acoplar para acceder a la lectura y al tratamiento de los datos adquiridos, la placa tendría que ser la más grande en cuanto a entradas y salidas analógicas y digitales se tratase. Su tamaño, a pesar de la gran cantidad de pines que alberga, es reducido, por lo que contaremos con un módulo de lo más compacto. Por último, este sistema tiene una visión de conjunto de las más completas y su relación calidad-precio le convierte en uno de los mejores del mercado.



Figura 3.16. Arduino Mega 2560 ADK (R3)
(parte delantera)

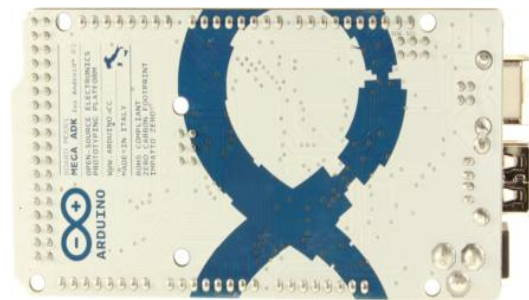


Figura 3.17. Arduino Mega 2560 ADK(R3)
(parte trasera)



La placa electrónica Arduino Mega 2560 ADK (R3) basa su funcionamiento en el microcontrolador Atmega2560. Es compatible con el Accessory Development Kit (ADK) de Android, permitiendo desarrollar accesorios y proyectos de control electrónico y robótico con un smartphone o tablet con sistema operativo Android. Para ello se modifica y se instala en la placa un puerto USB host interface para vincularlo con los dispositivos Android por medio del chip MAX3421E. Dispone de 54 pines digitales de entrada y salida, de los cuales 15 se usan para trabajar con señales de modulación de onda por pulsos, es decir, señales PWM (Pulse Wave Modulation). Cuenta con 16 entradas analógicas, 4 puertos serie (UARTs), un conector USB, pines para ICSP, un conector de alimentación Jack, un cristal de 16 Mhz que funciona como oscilador para el micro y un botón de RESET para resetear la placa de manera física.

Es importante resaltar que este módulo dispone de un conversor Serie/USB integrado, basado en un chip Atmega8U2 y de un LED, cuya referencia es el pin digital 13, que hace saber al usuario si un programa se carga y ejecuta de manera correcta.

Para este proyecto se utilizan 4 pines digitales que son: MISO, MOSI, SCK y SS; 2 pines de alimentación como son: GND y 5V; y uno de los puertos serie para realizar la conexión entre placa y ordenador.

❖ Características principales

En este apartado, se expone un resumen de las características más importantes y específicas de Arduino Mega 2560 ADK (R3).

- Microcontrolador: ATmega2560
- Voltaje de alimentación: 5V
- Voltaje de Entrada: 7-12V
- Voltaje límite de entrada (max): 6-20V
- Pines digitales: 54 (15 con PWM)
- Pines analógicos: 16
- Corriente por pin: 40 mA
- Corriente sobre pin 3,3V: 50 mA
- Memoria Flash: 256 KB (8 KB usados para el bootlader, (gestor de arranque))
- SRAM: 8 KB
- EEPROM: 4 KB
- Reloj: 16 MHz
- Puertos serie: 4
- USB Host chip: MAX3421E

❖ Características físicas

El módulo Arduino Mega ADK (R3) contiene unas dimensiones de 10,16 cm de largo por 5,33 cm de ancho. Es de aclarar, que debido a la colocación de los puertos serie, la dimensión relacionada con el largo de la placa se incrementa en unos cuantos milímetros más.

Esta placa cuenta con seis orificios para que, a través de una serie de tornillos, se instale y acople una superficie que ejecute la función de carcasa, protegiendo de este modo el sistema electrónico.

❖ Fuente de alimentación de la placa

Este dispositivo electrónico permite ser alimentado de dos modos. Uno de ellos es a través de una fuente de alimentación externa, ya bien sea por medio de una batería o un transformador. La instalación de la batería se realiza a partir del conexionado de sus cables con los conectores de alimentación GND y VIN. El acople del transformador se efectúa mediante un conector macho de 2,1 mm con centro positivo en el conector de alimentación de la placa. El segundo modo de alimentación es el proporcionado a través de un ordenador mediante una conexión USB.

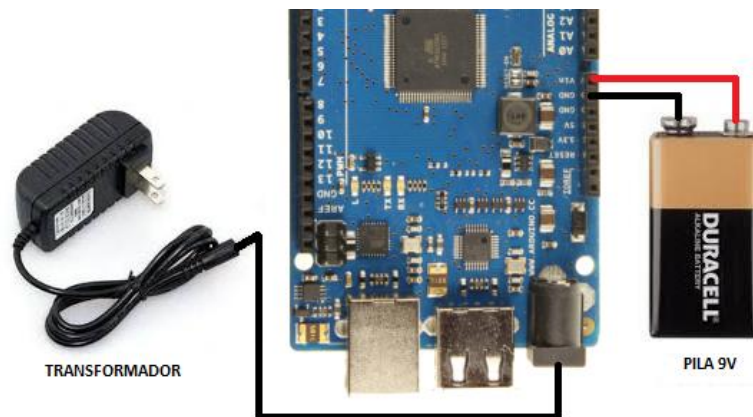


Figura 3.18. Fuentes de alimentación externa (conexionado)

Este proyecto se decantó por utilizar el conexionado USB como vía de alimentación, ya que para desarrollar la lectura, tratamiento, monitorización y análisis de datos se necesita trabajar en un puesto que cuente con un ordenador.



Figura 3.19. USB como fuente de alimentación (conexionado)

Hay que tener en cuenta que este modelo de placa dispone de un Host USB y por tanto permite interactuar con dispositivos y sistemas Android. No es el caso de este estudio, pero si se utilizan dichos dispositivos y sistemas se debe saber que extraerán y utilizarán la energía de la propia placa Arduino cuando estos estén conectados a ella. Esto sucederá tanto si la placa se alimenta a través del mini USB Host o por medio de una fuente de alimentación externa.

El módulo arduino puede desempeñar su trabajo con una alimentación externa de entre 6 a 20 voltios. En el caso de suministrar un voltaje inferior a los 7V, el pin de 5V podría abastecer a los elementos periféricos, conectados a la placa, con un voltaje inferior a los 5V, produciendo una inestabilidad en dicha placa. Sin embargo, si se emplean más de 12V, los reguladores de voltaje se pueden dañar a causa del sobrecalentamiento, dañando también la placa. Es por ello, que el rango de alimentación recomendado para Arduino es de 7 a 12 voltios.

❖ Estudio externo de la placa

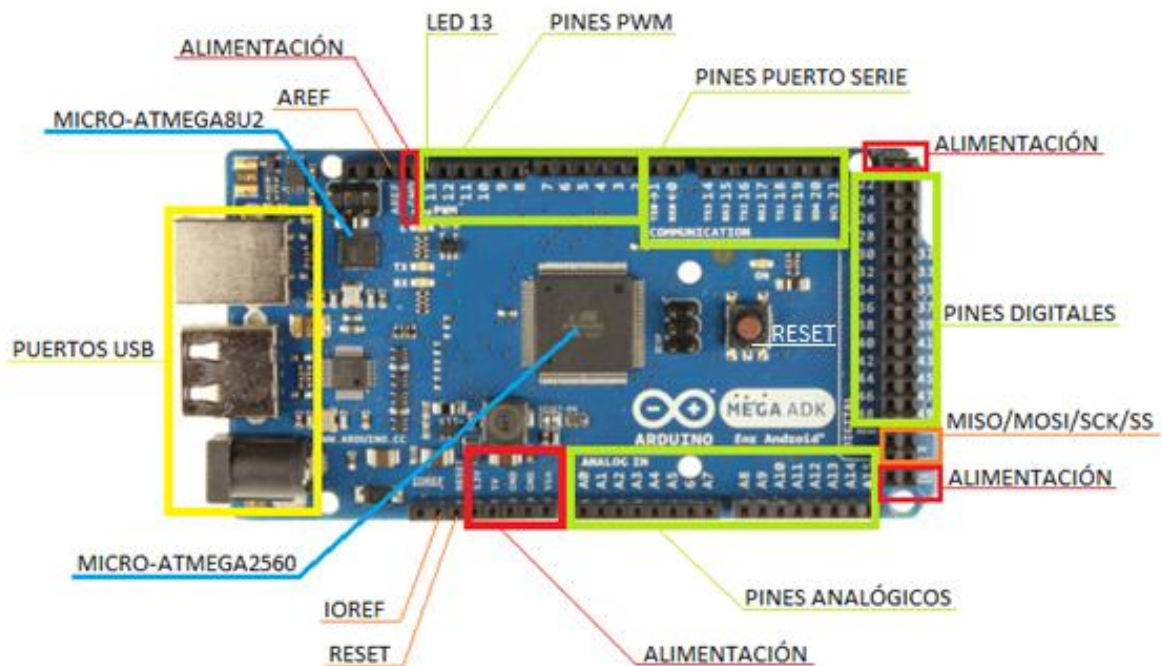


Figura 3.20. Diagrama externo de Arduino Mega ADK (R3)

- **Puertos USB:** Los puertos serie mostrados en el diagrama y ordenados de arriba a abajo son: puerto USB BM (macho), puerto mini USB Host (macho) y puerto USB Jack (macho). Para la ejecución de este trabajo de fin de grado ha sido utilizado el puerto USB BM (macho).
- **Micro-ATMEGA2560:** Este microcontrolador es el cerebro de la placa, se encarga de realizar todo el procesamiento de las funciones y datos con las que va a trabajar. Sus especificaciones técnicas junto con su pin mapping (diagrama de pines) pueden ser consultados y analizados en el anexo 2.
- **Micro-ATMEGA8U2:** El siguiente microcontrolador es el encargado de la conversión Serie/USB. Sus especificaciones técnicas se adjuntan, para poder ser examinadas, en el anexo 1.
- **Botón reset:** Esta placa incorpora un pulsador que al ser accionado de manera física resetea todo el dispositivo electrónico. Gracias a su implementación, ya no es necesario realizar siempre esta acción desde el programa de desarrollo con el que cuenta Arduino.
- **Pines analógicos:** El Mega ADK contiene 16 entradas analógicas, las cuales proporcionan una resolución de 10 bits (1024 valores). Los pines analógicos se pueden identificar por una nomenclatura alfanumérica, cuyo comienzo es siempre "A". Estos zócalos se inician en "A0" y finalizan en "A15". Por defecto la medición se ejecuta de tierra a 5 voltios, aunque existe la posibilidad de modificar el extremo superior de este rango utilizando el pin AREF y la función *analogReference()*.



- **Pines digitales:** Cada uno de los 54 pines digitales en el Arduino Mega ADK son empleados como una entrada o salida utilizando las funciones *pinMode()*, *digitalWrite()* y *digitalRead()*. Cada pin funciona a 5 voltios, pueden recibir o suministrar una intensidad máxima de 40 mA y cuentan con una resistencia interna, desconectada por defecto, de 20 a 50 kOhms. Además, exceptuando los pines digitales que están comprendidos entre los números 22 y 49, ambos inclusive, el resto tienen funciones especiales y los analizaremos a continuación:
 - ❑ **Pines PWM:** Su ubicación está situada entre los pines 0 y 13, ambos incluidos. Estos conectores aportan señales de modulación de onda por pulsos (PWM, Pulse Wave Modulation) de 8 bits de resolución (Valores de 0 a 255), empleando la función *analogWrite()*.
 - ❑ **Pines puertos series:** Los pines de comunicación están organizados de la siguiente manera: Serie 0: Pin 0 (RX) y 1 (TX); Serie 1: Pin 19 (RX) y 18 (TX); Serie 2: Pin 17 (RX) y 16 (TX); Serie 3: Pin 15 (RX) y 14 (TX). Se utilizan para recibir (RX) y transmitir (TX) datos a través de un puerto serie TTL. Los pines Serie 0: Pin 0 (RX) y 1 (TX) también están conectados a los pines correspondientes del micro ATmega8U2 USB a TTL, es decir, interactúan mediante la conexión USB con el ordenador. Cabe destacar también, la existencia de los pines de comunicación 20 (SDA) y 21 (SCL), los cuales permiten la comunicación TWI (Two Wired Serial Interface) a través de la librería Wire que proporciona Arduino. Estas conexiones se encargan de comunicar circuitos integrados que comparten una misma placa.
 - ❑ **Pines SPI:** Estos pines son capaces de admitir la comunicación SPI (Serial Peripheral Interface), la cual se puede desarrollar utilizando la librería SPI de Arduino. Este grupo está formado por 4 conectores que son: Pin 50 (MISO), 51 (MOSI), 52 (SCK) y 53 (SS) y se encargan de acondicionar la transferencia de información entre circuitos integrados de equipos electrónicos. En este proyecto son utilizados con el fin de transferir los datos de una tarjeta SD al microcontrolador ATmega2560 de la placa para su posterior lectura y tratamiento.
 - ❑ **Pines interrupciones:** La distribución de estos pines es bastante aleatoria a lo largo de toda la placa Mega ADK. Tienen la capacidad de habilitar interrupciones externas y se corresponden con el siguiente conexionado de pines: 2 (interrupción 0), 3 (interrupción 1), 18 (interrupción 5), 19 (interrupción 4), 20 (interrupción 3) y 21 (interrupción 2). Estos pines pueden ser configurados para activar una interrupción en un valor bajo (LOW, 0V), en flancos de subida o bajada (LOW → HIGH (5V) o viceversa) o en cambios de valor. Para realizar todo este trabajo se usa la función *attachInterrupt()*.



- **Pines alimentación:** los pines encargados de alimentar los elementos periféricos están formados por los siguientes tipos:
 - ❑ **3.3V:** Este tipo de pin suministra un voltaje de 3,3 voltios generado por el regulador de la placa. La corriente máxima que soporta el conector es de 50 mA. Esta placa cuenta con tan sólo uno de estos pines y está ubicado en la zona donde se encuentra el grueso de los pines de alimentación.
 - ❑ **5V:** Este modelo está formado por un grupo de 3 pines, los cuales están localizados de la siguiente manera, uno en la zona de los pines de alimentación y otros dos situados en la franja de los pines digitales que no disponen de funciones especiales. Representan la fuente de voltaje que suministra la energía necesaria para la alimentación del microcontrolador y otros elementos de la placa. Dicha energía puede ser originada a partir de un USB, una fuente externa estabilizada a 5V o puede ser proporcionada por VIN mediante un regulador integrado en la placa. Para la alimentación de este proyecto se cuenta con los servicios de uno de ellos.
 - ❑ **VIN:** Cuando se usa una fuente de alimentación externa, VIN es la entrada de voltaje a la placa Mega ADK. Se puede suministrar voltaje a través de este pin o, si está alimentado mediante la toma de 2,1mm, acceder a ella a partir de este pin. Solo hay un pin de este tipo y está ubicado en el grupo de alimentación.
 - ❑ **GND:** Son los pines de conexión a tierra. La placa alberga un grupo con un total de 5 pines distribuidos de la siguiente forma: Dos en la zona de alimentación, dos junto a los pines digitales sin funcionalidad especial y uno situado en la zona de los pines PWM. Para el funcionamiento de este estudio se ha hecho uso de uno de estos pines.
 - ❑ **IOREF:** Este modelo proporciona la medida de tensión con la que opera el microcontrolador de la placa Arduino. Un dispositivo periférico puede configurarse para leer el voltaje y seleccionar la fuente de alimentación adecuada a través de este pin. Solo existe uno de este tipo y se encuentra junto a los pines de alimentación.
- **Otros pines:** Debido a la importancia de sus funciones, es necesario mencionar los siguientes pines:
 - ❑ **Reset:** Ubicado junto a la zona de alimentación, es usado para proporcionar un valor de 0V para reiniciar el microcontrolador.
 - ❑ **LED 13:** Se trata de un led integrado en la placa que se encuentra junto a los leds de comunicación serie (TX) y (RX) y está conectado al pin digital 13. Si el led se enciende el pin se encuentra HIGH (5V) y si se apaga LOW (0V).
 - ❑ **AREF:** Situado junto a los pines PWM, es el voltaje de referencia para las entradas analógicas. Se utiliza mediante la función *analogReference()*.



❖ Protección de la placa

Arduino Mega ADK (R3) es un módulo consciente de la importancia que supone la protección de los equipos electrónicos. Debido a ello dispone de un fusible reajutable que protege de cortocircuitos y sobrecargas a los puertos USB de los ordenadores a los que este conectado la placa. A pesar de que la mayoría de los ordenadores cuentan con su propia protección interna, el fusible de la placa presta un servicio adicional de protección. Si el puerto USB percibe más de 500 mA, el fusible corta de forma automática la conexión hasta que se elimine la sobrecarga o el cortocircuito.

❖ Comunicación de la placa

Este dispositivo electrónico permite comunicarse de un modo sencillo con un ordenador, otro Arduino u otros microcontroladores. En el caso de este proyecto, la comunicación se realiza a través de un ordenador mediante conexión USB. Para que esta conexión sea posible se utiliza el puerto serie 0, que es uno de los encargados que hace posible la transferencia de datos de la placa al ordenador. De este modo los datos quedan preparados para efectuar su lectura y tratamiento desde el entorno de desarrollo que ofrece Arduino para el ordenador.

3.2.2.4. SD card shield

Otro de los componentes trascendentes e importantes que forman el hardware de tratamiento es la SD card shield. En este trabajo de fin de grado se necesita un periférico que transfiera la información recopilada sobre la motocicleta de competición entre la micro tarjeta SD y el microcontrolador de la placa Arduino Mega ADK (R3). Para ello se optó por elegir una de las opciones disponibles de Seed Studio. Dicha opción se transformó en el modelo SD card Shield y en especial en la versión V3.1. Los motivos que llevaron a tal decisión fueron la compatibilidad que ofrecían estos módulos con Arduino y la posibilidad de trabajar tanto con tarjetas SD como con micro tarjetas SD.

El SD card shield V3.1 es un dispositivo auxiliar e independiente, cuya función de transferencia permite el traspaso de datos entre dos elementos electrónicos. Este modelo es compatible con tarjetas SD, SDHC o MicroSD. Dispone de un interruptor de palanca integrado en el dispositivo para seleccionar, antes de comenzar a trabajar, el tipo de tarjeta SD. Este terminal está preparado para la comunicación SPI (Serial Peripheral Interface) y TWI (Two Wired Serial Interface, I2C), pero sólo utiliza los puertos SPI de Arduino. También cuenta con conectores Grove compatibles para los puertos TWI (I2C) y UART.

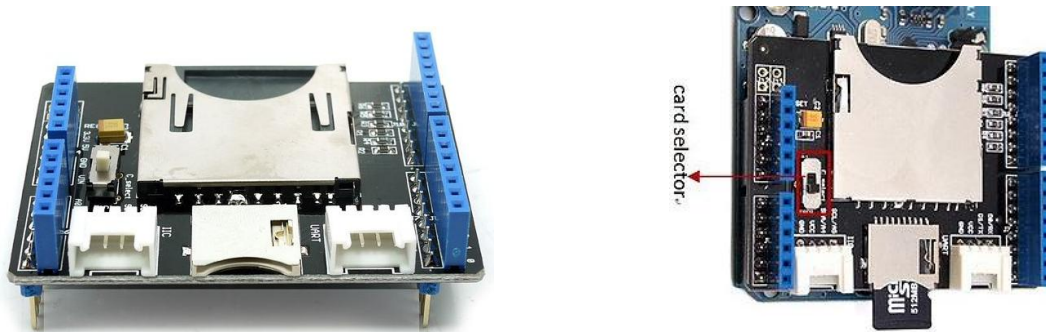


Figura 3.21. SD card shield V3.1

❖ Características principales

A continuación, se observa un resumen de las características más importantes de SD card shield V3.1.

- Compatibilidad con Arduino
- Compatibilidad con conectores Grove
- Compatible con tarjetas de formato SD, SDHC y MicroSD
- Voltaje de pines de 3,3V y 5V (voltaje lógico)
- Rango de voltaje por fuente de alimentación de 2.6V a 3,6V

❖ Características específicas

Tabla 3.3. Especificación

Artículo	Min	Típico	Max	Unidad
Voltaje	2.7	3.3	3.6	V
Corriente	0,159	40	200	mA
Tipo de tarjeta	Tarjeta SD (<= 2G); Tarjeta Micro SD (<= 2G); Tarjeta SDHC (<= 16G)			/
Dimensión	57.15x44.70x19.00			mm
Peso Neto	16.6			g

❖ Estudio externo de la placa SD card shield

Se realiza una descripción física de los distintos elementos que forman la placa Sd card shield V3.1 utilizada para este proyecto.

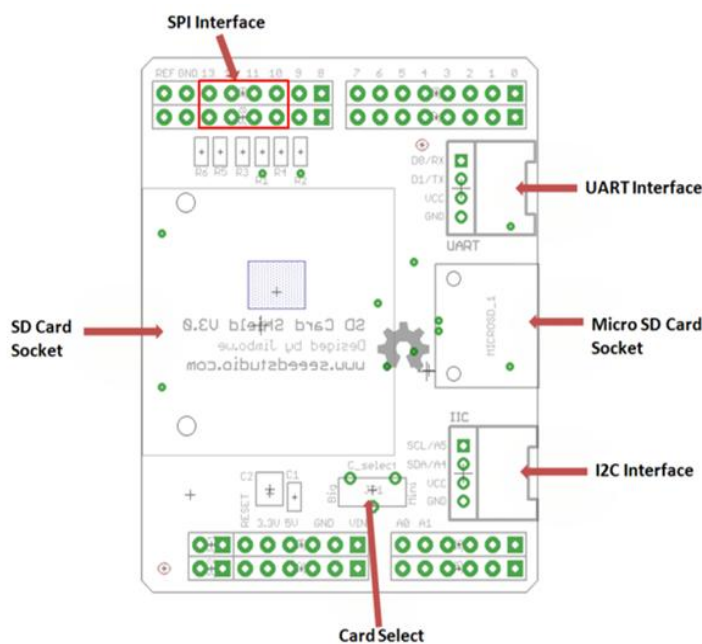


Figura 3.22. Diagrama externo SD card shield V3.1

Este dispositivo cuenta con 28 pines distribuidos en 14 pines digitales (pin 10, 11, 12 y 13 para comunicación SPI), 7 pines de alimentación, 6 pines analógicos y 1 pin destinado al reset. Dispone de un soporte para tarjetas SD y otro para micro tarjetas SD. Contiene dos conectores Grove, uno para puerto UART y otro para puerto I2C. Por último, incorpora una palanca para seleccionar la tarjeta incorporada.

3.2.2.5. Micro tarjeta SD

En este trabajo, la tarjeta SD a pesar de pertenecer a los elementos más sencillos del grupo que representa el hardware de tratamiento, es uno de los dispositivos electrónicos más importantes. Es el medio encargado de contener y transportar todos los datos adquiridos de la motocicleta de competición al puesto de análisis del sistema telemétrico.

El formato de tarjeta escogido es la MicroSD y el modelo seleccionado fue una SanDisk de 2 GB. Las razones sobre la elección de esta micro tarjeta se deben a su tamaño (15 mm x 11 mm), que nos permite recopilar toda la información necesaria de la prueba sin ocupar un gran espacio físico, y a su capacidad de almacenamiento, que cuenta con la máxima permitida por las especificaciones de la SD card shield donde es acoplada.



Figura 3.23. MicroSD SanDisk

La MicroSD está diseñada para soportar las condiciones más duras, son tarjetas de memoria resistentes a los golpes, al agua e incluso a una prueba de rayos X. Pueden operar en un rango de temperaturas entre los 13 y 185 grados Fahrenheit, logrando que se disfrute de los archivos de este dispositivo en casi cualquier clima.

3.2.2.6. Adaptador de micro tarjeta SD

Este dispositivo electrónico sirve, como su nombre indica, para adaptar la tarjeta MicroSD, expuesta con anterioridad, al ordenador. El adaptador de tarjeta se utiliza en este proyecto como



Figura 3.24. Adaptador de micro tarjeta SD

complemento al trabajo o a las funciones que se pueden realizar con la MicroSD. Gracias a él se puede observar en la pantalla del ordenador, sin necesidad de ninguno de los demás elementos hardware, los datos que almacena la micro tarjeta, y por consiguiente, analizar si la escritura de los datos ha sido correcta o formatear y limpiar la información existente para dejar preparada la MicroSD para otra posible sesión de datos de la motocicleta.

La decisión que se toma para la elección del adaptador sigue los mismos pasos que los dados con la tarjeta MicroSD, es decir, se elige un adaptador de la marca SanDisk. En concreto se escoge el modelo SDXCAdapter, compatible con micro tarjetas SD, SDHC y SDXC. Dispone de unas dimensiones de (3.2 cm x 2.4 cm x 0.1cm) y de un peso de 1 gramo. Los motivos de su elección, en esta ocasión, no tienen ninguna singularidad, ya que cualquier dispositivo que desempeñe esta función es válido.

3.2.2.7. Cable USB

Por último mencionar el componente que cierra el grupo de los elementos del hardware de tratamiento. El cable USB se utiliza para comunicar el ordenador con todo el grueso del hardware de este proyecto, y así abastecerlo de energía. El cable, un USB 2.0, está formado por una conexión de tipo A (macho) a tipo B (hembra) y para este estudio vale con cualquier modelo que disponga de estas características.



Figura 3.25. Cable USB



3.3. SOFTWARE DEL SISTEMA

La composición del software está formada, a su vez, por dos grupos de entornos de desarrollo integrado. El primer grupo es denominado como software de adquisición y se encarga de la creación de un programa que escriba sobre una micro tarjeta SD todas las medidas que nos proporcionan la información necesaria sobre la motocicleta. El segundo grupo se define como software de tratamiento y tiene la función de elaborar programas capaces de leer, procesar y monitorizar toda la información recopilada en la micro tarjeta SD, y que es transferida por el hardware de tratamiento. Los programas informáticos que forman ambos grupos se observan en la tabla siguiente:

Tabla 3.4. Estructura del software del sistema de telemetría

SOFTWARE DE ADQUISICIÓN	SOFTWARE DE TRATAMIENTO
<ul style="list-style-type: none"> ➤ Entorno de desarrollo integrado → Plataforma Arduino (escritura de datos) 	<ul style="list-style-type: none"> ➤ Entorno de desarrollo integrado → Plataforma Arduino (lectura de datos)
	<ul style="list-style-type: none"> ➤ Entorno de desarrollo integrado → Plataforma MATLAB (procesado de datos)
	<ul style="list-style-type: none"> ➤ Entorno de desarrollo integrado → Plataforma MATLAB <ul style="list-style-type: none"> • GUIDE (monitorizado de datos)

3.3.1. SOFTWARE DE ADQUISICIÓN

Al igual que sucede en el "Hardware de adquisición" perteneciente al epígrafe "3.2.1.", el software de adquisición no es desarrollado en este trabajo de fin de grado. No obstante, es necesario realizar una pequeña introducción explicativa sobre el mismo para posteriormente entender, de un modo correcto, el funcionamiento del software de tratamiento, el cual si se desarrolla en el presente informe. Para comprender mejor esta idea es preciso examinar, de un modo resumido, la función que desempeña la programación del software de adquisición mencionada en la "tabla 3.4."

Este tipo de software utiliza un entorno de desarrollo habilitado por la plataforma Arduino. Su finalidad es la programación de funciones que calculen las medidas necesarias a partir de los datos adquiridos a través de los sensores que están distribuidos por la motocicleta de competición. Una vez conseguido este propósito, se desarrolla una función general de escritura que englobe a las funciones programadas para las medidas de cada sensor e inscriba dichas medidas, siendo colocadas y diferenciadas por columnas, en la micro tarjeta SD. Como última observación a la función general, la última columna inscrita determina el número de datos que se han obtenido en la prueba. Una vez programado el almacenamiento de toda la información, queda preparada para su posterior programación en cuanto a lectura, tratamiento y monitorización.



3.3.2. SOFTWARE DE TRATAMIENTO

El software de tratamiento, a pesar de constituir uno de los dos grupos que dan forma al armazón del software del sistema de telemetría, es el único que, en realidad, cumple con los cometidos de este proyecto. Una vez escritos los datos y acondicionado todo el hardware de tratamiento, se completa la lectura de esta información programando y desarrollando una función de lectura a través del entorno de desarrollo integrado proporcionado por la plataforma Arduino. Con la lectura de los datos finalizada, estos quedan competentes para su posterior procesamiento y monitorización. Estas dos funciones son programadas mediante el entorno de desarrollo que nos ofrece MATLAB y MATLAB (GUIDE) respectivamente. De este modo, las medidas quedan preparadas para la realización de un análisis a través de la interfaz gráfica programada, también, por MATLAB (GUIDE). En este epígrafe se examina en profundidad los entornos de desarrollo integrado utilizados en el software de tratamiento y expuestos en la "Tabla 3.4."

3.3.2.1. Entorno de desarrollo integrado

Un entorno de desarrollo integrado, denominado como IDE (Integrated development environment), es un programa informático formado por un conjunto de herramientas de programación. Dicho programa está empaquetado como un programa de aplicación, es decir, está compuesto por elementos como: un editor de código, que permite crear y modificar archivos digitales como archivos de configuración, scripts o el código fuente de algún programa; un compilador capaz de traducir el lenguaje de programación de un programa escrito en otro lenguaje de programación más manejable para un ordenador; un depurador utilizado para probar y depurar (eliminar) los errores hallados en un programa, y un constructor de interfaz gráfica, que utiliza un conjunto de imágenes y objetos para representar la información disponible en un entorno visual. Los IDE pueden ser aplicaciones por sí mismas o pueden ser parte de aplicaciones ya existentes. Cuentan con un marco de trabajo compatible con la mayoría de los lenguajes de programación, teniendo la capacidad de trabajar con uno o varios de estos lenguajes: C++, C#, Java, PHP, Python, Delphi, Visual Basic, etc.

3.3.2.2. Entorno de desarrollo integrado de la plataforma Arduino

La elección de este entorno de desarrollo tiene su justificación en la posibilidad que tiene la plataforma Arduino para compatibilizar la comunicación entre componentes hardware y software. En este proyecto se consigue, gracias a ello y mediante la comunicación proporcionada por el puerto serie 0 de la placa Arduino, la conexión entre el software de tratamiento y el hardware de tratamiento.

El software de Arduino consiste en un entorno de desarrollo que está basado en el lenguaje de programación Processing, el cual se emplea como medio para la enseñanza y producción de proyectos multimedia de diseño digital. Debido a la transmisión de datos por puerto serie que permite Arduino, puede soportar el uso de otros lenguajes de programación. Sin embargo, para aquellos lenguajes que no soportan el formato serie, se puede utilizar un software intermediario que traduzca los mensajes enviados para conseguir una comunicación más fluida. Algunos ejemplos son: Adobe Director, C, C++, C#, Flash, Isadora, Instant Reality, Java, Liblertlab, Mathematica, MATLAB, MaxMSP, Php, Physical Etoys, VBScript, Visual Basic, etc.

❖ Instalación y preparación del entorno de desarrollo integrado

Para poder disponer y acceder a este entorno de desarrollo, Arduino lo facilita de forma gratuita a través de su página web oficial. Dependiendo del sistema operativo se puede elegir la versión, las cuales se van actualizando de forma temporal.

Una vez descargado e instalado el entorno de desarrollo, se prepara para poder trabajar con él. Para ello, se conecta la placa Arduino Mega 2560 ADK (R3) con el ordenador y se abre el entorno de desarrollo, también, de Arduino. A continuación, primero se selecciona el tipo de placa. Para ello se pulsa el menú "Herramientas", opción "Tarjeta" y se selecciona el modelo de placa que se utiliza. En este caso, se escoge la opción Arduino Mega 2560 or Mega ADK.

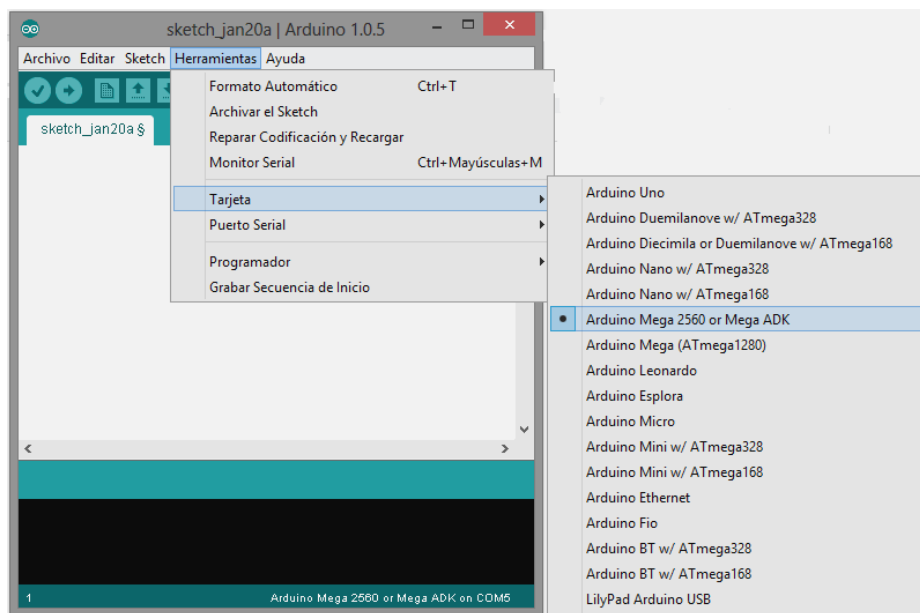


Figura 3.26. Elección de la placa Arduino

Resuelta la selección de la placa, es necesario seleccionar el puerto serie donde se encuentra la misma para poder comenzar la comunicación. Para ello volvemos a pulsar el menú "Herramientas", opción "Puerto Serial" y se selecciona o se muestra el tipo de puerto que se utiliza.

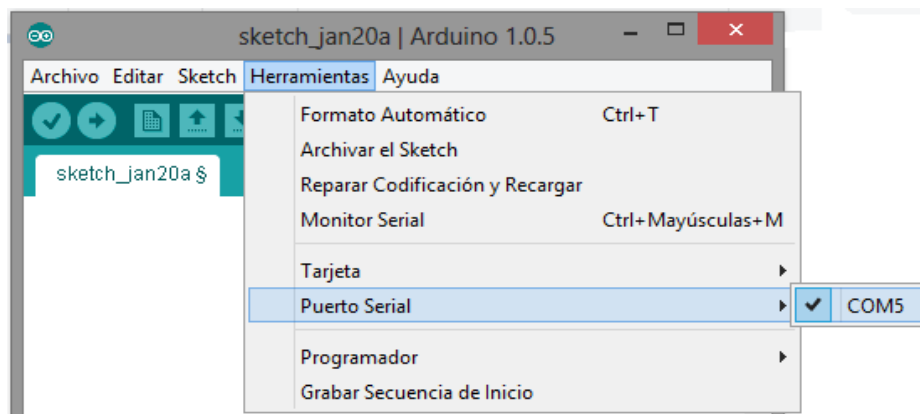


Figura 3.27. Elección del puerto serial

❖ Estudio del entorno de desarrollo integrado

La plataforma Arduino dispone de un entorno de trabajo compuesto por: un editor de texto empleado para escribir el código de un programa, denominado "Sketch"; un área de mensajes, cuya función es la de mostrar información y errores acerca de los programas mientras estos se cargan; una consola de texto, que se encarga de mostrar el texto de salida para el entorno Arduino, incluyendo mensajes de error completos y otras informaciones; una barra de herramientas con diferentes botones para las funciones comunes, que posibilita

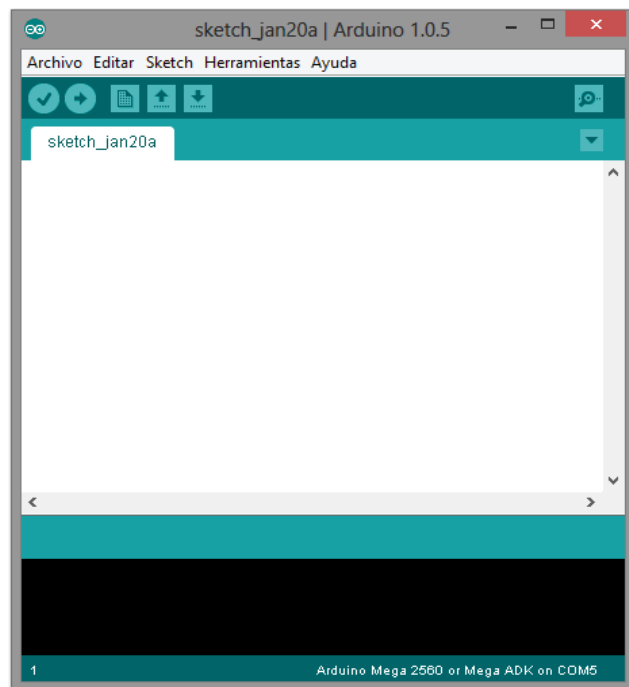


Figura 3.28. Entorno de desarrollo integrado Arduino







confirmar el proceso de carga, creación, apertura, guardado y monitorización serie de programas y por último, una serie de menús desplegables con distintas opciones.

Acto seguido, se examinan e ilustran los diferentes elementos que forman este entorno:

- **Barra de herramientas:** Seis botones dan forma y componen la barra de herramientas de este entorno de desarrollo integrado.



Figura 3.29. Barra de herramientas

-  Chequea el código del sketch programado en busca de errores.
-  Compila el código y lo carga en el microcontrolador de la placa Arduino utilizada. En el caso de este proyecto en la placa Arduino Mega 2560 ADK (R3).
-  Crea un nuevo programa (sketch).
-  Accede a un menú con todos los programas que contiene la librería "sketchbook". Al pulsar en uno de ellos se abre en la ventana actual.
-  Guarda el programa que se realiza.
-  Abre una ventana nueva, correspondiente al monitor serie, iniciando la monitorización de los resultados del programa, obtenidos de la placa Arduino.



- **Barra de menús:** Está compuesta por un grupo de cinco menús desplegables, los cuales son sensibles al contexto, es decir, estarán disponibles aquellas opciones relevantes para el trabajo que se esté realizando en ese momento. Cada menú está representado en la "Figura 3.30."

Archivo Editar Sketch Herramientas Ayuda

Figura 3.30. Barra de menús

- ❏ **Menú Archivo:** Dispone de opciones para trabajar con archivos como: crear, abrir, cerrar, guardar, configurar, sketchbook, imprimir y salir. También cuenta con dos opciones más que destacan sobre el resto y son:
 - **Ejemplos:** Opción que habilita la utilización de programas ya creados, con el fin de aprender y observar el funcionamiento del "mundo Arduino".
 - **Preferencias:** Opción que permite elegir donde guardar los programas y como realizar ciertos aspectos de su programación.

- ❏ **Menú Editar:** Está formado con elementos de edición de archivo como: deshacer, rehacer, cortar, copiar, pegar, comentar, descomentar, incrementar y reducir margen, seleccionar y buscar. Las opciones que destacamos en este menú son las dos siguientes:
 - **Copiar para el foro:** Se encarga de copiar, con el formato adecuado y sintaxis coloreada, el código del sketch para publicarlo en el foro.
 - **Copiar como HTML:** Copia el código del sketch adecuándolo al formato HTML para "subirlo" en una página web.

- ❏ **Menú Sketch:** Para el tercer menú destacamos las cuatro opciones con las que cuenta.
 - **Verificar/compilar:** Comprueba los errores del programa (sketch).
 - **Mostrar la carpeta de sketch:** Abre la carpeta de programas en el escritorio.
 - **Agregar archivo:** Se encarga de añadir un fichero fuente al programa.
 - **Importar librería:** Permite utilizar librerías, incluyéndolas al principio del código, para agilizar y simplificar la programación.

- ❏ **Menú Herramientas:** Compuesto por opciones como: archivar, reparar, recargar, monitorizar, grabar y programador. A continuación examinamos otras tres opciones importantes:
 - **Formato automático:** Proporciona al código un formato estético, es decir, organiza la programación mediante llaves y tabulaciones.
 - **Tarjeta:** Permite seleccionar la placa con la que se trabaja. Es indispensable para la comunicación entre la placa y el entorno de desarrollo Arduino.
 - **Puerto Serial:** Identifica el nombre del puerto Serie con el que estamos operando.

- ❏ **Menú Ayuda:** Consiste en elementos de ayuda conectados a internet para entender y comprender el entorno de desarrollo de Arduino. Todos los elementos resultan importantes de cara al aprendizaje de Arduino.

- **Entorno de ventanas:** En este apartado se quiere diferenciar, de una forma clara, la distribución del editor de texto, el área de mensajes y la consola de texto, los tres elementos que faltaban por completar este entorno de desarrollo. Para ello se ha utilizado como ejemplo la "Figura 3.31.", que contiene una parte del programa "LecturaTarjetaSD" desarrollado para este proyecto.

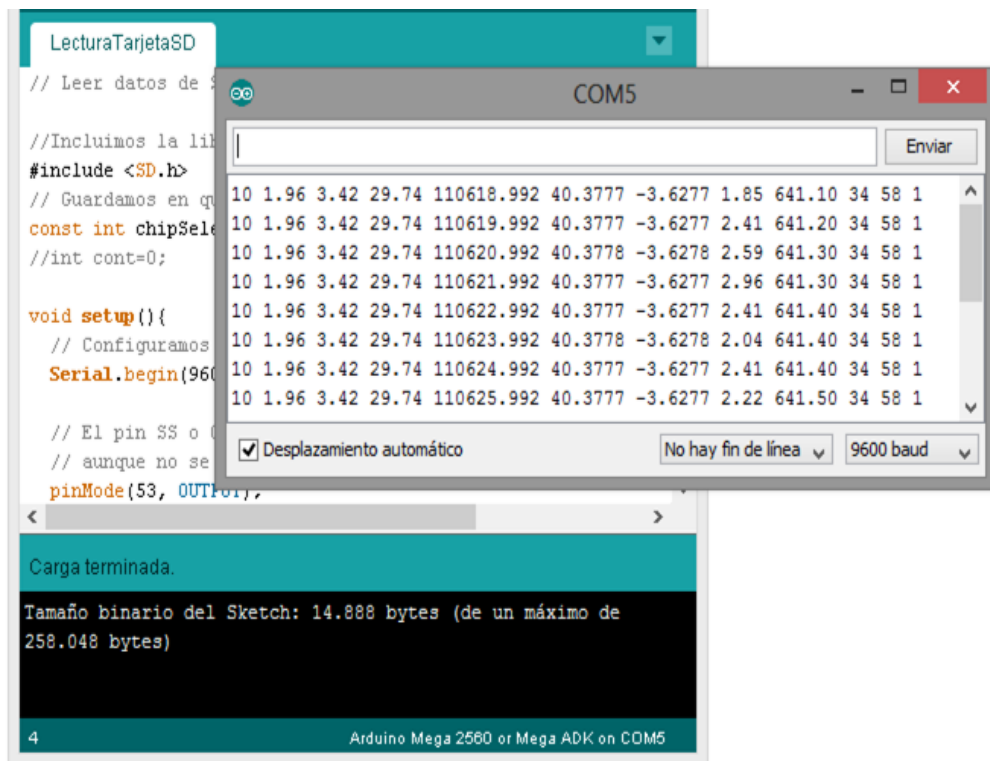



Figura 3.31. Entorno de ventanas del entorno de desarrollo integrado

El editor de texto es toda la zona blanca, tras el monitor serie (marco gris), donde se encuentra escrito el código del programa, la cual incluye una pestaña con el nombre de dicho programa.

El área de mensajes es la zona azul turquesa, en la cual se puede leer "Carga terminada". En esta barra también se puede observar un cambio de color según sean los mensajes que transmita.

La consola es la zona negra que apreciamos en la "Figura 3.31.". Se aprecia información sobre el código y los errores del programa.

Se aprovecha este espacio, dedicado a la distribución del entorno de ventanas, para destacar otros dos elementos importantes. El primero de ellos es el monitor serie, se accede a él mediante el botón  de la barra de herramientas de la "Figura 3.29.". Se trata de la ventana (COM5) de marco gris que contiene el resultado del programa cargado, y que se puede observar en la "Figura 3.31.".

El segundo elemento a destacar es la última franja de la "Figura 3.31.", en ella se encuentra la siguiente información. En el lado izquierdo se observa el número de línea de código en el que se encuentra el programa y en el lado derecho se aprecia el nombre de la placa Arduino y del puerto serie utilizado.



3.3.2.3. Entorno de desarrollo integrado de la plataforma MATLAB

MATLAB se crea en 1984 por el estadounidense Cleve Moler, un prestigioso matemático y programador de computadoras. Su nombre proviene de la abreviatura de MATrix LABoratory (laboratorio de matrices). La primera versión nace con la idea de desarrollar la programación de esta plataforma con paquetes de subrutinas escritas en lenguaje Fortran. Su resultado se dejó notar en el software de matrices de los proyectos LINPACK y EISPACK desarrollados en el Argonne National Laboratory. Con el transcurso de los años se fue complementando y mejorando con lenguaje M (lenguaje propio de MATLAB) para proporcionar un software de uso más sencillo. En la actualidad MathWorks Inc es propietario de su licencia.

MATLAB es una potente herramienta de software matemático e interactivo destinado al diseño de aplicaciones científico-técnicas. Utiliza un poderoso lenguaje industrial orientado a la resolución de problemas teóricos y reales, que permite al entorno de la ingeniería e investigación expresar sus ideas técnicas con simplicidad. Se trata de un entorno de desarrollo integrado, el cual desarrolla computación numérica y visualización gráfica de datos en 2D y 3D. Está basado en un sofisticado software de matrices, que integra computación matricial, representación de datos y funciones, implementación de algoritmos, proceso y análisis de señal y creación de interfaces de usuario (GUI (graphical user interface)). Cuenta también con una extensa librería, la cual contiene multitud de funciones matemáticas, estadísticas y científicas utilizadas dentro de la computación. MATLAB contiene un amplio abanico de programas de apoyo especializado, denominado Toolboxes, que amplían sus capacidades. El paquete MATLAB dispone de dos elementos adicionales que expanden sus prestaciones. Uno de ellos es Simulink, que es una plataforma de simulación multidominio, la cual se complementa con los paquetes de bloques (blocksets). El otro elemento se trata de GUIDE, que es un editor de interfaces de usuario (GUI). Por último destacar que MATLAB está disponible para las plataformas Unix, Mac OS X, GNU/Linux y Windows.

Por todo ello, las razones por las que hemos elegido MATLAB 7.10.0 (R2010a) quedan presentes y justificadas. No obstante, también es de destacar que su selección viene precedida por los buenos resultados obtenidos por muchas universidades, empresas y centros de investigación y desarrollo a la hora de utilizar sus servicios.

❖ Lenguaje de programación

El lenguaje de programación utilizado por MATLAB es del tipo interpretado o de scripting. Un "script" es una serie de ordenes escritas en un determinado lenguaje, que son tratadas en un intérprete para que las ejecute. Este lenguaje no cuenta con un compilador, pero debido a ello tiene la gran ventaja de poder modificar una rutina o función sin interrumpir la ejecución del programa. Esto significa que cuando se llame a la rutina deseada, se ejecutará sin necesidad de compilaciones ni enlazados. Tampoco es necesario reservar memoria como ocurre en otros lenguajes, por lo que se convierte en otra ventaja más a destacar. Otra de las características reseñables de las que dispone el lenguaje MATLAB es la orientada a objetos. Gracias a ello, se pueden desarrollar interfaces gráficas (GUI) mediante clases gestionadas por valor o, por lo que se denomina en MATLAB, "handles". Este lenguaje permite también ejecutar programas escritos en C o Fortran.

❖ Estudio del entorno de desarrollo integrado

Al inicializar MATLAB nos aparece un entorno de trabajo principal muy intuitivo, el cual está formado por: una serie de menús desplegables, una barra de herramientas, la cual contiene dos opciones adicionales como Simulink y GUIDE y donde ambas ofrecen un subentorno de desarrollo; un menú, denominado "Start", que incorpora un elenco de funciones matemáticas y programas interactivos para diferentes áreas de conocimiento; y una serie de ventanas distribuidas por todo el entorno que desarrollan unas determinadas funciones. Estas ventanas pueden ser configuradas y acopladas en distintos lugares de la zona de trabajo según las necesidades que tenga el usuario.

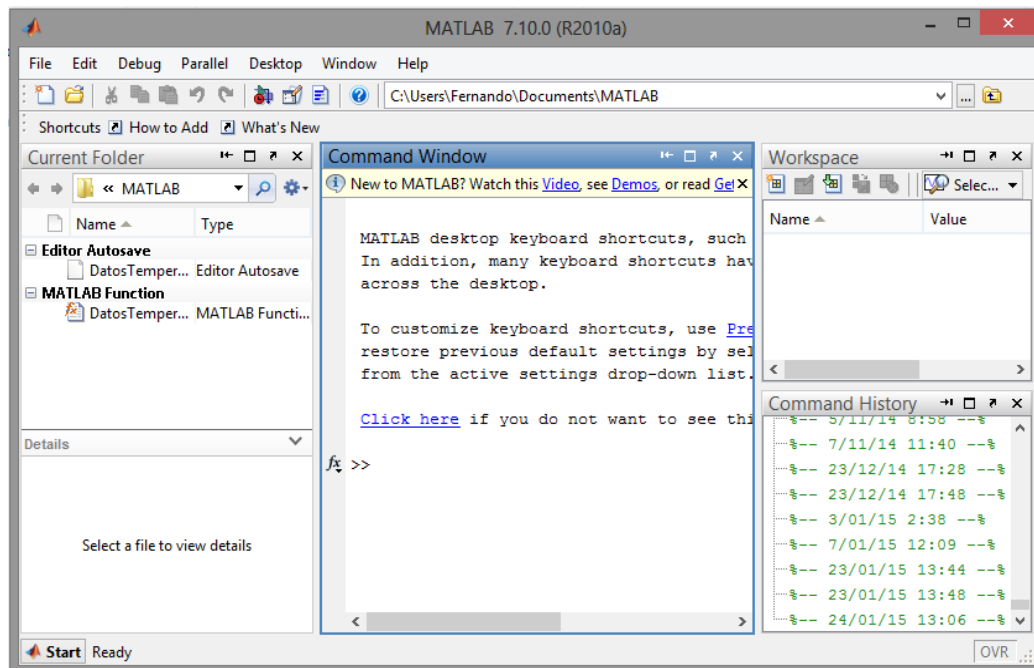


Figura 3.32. Entorno de desarrollo integrado de MATLAB

A continuación, se analizan e ilustran los distintos elementos que forman este entorno de desarrollo:

- **Barra de herramientas:** Un grupo de trece botones dan forma y componen la barra de herramientas de este entorno de desarrollo integrado.

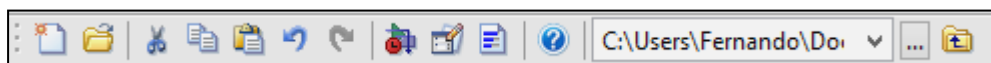














Figura 3.33. Barra de herramientas

-  Crea una nueva ventana, la cual contiene un editor para desarrollar un nuevo programa o "script".
-  Abre un archivo guardado.
-  Corta código.  Copia código.  Pega código.  Deshacer y rehacer.
-  Simulink.  GUIDE.  Identifica las funciones que consumen más tiempo.
-  Establece una ayuda.
-  Busca archivos en carpetas.
-  Acceso a carpetas superiores.



- **Barra de menús:** Para esta barra de herramientas, se han utilizado siete menús desplegables que serán sensibles al contexto, es decir, estarán activadas aquellas tareas relevantes para el trabajo que se esté realizando en ese momento.



Figura 3.34. Barra de menús

- ❏ **Menú File:** Encargado del tratamiento de archivos, está formado por un conjunto de opciones como: nuevo, abrir, cerrar, guardar, importar, añadir fichero, preferencias, opciones de página, imprimir y salir. Las dos opciones que más destacan son:
 - **Importar Datos:** Importa los datos a la ventana Workspace (espacio de trabajo).
 - **Guardar Workspace:** Guarda las variables importadas al Workspace.

- ❏ **Menú Edit:** Contiene tareas para editar archivos como: deshacer, rehacer, cortar, copiar, pegar, seleccionar, borrar, encontrar y limpiar. Las selecciones más reseñables son:
 - **Encontrar archivos:** Facilita la búsqueda de archivos.
 - **Limpiar Ventanas:** Limpia las ventanas del Command Window, Command History y Workspace de código, datos o variables.

- ❏ **Menú Debug:** Dispone de opciones para depurar el código programado como: iniciar, parar, continuar y salir. También se destaca otras dos opciones importantes como son:
 - **Limpiar Breakpoints:** Se encarga de eliminar los breakpoints impuestos para la depuración de errores sobre el código de un programa.
 - **Parada por error:** Avisa de los errores encontrados en el script, mediante los denominados "Warnings".

- ❏ **Menú Parallel:** Lo forman las dos únicas opciones que tiene, que son selección de configuración y gestión de configuración encargadas de la configuración del programa.

- ❏ **Menú Desktop:** Incorpora opciones de diseño para configurar el aspecto del entorno de trabajo de MATLAB como: minimizar, maximizar, mover y desacoplar. También dispone de un grupo de tareas que manipulan las diferentes ventanas.

- ❏ **Menú Window:** Es un menú que de forma básica se dedica a moverse por el entorno de ventanas que forman este entorno de trabajo.

- ❏ **Menú Help:** Integra un manual de autoayuda conectado a internet para ayudar a comprender mejor todo lo relacionado con MATLAB. Como opción importante para analizar:
 - **Demos:** Cuenta con una colección de videos explicativos sobre las tareas que se pueden realizar con la herramienta MATLAB.



- **Entorno de ventanas:** MATLAB consiste en un entorno de ventanas con cinco partes:

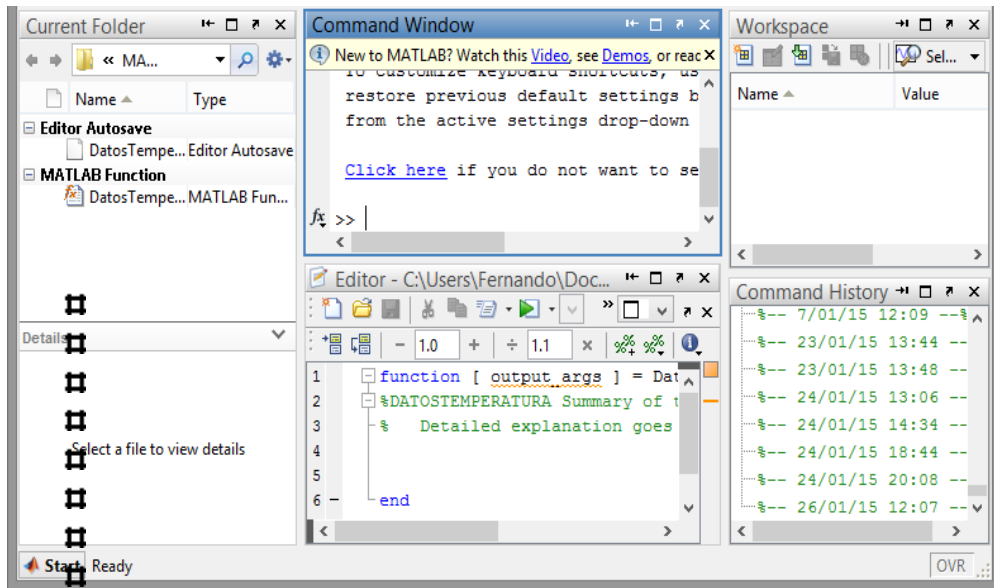



Figura 3.35. Entorno de ventanas

- ❑ **Command Window:** Es la ventana principal donde se programan las instrucciones que se quieren ejecutar.
- ❑ **Current Folder:** Muestra el contenido de la carpeta de trabajo. La dirección de la carpeta de trabajo se puede cambiar mediante la barra desplegable que aparece encima de las ventanas.
- ❑ **Workspace:** Contiene la información sobre las variables y objetos definidos en el programa.
- ❑ **Command History:** En esta venta se aprecia el historial de comandos o instrucciones que han sido ejecutados.
- ❑ **Editor:** Es la ventana encargada de habilitar la creación del código de los programas (scripts) para desarrollar las funciones que hacen posible el desarrollo de este proyecto. Es de remarcar que este editor cuenta con un conjunto de elementos nuevos que se acoplan y complementan la barra de herramientas de la "Figura 3.33."



Figura 3.36. Complemento de elementos de la barra de herramientas de MATLAB

- Imprime el código. Publica el código en internet.
- Encuentra código. Deshacer y rehacer. Funciones.
- Ejecuta el programa. Breakpoint. Borra los breakpoints.
- Analizan el código línea por línea. Sale del análisis.
- Continúa con el análisis. Finaliza la depuración.

- **Menú Start:** Se trata de un botón situado en la esquina inferior izquierda del entorno de desarrollo de MATLAB  que contiene una serie de programas para las siguientes utilidades: cálculo paralelo, matemáticas, estadística, optimización, diseño y análisis de sistemas de control, procesamiento de señales y comunicaciones, procesamiento de imágenes, visión artificial, prueba y medición, finanzas computacionales, biología computacional, generación de código y verificación, implementación de aplicaciones y acceso a base de datos e informes.

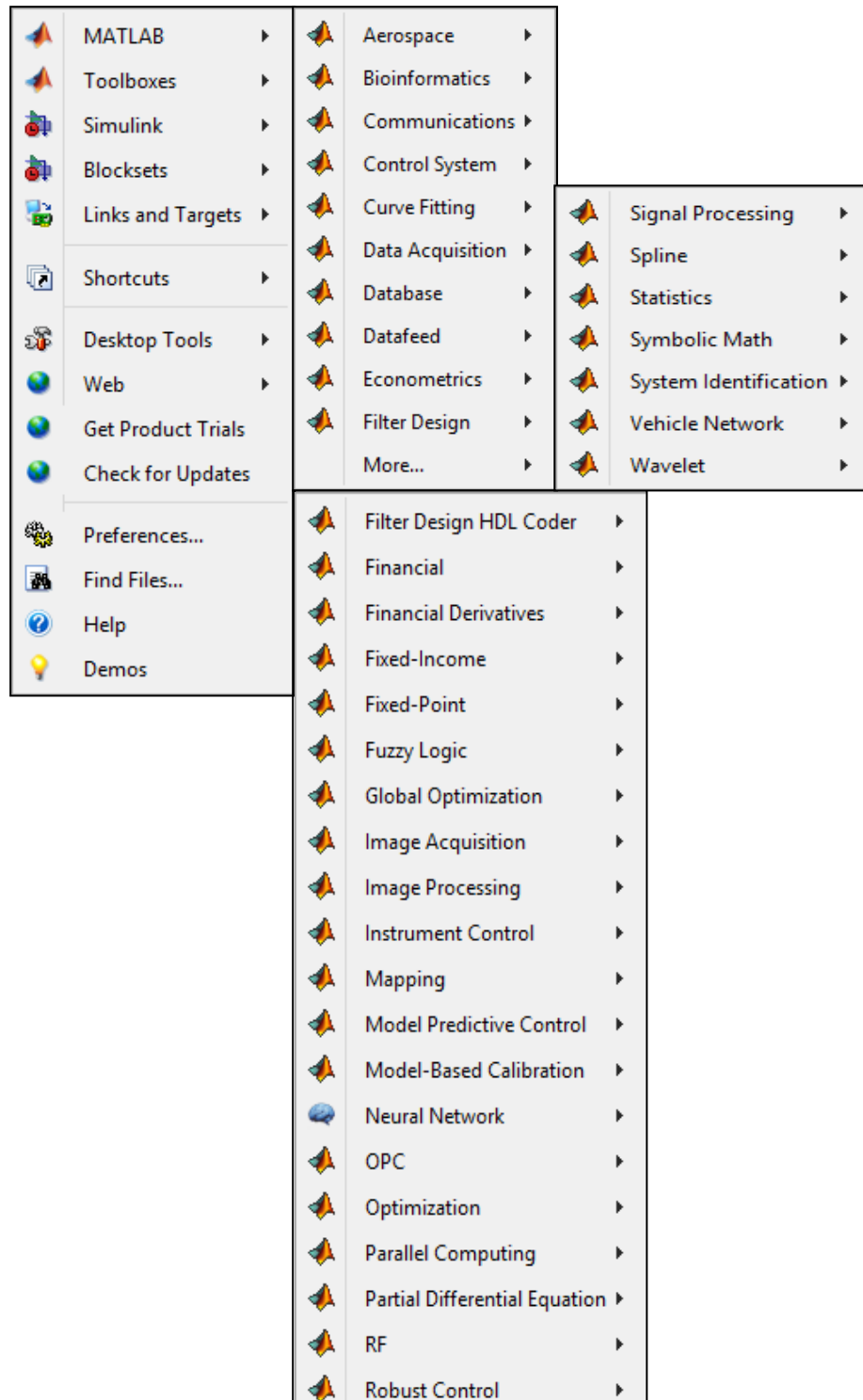



Figura 3.37. Menú Start / Toolboxes

- **Subentorno de desarrollo Simulink:** Debido a que no es utilizado en este proyecto, se analiza de una forma muy breve la utilidad de este software. 

Simulink es un entorno de diagramas de bloque para la simulación multidominio y el diseño basado en modelos. Admite el diseño y la simulación a nivel de sistema, la

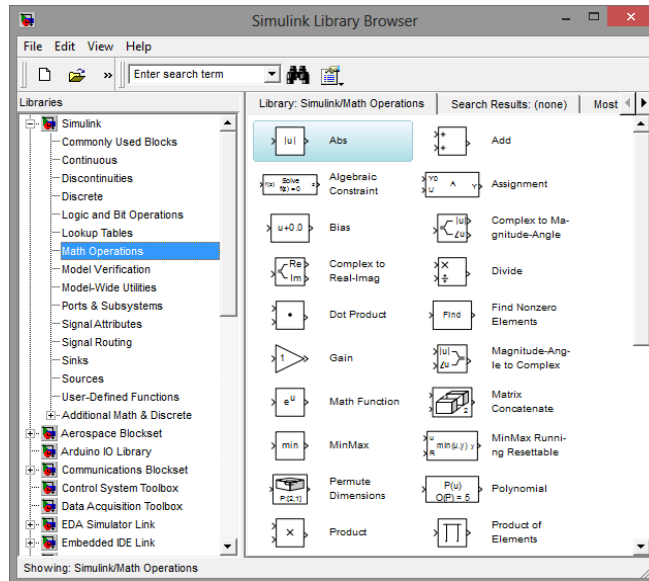



Figura 3.38. Subentorno Simulink

generación automática de código y la prueba y verificación continuas de los sistemas embebidos. A su vez, es uno de los dos subentornos que integra y ofrece el entorno principal de desarrollo de la plataforma MATLAB. Es por ello que permite incorporar algoritmos de MATLAB en los modelos y exportar los resultados de la simulación a MATLAB para examinar más análisis.

- **Subentorno de desarrollo GUIDE:** Por el contrario este tipo de subentorno, el segundo con el que cuenta MATLAB, se utiliza como herramienta para el desarrollo de la interfaz gráfica del sistema de telemetría de este proyecto. 

Si se quiere obtener todo el potencial de una herramienta software se debe permitir que los usuarios puedan interactuar con los programas de una forma cómoda. GUIDE (Graphical User Interface Development Environment) es un entorno de desarrollo de interfaz gráfica de usuario que permite un control sencillo de las

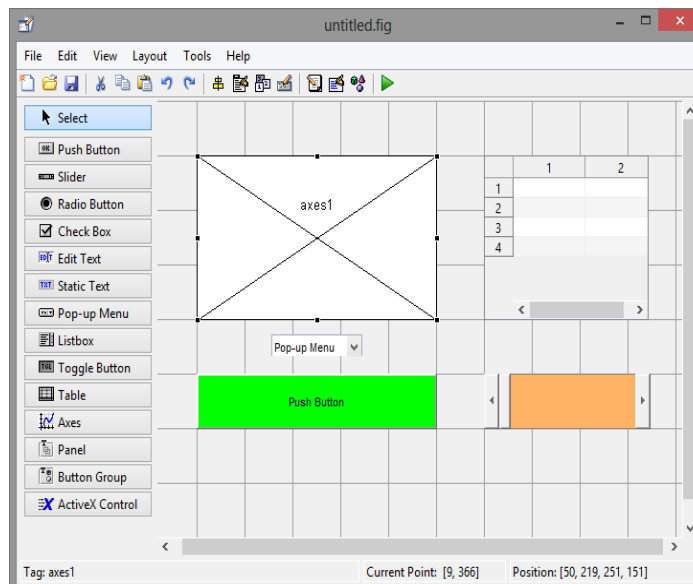


Figura 3.39. Subentorno GUIDE

aplicaciones de software, sin necesidad de saber o aprender un lenguaje con la finalidad de ejecutar una aplicación. A través de MATLAB se puede programar y modificar el comportamiento de la interfaz de usuario. Los elementos de los que dispone son: barra de menús, barra de herramientas y barra de elementos GUIDE.

En los puntos siguientes, se analizan los distintos componentes que forman el subentorno GUIDE:

- ❖ **Barra de herramientas:** En esta ocasión, está compuesta por un conjunto de dieciséis botones que habilitan sus correspondientes funciones.

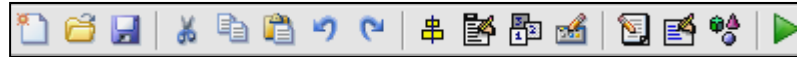


















Figura 3.40. Barra de herramientas

-  Nuevo GUIDE.  Abrir un GUIDE guardado.  Guardar GUIDE.
-  Corta objeto.  Copia objeto.  Pega objeto.
-   Deshacer y Rehacer.
-  Alinear objeto.  Editor de menús.  Ordena los objetos.
-  Editor de la barra de herramientas.
-  Guardar como (GUIDE).  Cambia las características de los objetos.
-  Navegador de objetos.  Ejecuta GUIDE.

- ❖ **Barra de menús:** Grupo de seis menús desplegable sensibles al contexto que activarán sus tareas según la relevancia del trabajo a realizar.

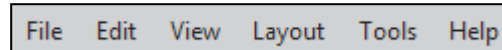


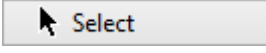

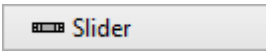
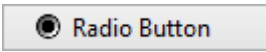
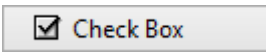
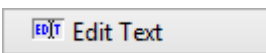
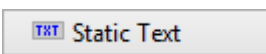
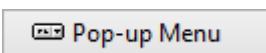
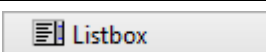
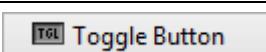
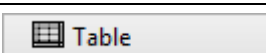
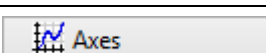
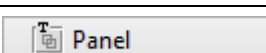
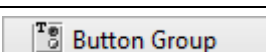
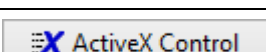
Figura 3.41. Barra de menús

- **Menú File:** Incorpora acciones como las de: crear, abrir, cerrar, guardar, imprimir y preferencias sobre GUIDE. Se resaltarán también la siguiente opción:
 - **Exportar:** Exporta datos del GUIDE al Workspace.
- **Menú Edit:** Contiene opciones para editar los objetos de la interfaz gráfica que se está creando. Dichas opciones son: rehacer, deshacer, cortar, copiar, pegar, limpiar, seleccionar y duplicar. De todas ellas hacemos incapie en:
 - **Duplicar:** Duplica el objeto junto con sus características, haciendo ganar tiempo en el diseño de la interfaz de usuario.
- **Menú View:** Dispone de una serie de tareas que permiten mostrar las distintas barras de herramientas y objetos relacionados con GUIDE. Se destaca:
 - **View Callbacks:** Permite crear, eliminar y situar el código de la función que va asociada a cualquier objeto creado para la interfaz gráfica.
- **Menú Layout:** Cuenta con distintas opciones para ajustar todo lo relacionado con la zona de la cuadrícula.



- **Menú Tools:** Incorpora elementos que organizan, modifican, ejecutan y configuran el aspecto de las barras de herramientas y la interfaz gráfica.
 - **Menú Help:** Integra un manual, videos de demostración y ejemplos de programas en el entorno web del cual dispone MATLAB para ayudar en todo momento a resolver las dudas del usuario.
- ❖ **Barra de elementos GUIDE:** Dispone de 14 botones que desarrollan distintas funciones y crean diferentes objetos para la construcción del GUIDE. Estos botones se pueden analizar en la "Tabla 3.5."

Tabla 3.5. Elementos GUIDE

BOTÓN	DESCRIPCIÓN
 Select	Selecciona los objetos de la interfaz gráfica.
 Push Button	Creación de un botón que al ser pulsado ejecuta una determinada acción.
 Slider	Genera un objeto que representa un rango de valores.
 Radio Button	Son botones de selección. Si hay varios son mutuamente excluyentes.
 Check Box	Es un pulsador de estado. Indica si una opción está activada o no.
 Edit Text	Es una ventana en la cual se puede editar tanto texto como números.
 Static Text	Muestra un mensaje de texto permanentemente.
 Pop-up Menu	Creación de un menú desplegable con un listado de opciones programables.
 Listbox	Se genera un tablero deslizante para incluir información en su interior.
 Toggle Button	Es un botón que proporciona una función on/off.
 Table	Produce un tablero para introducir datos ordenados en filas y columnas.
 Axes	Esta opción dispone de una gráfica para representar datos.
 Panel	Espacio destinado a visualizar varios botones de la barra de elementos.
 Button Group	Es un panel exclusivo para organizar push, radio y toggle button.
 ActiveX Control	Es una opción que despliega controles ActiveX en GUI.

Capítulo 4: ACONDICIONAMIENTO Y DISEÑO DEL SISTEMA DE TELEMETRÍA

4.1. INTRODUCCIÓN

Una vez conseguido armar la estructura de nuestro proyecto, es hora de poner en común todos los elementos empleados y darle forma a este sistema de telemetría. En los próximos análisis se estudia de un modo minucioso tanto el conexionado final de todos los elementos empleados en el sistema, como el diseño del código de la programación que complementa, con un rendimiento óptimo, la telemetría para esta motocicleta de competición. Por ello, se puede considerar lo que expone este capítulo como la parte más importante en cuanto a carga de trabajo y elaboración experimental del proyecto.

En cuanto al diseño del software, se divide en un proceso detallado de cuatro partes. En ellas se encuentra todo el código de programación del proyecto, el cual puede consultarse en su correspondiente anexo. Las cuatro partes diferenciadas son:

- ❖ Diseño del software de desarrollo del sistema de lectura de datos.
- ❖ Diseño del software de desarrollo del sistema de procesamiento de datos.
- ❖ Diseño del software de desarrollo del sistema de monitorización de datos.
- ❖ Diseño de la interfaz gráfica de usuario del sistema de análisis de datos.

4.2. CONEXIONADO

En este epígrafe se describe de una forma detallada y a través de tres pasos las conexiones empleadas entre los distintos elementos para formar el bloque que define el sistema telemétrico propuesto. Se adjuntan imágenes para observar como estos tres pasos se llevaron a la práctica.

- ❖ **Paso 1:** Una vez obtenida la micro tarjeta SD con todos los datos analíticos de la prueba realizada con la motocicleta de competición, se introduce en la ranura del periférico SD card shield V3.1 correspondiente para su formato. A continuación, para el buen funcionamiento del proceso, en el mismo periférico, en la zona del card select, se selecciona, mediante la palanca de selección, la opción Mini.

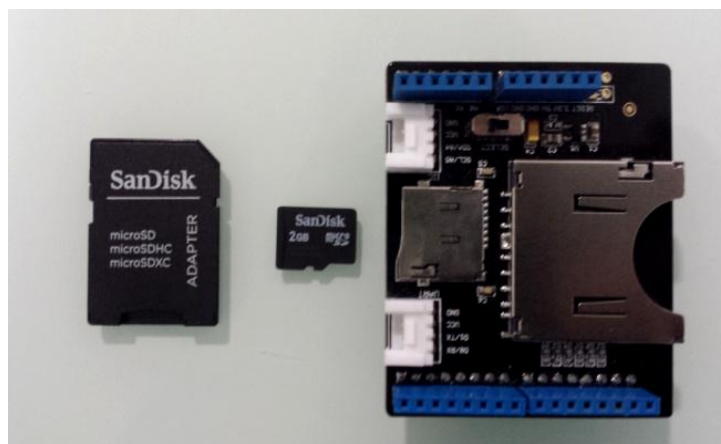


Figura 4.1. Paso 1

- ❖ **Paso 2:** Se prepara la función de transferencia de la que dispone el dispositivo SD card shield V3.1. Dicha función permite transferir, a través de comunicación SPI, la información de la micro tarjeta SD a la placa Arduino. Para habilitar su funcionamiento, se realizan las siguientes conexiones entre el periférico SD card shield y la tarjeta Arduino:

Tabla 4.1. Conexiones de la función de transferencia

SD CARD SHIELD V3.1		ARDUINO MEGA ADK (R3)
PIN 10	→	PIN SS
PIN 11	→	PIN MOSI
PIN 12	→	PIN MISO
PIN 13	→	PIN SCK
PIN 5V	→	PIN 5V
PIN GND	→	PIN GND

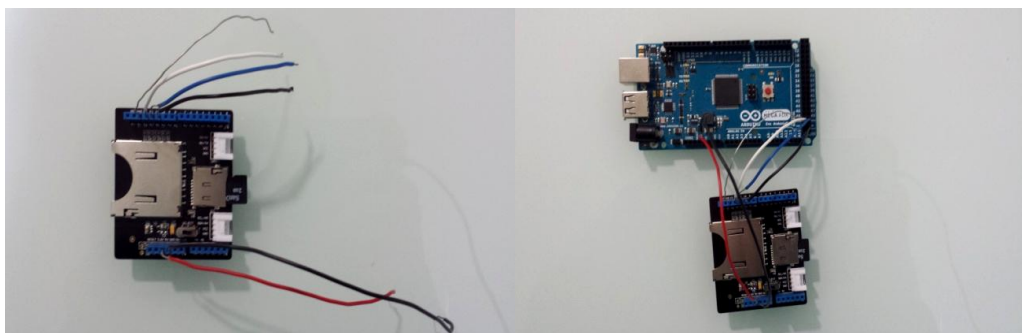


Figura 4.2. Paso 2

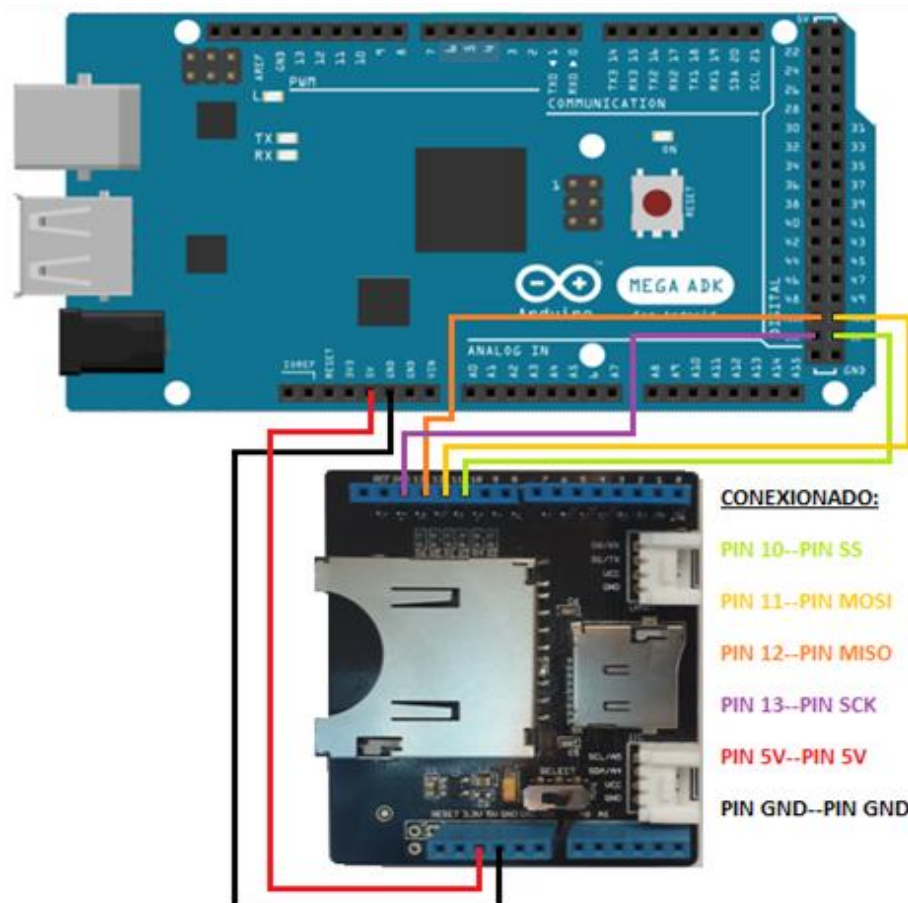


Figura 4.3. Diagrama del conexionado de pines entre SD card shield y Arduino (según "Tabla 4.1.")



- ❖ **Paso 3:** La conexión entre Arduino y el ordenador se efectúa a través del cable USB, consiguiendo que el puerto serie 0 traspase los datos leídos y tratados por el módulo de Arduino al ordenador.



Figura 4.4. Paso 3

4.3. DISEÑO DEL SOFTWARE DE DESARROLLO DEL SISTEMA DE LECTURA DE DATOS

En el diseño del sistema de lectura de datos se comienza con la idea de crear un programa, utilizando el entorno de desarrollo de Arduino, que consiga leer toda la información transferida desde la herramienta electrónica SD card shield al microcontrolador de la placa Arduino.

4.3.1. ESTRUCTURA BÁSICA DE PROGRAMACIÓN ARDUINO

La estructuración para el desarrollo básico de la programación de Arduino se divide en tres partes: la zona de librerías y variables, la función *void setup ()* y la función *void loop ()*. En la primera parte se incluyen las librerías necesarias que se van a utilizar y se declaran las variables y constantes con las que trabaja el programa a diseñar. La segunda, *void setup ()*, es la encargada de la preparación del programa y se trata de la primera función que el programa ejecuta. Esta ejecución se realiza de forma única y se utiliza para inicializar la comunicación serie y para configurar el *pinMode ()* (es la instrucción que determina si un pin digital es de entrada o salida). La tercera parte, *void loop ()*, está formada por todo el código programado que va a ser ejecutado de manera continua. También se encarga de leer las entradas y salidas de la placa.

```
//LIBRERÍAS
#include <SD.h>
//VARIABLES Y CONSTANTES
const int a;
double b;
//FUNCIÓN VOID SETUP()
void setup(){
    //CÓDIGO DE LA FUNCIÓN
}
//FUNCIÓN VOID LOOP()
void loop(){
    //CÓDIGO DE LA FUNCIÓN
}
```

Figura 4.5. Estructura básica de programación de Arduino



4.3.2. PROGRAMA *LecturaTarjetaSD*

El programa *LecturaTarjetaSD* sirve como enlace final para poder obtener y tratar en el ordenador los datos transportados y almacenados, desde un inicio, por la micro tarjeta SD. Se diseña cumpliendo y manteniendo la estructura citada en el epígrafe anterior.

Primero se selecciona la librería SD , que proporciona Arduino, para incluirla en el programa. Esta librería permite y habilita la lectura y escritura en tarjetas SD, en el caso de este estudio la lectura. Lo siguiente que se hace es declarar una constante que guarde en que número de entrada, de la placa Arduino, se encuentra conectado el pin SS. Este pin será el encargado de comenzar el uso de la comunicación SPI que mantendrán la SD card shield y el módulo Arduino Mega ADK para la transferencia de datos entre ambos.

A continuación se acondiciona la función *void setup ()* para su programación. En ella se desarrolla todo el grueso del programa y se observan varias tareas. Comienza con la configuración del puerto serie para poder ordenar y trabajar con el microcontrolador de la placa Arduino de tal modo que, nos informe de algún fallo o nos transmita los datos ya procesados por el programa. Esta transmisión de datos en serie se realizará a la velocidad adecuada para el buen funcionamiento de este tipo de dispositivo Arduino, es decir, se programa a 9600 baudios (bits por segundo). La tarea siguiente consiste en configurar el pin SS, ya citado, como salida para que las funciones de la librería SD funcionen. Después, se crea un bucle para transmitir, en caso de que se produzca algún fallo al leer la tarjeta, un error al usuario por medio del puerto serie. Si esto no es así, y se lee de la forma adecuada, damos paso al código que realiza la función de la lectura de datos. Es de reseñar, que el código encargado de la lectura debería programarse, de forma habitual, en la función *void loop ()* para que el programa fuera ejecutado de forma continua, pero puesto que sólo interesa que la lectura de los datos se realice una sola vez, sin necesidad de leer el mismo "paquete de datos" una y otra vez, es fundamental trasladar el código en el interior de la función *void setup ()* para que se ejecute de forma única. También cabe destacar el acondicionamiento de la función *void loop ()*. A pesar de lo citado con anterioridad, el entorno de desarrollo Arduino no permite el incumplimiento de la estructura de programación. Por tanto aunque la función *void loop ()* no contenga código, debe ser escrita.

Debido a todo ello se consigue el sistema de lectura buscado desde la primera idea, permitiendo al usuario la lectura y visualización del "paquete de datos", adquiridos sobre la motocicleta de competición, mediante el ordenador del sistema telemétrico.

En la siguiente figura se puede observar el flujograma del programa *LecturaTarjetaSD*.

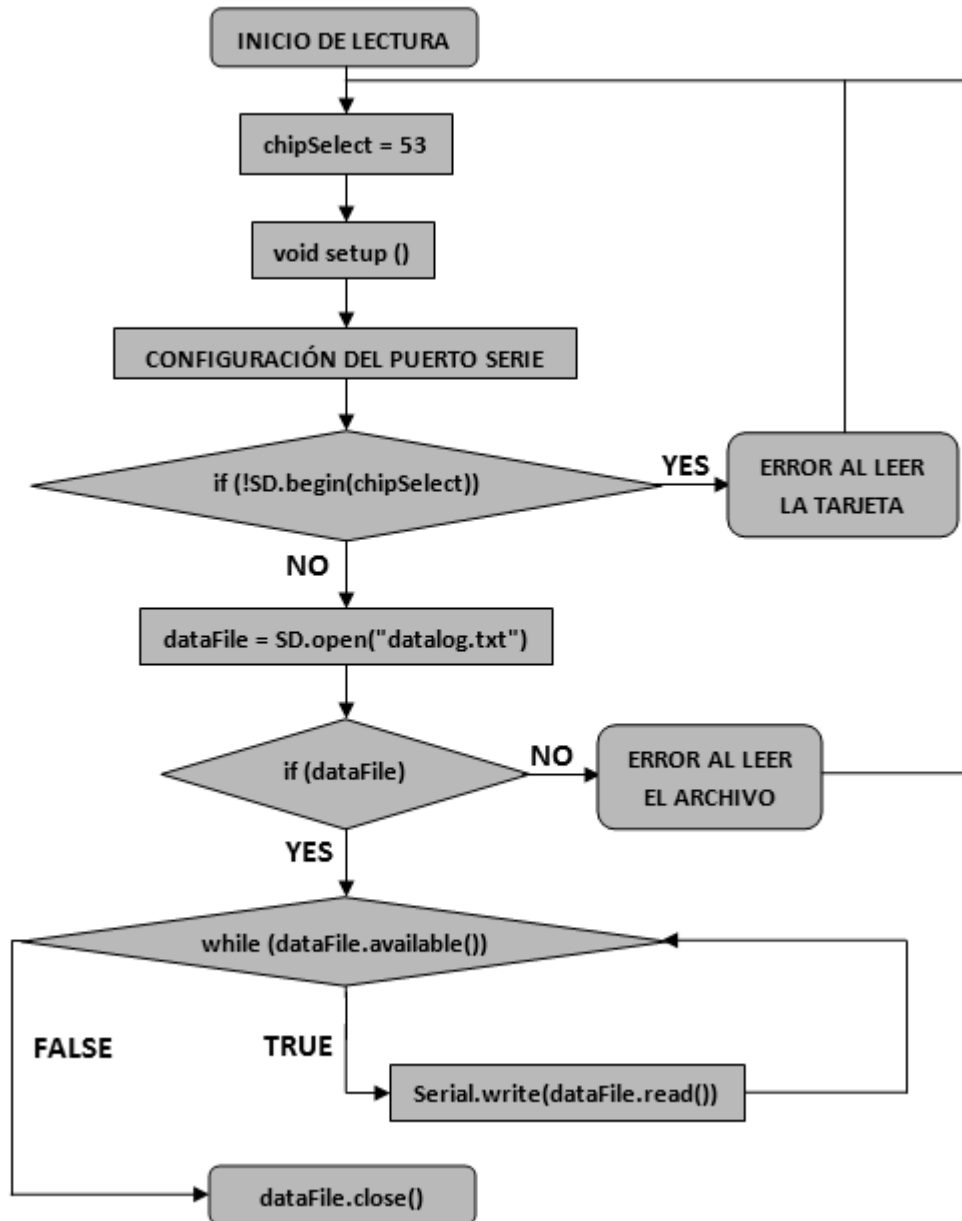


Figura 4.6. Flujograma LecturaTarjetaSD



4.4. DISEÑO DEL SOFTWARE DE DESARROLLO DEL SISTEMA DE PROCESAMIENTO DE DATOS

En el sistema de procesamiento se puede observar cómo se someten los datos, adquiridos mediante el sistema de lectura y el puerto serial anterior, a una serie de operaciones programadas. Estas operaciones se desarrollan a lo largo de una función principal, denominada *datosSesion()*, y a través de cinco funciones específicas para cada sensor. El cometido general de todo este conjunto de funciones es clasificar los datos de las medidas de cada tipo de sensor por matrices. Las medidas que se analizan son once: la temperatura ambiente del sensor temperatura; el eje (x), (y), (z) del acelerómetro; el tiempo horario, latitud, longitud, velocidad y altitud del GPS y la suspensión delantera y trasera de los sensores de ambas suspensiones. Para esta tarea se emplea el entorno de desarrollo de MATLAB como herramienta de programación.

4.4.1. FUNCIÓN PRINCIPAL *datosSesion()*

La función principal *datosSesion()* de este sistema de procesamiento de datos consiste en obtener la matriz final y principal de todos los datos que nos proporcionan los diferentes sensores de la motocicleta, es decir, se consigue un tablón informativo con las medidas de la sesión realizada por la motocicleta de competición. Dicho tablón está organizado por columnas y filas para proporcionar una mejor comprensión de los datos al usuario. Las filas son destinadas al número de medidas realizadas y las columnas son formadas por el número de vueltas dadas y por el tipo de medida que proporcionan cada uno de los sensores.

Para desarrollar esta idea, se comienza atribuyendo un serie de variables internas y globales a la función. Las variables internas son aquellas que se utilizan de manera única en esta función, entre ellas se encuentran las variables para cada medida de cada sensor, contadores de vueltas, contadores de muestras e incrementadores utilizados para desplazarse por las dimensiones de las matrices. Por otro lado, las variables globales sirven para trabajar en una nueva función con el contenido de una variable interna de una determinada función. De este modo se puede trasladar y utilizar valores e información entre los distintos códigos de las diferentes funciones del sistema de telemetría, consiguiendo una programación más eficaz y fluida.

Se continua con la acción de enlazar Arduino con MATLAB, pues Arduino dispone de los datos con los que los sistemas de desarrollo de software de MATLAB tienen que trabajar para completar el sistema telemétrico de este estudio. Para ello se inicializa el puerto serie para la apertura de este y por consiguiente llevar a cabo la adquisición de los datos para su posterior procesamiento. Para capacitar esta tarea se decide programar el puerto serie a 9600 baudios, ya que se recuerda que es la velocidad de programación adecuada para la placa Arduino utilizada.

Una vez optimizado el enlace entre Arduino y MATLAB se inicia la construcción de ciertas matrices que sirven como base para la creación de la matriz principal y, por tanto, al tablón informativo de la sesión sobre los datos de la moto. Este proceso se empieza generando diferentes matrices, a las cuales se les atribuye una serie de datos de una medida y de un sensor en concreto. El modo de esta atribución se realiza a través del puerto serie, el cual se comporta como si se tratase de un vector, es decir, tiene tantas posiciones como tipos de medidas contiene. Por tanto para saber qué medidas se atribuyen a que matrices, basta con organizar los datos en columnas y en el orden que queramos a la hora de escribirlos y por consiguiente leerlos en la micro tarjeta SD. De este modo si se quiere obtener una matriz que contenga datos de velocidad, es suficiente con igualar la matriz velocidad con la posición del "vector" puerto serie donde se encuentra la medida velocidad. Ejemplo: `matriz_velocidad = puerto serie (6)`. Este



proceso se repite tantas veces como número de "paquetes de datos" haya, número que se lee en el programa de LecturaTarjetaSD y que se observa en el último dato de la última columna del "paquete de datos" escritos en la micro tarjeta SD.

Conseguidas las distintas matrices con las distintas medidas, se calculan el número de vueltas que ha dado la motocicleta y el número de muestras que contienen esas vueltas. Estos dos cálculos son de vital importancia para construir la matriz principal que contiene la información de todas las medidas de los diferentes sensores. El número de vueltas se obtiene a través de un contador que acumula las vueltas cada vez que la latitud y la longitud de la moto coincide con la latitud y longitud de la línea de meta, o lo que es lo mismo, cuando la moto pasa por línea de meta. Con este resultado y conociendo el número de tipos de medidas que se realizan para este sistema de telemetría, podemos calcular las columnas del tablón de información. Para ello se resuelve la siguiente ecuación:

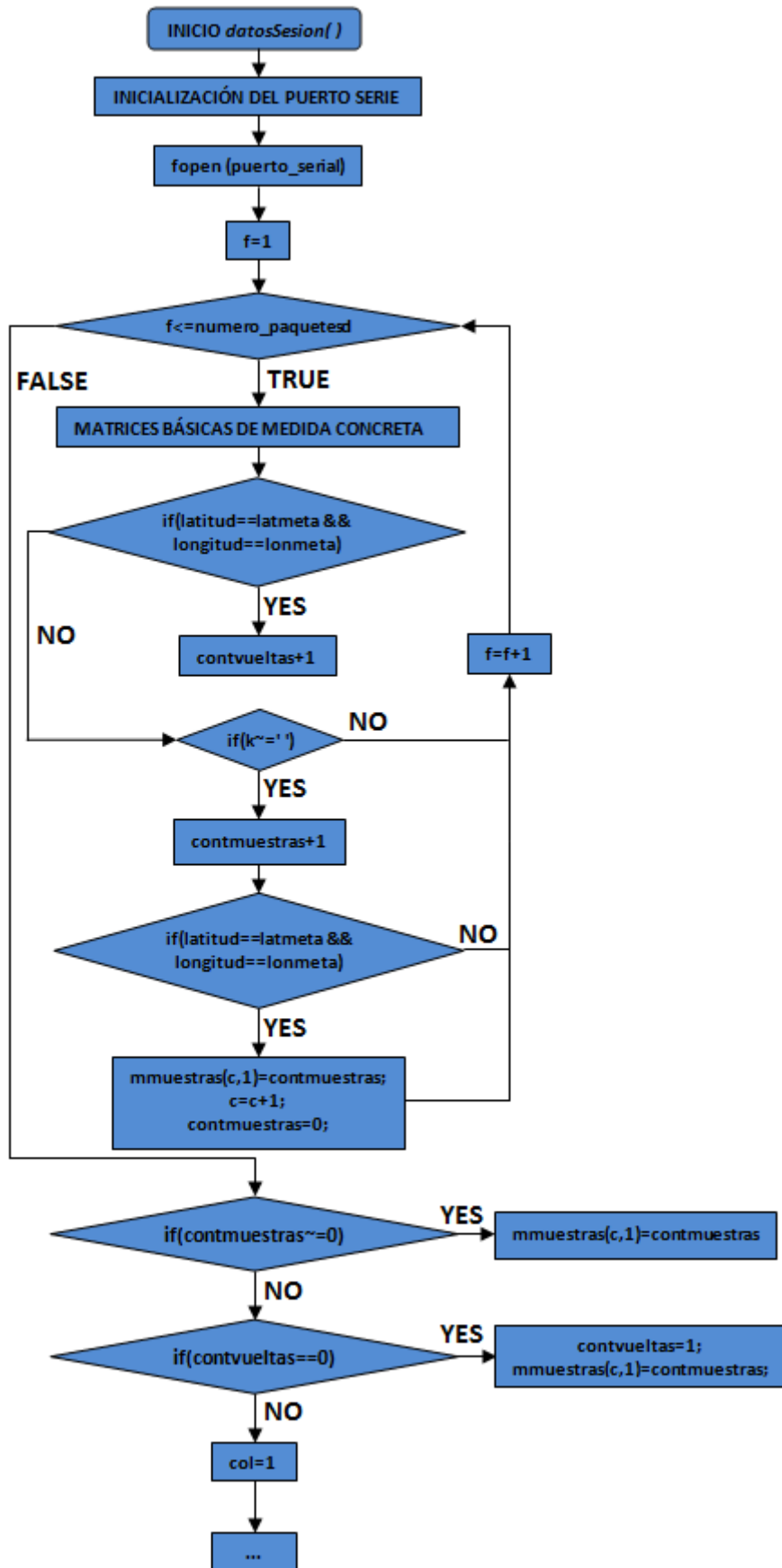
$$\text{nº de columnas} = \text{nº de vueltas} \cdot \text{nº de tipos de medidas}$$

El número de muestras se calcula mediante un contador que se incrementa si al leer un dato del puerto serie es distinto del carácter espacio en blanco. Después se crea una matriz de muestras para almacenar, en cada paso por vuelta de la motocicleta, el número de muestras que se han recogido para esa vuelta. Haciendo este proceso, para cada vuelta completada, se consigue saber cuántos datos contiene cada columna, y por tanto, las filas del tablón de información.

Conseguidas las matrices con las distintas medidas y obtenidos los resultados del número de vueltas y del número de muestras, se tienen los elementos necesarios para la creación de la matriz principal, la cual da pie al tablón principal de información de cualquier sesión de la motocicleta de competición. En esta función también se procesan los datos que recogen los límites para situar y mostrar el dibujo que proyecta el recorrido trazado por la motocicleta. Este cometido se desarrolla con las medidas latitud, longitud y altitud proporcionadas y facilitadas por el GPS. Finalmente cortamos la conexión con el puerto serie sin eliminar los datos obtenidos.

Hay que reseñar que MATLAB no permite dejar incompletas las matrices con las que se trabaja, por ejemplo: Si tenemos una matriz de 4x12, no sería posible que en ciertas columnas se tuvieran las dos primeras filas con datos y las dos últimas sin ninguno. Cuando ocurre esto, MATLAB de forma automática rellena con ceros estos espacios sin datos, dando lugar a un error y a una posible falta de compresión, ya que la motocicleta transmite el fin de la lectura de datos y el usuario puede analizarlo considerando una medida cuyo valor es cero. Para resolver este problema y diferenciar este "cero automático" de un valor "cero real" medido por los sensores, se recorre la matriz principal, ya creada, en busca de columnas (medidas) que compartan filas (datos) llenas de ceros, ya que es imposible que, con el sistema de telemetría funcionando en buenas condiciones, todas las medidas de todos los sensores de la motocicleta midan el valor de cero. Una vez localizada toda esa coincidencia de "ceros automáticos", se sustituyen por el valor numérico de infinito, apareciendo en la casilla correspondiente del tablón la palabra (inf), la cual se utilizará como abreviatura del aviso informativo, "información de finalización de datos" que se le proporciona al usuario para ponerle en conocimiento de que la lectura de datos finaliza en ese punto.

A continuación se muestra el flujograma de la función principal *datosSesion()*.



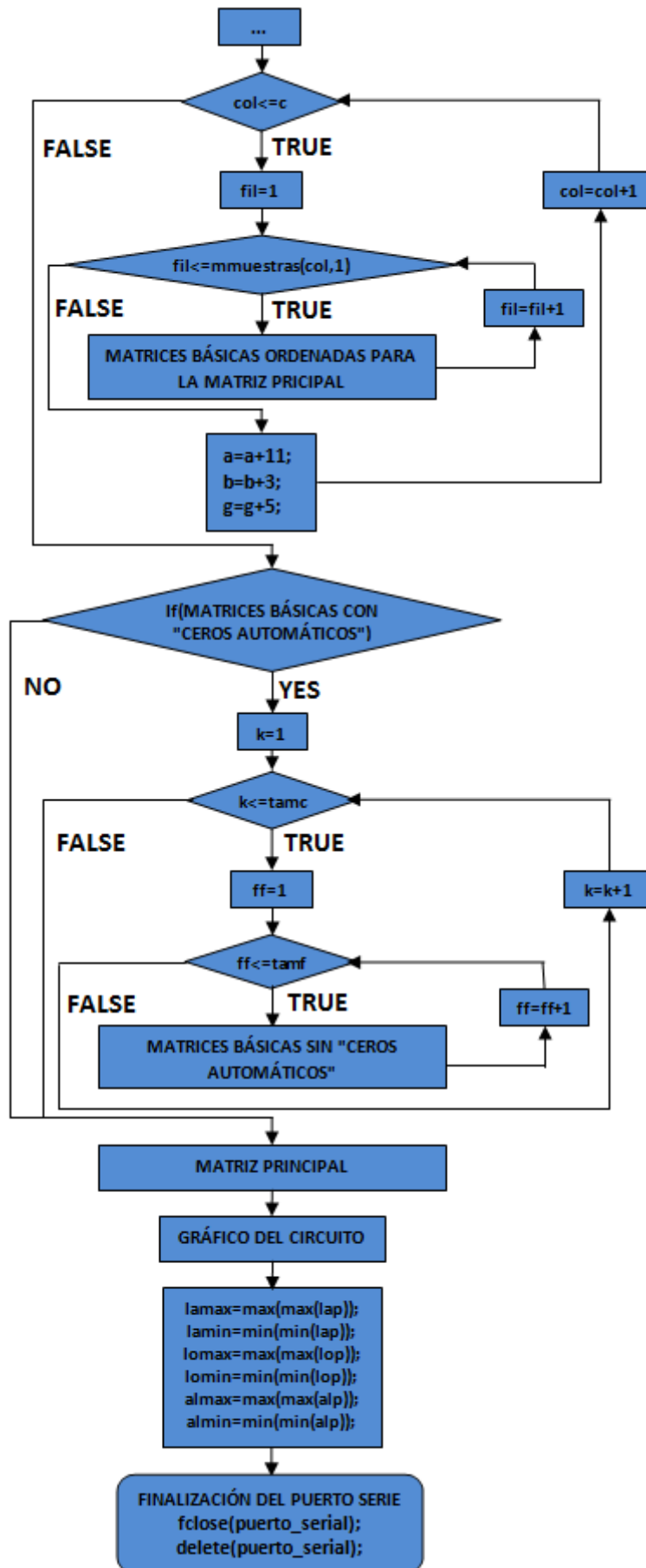


Figura 4.7. Flujograma función principal datosSesion()



4.4.2.FUNCIÓN *datosTemperatura()*

Está diseñada para procesar las medidas realizadas por el sensor de temperatura y estudiar de forma personalizada la evolución que sufre la temperatura ambiente entorno a la motocicleta. Este análisis proporciona un tablón informativo y un gráfico que muestran la evolución de los registros de temperaturas obtenidas a lo largo de las vueltas completadas por la moto.

El desarrollo de la función *datosTemperatura()* comienza con la declaración de variables, en concreto con la declaración de variables globales, las cuales no son generadas en esta función, sino que provienen de la función *datosSesion()* para hacer uso del número de vueltas y muestras que se adquieren de ella.

Se inicializa la comunicación con el puerto serie mediante una programación a 9600 baudios, la cual se mantendrá para todas las funciones desarrolladas en MATLAB. Acto seguido se produce la apertura del puerto serial.

A continuación, se crea una ventana para desarrollar la gráfica que muestra el comportamiento de las distintas temperaturas del ambiente. Este gráfico hace que el usuario observe la evolución de dichas temperaturas cuando se trata de analizar el traspaso de datos de una nueva sesión, es decir, siempre que la consulta, hacia esta función, no sea sobre una sesión ya almacenada. Al ser así, el usuario simplemente se encuentra con la gráfica dibujada y finalizada.

Una vez conseguido todo lo anterior, se construye la matriz de la temperatura ambiente. Sus filas y sus columnas son proporcionadas por las variables globales citadas con anterioridad. Conseguida la dimensión de la matriz se procede a completarla con registros de temperatura mediante la transferencia de los datos, correspondientes a esta medida, del puerto serie.

Cabe destacar que, según se completa la matriz, la gráfica dibuja punto a punto cada una de las temperaturas en función de los datos que recibe por el puerto serie. Este proceso se desempeña hasta conseguir completar el dibujo de la gráfica. Por último, se corta la conexión del puerto serie sin eliminar la información obtenida.

En la "Figura 4.8." se puede analizar el flujograma de la función *datosTemperatura()*.

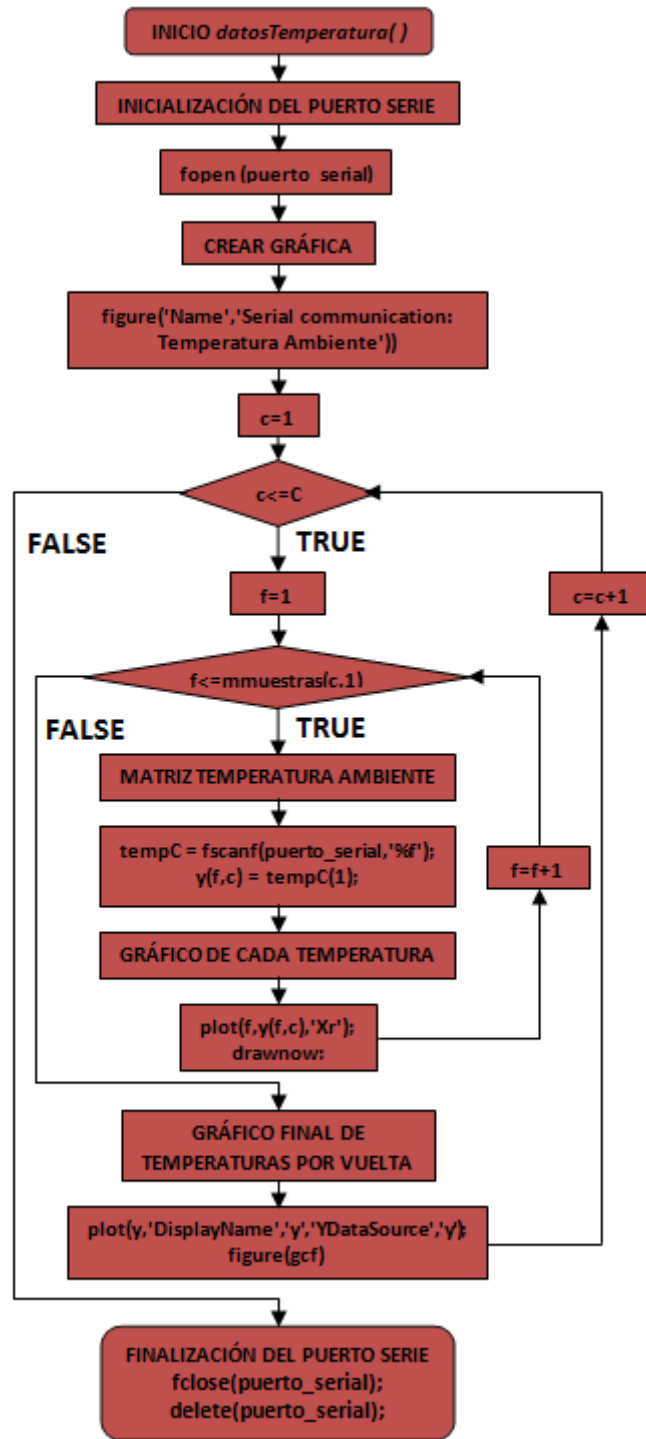


Figura 4.8. Flujo de datosTemperatura()

4.4.3. FUNCIÓN *datosSuspDelantera()*

Es la encargada de estudiar y analizar, en concreto, los datos sobre el desplazamiento sufrido por la suspensión delantera de la motocicleta. La función mantiene de forma exacta la explicación, el diseño y la estructura del código que tiene la función *datosTemperatura()*. La medida y el sensor son las únicas diferencias, las cuales son encontradas en el flujograma y en los resultados obtenidos. En ellos cambian ciertas variables y el número de la posición del "vector" puerto serie, el cual se corresponde con la medida para la suspensión delantera. Por ello no se realiza de nuevo la explicación y se muestra de forma directa su flujograma.

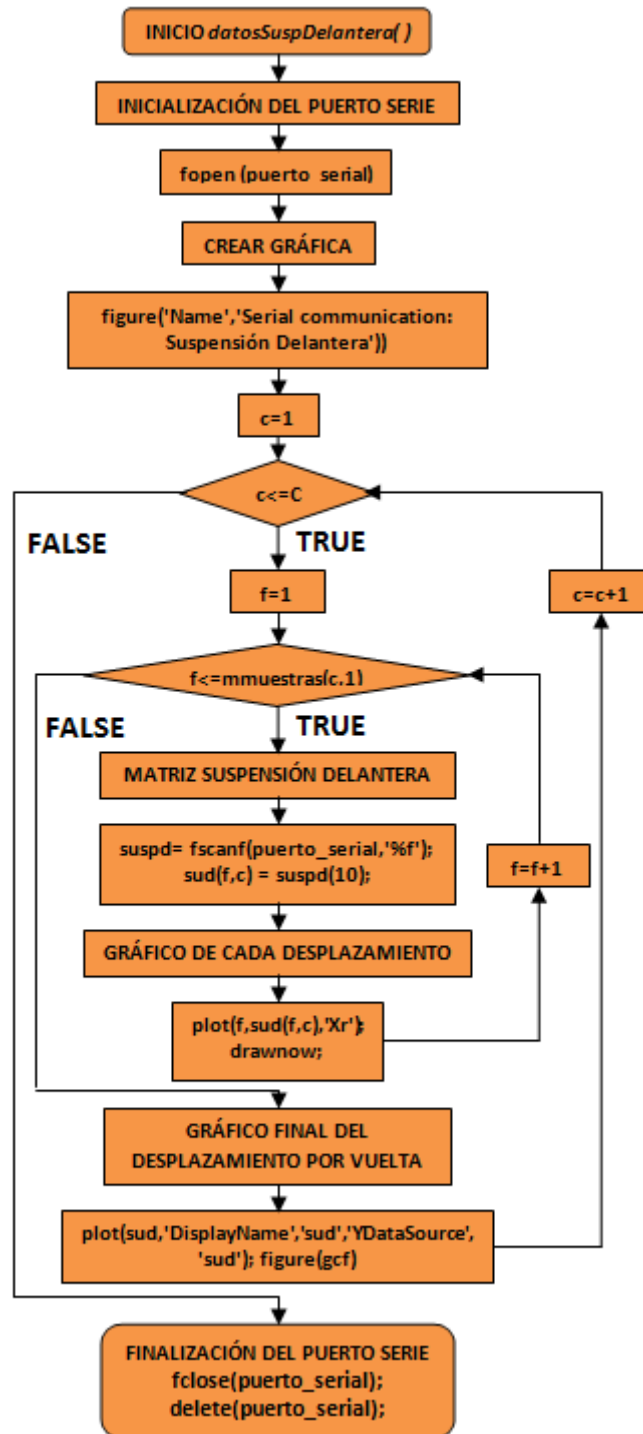


Figura 4.9. Flujograma función *datosSuspDelantera()*

4.4.4. FUNCIÓN *datosSuspTrasera()*

La función *datosSuspTrasera()* tiene la exclusiva tarea de adquirir las medidas del desplazamiento originado en la suspensión trasera de la motocicleta de competición. Gracias al sensor de la suspensión trasera es posible examinar las distintas muestras registradas. A esta función se le aplica la misma observación que a la función *datosSuspDelantera()*. Por consiguiente, se muestra su flujograma.

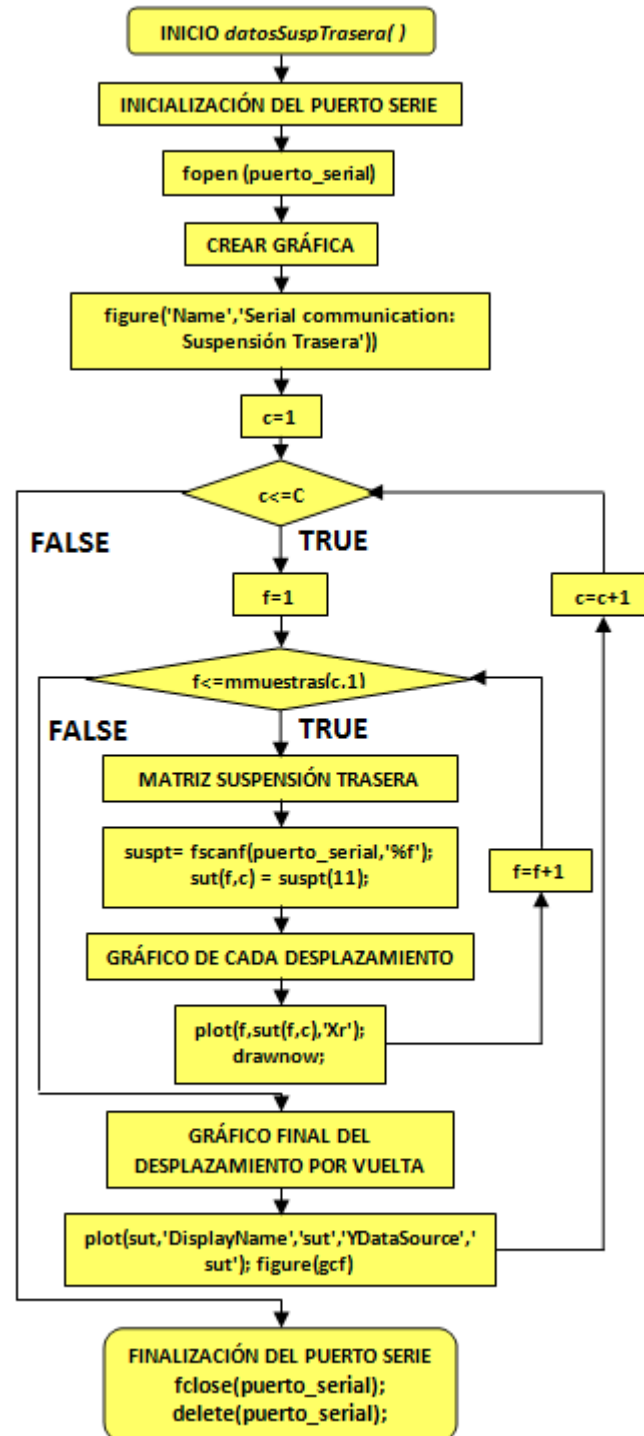


Figura 4.10. Flujograma función *datosSuspTrasera()*



4.4.5. FUNCIÓN *datosAcelerometro()*

Se desarrolla para realizar el estudio único de las aceleraciones que sufre la motocicleta en referencia a su eje de coordenadas. Se obtienen tres medidas del propio acelerómetro, consiguiendo un análisis del eje (x), (y), (z) del ya mencionado eje de coordenadas.

En cuanto al diseño comparte ciertas características que las funciones anteriores, es decir, dispone de la misma estructura que la función *datosTemperatura()*. Sin embargo, *datosAcelerometro()* sólo se asemeja hasta el punto de la creación de la ventana para el desarrollo de la gráfica. El resto se explica a continuación, ya que difiere en cuanto a la creación de la matriz para el acelerómetro (tabla informativa) y en cuanto a la elaboración de su gráfica.

En esta ocasión la construcción de la matriz informativa se procesa de la siguiente manera. En lugar de recibir un valor, se reciben tres valores distintos a través de las tres posiciones del "vector" puerto serie destinadas al eje (x), (y), (z) del acelerómetro. Esto produce que por cada muestra haya tres valores, y por tanto, la ocupación de tres columnas de la matriz. Debido a ello se genera un incrementador de columnas para que, por cada vuelta que supere la motocicleta, los datos de los tres ejes ocupen las siguientes tres columnas correspondientes, consiguiendo la distribución correcta de los datos en la matriz.

En cuanto al desarrollo de la gráfica, sigue el mismo patrón que la construcción de la matriz del acelerómetro. Para cada vuelta se dibuja dato a dato los valores de los tres ejes. Al finalizar la vuelta se plasma la unión de esos valores para observar la evolución de estas aceleraciones.

Acto seguido, se puede examinar el siguiente flujograma perteneciente a la presente función.

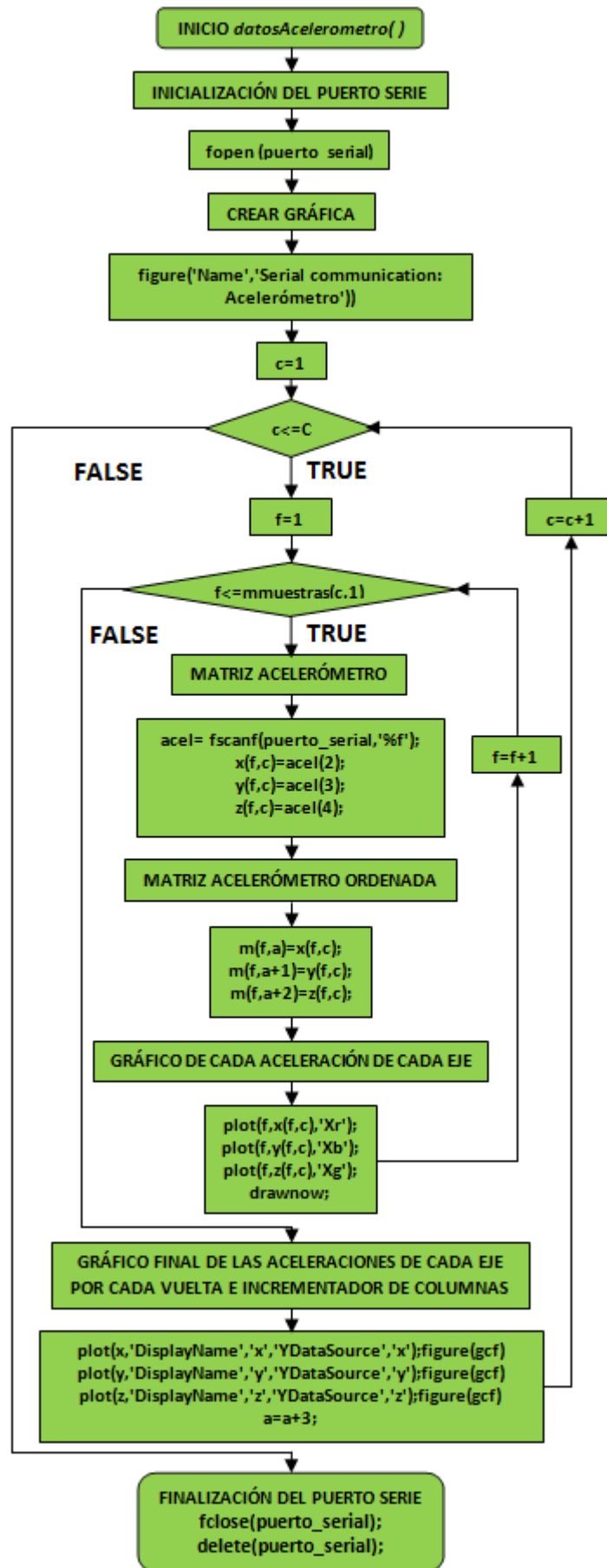


Figura 4.11. Flujograma función datosAcelerometro()



4.4.6. FUNCIÓN *datosGps()*

La función *datosGps()* es creada y procesada con el objetivo de estudiar en profundidad las diferentes medidas que percibe y nos proporciona el GPS. Se analizan medidas como la latitud, longitud y altitud para saber el posicionamiento de la moto, el tiempo horario para poder conocer el tiempo por vuelta de la motocicleta y la velocidad para conocer la rapidez con la que se desplaza el piloto.

El diseño de esta nueva función es una mezcla entre la estructura del código de *datosTemperatura()* y del código de *datosAcelerometro()*. En este proceso se mantiene todo el desarrollo de la función *datosTemperatura()* hasta la creación de su ventana para la gráfica, a partir de ahí, la obtención de su matriz y la creación de su gráfica es diferente, asemejándose a lo ocurrido en la función *datosAcelerometro()*. También cabe destacar, como otra diferencia, la utilización de nuevas variables para esta función, ya que en *datosGps()* también se construye una nueva matriz y una nueva gráfica para la proyección del circuito recorrido por la motocicleta.

Para el desarrollo de la matriz del GPS se reciben los cinco valores de las cinco medidas, ya citadas, mediante las correspondientes posiciones del "vector" puerto serie. Esto origina un proceso similar al ocurrido con el sensor acelerómetro, es decir, por cada valor del GPS de la muestra recibida se ocupa una columna de la matriz. Esto produce la programación de un incrementador de columnas para conseguir que, por cada vuelta completada por la moto, los cinco datos del GPS ocupen las siguientes cinco columnas, construyendo en el orden correcto la matriz. Por otro lado, utilizando el mismo proceso de colocación de datos, se construye la matriz destinada al circuito recorrido.

En esta ocasión, la gráfica asociada a los datos recopilados se divide en tres subgráficas. En la primera se representa paso a paso como se construye en 3D el recorrido trazado por la motocicleta. En la segunda se muestra la medida de la velocidad, siendo representada dato a dato para observar las variaciones que ha sufrido en cada vuelta. Por último, en la tercera, se da a conocer en 2D la silueta del circuito recorrido.

Para dejar más claro el diseño que sigue esta función, se muestra la ejecución de su código en el siguiente flujograma.

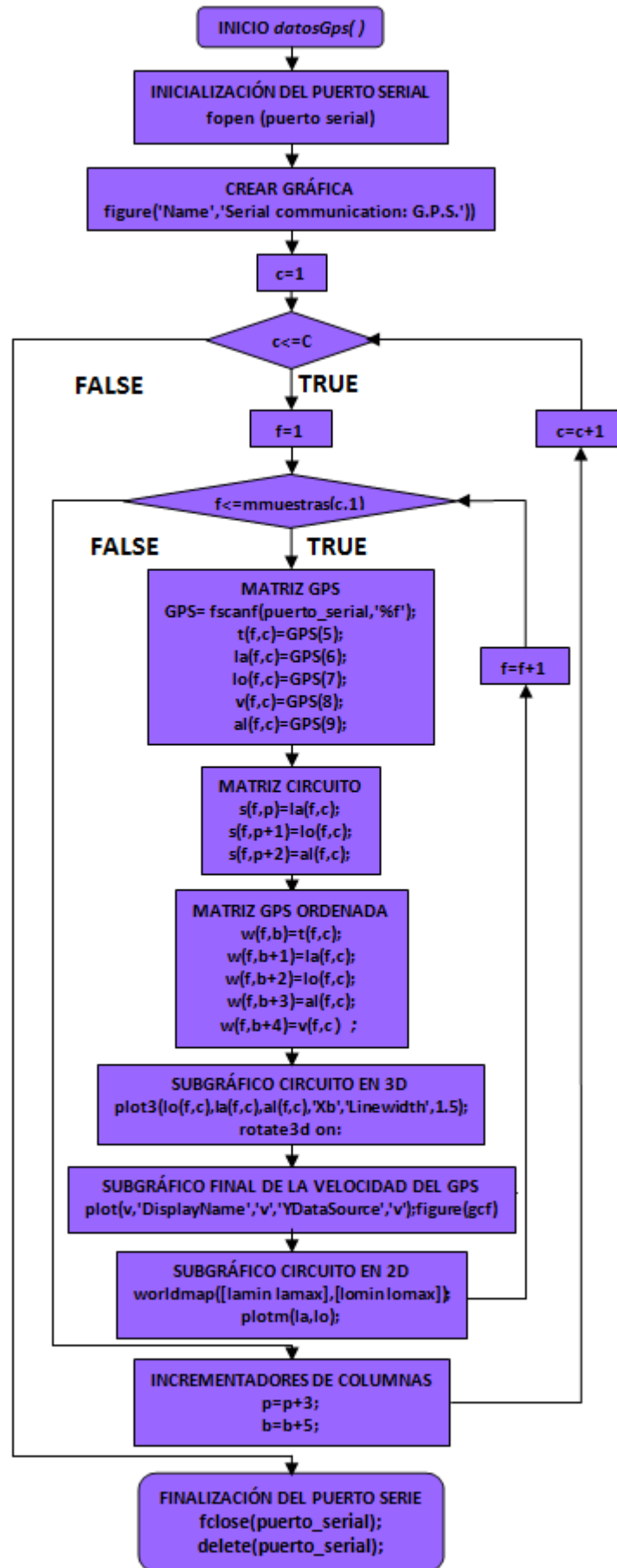


Figura 4.12. Flujograma función datosGps()



4.5. DISEÑO DEL SOFTWARE DE DESARROLLO DEL SISTEMA DE MONITORIZACIÓN DE DATOS

En el sistema de monitorización de datos se desarrolla la programación de todas y cada una de las funciones de entorno necesarias para la correcta ejecución de los cometidos a realizar por la interfaz gráfica. Una vez que los datos están procesados, este diseño se encarga de configurar toda la información para ser mostrada al usuario, haciendo que interactúe con la interfaz de un modo correcto y sencillo. El diseño de este sistema lo componen 31 funciones de entorno examinadas y clasificadas según su labor, las cuales se caracterizan por contener multitud de variables globales, ya que este nuevo código emplea los datos obtenidos en el sistema de procesamiento de datos y "salta" de una función (GUI) a otra de manera continua. También disponen de una estructura de código común, basada en "minifunciones" y llamadas a función.

Para llevar a cabo todas estas tareas se utiliza el editor de funciones que habilita, de forma automática, el entorno de MATLAB GUIDE. Esto origina la asociación de la estructura del código de las funciones de este sistema con los elementos que componen la interfaz gráfica, dotándolos de la funcionalidad correspondiente para cumplir con su cometido.

Debido a esta asociación, el sistema de monitorización de datos va de la mano del sistema de análisis de datos, es decir, son dos sistemas con estructuras distintas pero con el objetivo común de proporcionar al usuario el estudio del comportamiento de una motocicleta de competición durante su rodaje en pista. Esto significa que las funciones de este sistema de monitoreo son explicadas, pero no acompañadas de sus respectivos flujogramas. La razón de esta decisión se debe a que el conjunto de flujogramas, correspondientes a cada una de las funciones de monitoreo, son idénticos al flujograma general de los elementos que componen el diseño de la interfaz gráfica del sistema de análisis de datos. Por lo que será en el flujograma general del capítulo 5, "Funcionamiento del sistema de telemetría", donde se entenderá de una forma más ordenada y sencilla las funciones destinadas a la monitorización de datos, puesto que son utilizadas según la selección de los elementos que forman la interfaz gráfica para el análisis de toda la información de la motocicleta.

4.5.1. FUNCIÓN DE ENTORNO *guidePrincipal()*

Es la encargada de gestionar todo lo que ocurre en el menú principal de la interfaz gráfica de usuario (GUI). La función comienza recibiendo el número de "paquetes de datos", que han sido identificados y leídos, junto con la latitud y longitud de la línea de meta del circuito en cuestión. Una vez obtenidos estos datos, se encarga de tareas como: mostrar imágenes, cargar una sesión nueva, abrir una sesión ya almacenada y limpiar los datos recibidos.

Para cargar una sesión nueva, el programa realiza una llamada a la función *datosSesion()*. La apertura de una sesión ya almacenada se desarrolla con la llamada de la función *abrirS()*, la cual explicaremos más adelante. Tanto para una sesión cargada como para una sesión almacenada se prepara la monitorización de sus datos obtenidos en su correspondiente zona de la interfaz (GUIDE), la cual pertenece a la función *guideSesion()*. Por último tanto la limpieza de datos como la muestra de imágenes, se realiza mediante una serie de instrucciones propias del entorno de programación GUIDE.



4.5.2. FUNCIÓN DE ENTORNO *guideSesion()*

Se diseña para recibir y mostrar tanto la información general de una sesión nueva como la de una sesión almacenada. Tiene la capacidad de proporcionarnos las medidas sobre la motocicleta y el circuito realizado. Para ello se desarrollan una serie de instrucciones que habilitan tablas y gráficas para albergar los datos y circuitos. Esta función, a través de la utilización de las funciones *datosTemperatura()*, *datosSuspDelantera()*, *datosSuspTrasera()*, *datosGps()* y *datosAcelerometro()*, permite realizar y monitorizar un estudio más concreto sobre los datos de todas las medidas adquiridas. En *guideSesion()* se programa la apertura y el almacenamiento de sesiones. También se prepara, mediante la llamada de la función *guideScompararsesion()*, la monitorización de un entorno que compara sesiones y medidas concretas entre sí. Otra de las posibilidades que ofrece es la capacidad para volver al menú principal mediante las llamadas a las funciones GUIDE. Por último, en la función *guideSesion()*, se desarrolla la programación de una "minifunción" que proporciona al usuario el acceso al entorno de internet que dispone MATLAB para consultar, desde una página web, el tiempo meteorológico detallado de cualquier parte del mundo. De este modo, también, se pueden tomar decisiones importantes en cuanto a los reglajes necesarios para la motocicleta en función del clima, completando, aún más, el sistema de telemetría.

4.5.2.1. Instrucción guardar sesión

Para el buen entendimiento del diseño del sistema de monitorización es necesario explicar cómo se realiza, en *guideSesion()*, el almacenamiento de una sesión que interesa y se quiere guardar. Primero de todo se aclara que para guardar una sesión, debido a la sencilla programación que conlleva, no tiene una función informática que la desarrolle, sino que con una instrucción que proporciona el entorno de programación MATLAB GUIDE es suficiente para la ejecución de esta tarea de almacenamiento.

A continuación cabe destacar en este epígrafe, el problema que surge en el diseño de este sistema cuando se quiere comparar dos sesiones guardadas, o una sesión nueva con una guardada, en el entorno gráfico que nos facilita *guideScompararsesion()*. Este entorno cuenta con dos tablas informativas, A y B, a las cuales se les asocia dos sesiones distintas utilizando una misma variable global (ST, Sesión Total) encargada de contener la información respectiva a mostrar. Al abrir estas sesiones, ambas tablas siempre muestran los datos de la última sesión que es abierta, es decir, si en primer lugar abrimos la sesión (A) y en segundo lugar la sesión (B), en vez de aparecer en la primera tabla la sesión (A) y en la segunda la sesión (B), se superpone la sesión (B) tanto en la primera como en la segunda tabla. Esto se debe a que, a pesar de que la información es distinta, la variable (ST) encargada de ese tipo de información es la misma, propiciando que MATLAB sea capaz de distinguir entre variables pero no entre la información de una misma variable. Como la creación de diferentes variables para cada sesión nueva no es posible, debido a que no se puede conocer cuantas sesiones se pueden cargar y querer guardar en el transcurso del tiempo, sería imposible asociar las tablas informativas a dichas variables.

Por lo tanto, para resolver este problema se lleva a cabo la siguiente solución. Primero, debido a que la comparación se desarrolla entre dos sesiones mediante las tablas informativas (A y B), se necesita asociar, a dichas tablas, dos variables globales que tengan distinto nombre pero que realicen la misma función, es decir, que ambas contengan la información de sus respectivas sesiones. La primera variable global (ST), la cual ya fue programada, se asocia con la tabla (A). La segunda variable global es (STB), la cual ha sido creada para completar esta asociación con



la tabla (B). Ambas variables son programadas en la función *datosSesion()* del sistema de procesamiento de datos, la cual se encarga de obtener toda la información que queremos comparar. Una vez conseguido esto, se desarrolla la instrucción guardar sesión, perteneciente a la función *guideSesion()*, de la siguiente manera. En el caso de que la sesión examinada sea una nueva sesión cargada, se procede a guardar hasta en dos ocasiones la misma sesión. En la primera se le atribuye, a la sesión, el nombre que se quiera y en la segunda ocasión se le da el mismo nombre pero terminado con la letra (B). De este modo, las sesiones quedan distinguidas en sesiones (A) y sesiones (B) en el entorno que almacena los archivos guardados. Para finalizar se desarrollan dos funciones con el objetivo de abrir las sesiones guardadas. Estas funciones se diseñan para cada uno de los dos tipos de sesión que hemos logrado diferenciar con anterioridad. Gracias a ello se consigue una apertura de datos independiente que logra diferenciar las sesiones a comparar, evitando el error de la superposición entre la tabla informativa (A) y (B).

4.5.3. FUNCIONES DE ENTORNO DE APERTURA DE SESIONES

Se tratan de las funciones *abrirS()* y *abrirSB()*. La función *abrirS()* es utilizada en todos los entornos que se quiera abrir una sesión guardada. Por otro lado, la función *abrirSB()* se emplea, únicamente, para la apertura de sesiones guardadas que quieran ser comparadas en su entorno correspondiente. Las dos funciones proporcionan toda la información sobre las sesiones seleccionadas. Su diseño se basa en una estructura programada para la consecución y colocación correcta de los datos obtenidos en sus correspondientes matrices.

4.5.4. FUNCIONES DE ENTORNO DE ESTUDIO PERSONALIZADO DE SENSORES

El conjunto de funciones de entorno que forman este grupo son: *guideStemperatura()*, *guideSsuspdelantera()*, *guideSsusptrasera()*, *guideSgps()* y *guideSaccelerometro()*. Cada una de ellas crea un entorno donde se puede analizar la información y la evolución de los datos de un sensor concreto. Todos esos entornos disponen de leyendas para mejorar la comprensión de sus gráficas. Cada una de las funciones de entorno de este epígrafe adquiere los datos con los que trabajar mediante sus respectivas llamadas a función de: *datosTemperatura()*, *datosSuspDelantera()*, *datosSuspTrasera()*, *datosGps()* y *datosAcelerometro()*,

4.5.5. FUNCIÓN DE ENTORNO *guideScompararsesion()*

Es la función que contiene la estructura del código que hace posible el entorno para comparar sesiones realizadas por la motocicleta. En ella contiene tres conjuntos de "minifunciones" de entorno. Dos para la comparación entre variables de una misma sesión y uno para comparar sensores de distintas sesiones.

4.5.6. FUNCIONES DE ENTORNO DE COMPARACIÓN DE SENSORES

El grupo de funciones que generan estos entornos de comparación son: *guideCsensortempambiente()*, *guideCsensorsuspdelantera()*, *guideCsensorsusptrasera()*, *guideCsensorgps()* y *guideCsensoraccelerometro()*. Todas ellas estudian, a través de tableros informativos y gráficas, las diferencias entre las distintas sesiones realizadas por la motocicleta mediante la comparación de sus sensores.



4.5.7. FUNCIONES DE ENTORNO DE COMPARACIÓN DE VARIABLES

Este conjunto de funciones de entorno están destinadas para analizar cualquiera de las medidas de cualquier sensor de una misma sesión. Para desarrollarlo se utiliza tanto tableros informativos como una gráfica comparativa en común, en la cual se superponen dos variables para examinar la relación que existen entre ellas. Esto se aplica tanto para una sesión cargada como para una sesión guardada. Las funciones de entorno que desempeñan esta tarea se muestran en la siguiente tabla:

Tabla 4.2. Funciones de entorno de comparación de variables

SESIÓN A	SESIÓN B
<i>guideCvariableAaltitudtempambiente()</i>	<i>guideCvariableBaltitudtempambiente()</i>
<i>guideCvariableAsuspDaltitud()</i>	<i>guideCvariableBsuspDaltitud()</i>
<i>guideCvariableAsuspDsuspT()</i>	<i>guideCvariableBsuspDsuspT()</i>
<i>guideCvariableAsuspTaltitud()</i>	<i>guideCvariableBsuspTaltitud()</i>
<i>guideCvariableAvelocidadAltitud()</i>	<i>guideCvariableBvelocidadAltitud()</i>
<i>guideCvariableAvelocidadSuspD()</i>	<i>guideCvariableBvelocidadSuspD()</i>
<i>guideCvariableAvelocidadSuspT()</i>	<i>guideCvariableBvelocidadSuspT()</i>
<i>guideCvariableAvelocidadtempambiente()</i>	<i>guideCvariableBvelocidadtempambiente()</i>

4.6. DISEÑO DE LA INTERFAZ GRÁFICA DE USUARIO DEL SISTEMA DE ANÁLISIS DE DATOS

La culminación del sistema telemétrico viene precedida por el sistema de análisis de datos. La funcionalidad de su interfaz gráfica permite completar el proceso informativo, logrando hacer partícipe al usuario de la comprensión analítica del comportamiento de la motocicleta de competición.

El sistema de análisis consta de 30 ventanas multifunción conectadas entre sí. En el diseño general de la interfaz gráfica del sistema de telemetría se intenta introducir aquellos aspectos que logran el uso satisfactorio de la misma. Se comienza escogiendo, para este proyecto, una distribución ramificada, ya que es la más apropiada para comunicar interfaces que realizan sus tareas mediante menús de opciones. Se facilita la comprensión de su uso, buscando un diseño intuitivo en su aprendizaje. Se utilizan colores para relacionar las medidas que se examinan con los entornos donde se encuentran, de tal modo, que el usuario sepa en todo momento, a través de un color, que se analiza y donde lo analiza. Se desarrolla un ambiente claro, en el cual el usuario identifica de una forma rápida y precisa los elementos que tiene que utilizar para obtener la información deseada. Otra característica principal es la cercanía entre los controles y los objetos, a los cuales queremos acceder o cambiar. Cada control contiene una etiqueta con la que "pide" al usuario, en todo momento, lo que puede obtener al hacer uso de él. Para el uso correcto, también, se le proporciona al usuario mensajes que le informan y guían sobre las posibilidades de lo que puede hacer. Por último todo el estudio de la moto se divide en varias iteraciones para proporcionar al usuario una interfaz más manejable. A continuación, en los siguientes epígrafes, se describe el diseño más gráfico y personalizado de cada una de las distintas interfaces que nos proporciona la interfaz telemétrica.

4.6.1. INTERFAZ GRÁFICA TELEMETRÍA

El entorno gráfico que ofrece esta interfaz, denominada "TELEMETRÍA", corresponde con el menú principal del sistema telemétrico.

Su diseño está dividido en dos zonas. La primera zona está compuesta por el menú principal, el cual ofrece dos tipos de opciones. La opción uno permite cargar los datos de una nueva sesión realizada por la moto, propiciando el análisis de un nuevo estudio. Los elementos que forman esta opción son: tres pequeñas ventanas para la introducción, por teclado, de los datos "NÚMERO DE PAQUETES DE DATOS", "LATITUD META" y "LONGITUD META", los cuales son necesarios para cargar cada nueva sesión; el botón "LIMPIAR DATOS", el cual es utilizado para limpiar los datos de las tres ventanas anteriores en el caso de que se quiera introducir los datos de una nueva sesión; y el botón "CARGAR SESIÓN" que se utiliza para cargar y analizar los datos de una sesión nueva. La segunda opción, compuesta por el botón "ABRIR DATOS", proporciona el acceso a los datos de una sesión que, de forma previa, ha sido guardada.

En la segunda zona, la menos importante, simplemente destacar que está formada por tres imágenes. La primera, situada en la zona central de la interfaz, representa el circuito donde se desarrolla la competición. La segunda y la tercera imagen se encuentran en la parte superior derecha y representan a los logos de la Universidad Carlos III de Madrid y MotoStudent respectivamente.

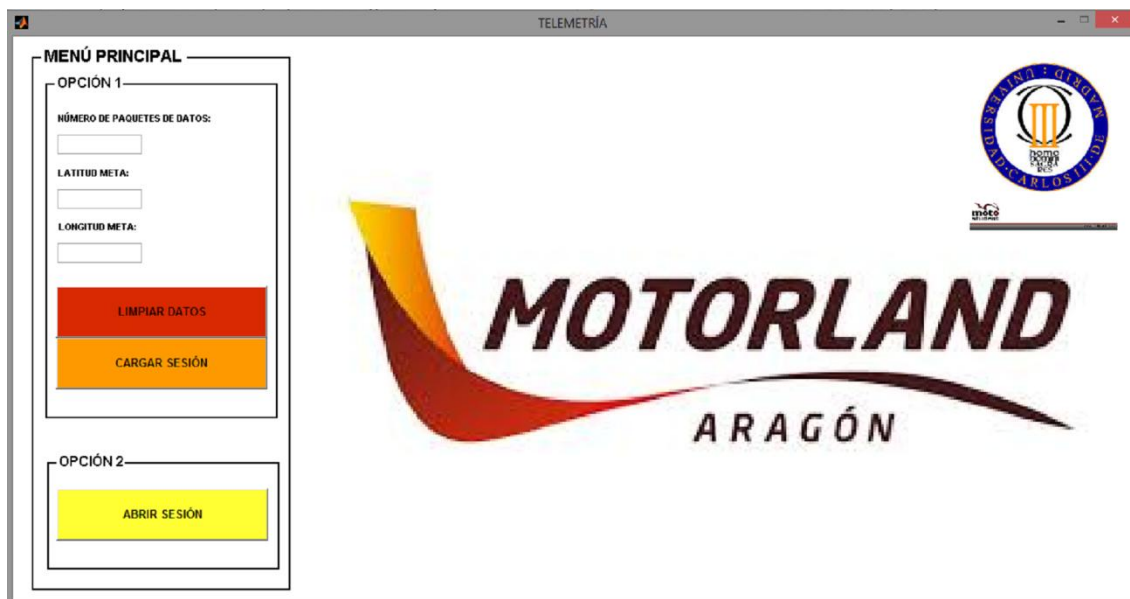


Figura 4.13. Interfaz gráfica TELEMETRÍA del sistema de telemetría

4.6.2. INTERFAZ GRÁFICA TELEMETRÍA-SESIÓN

La interfaz que pertenece a "TELEMETRÍA-SESIÓN", proporciona un espacio con multitud de selecciones para un análisis completo de cualquier sesión.

Su diseño lo forma dos partes bien diferenciadas. La primera está destinada al estudio y a la utilización de los datos de las sesiones cargadas o abiertas. Esta parte integra un tablón informativo, el cual contiene toda la información sobre la sesión de la moto. Este tablón está organizado por filas, destinadas al número de muestras, y por columnas, reservadas para ordenar las diferentes medidas de cada tipo de sensor por vueltas. Otro elemento importante de esta primera parte, es el menú sesión, dividido a su vez, en un submenú de sensores y en otro de opciones.

El submenú de los sensores, lo componen seis botones de colores vivos, que son: "Temperatura Ambiente", de color rojo; "Suspensión Delantera", de color naranja; "Suspensión Trasera", de color amarillo; "Acelerómetro", de color verde; "Climatología", de color azul y "GPS", de color morado. Las funciones de estos botones se centran en el estudio personalizado de las medidas que proporciona el sensor que se ha seleccionado. Cabe destacar que los colores de los botones de este submenú en concreto, coinciden con el color del entorno gráfico al que se accede después de pulsarlos, consiguiendo que, precisamente por los colores, el usuario identifique en todo momento dónde y qué está analizando. El segundo submenú de las opciones está integrado por cuatro botones de colores más suaves, que son: "Comparar Sesión", de color marrón y encargado de acceder al entorno de comparación para comparar datos entre dos sesiones diferentes; "Guardar Sesión", de color azul grisáceo y destinado a guardar la sesión que nos interesa; "Abrir Sesión", de color granate y utilizado para la apertura de sesiones guardadas que se quieran volver a analizar y "Menú principal", de color azul marino y empleado para volver al menú principal.

La segunda parte de este diseño, está formada por dos gráficas y un botón, de color negro, denominado "Comparar circuito". La gráfica situada más arriba, da cabida al circuito de la sesión (A), en cambio, la gráfica situada por debajo recibe, mediante la pulsación previa del botón "Comparar circuito", el circuito de la sesión que se quiere comparar con el circuito de la sesión (A).

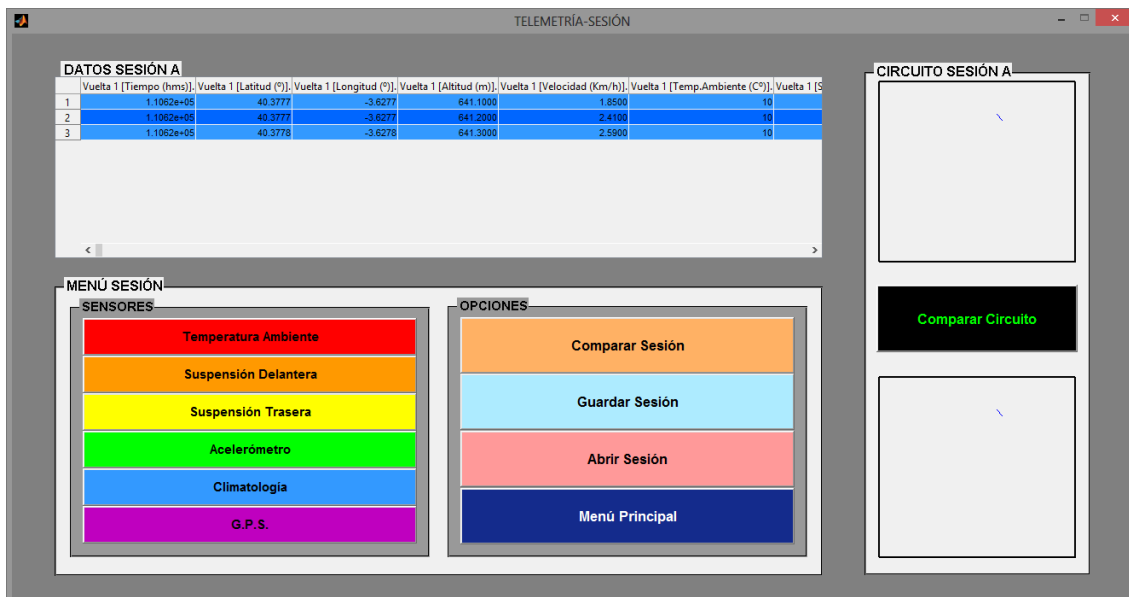


Figura 4.14. Interfaz gráfica TELEMETRÍA-SESIÓN del sistema de telemetría



4.6.3. INTERFACES GRÁFICAS DE SENSORES

Este conjunto de interfaces está formado por aquellos entornos gráficos, a los cuales se accede a través de los distintos botones que componen el menú de sensores de la interfaz "TELEMETRÍA-SESIÓN".

Todas las interfaces relacionadas con estos sensores cuentan con un diseño en común, a excepción del entorno gráfico que proporciona el botón "Climatología", que será puntualizado de un modo posterior y del entorno gráfico del botón "GPS", que mantiene en común todo, exceptuando el diseño de la zona para la gráfica y la introducción de algunos elementos más, donde ambos aspectos se explicarán a posteriori.

En cuanto al resto de interfaces se componen de una gran gráfica para observar de manera precisa y clara la evolución de las medidas proporcionadas por sus respectivos sensores. Disponen de un tablón informativo, con las mismas características que el tablón informativo, para ordenar los datos, de "TELEMETRÍA-SESIÓN", pero mostrando sólo los datos de las medidas del sensor en concreto que se haya seleccionado. Contienen un menú sencillo compuesto por tres botones, que son: "LEYENDA ON/OFF", el cual es un toggle button que proporciona una función on/off, para mostrar, o no, la leyenda de la gráfica; "Menú Sesión", que permite volver al menú de la sesión de la interfaz "TELEMETRÍA-SESIÓN"; y "CUADRÍCULA ON/OFF", el cual, también, es un toggle button y puede añadir o quitar la cuadrícula de la gráfica para un mejor entendimiento de la misma, en el caso de que sea necesario. Tanto el tablón informativo como el menú de opciones de estos entornos, comparten el color que se le asocia al sensor que hemos seleccionado. Para terminar, todas ellas, contienen debajo del menú de opciones una imagen del respectivo sensor utilizado.

En el caso de la interfaz gráfica del botón "GPS", destacar que la zona destinada para la gran gráfica, en este caso, se utiliza para incorporar tres gráficas que representan la velocidad, la altitud y el recorrido. Remarcar, también, que debido a que la gráfica del recorrido es programada para soportar el visionado 3D, se incorpora un nuevo toggle button, denominado "3D ON/OFF", que activa o desactiva el 3D de la gráfica.

Por último, para el botón "Climatología", explicar que al pulsarlo se obtiene un entorno de internet para acceder a la página web que proporciona la información meteorológica del lugar donde se desarrolla la prueba de la motocicleta de competición.

A continuación se adjuntan las diferentes interfaces gráficas que forman este conjunto.

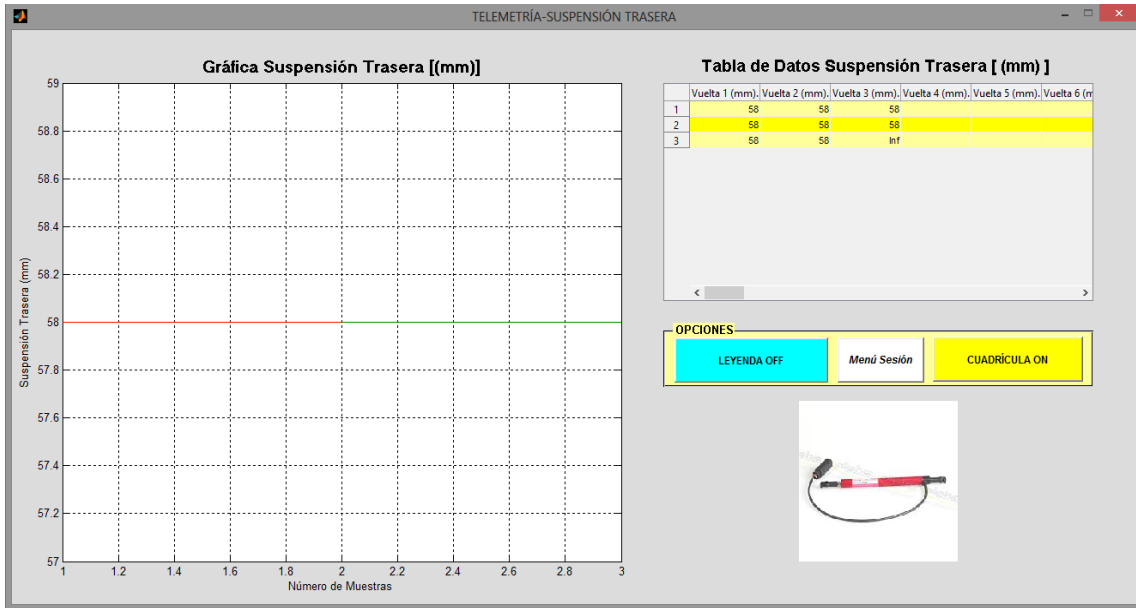


Figura 4.17. Interfaz gráfica TELEMETRÍA-SUSPENSIÓN TRASERA del sistema de telemetría

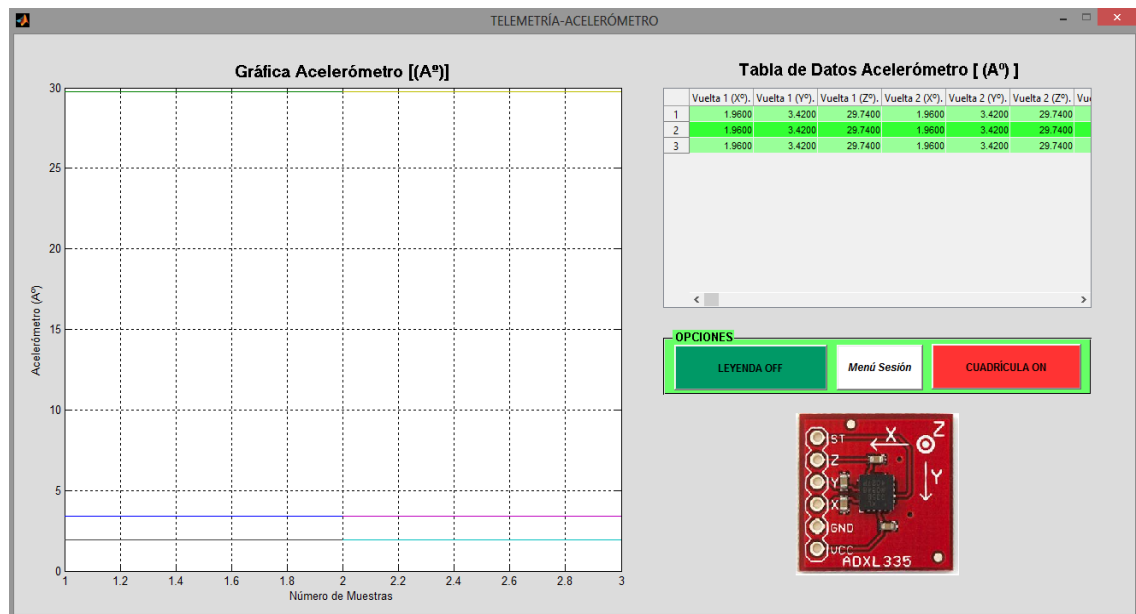


Figura 4.18. Interfaz gráfica TELEMETRÍA-ACELERÓMETRO del sistema de telemetría

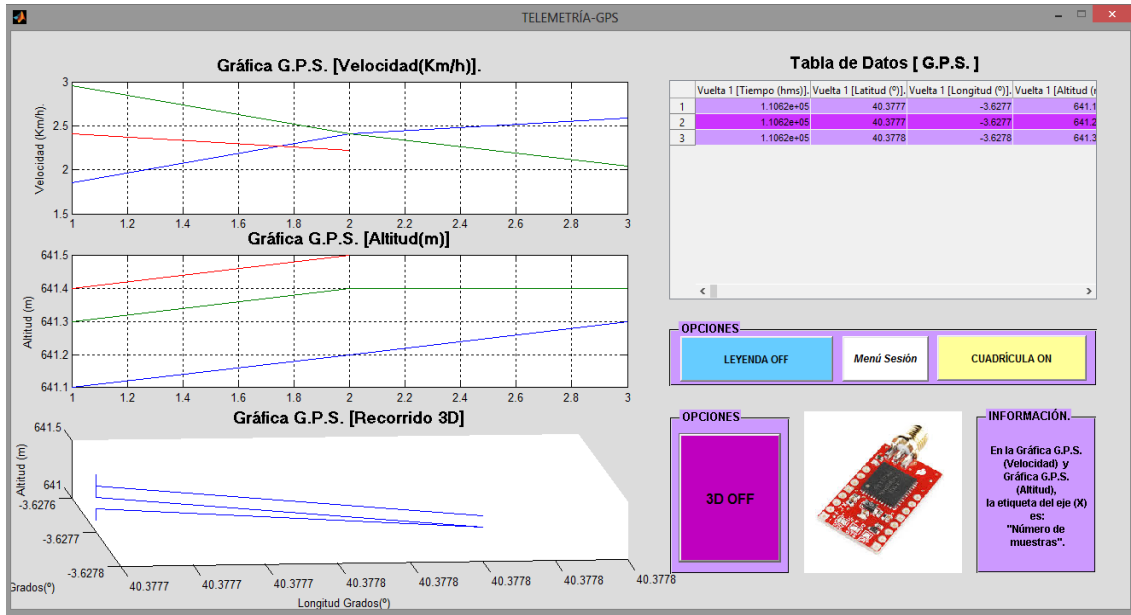


Figura 4.19. Interfaz gráfica TELEMETRÍA-GPS del sistema de telemetría

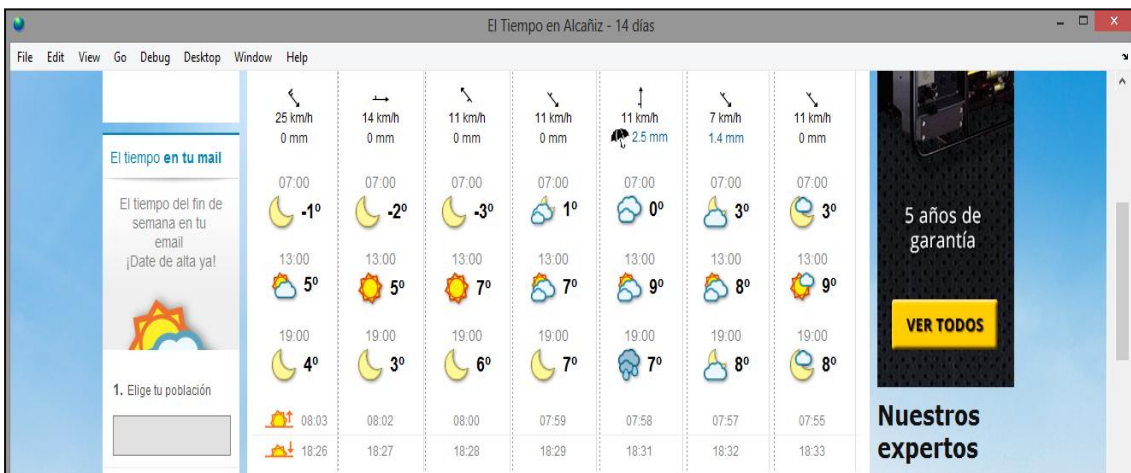


Figura 4.20. Interfaz gráfica TELEMETRÍA-CLIMATOLOGÍA del sistema de telemetría



4.6.4. INTERFAZ GRÁFICA TELEMETRÍA-COMPARAR SESIÓN

Esta interfaz proporciona un entorno gráfico destinado al análisis de datos a través de la comparación de sesiones. Para el diseño de "TELEMETRÍA-COMPARAR SESIÓN" se hace una descripción siguiendo un orden gráfico descendente y de izquierda a derecha del conjunto de elementos que lo componen.

El color del entorno es marrón claro para seguir guardando la relación entre los botones que nos proporcionan datos y sus correspondientes interfaces al pulsarlos. Cuenta con dos tablas informativas, cuyas estructuras coinciden con la del tablón informativo de "TELEMETRÍA-SESIÓN". La primera tabla se corresponde con los datos de la sesión (A), los cuales, de manera inicial, son los datos de una nueva sesión cargada desde Arduino. La segunda tabla proporciona los datos de la sesión (B), los cuales siempre provienen de una sesión guardada.

Dispone de un menú, denominado "COMPARAR", formado por tres botones y tres menús desplegables (pop-up menú) destinados a la comparación. Dos de los tres botones son de color azul marino y se utilizan para la apertura tanto de la sesión (A) como de la sesión (B), ambas, en este caso, ya guardadas. El tercer botón, denominado "MENÚ SESIÓN" y de color blanco, permite volver a la interfaz "TELEMETRÍA-SESIÓN". En cuanto a los menús desplegables se dividen en los menús "Variables Sesión (A)", "Sensores [Sesión A (VS) Sesión B]" y "Variables Sesión (B)". El primero se compone de una serie de variables pertenecientes a la sesión (A) y sirve para analizar la relación que existe entre ellas. El segundo menú está formado por todas las medidas que proporcionan cada sensor y se utiliza para examinar las diferencias del comportamiento de la motocicleta en pruebas distintas. El tercero y último, integra unas determinadas variables de la sesión (B) para estudiar la evolución que proporcionan entre sí. Para finalizar, destacar que los tres menús desplegables proporcionan una serie de interfaces gráficas de usuario para observar las, ya explicadas, funciones que realizan.

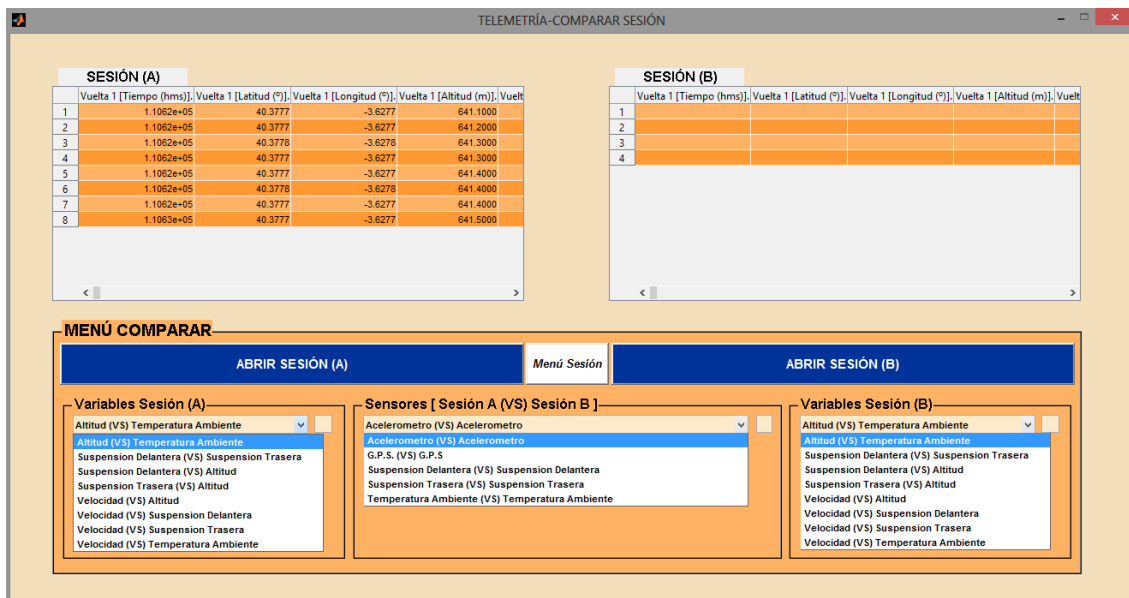


Figura 4.21. Interfaz gráfica TELEMETRÍA-COMPARAR SESIÓN del sistema de telemetría

4.6.5. INTERFACES GRÁFICAS DE COMPARACIÓN ENTRE VARIABLES DE UNA MISMA SESIÓN

Esta colección se forma por aquellas 16 interfaces gráficas que proporcionan las opciones de comparación de los menús "Variables Sesión (A)" y "Variables Sesión (B)". Hemos agrupado todas estas interfaces en un mismo grupo porque, todas ellas, comparten el mismo diseño gráfico y realizan las mismas funciones. La única diferencia es los tipos de variables que se comparan y el tipo de sesión al que pertenecen dichas variables.

Se comienza con la aplicación de un color azul para todos los entornos de los menús de comparación. Cada una de sus interfaces cuenta con dos tableros informativos, los cuales contienen los datos de su correspondiente variable. Estos tableros son coloreados según el sensor que pertenezca la variable que muestran, es decir, tienen y guardan la misma relación de colores que los asociados a los botones que componen el menú "SENSORES" de la interfaz "TELEMETRÍA SESIÓN". Por ejemplo, si seleccionamos la opción "Suspensión trasera (VS) altitud" del menú "Variables Sesión (A)" de la interfaz "TELEMETRÍA COMPARAR-SESIÓN". Al ser la longitud de desplazamiento una medida del sensor de la suspensión trasera y al estar dicho sensor asociado con el color amarillo, el tablón que contiene la medida de este desplazamiento será de color amarillo. Lo mismo ocurre con la altitud pero con el color morado que es el asociado al GPS.

Se continúa, para cada una de las interfaces, con la colocación de una misma gráfica para mostrar las dos variables a comparar. De este modo se consigue ver la relación que mantienen ambas variables y obtener así conclusiones sobre lo ocurrido a la moto en pista. Por último, estos entornos cuentan con un menú de opciones compuesto por dos botones marrones y uno blanco. Los dos primeros, denominados "LEYENDA ON/OFF" y "CUADRÍCULA ON/OFF", se encargan de activar o desactivar tanto la leyenda como la cuadrícula de la gráfica. El tercer botón, denominado "MENÚ COMPARAR" y de color blanco, permite volver a la interfaz "TELEMETRÍA-COMPARAR SESIÓN".

En este caso se ha decidido mostrar la interfaz que coincide con el ejemplo mencionado con anterioridad sobre la suspensión trasera y la altitud del GPS de una sesión (A).

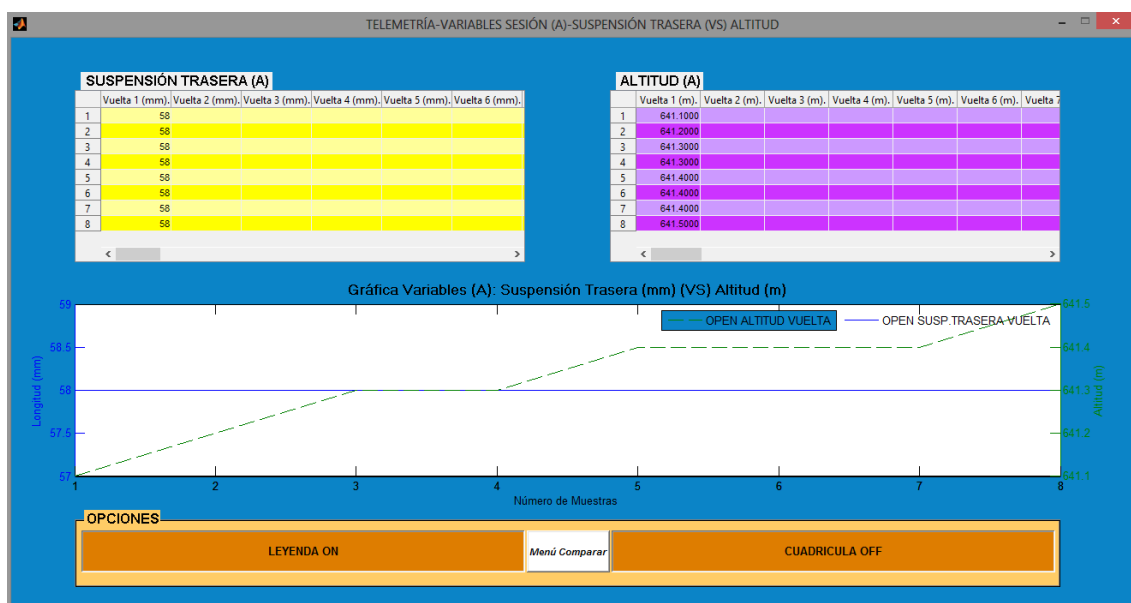


Figura 4.22. Interfaz gráfica TELEMETRÍA-COMPARAR VARIABLES del sistema de telemetría

4.6.6. INTERFACES GRÁFICAS DE COMPARACIÓN ENTRE SENSORES DE DISTINTA SESIÓN

En este conjunto se encuentran las cinco interfaces gráficas que se emplean para la comparación entre las medidas de los sensores de las distintas sesiones. Estos espacios facilitan el análisis del comportamiento de la motocicleta entre sesiones de distintos circuitos o entre sesiones de un mismo circuito.

El diseño de todas estas interfaces gráficas sigue los mismos patrones que el creado para cada una de las interfaces del epígrafe "4.6.5. INTERFACES GRÁFICAS DE COMPARACIÓN ENTRE VARIABLES DE UNA MISMA SESIÓN", pero difiere en dos aspectos. El primero de ellos son los datos mostrados en sus tableros informativos. En este caso, dichos datos, provienen de las medidas que recogen cada sensor comparado. El otro aspecto a diferenciar se encuentra en la zona de la gráfica, es decir, para esta ocasión se construyen dos gráficas. Cada una de ellas representa la información de su sesión correspondiente, (A) o (B), para su posterior comparación. El motivo de emplear gráficas diferentes se debe a MATLAB GUIDE, pues no permite diferenciar en una misma gráfica sesiones distintas en cuanto a número de vueltas y muestras.

A continuación se muestra, en la siguiente figura, la interfaz que compara los datos obtenidos por los sensores GPS de las sesiones (A) y (B) y que pertenece a una de las cinco interfaces de este grupo.

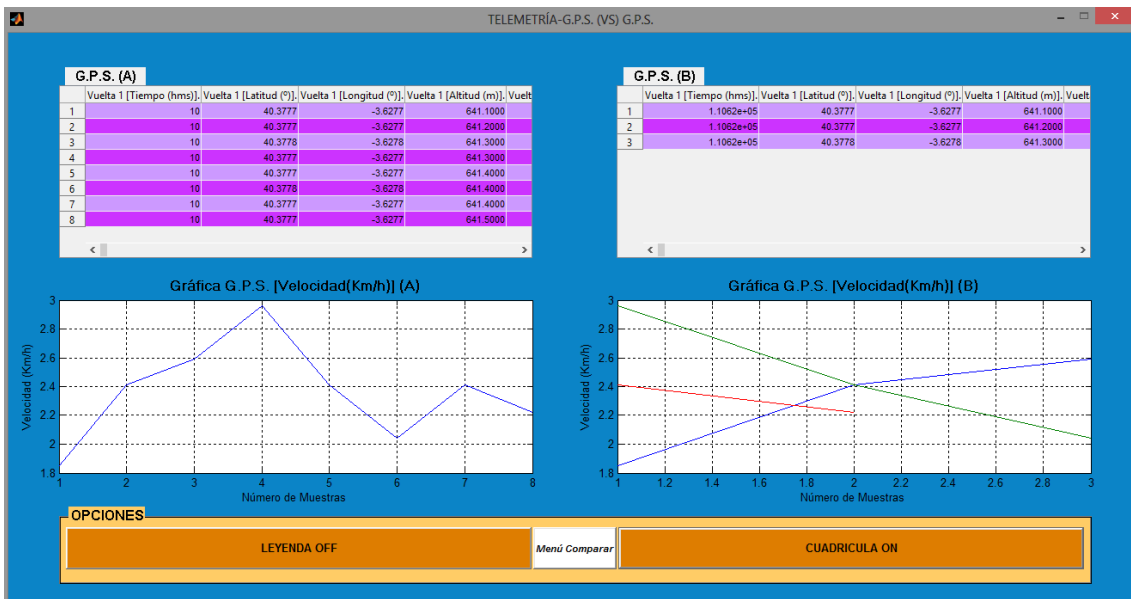


Figura 4.23. Interfaz gráfica TELEMETRÍA-COMPARAR SENSORES del sistema de telemetría



Capítulo 5: FUNCIONAMIENTO DEL SISTEMA DE TELEMETRÍA


5.1. INTRODUCCIÓN


Una vez finalizado el proceso de construcción y procesamiento que conlleva la cadena telemétrica, se completa el objetivo principal de este trabajo de fin de grado. Por tanto, se consigue la creación del primer sistema de telemetría para una motocicleta de competición en la Universidad Carlos III de Madrid.

Este capítulo está vinculado con el aprendizaje y la familiarización del usuario con el funcionamiento y manejo del sistema de telemetría. Se explica, de manera minuciosa, los pasos a seguir por el usuario para conseguir realizar un estudio óptimo de la conducta que experimenta la motocicleta de competición. Se adjunta el flujograma con el que seguir los pasos para un buen uso del sistema. Por último se expone como convertir el sistema de telemetría en una aplicación para que pueda usarse en el ordenador sin necesidad de tener el soporte del programa MATLAB.

5.2. FUNCIONAMIENTO

Este sistema de telemetría, según el tipo de estudio que se quiera realizar, necesita de una preparación previa para poder funcionar. Se pueden hacer dos tipos de estudio sobre la motocicleta. Uno sobre una sesión nueva y otro sobre una sesión ya realizada y guardada por este mismo sistema. A continuación se explica con detalle lo que sucede en ambos estudios antes de utilizar este sistema telemétrico.

Se decide comenzar con la preparación del estudio destinado para una sesión nueva, pues sin ella es imposible estudiar una sesión guardada o almacenada. Lo primero de todo es obtener y anotar los tres datos que hacen posible este primer estudio. El primer dato y el segundo se consiguen a la vez y corresponden a la latitud y longitud de la línea de meta de nuestro circuito en cuestión. Para conseguirlos es necesario realizar un reconocimiento posicional, a través del GPS, de la línea de meta. Una vez conseguidos y anotados estos dos primeros datos, se obtiene el tercero de ellos de la siguiente manera. Se carga el programa del sistema de lectura perteneciente al soporte Arduino. Una vez obtenidos y leídos los datos de la sesión contenida en la micro tarjeta SD, se pulsa el icono scope  y se abre el monitor serial desde donde podemos observar todo el conjunto de datos. Por último, en ese conjunto de datos, se anota el último número de la última columna, consiguiendo así el número de "paquetes de datos" y por tanto el tercer dato.

Una vez anotados los tres datos, se continua con la apertura de la interfaz del sistema de telemetría. Para ello, se abre el programa MATLAB y se accede a la función *guidePrincipal()*. Después, se pulsa el icono Run  y se habilita el menú principal del sistema telemétrico con el que se va a trabajar. A partir de este punto el usuario escoge la opción uno, de las dos posibles, e introduce los tres datos necesarios para este tipo de estudio. Acto seguido, queda preparado el primer estudio para su análisis con el sistema de telemetría.

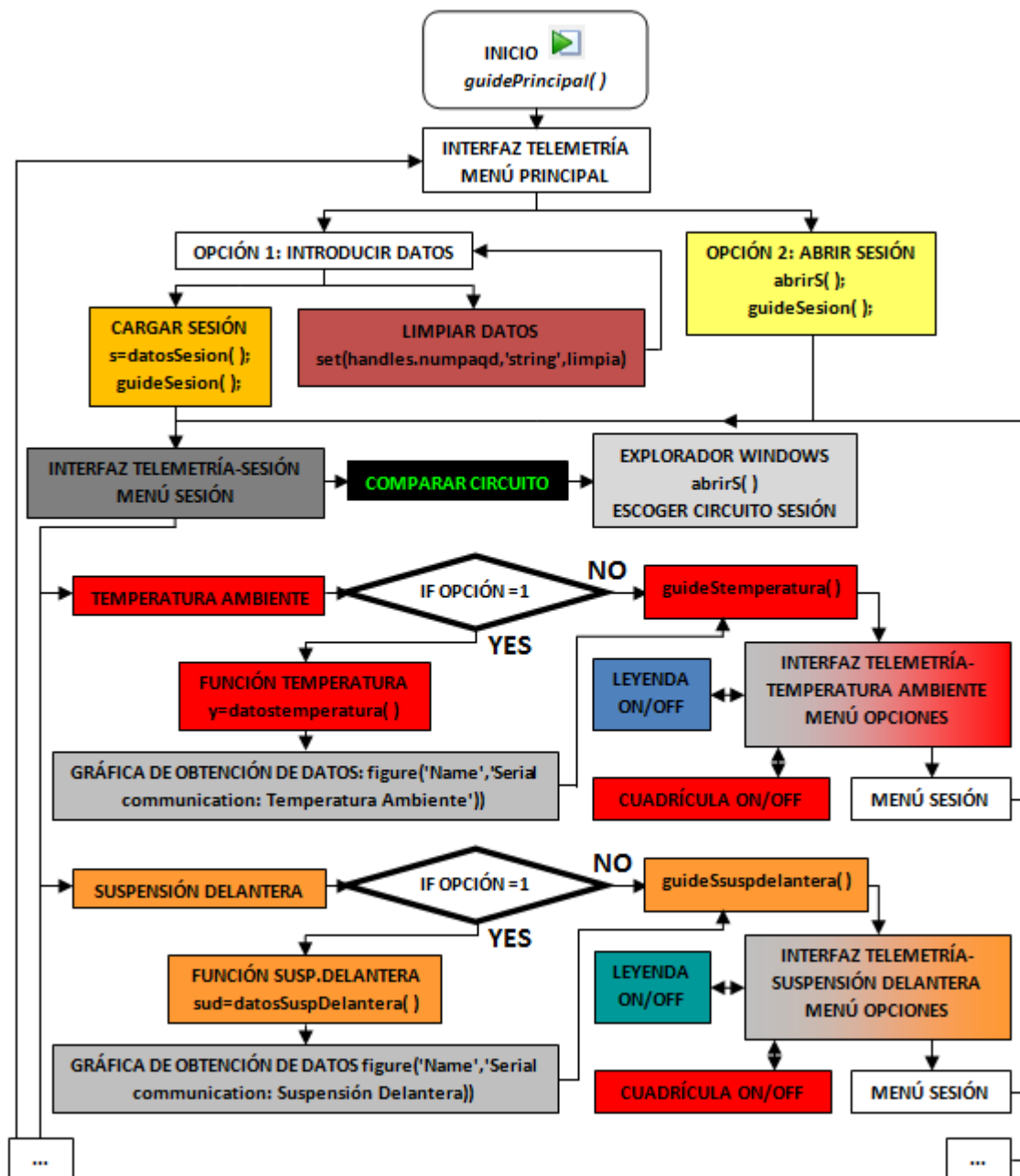


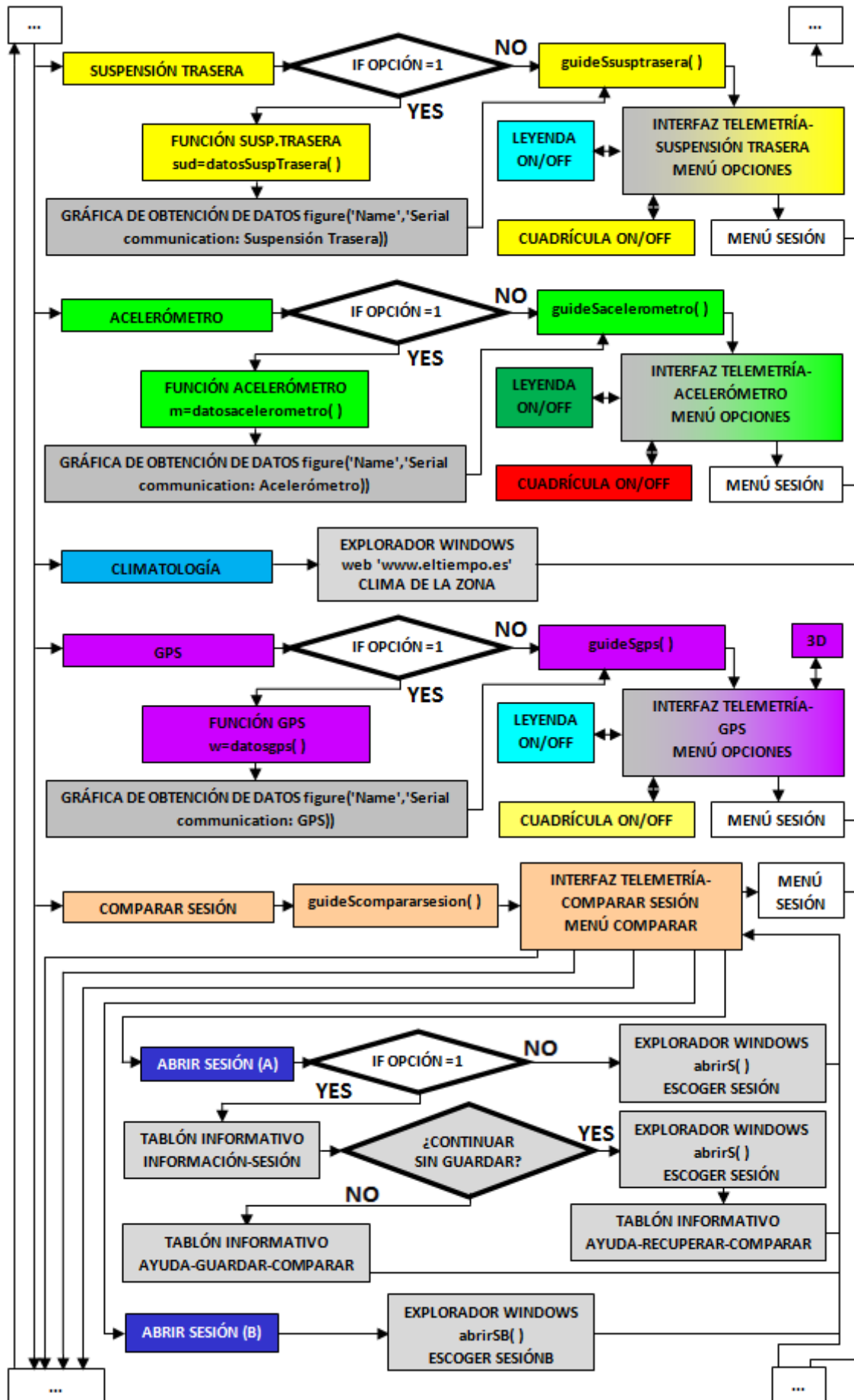
La preparación para el estudio de una sesión guardada es mucho más sencilla. De forma directa y sin necesidad de obtener y anotar ningún tipo de dato, se abre el programa MATLAB y se accede, de la misma forma que en el estudio anterior, al menú principal de la interfaz gráfica del sistema de telemetría. A continuación, se elige la opción dos y queda preparado el segundo estudio para ser examinado por el sistema telemétrico.

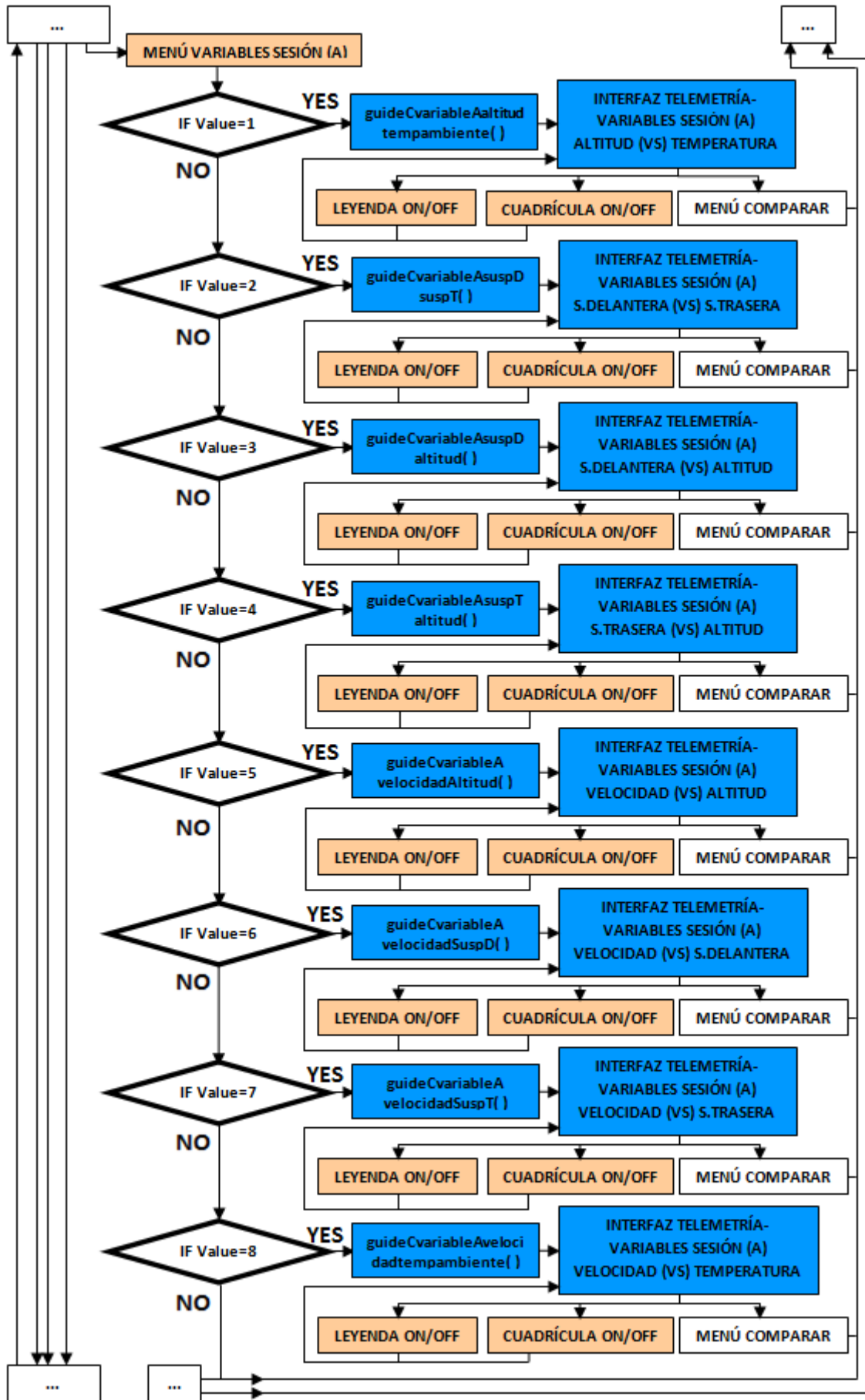
Finalizadas ambas preparaciones, se explica al usuario el funcionamiento de la interfaz del sistema telemétrico mediante su propio flujograma. De este modo, el usuario puede completar el análisis de cualquiera de sus estudios de un modo más sencillo y metódico.

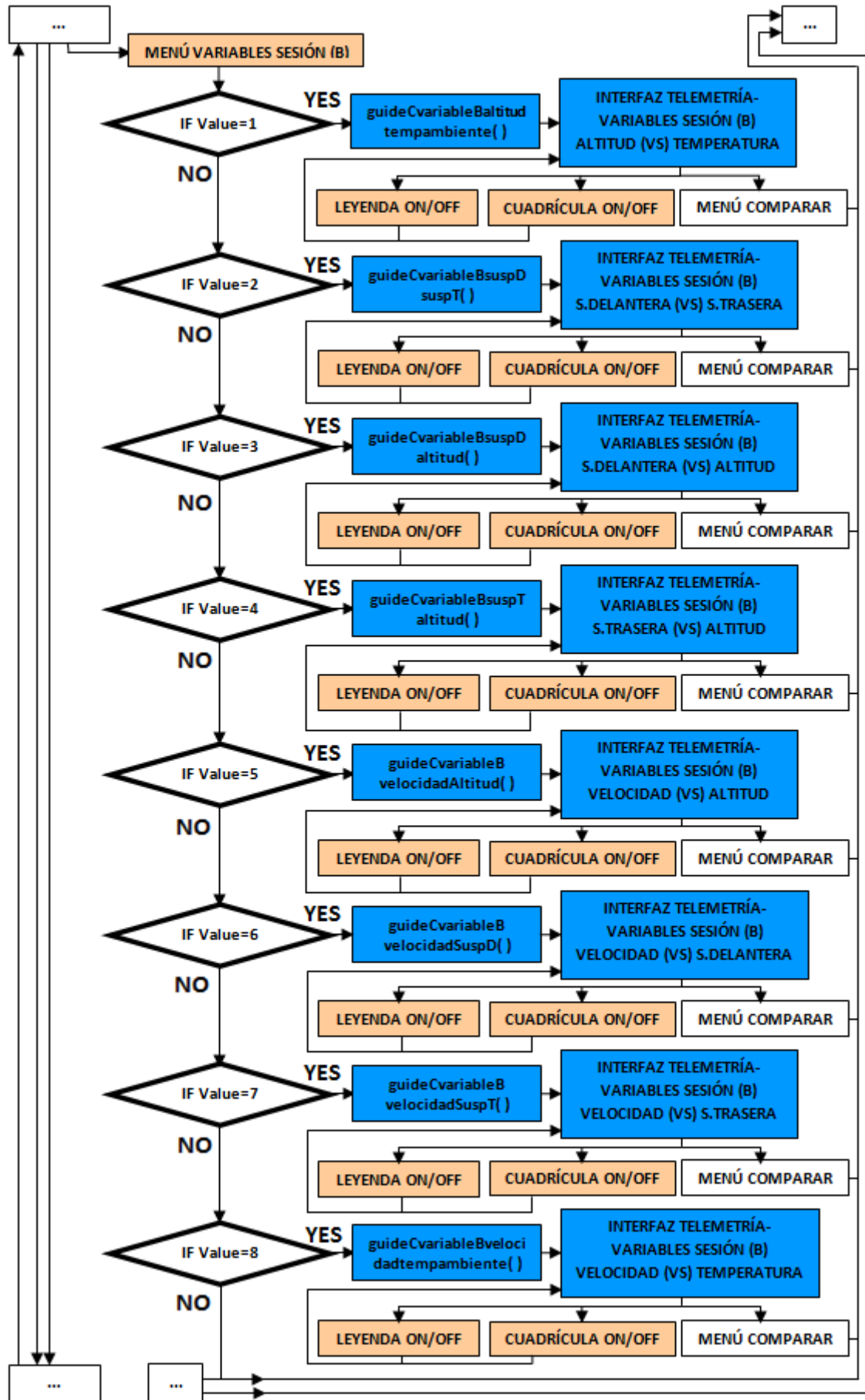
5.2.1. FLUJOGRAMA FUNCIONAL DE LA INTERFAZ TELEMÉTRICA

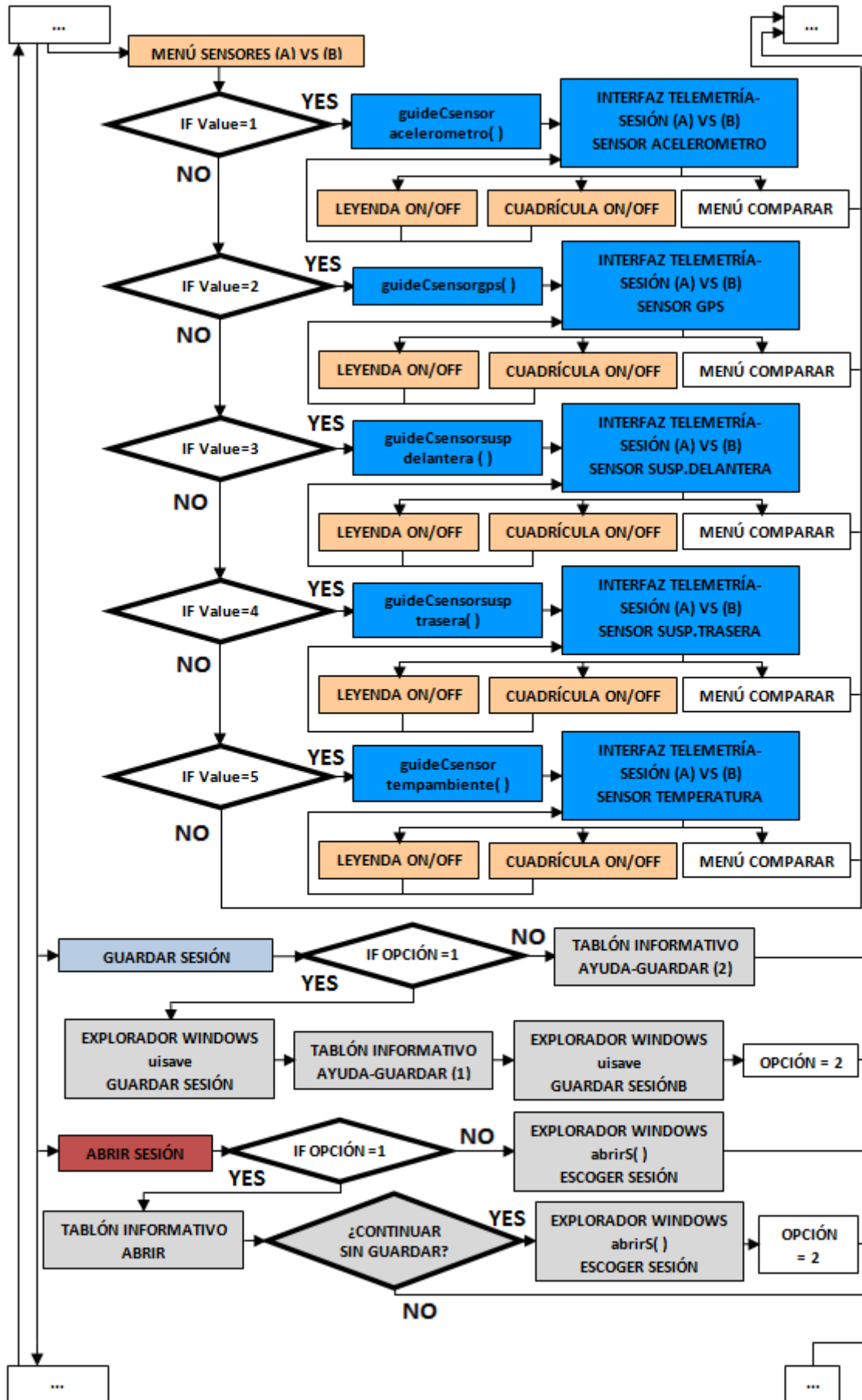
Se aclara que los colores observados, durante el transcurso de las acciones de este flujograma, se corresponden con los colores utilizados en la interfaz gráfica del sistema de telemetría para los entornos a los que se accede y para los botones y menús que se utilizan.











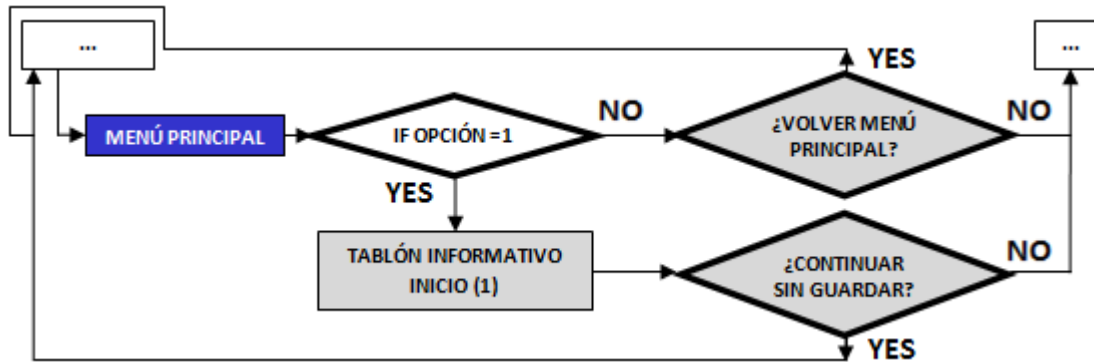


Figura 5.1. Flujograma funcional de la interfaz gráfica del sistema de telemetría

5.3. APLICACIÓN PARA EL SISTEMA DE TELEMETRÍA

Una vez comprendido el funcionamiento y el manejo de la interfaz telemétrica, se le proporciona al usuario la posibilidad de obtener el sistema de telemetría de este proyecto a través de una aplicación para su propio ordenador.

La aplicación se obtiene del siguiente modo. Mediante la propia herramienta MATLAB, se lleva a cabo el proceso de empaquetamiento del código y de los entornos gráficos que desarrollan todo el sistema de telemetría. Tras ser finalizado dicho proceso, se obtiene un programa ejecutable (.exe), o lo que es lo mismo, la aplicación del sistema telemétrico para el ordenador.

Gracias a esta aplicación no es necesario disponer de la instalación de MATLAB. Por consiguiente, el usuario puede disponer de todas las funciones del sistema de telemetría de un modo óptimo y económico, ya que no es necesario pagar la licencia de uso de MATLAB. Por otro lado, esta aplicación puede ser instalada en cualquier ordenador que comparta la mismas características de estructura. La única diferencia experimentada entre el uso de MATLAB y la aplicación es la velocidad a la que se inicia el sistema de telemetría. MATLAB consigue ser más rápida que la propia aplicación, aunque el desarrollo posterior de la ejecución es idéntico.

A continuación se explica, en cinco pasos, el procedimiento para obtener la aplicación del sistema de telemetría, la cual recibe el nombre de "telemetría".

- ❖ **Paso 1:** Se escribe en el Command Window de MATLAB la instrucción `deploytool`, para acceder a la herramienta que permite este proceso. Acto seguido, para iniciar dicho proceso, aparece la ventana "Deployment project", en la cual se introduce el nombre de la aplicación a generar, el lugar donde querer almacenarla y el tipo de trabajo a realizar, para esto último, se selecciona la opción "Windows Standalone Application", la cual se encarga de crear la aplicación para Windows. Con todo completado, se pulsa el botón "OK" y se abre el entorno donde continuar con los siguientes pasos.

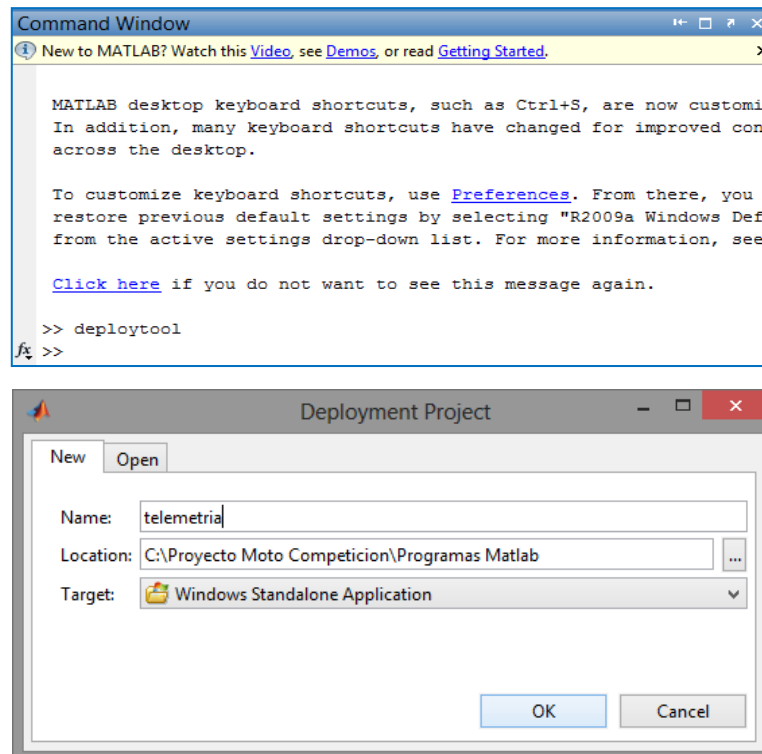



Figura 5.2. Paso 1

- ❖ **Paso 2:** Ubicados en el entorno "Deployment Tool", se selecciona la pestaña "Build" y se observan dos zonas. En la primera de ellas se añade la función principal y en la segunda se incluyen el resto de funciones e imágenes necesarias que completan, en este caso, al sistema de telemetría. A continuación, se pulsa la pestaña "Package" y se pulsa la opción "Add MCR" para añadir elemento que hace posible que la aplicación, la cual se va a crear, funcione sin la necesidad de MATLAB. Por último se pulsa el icono "Build"  para iniciar la construcción de la aplicación.

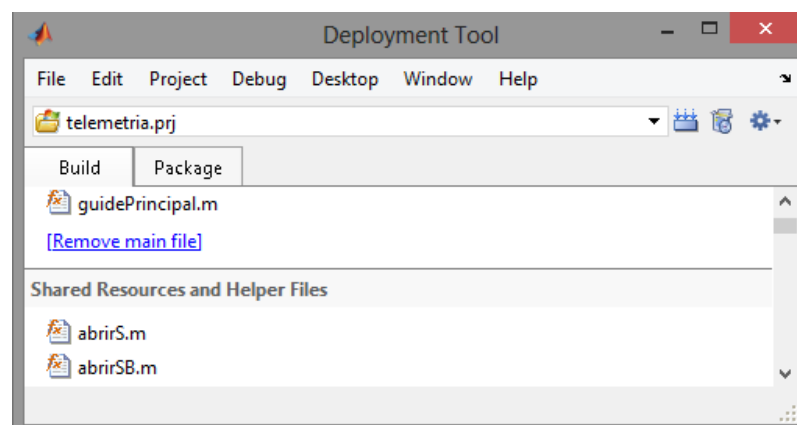


Figura 5.3. Paso 2



- ❖ **Paso 3:** Una vez finalizada la construcción de la aplicación, se comprueba que el programa de tipo aplicación se encuentra generado en la carpeta seleccionada, en el paso 1, para su almacenamiento. Para ello, en esa misma carpeta, se busca y se accede a la carpeta de la aplicación construida, se selecciona la carpeta "distrib" y se localiza el archivo de la aplicación. Acto seguido, se vuelve a la carpeta principal y se localiza el archivo de la aplicación con extensión (.prj). Ambos archivos atienden al nombre indicado en el paso 1 y ambos archivos, una vez localizados, se empaquetan para poder crear la aplicación.

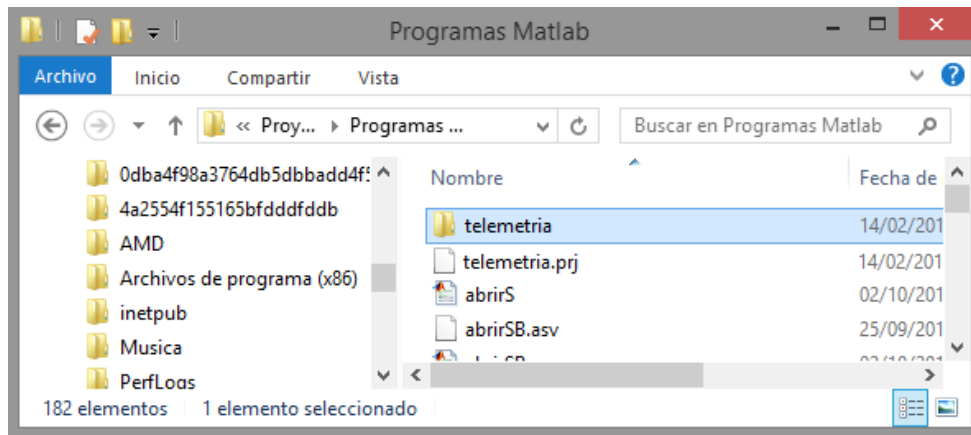


Figura 5.4. Paso 3


- ❖ **Paso 4:** Para empaquetar los archivos mencionados en el paso anterior, se vuelve al entorno "Deployment Tool" y se pulsa el icono "Package" . En consecuencia el proceso pide el lugar donde almacenar este empaquetamiento. Para ello se crea una nueva carpeta, denominada "pkg", en el interior de la carpeta seleccionada en el paso 1 para el almacenamiento. Finalizada esta tarea anterior, se accede a la carpeta "pkg" y se ejecuta la aplicación empaquetada. Acto seguido se instala, de forma automática, el MCR y se instala el "Compiler Runtime" de MATLAB. Para la instalación de este último basta con pulsar "Next" hasta el final de la instalación.



Figura 5.5. Paso 4

- ❖ **Paso 5:** Como último paso, se accede de nuevo a la carpeta "pkg", se busca la aplicación final, generada en el paso 4, y se ejecuta. Por lo tanto se consigue la transformación de un programa a una aplicación para el ordenador sin la necesidad de MATLAB, consiguiendo, en el caso de este proyecto, la aplicación para ordenador de un sistema de telemetría.

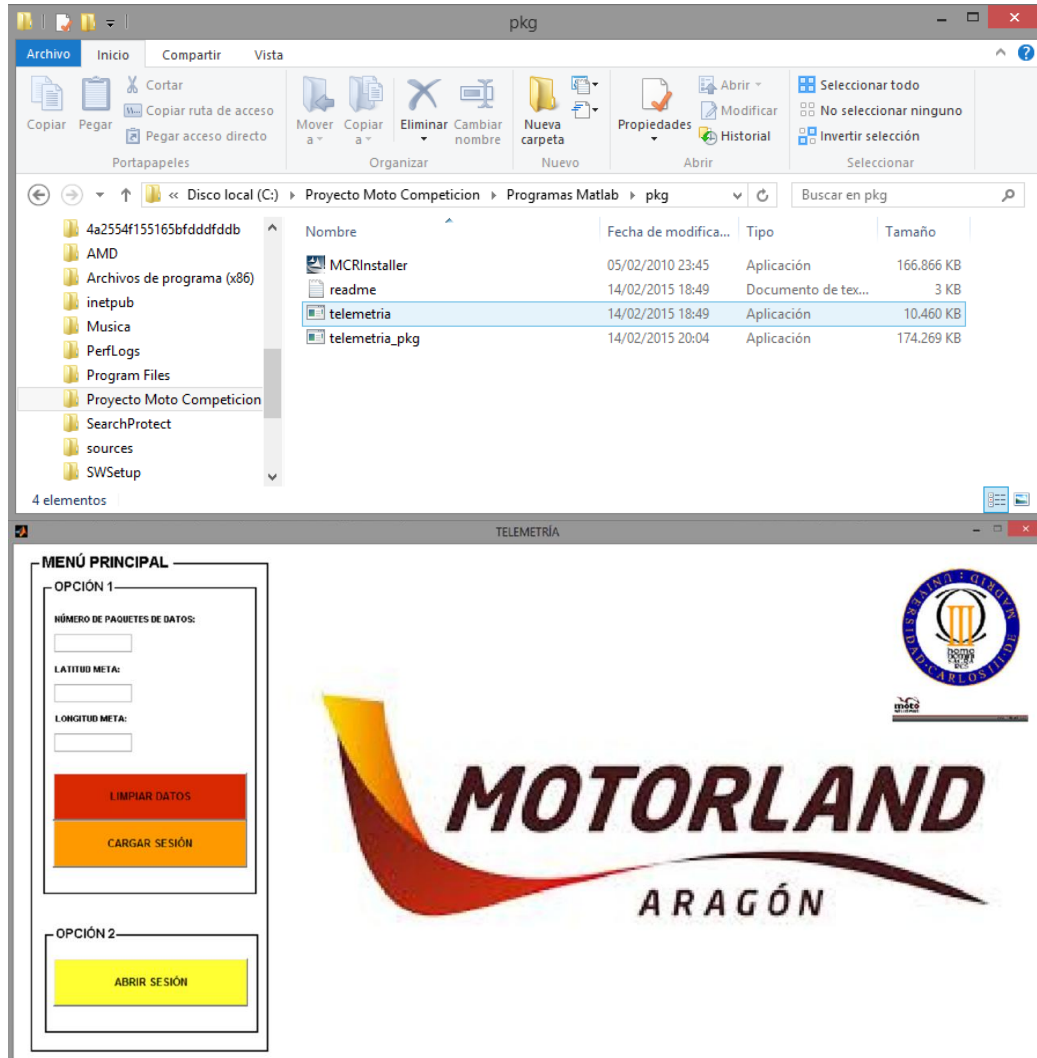


Figura 5.6. Paso 5. Apertura del sistema de telemetría desde la aplicación creada



Capítulo 6: VISUALIZACIÓN DE DATOS EN GOOGLE EARTH

6.1. INTRODUCCIÓN

Para hacer mucho más completo el análisis de datos de la motocicleta de competición, se pensó en complementar el sistema de telemetría con un programa capaz de visualizar el recorrido realizado sobre un mapa real de la zona. El programa seleccionado y que presta estos servicios es Google Earth, el cual se puede descargar de forma gratuita a través de internet. Los datos representados son la latitud, longitud, altitud y velocidad, los cuales son transmitidos a Google Earth mediante un archivo en lenguaje KML (Keyhole Markup Language). Este lenguaje permite representar datos geográficos en tres dimensiones.

Se destaca que la intención de realizar todo este proceso es a través del propio MATLAB, es decir, la idea es la creación de un archivo KML, con los datos que se quieren representar, para visualizarlos en Google Earth utilizando MATLAB. El problema surge que para el desarrollo de este proceso es necesario internet. MATLAB proporciona su propio navegador o entorno de internet pero debido a la versión proporcionada para este proyecto y tras sus respectivas pruebas, no soporta de manera fiable los actuales navegadores de internet. Esto origina que dependiendo de según qué páginas o aplicaciones web, permite, o no, la apertura o la utilización de todas las funciones de dicha página o aplicación. La solución para este problema se puede encontrar explicada entre los siguientes epígrafes de este capítulo

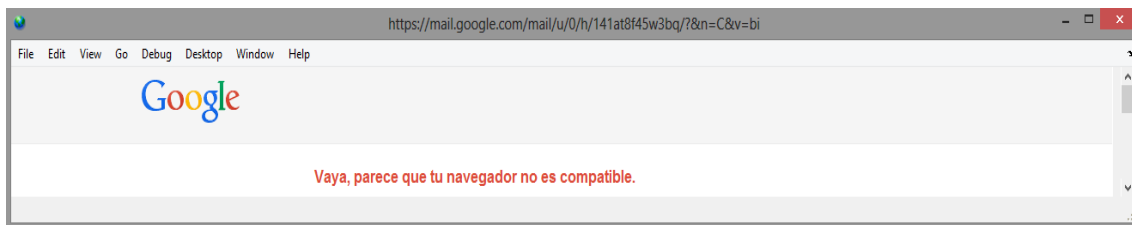


Figura 6.1. Aviso de internet al navegador de MATLAB sobre su compatibilidad al acceder a Google Earth

6.2. PROCESO DE VISUALIZACIÓN DE DATOS

Debido al contratiempo explicado con anterioridad, es necesario explicar cómo sería el proceso de visualización de datos en el caso de no existir problemas con la compatibilidad de los navegadores de internet, y como es el proceso de visualización de datos tras aplicar una solución al problema existente.

Para el primer proceso, como ya ha sido comentado en el epígrafe anterior, la idea era conseguir su desarrollo mediante MATLAB. Para ser más precisos se pensaba en la consecución de esta tarea a través de la utilización de la interfaz gráfica del sistema de telemetría de este proyecto. Para ello es necesario la descarga de la aplicación específica de Google Earth para MATLAB e incluirla en la caja de herramientas ("Toolbox") que dispone el propio MATLAB.



A continuación mediante la programación de la instrucción "kmlwrite", que proporciona MATLAB, se crea un archivo KML con los datos que se quieren representar. Por último, con realizar la llamada correspondiente a Google Earth se observa la trazada realizada sobre un mapa real.

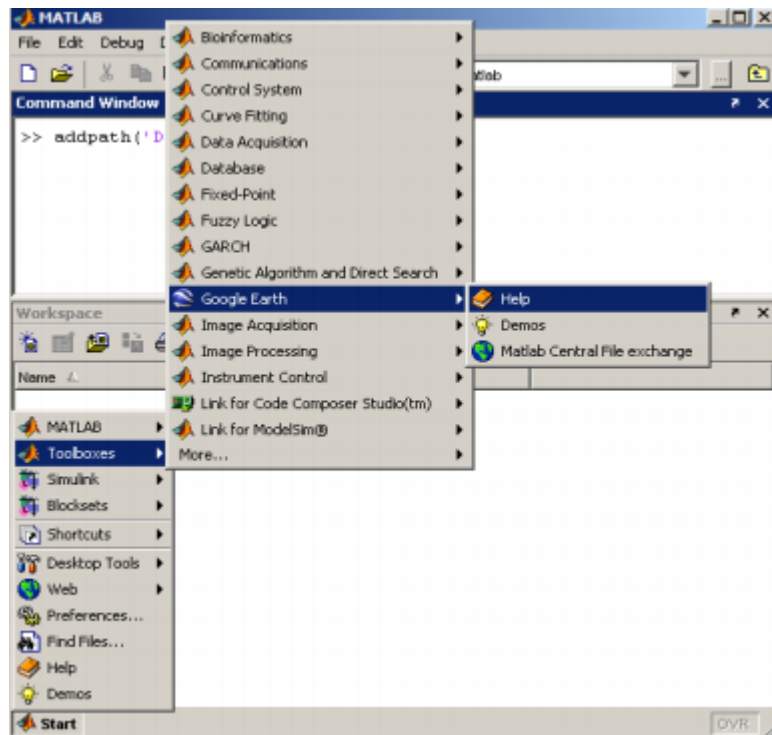


Figura 6.2. Proceso 1: inclusión de Google Earth en MATLAB

Para el segundo proceso, y por tanto como solución al problema, se piensa en el siguiente desarrollo. Se utiliza Google Earth como "programa complemento" del sistema de telemetría en vez de "aplicación" o herramienta del sistema de telemetría. Para ello simplemente con la descarga del programa Google Earth al ordenador es válido. Acto seguido, al no estar asociado Google Earth a MATLAB, no se puede crear el archivo KML mediante la programación de las instrucciones de MATLAB. Por tanto, tras la investigación sobre la conversión de archivos a KML, se encuentra una página web (<http://www.gpsvisualizer.com/mapinput?form=googleearth>), la cual se encarga de la creación del archivo KML de los datos a examinar y permite, mediante una gran cantidad de opciones, representar los datos del modo que se quiera. Para ello, antes, se debe subir, por medio de una de las opciones de la "página", el archivo con los datos ordenados correctamente. Este archivo de datos se crea en un block de notas, en el cual se copian, de los datos almacenados en la micro tarjeta SD, aquellos que se necesitan para esta labor. En cuanto

a la organización de los datos es fundamental, para que se pueda originar el archivo KML, que estén separados por comas y organizados por columnas, las cuales deben ir nombradas con el tipo de dato que contienen (Ejemplo: latitud,longitud,velocidad,altitud). Por último se pulsa sobre el archivo KML creado y los datos son mostrados por Google Earth de forma automática.

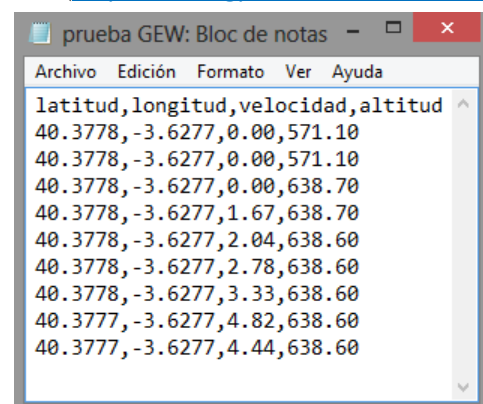


Figura 6.3. Ejemplo de archivo de datos para convertir a KML



The figure consists of three sequential screenshots from a web browser:

- Top Screenshot:** The 'Convert your GPS data for use in Google Earth' page. It features a title, a brief description of the tool's function, and several configuration sections:
 - General map parameters:** Includes options for output file type (.kml (zipped)), units (Metric), Google Earth doc name, and DEM elevation data.
 - Track options:** Includes track opacity (100%), line width (4), colorization (Track), altitude mode (Clamped to ground), and drawing options like shadows and tickmarks.
 - Waypoint options:** Includes labels, icons, and show options for waypoints.
 - Upload your GPS data files here:** Three file selection boxes for uploading data.
 - Or paste your data here:** A text area for pasting coordinates.
 - Or provide the URL of data on the Web:** A text area for web-based data.
- Middle Screenshot:** The 'GPS Visualizer' output page. It displays the 'Google Earth output' section with a download link for the file '20150216072815-00093-map.kmz'. It also includes a sidebar with navigation links like 'MAKE A MAP', 'MAKE A PROFILE', and 'Geocode addresses', along with social media icons for Facebook and Me.gi.
- Bottom Screenshot:** A view of Google Earth showing a 3D aerial map of a city. A red line representing the GPS track is visible, with a callout box labeled 'prueba GEW' pointing to a specific location on the track.

Figura 6.4. Conversión del archivo a KML (Primera imagen), Creación del archivo KML(segunda imagen) y Visualización de los datos (KML) del ejemplo "prueba GEW" mediante Google Earth (tercera imagen)



Capítulo 7: ENSAYOS EXPERIMENTALES Y RESULTADOS REALES

7.1. INTRODUCCIÓN

Con el sistema de telemetría dispuesto y preparado, se experimenta y trabaja con él para observar los resultados obtenidos de los múltiples y diferentes ensayos que se han realizado. Estos ensayos han sido completados abordando diferentes localidades de Madrid. Las calles de Leganés, Móstoles y Vallecas han servido como "circuitos" improvisados para la adquisición y análisis de datos experimentales.

7.2. DESARROLLO DE LOS ENSAYOS EXPERIMENTALES

Debido a que no ha sido posible el acceso a la motocicleta de competición para realizar este tipo de pruebas, no ha sido posible la lectura y el tratamiento de datos correspondientes a la misma. Por tanto, para poder hacer viable la finalidad de este proyecto, se utilizó, como solución entre los medios que se disponían, un automóvil propio. A pesar de ello, gracias a esta solución, se ha conseguido de un modo óptimo y satisfactorio cumplir con el objetivo propuesto para este proyecto. Se ha logrado mostrar por pantalla un análisis completo de los datos adquiridos de un vehículo, capacitando al usuario de un estudio del comportamiento de dicho vehículo. Por tanto, gracias a este "contratiempo" se llega a la conclusión de que, instalados los sensores adecuados de una forma correcta, este sistema de telemetría es perfectamente válido para cualquier tipo de vehículo, lo cual incluye a la motocicleta de competición del equipo MotoStudent de la Universidad Carlos III de Madrid.

Los ensayos, para el buen funcionamiento del sistema de telemetría, han sido multitudinarios. Eran necesarias una gran cantidad de pruebas para corroborar el buen hacer de cada una de las funciones que proporciona la interfaz telemétrica.

Los diferentes ensayos se dividen por recorridos. Para cada uno de ellos se realiza una prueba de análisis de datos mediante la interfaz del sistema de telemetría. Esta prueba se fragmenta, a su vez, en cuatro partes. La primera parte consiste en el correcto "volcado de datos" de Arduino a MATLAB, si se quiere cargar una sesión, o en la apertura correcta de una sesión ya guardada. La segunda parte consta del buen funcionamiento para mostrar tanto los datos conjuntos de la sesión como los datos por separado de cada sensor. La tercera parte trata de realizar un estudio comparativo entre los distintos sensores de distintas sesiones y entre las distintas variables de una misma sesión. La cuarta parte se fundamenta en el almacenamiento y apertura de sesiones y en la buena comunicación a través de los distintos menús que ofrece la interfaz gráfica. Completada la prueba se pueden sacar conclusiones y en consecuencia, actuar en beneficio de la motocicleta de competición de un modo certero y con un mayor conocimiento de sus necesidades.



7.3. RESULTADOS REALES

Para observar los resultados reales de este proyecto, se ha escogido una de las pruebas desarrolladas en los alrededores de la Universidad Carlos III de Madrid, situada en la localidad de Leganés, y una de las pruebas realizadas en las calles de Vallecas.

El grueso de los resultados se evalúan para la prueba relacionada con la universidad, por tanto se comienza introduciendo, por teclado, los datos necesarios para obtener los resultados reales de este primer ensayo y se pulsa "CARGAR SESIÓN". Dichos datos son adquiridos según lo explicado en el epígrafe 5.2. "Funcionamiento" del capítulo 5 del presente documento.



Figura 7.1. Introducción de datos para el ensayo Universidad Carlos III de Madrid

A continuación, en el "MENÚ SESIÓN", se muestran todos los datos adquiridos y el trazado del recorrido que forman el ensayo de la sesión de la Universidad Carlos III de Madrid.

DATOS SESIÓN A						
	Vuelta 1 [Tiempo (hms)]	Vuelta 1 [Latitud (°)]	Vuelta 1 [Longitud (°)]	Vuelta 1 [Altitud (m)]	Vuelta 1 [Velocidad (Km/h)]	
1	1.0593e+05	40.3350	-3.7630	602.2000	26.8500	
2	1.0593e+05	40.3349	-3.7631	612.8000	25.5600	
3	1.0594e+05	40.3349	-3.7631	621.8000	25	
4	1.0594e+05	40.3348	-3.7632	629.2000	25.7400	
5	1.0594e+05	40.3348	-3.7633	635.5000	27.0400	
6	1.0594e+05	40.3348	-3.7634	640.7000	27.9700	
7	1.0594e+05	40.3348	-3.7635	645.2000	31.3000	
8	1.0594e+05	40.3348	-3.7636	648.8000	33.8900	
9	1.0594e+05	40.3347	-3.7637	652	35.1900	
10	1.0594e+05	40.3347	-3.7638	654.6000	36.8500	

Figura 7.2. "MENÚ SESIÓN" con el tablón de datos y el recorrido del ensayo Universidad Carlos III de Madrid

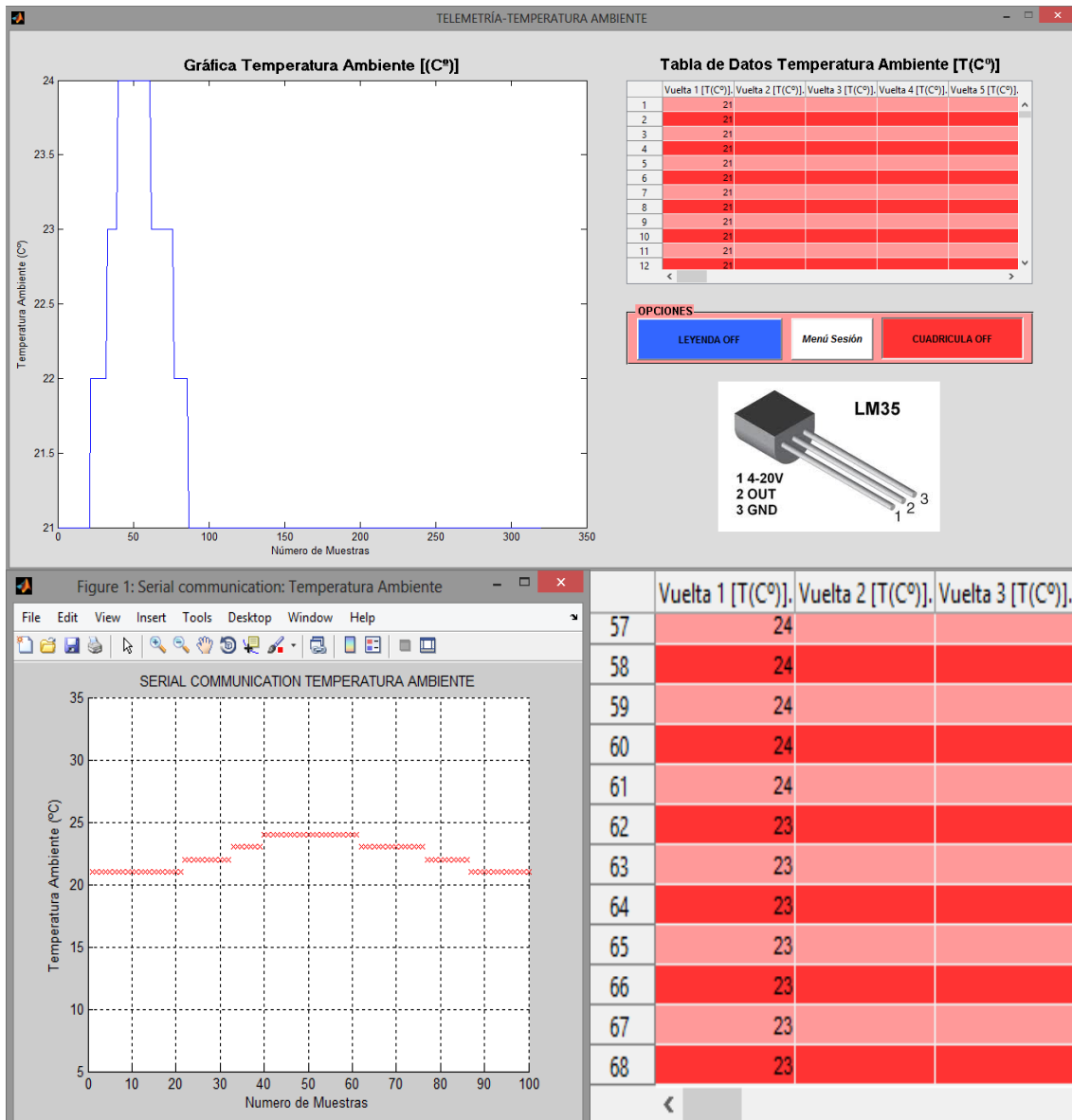


Figura 7.3. Estudio de los resultados del sensor temperatura del ensayo Universidad Carlos III de Madrid

En la "Figura 7.3." se analizan las variaciones que experimenta la temperatura ambiente del automóvil a lo largo del recorrido del ensayo. Debido a que los cambios de la temperatura en el ambiente se producen de manera suave a lo largo del tiempo, para obtener una respuesta evidente del funcionamiento del sensor LM35 durante el transcurso del recorrido, se procedió a presionar con los dedos dicho sensor. Se puede observar como la temperatura se mantiene estable a 21°C y con el paso del tiempo, entre las muestras número 21 y 87 (ambas incluidas), sufre una evolución hasta los 25°C para después volver a disminuir y recuperar la estabilidad a los 21°C.

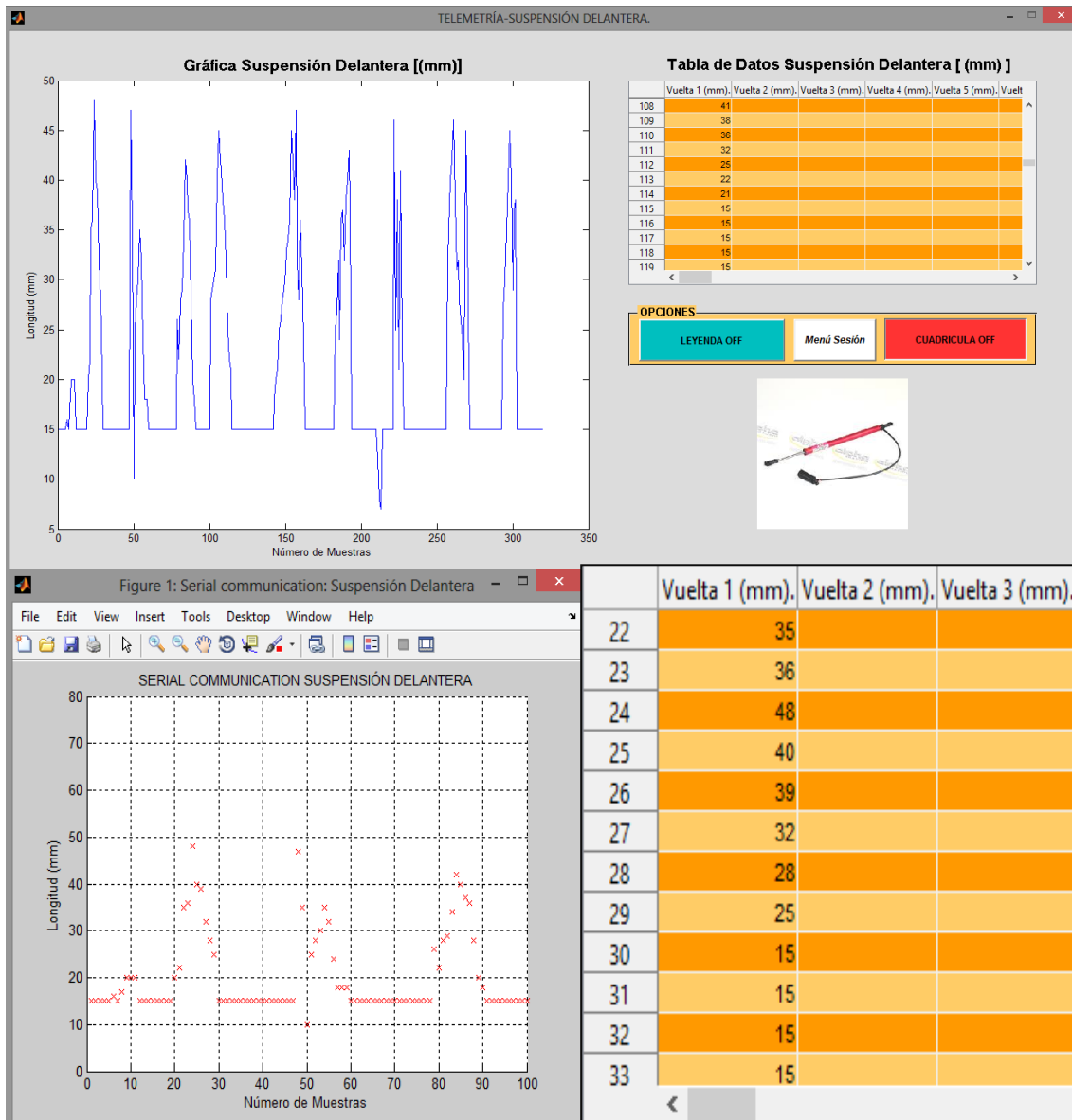


Figura 7.4. Estudio de los resultados del sensor suspensión delantera del ensayo Universidad Carlos III de Madrid

El análisis de las medidas de desplazamiento de las suspensiones no pudo realizarse con datos reales. Esto es debido a que los sensores de desplazamiento, los cuales se proporcionaron para este proyecto, son sensores con dimensiones destinadas a las suspensiones de la motocicleta MotoStudent, por lo tanto, a pesar de intentarlo, fue imposible acoplarlos al automóvil. Para solucionar la evaluación de esta medida, se intentó simular el trabajo desarrollado por estos sensores comprimiéndolos de manera manual en cada punto crítico del recorrido, es decir, zonas con giros pronunciados (rotondas), zonas de acelerado y frenado y zonas con badenes. Esto afecta a todos los entornos gráficos relacionados con la suspensión delantera y trasera.

A pesar de todo, el cometido para este proyecto en concreto está cumplido, pues sean datos reales o experimentales se pueden mostrar y analizar mediante la creación de este sistema de telemetría. Prueba de ello es la "Figura 7.4.", la cual permite con sus resultados hacerse una idea de los desplazamientos sufridos en los momentos de más trabajo para estos sensores.

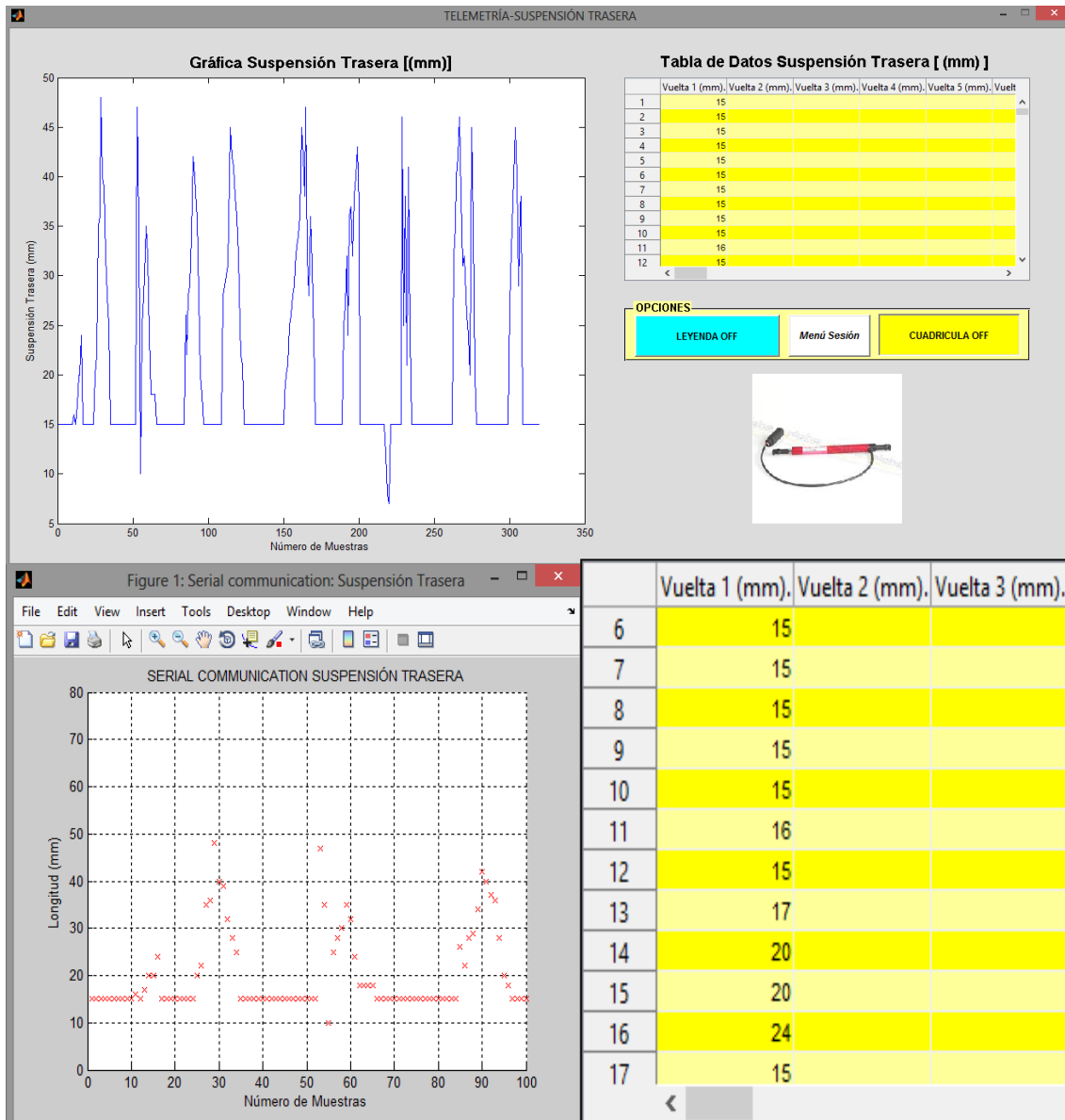


Figura 7.5. Estudio de los resultados del sensor suspensión trasera del ensayo Universidad Carlos III de Madrid

Las medidas adquiridas por el sensor suspensión trasera atienden a la misma puntualización explicada con anterioridad para la suspensión delantera. Por tanto los datos mostrados por la "Figura 7.5." parten de una simulación experimental durante el ensayo del recorrido real. Gracias al funcionamiento de este sistema es posible generar una idea muy cercana al comportamiento real de la suspensión trasera de un automóvil. Se observa en que momentos se comprime dicha suspensión.

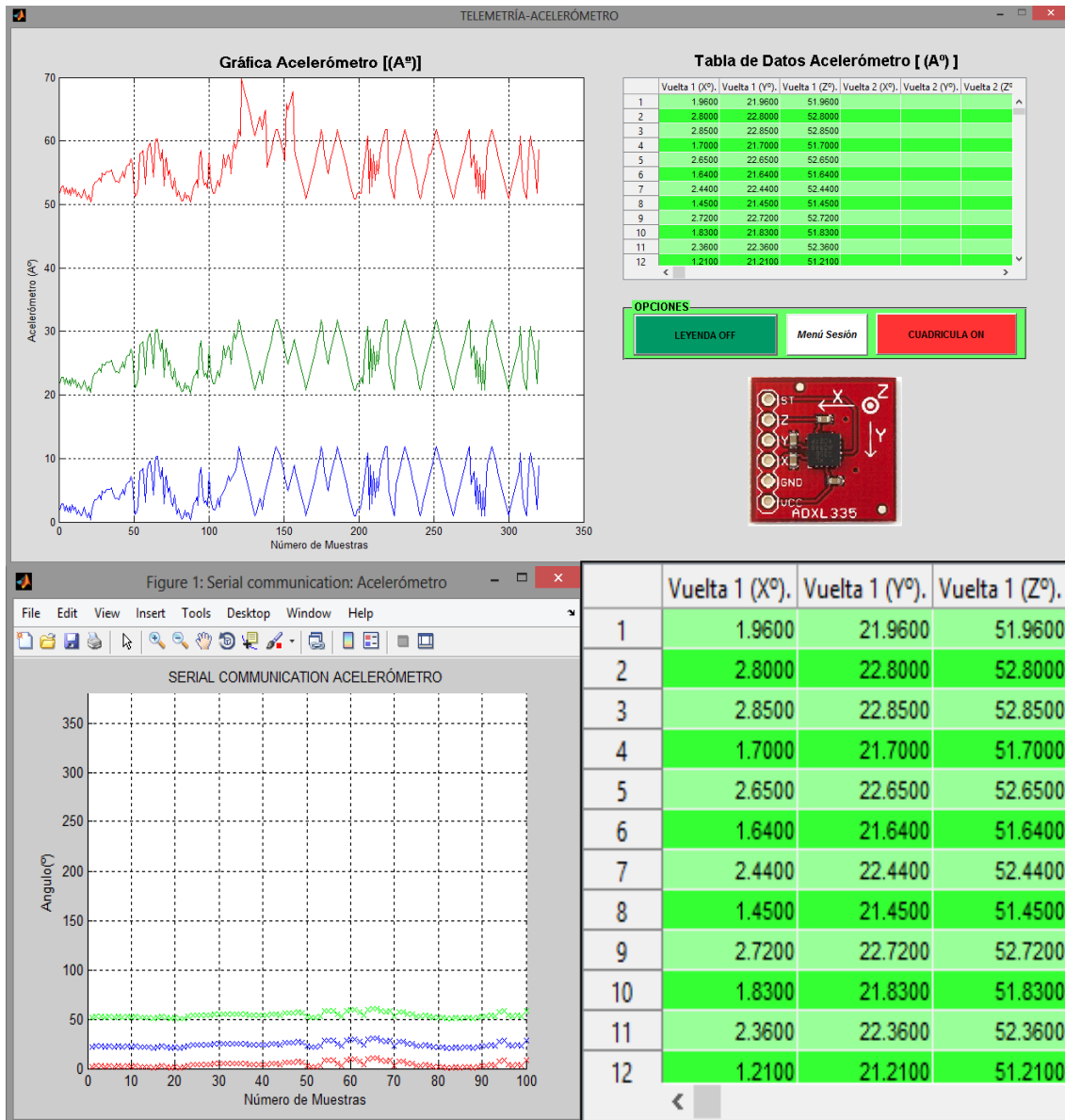


Figura 7.6. Estudio de los resultados del sensor acelerómetro del ensayo Universidad Carlos III de Madrid

Analizando la presente captura, se ve de una forma clara la progresión de las aceleraciones de cada eje de referencia del automóvil. La aceleración de color rojo representa al eje (x), de color azul se identifica la aceleración para el eje (y) y de color verde se muestra el análisis de la aceleración para el eje (z). Todas estas aceleraciones experimentan variaciones manifestadas por grados angulares (°). Se pueden observar las perturbaciones que transmiten cada uno de los ejes en determinados momentos del recorrido. Algunas de estas aceleraciones se pueden manifestar hasta los 360(°).

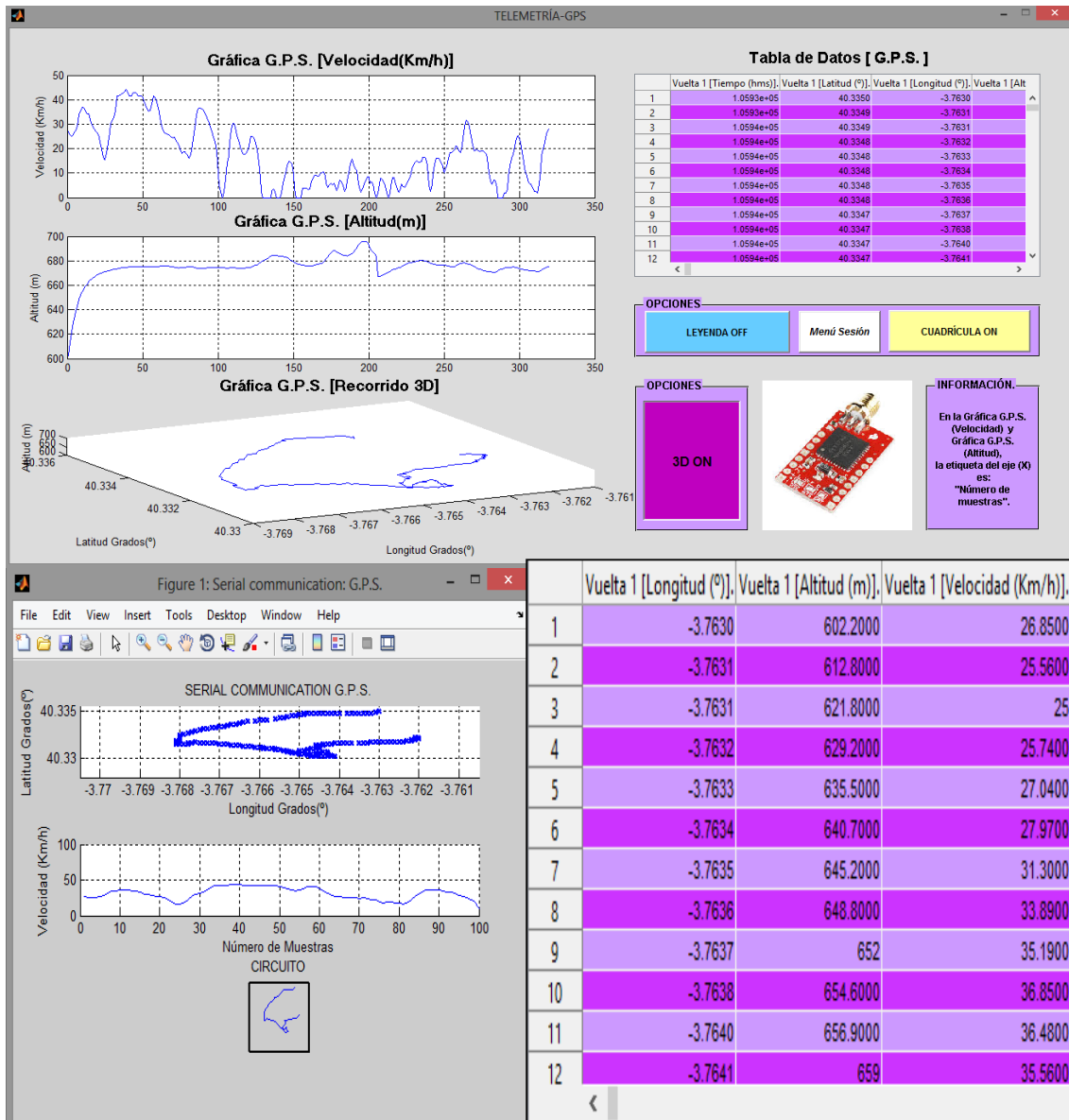


Figura 7.7. Estudio de los resultados del sensor GPS del ensayo Universidad Carlos III de Madrid

El GPS permite el análisis de las medidas del tiempo horario, latitud, longitud, altitud y velocidad. La "Figura 7.7." proporciona el análisis de posicionamiento a través de su recorrido, el estudio de la velocidad a la que se ejecuta el trazado y la altitud a la que se encuentra cada parte del circuito. Con el trazado se pueden analizar los puntos claves del circuito. Para este ensayo, gracias a las variaciones de la velocidad, se observa y se interpreta perfectamente que el tipo de recorrido llevado a cabo es urbano. Por otro lado, los constantes "picos y valles" reflejan tanto la puesta en marcha y avance normalizado, como las reducciones y paradas del vehículo durante todo el trayecto. Todo este conjunto permite conocer y aportar mejoras a la conducción del piloto.



Se continua obteniendo más resultados en el entorno gráfico que nos proporciona el "MENÚ COMPARAR". En este espacio se examinan tanto los resultados obtenidos de relacionar variables de la misma sesión o ensayo, como los resultados de comparar sensores entre distintas sesiones o pruebas.

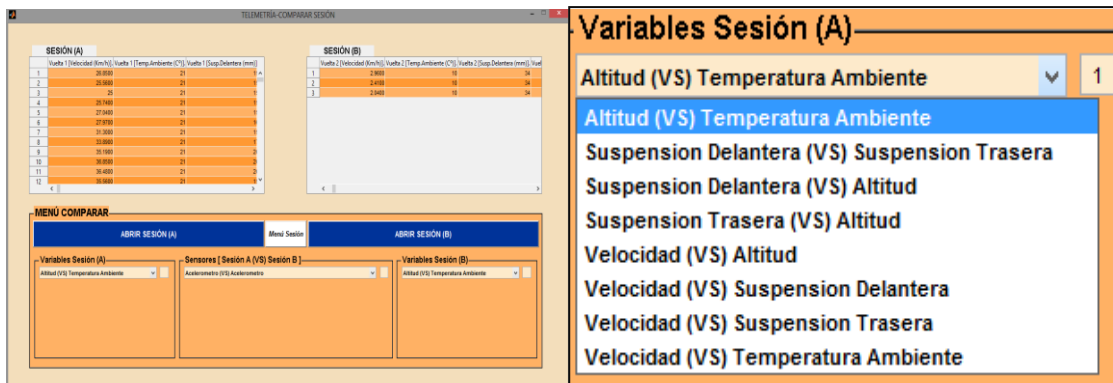


Figura 7.8. "MENÚ COMPARAR" variables de una misma sesión del ensayo Universidad Carlos III de Madrid

A continuación se muestran los datos comparativos entre las variables del ensayo de la Universidad Carlos III de Madrid.

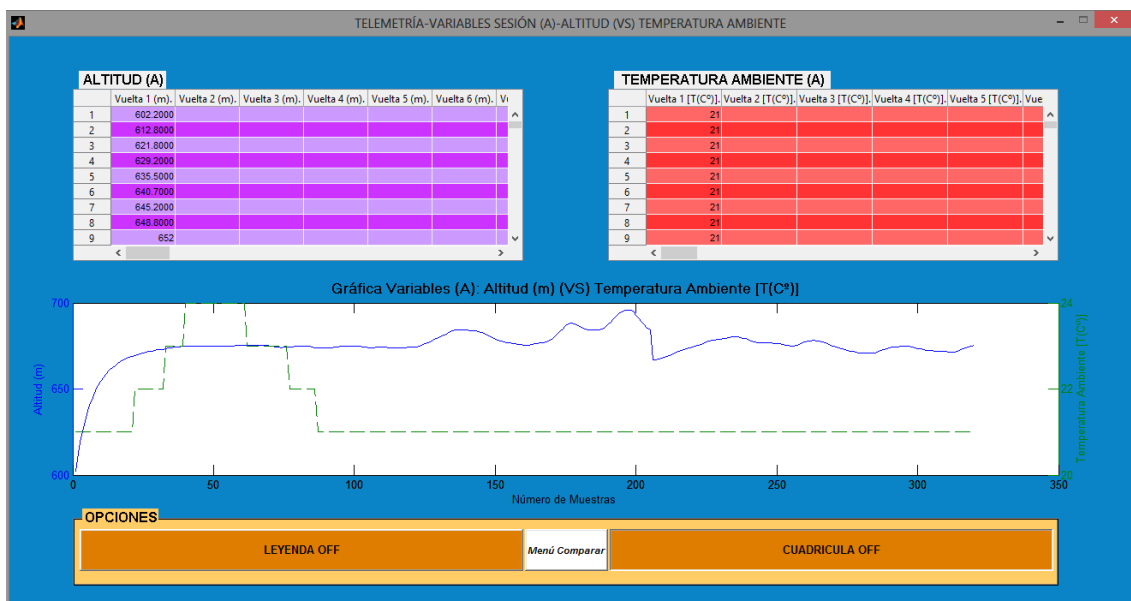


Figura 7.9. Estudio de la relación entre las variables altitud y temperatura del ensayo Universidad Carlos III de Madrid

En la "Figura 7.9." aparecen la comparación entre los datos que comprenden la altitud y la temperatura ambiente. La altitud es representada mediante la línea continua azul y la temperatura es construida por una línea discontinua de color verde. Desde este entorno gráfico se puede examinar, el comportamiento que contienen una variable respecto a la otra. Lo esperado es encontrar una reacción en la temperatura según varía la altitud, es decir, dependiendo de si los cambios de altitud son notables o no, la temperatura ambiente debería disminuir con el incremento de la altitud y aumentar con el descenso de la altitud.

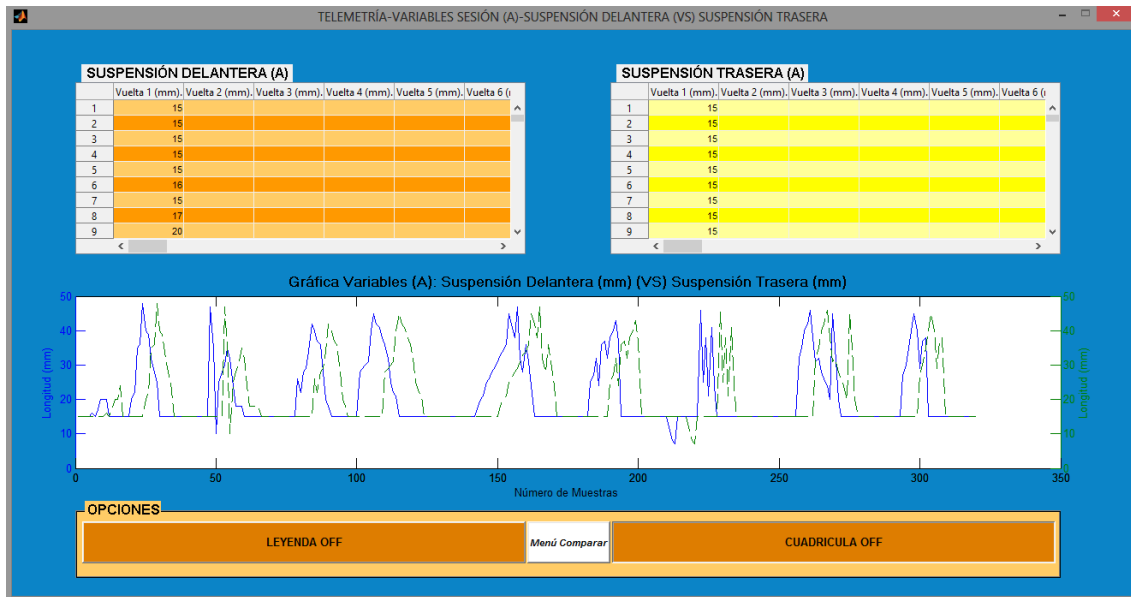


Figura 7.10. Estudio de la relación entre las variables suspensión delantera y suspensión trasera del ensayo Universidad Carlos III de Madrid

En el siguiente espacio visual, se muestra el enfrentamiento entre las dos suspensiones del automóvil. La línea azul continua representa a la suspensión delantera y línea discontinua de color verde a la suspensión trasera. Esta gráfica representa las variaciones de dichas suspensiones al encontrar alguna inestabilidad por el recorrido o al transmitir ciertas maniobra al vehículo. En el caso de una motocicleta de competición se observaría una compresión clara de la suspensión delantera en el momento de frenado. También pueden surgir variaciones leves debido al peso del piloto al moverse.

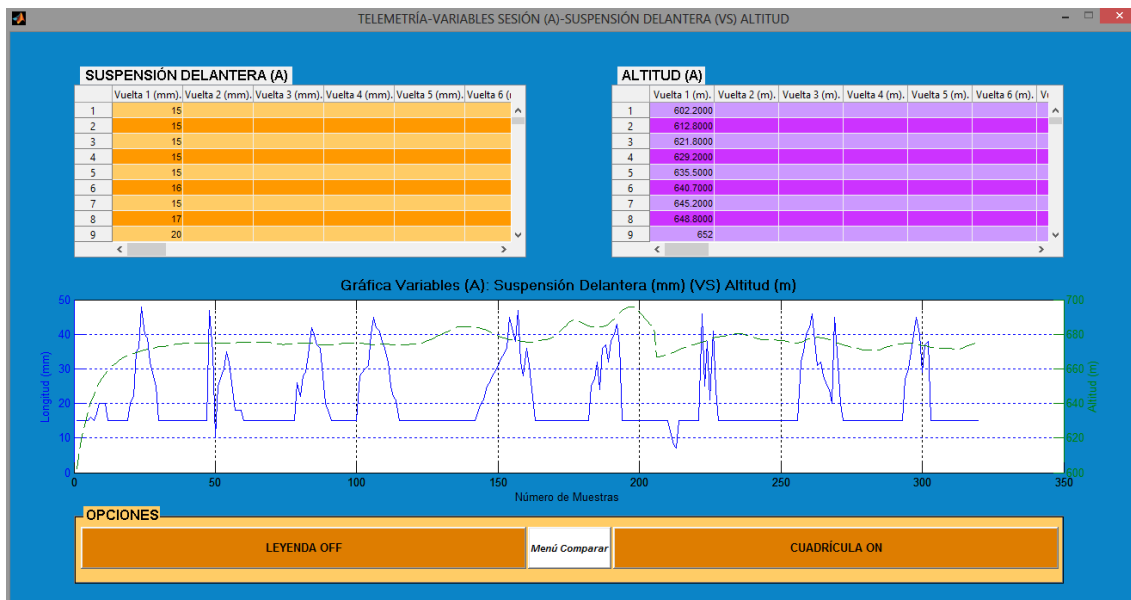


Figura 7.11. Estudio de la relación entre las variables suspensión delantera y altitud del ensayo Universidad Carlos III de Madrid

En la captura de imagen de "la Figura 7.11." se muestra la relación que mantienen la altitud (verde) y el desplazamiento de la suspensión delantera (azul). Dependiendo de cómo sean de pronunciadas las pendientes de ciertas zonas del recorrido, la suspensión delantera tiende a comprimirse y emitir variaciones.

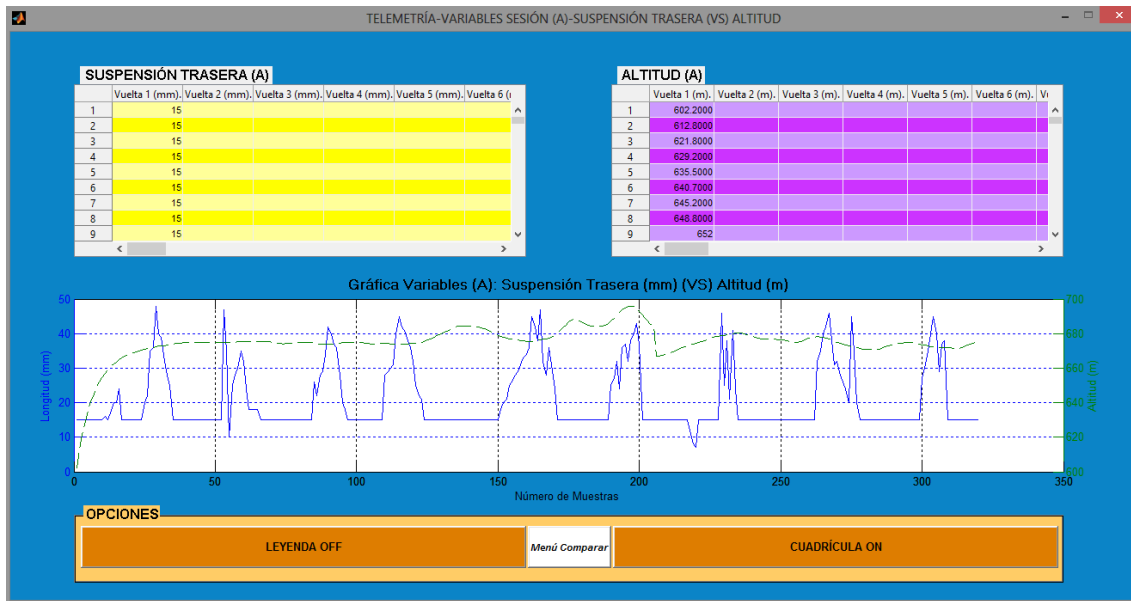


Figura 7.12. Estudio de la relación entre las variables suspensión trasera y altitud del ensayo Universidad Carlos III de Madrid

En el gráfico de la "Figura7.12." se comparan los datos obtenidos por la suspensión trasera (azul) con la altitud (verde) registrada durante el trayecto. Los desplazamientos de la suspensión trasera son producidos cuando en el recorrido se tiene que subir una pendiente, ya que la mayoría del peso tiene que ser soportado por la parte trasera del vehículo.

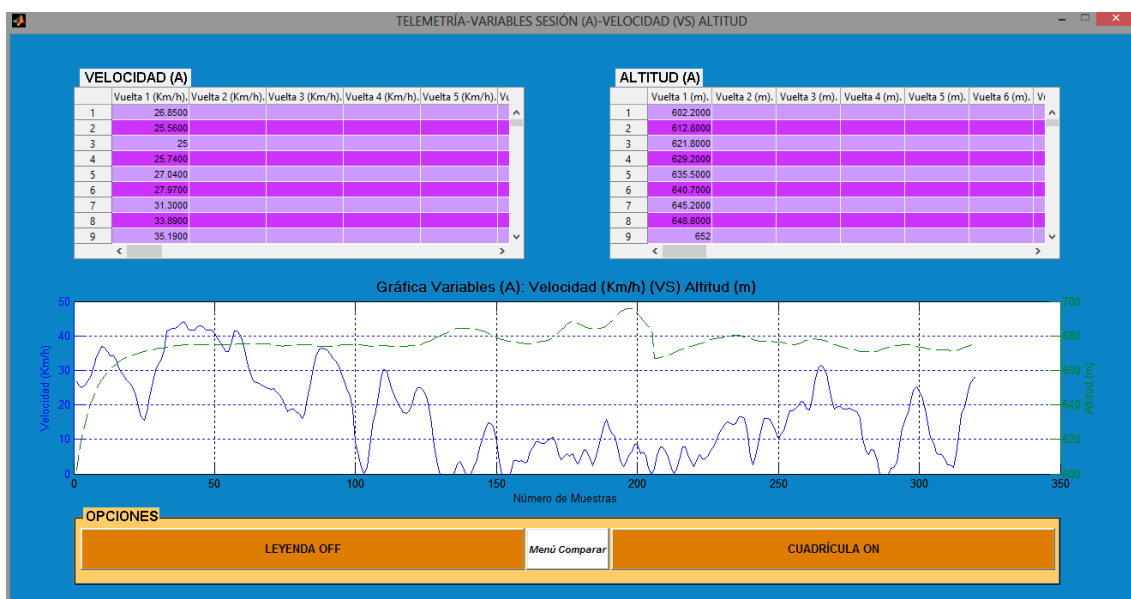


Figura 7.13. Estudio de la relación entre las variables velocidad y altitud del ensayo Universidad Carlos III de Madrid

Para esta gráfica la relación que se examina es la que se produce entre la velocidad (azul) y la altitud (verde). En esta ocasión se analiza que sucede con la velocidad al encontrarse diferentes pendientes durante el recorrido. Si la disminución o el aumento de la altitud es muy pronunciado se debe presenciar alguna variación en la velocidad. En el caso de este ensayo, se destaca que su el recorrido se produce en zona urbana por lo que no se llega a apreciar muy bien este fenómeno. En el caso de realizar un recorrido con una motocicleta en un circuito, se puede experimentar esta relación de un modo más satisfactorio.

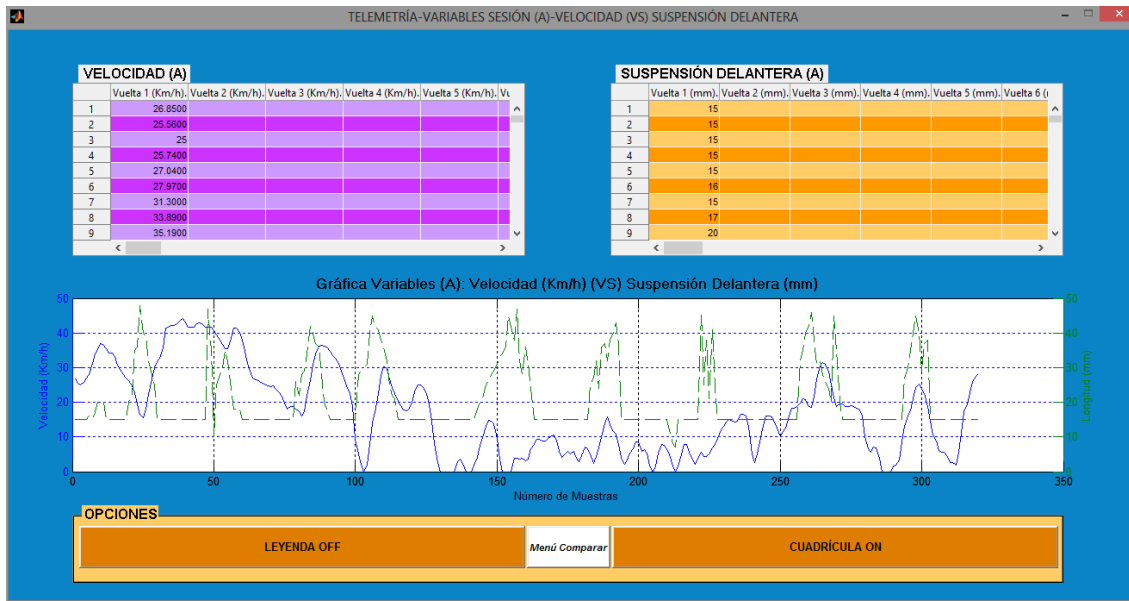


Figura 7.14. Estudio de la relación entre las variables velocidad y suspensión delantera del ensayo Universidad Carlos III de Madrid

La "Figura 7.14." evalúa la relación que establece la suspensión delantera (verde) con la velocidad (azul). En este estudio se manifiesta el comportamiento de la suspensión delantera según la velocidad a la que se realiza el recorrido. Es muy útil para los pasos por curva. En este ensayo se pueden observar las relaciones de frenado y aceleración respecto al desplazamiento de la suspensión. En el caso del uso de una motocicleta rodando en un circuito el fenómeno se puede manifestar de manera más pronunciada.

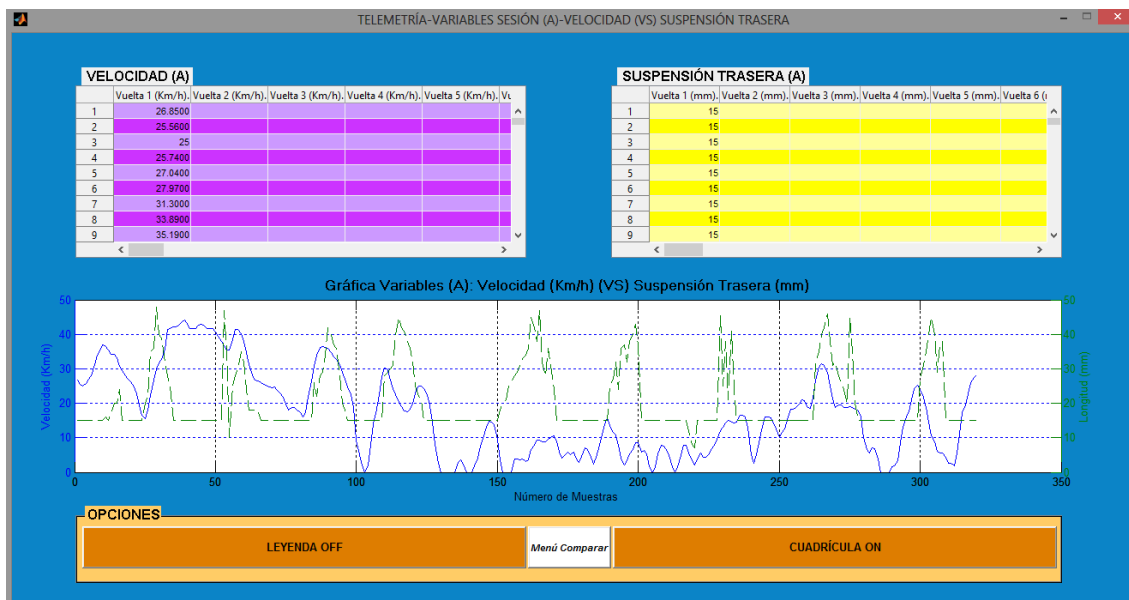


Figura 7.15. Estudio de la relación entre las variables velocidad y suspensión trasera del ensayo Universidad Carlos III de Madrid

En el caso de la relación de la velocidad (azul) y del desplazamiento surgido por la suspensión trasera (verde), atiende al mismo razonamiento expuesto en la "Figura 7.14." del caso anterior.

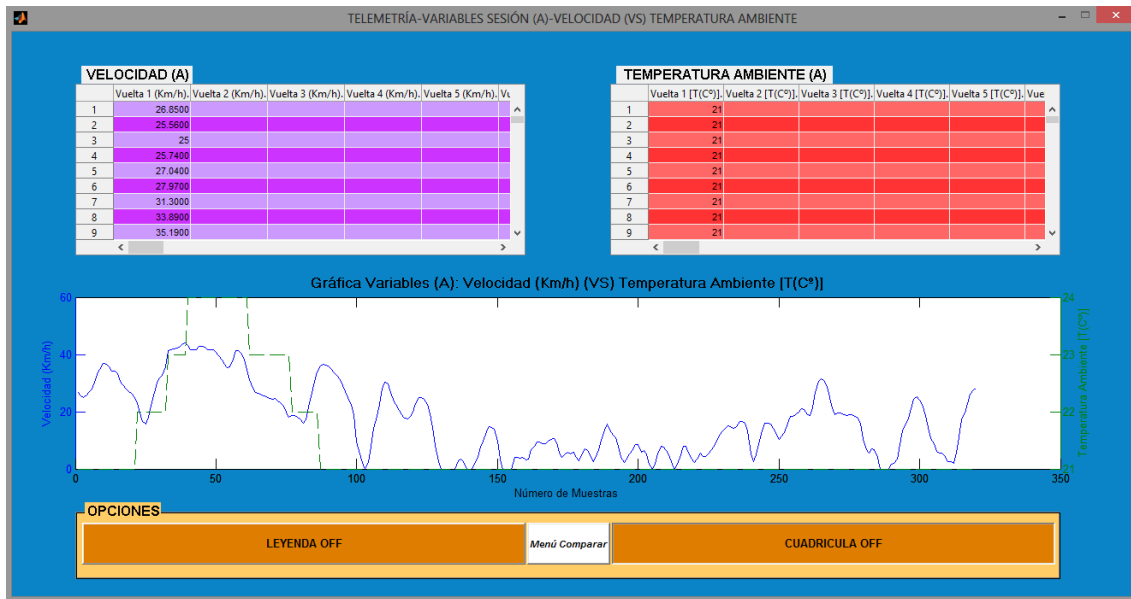


Figura 7.16. Estudio de la relación entre las variables velocidad y temperatura del ensayo Universidad Carlos III de Madrid

En la imagen captada de la "Figura 7.16." se estudia la relación de la temperatura (verde) con la velocidad (azul). Se debe observar como la temperatura que recopila el LM35 disminuye con el aumento de la velocidad y viceversa. Desde el ensayo realizado se aprecia de una forma sutil. Esto es debido a la poca velocidad que se alcanza en un recorrido urbano. Con la motocicleta en un circuito cerrado se observarían de forma notable estos registros.

Acto seguido, se mostrarán los resultados que se obtienen de comparar los sensores de dos sesiones de ensayos y recorridos distintos. De este modo se pueden observar los distintos comportamientos del vehículo en distintos trazados de distintos circuitos. Para ello se compara el ensayo Universidad Carlos III de Madrid con el realizado en las calles de Vallecas. Se recuerda que, también, es posible obtener conclusiones de la comparación entre sensores del mismo recorrido.



Figura 7.17. "MENÚ COMPARAR" ensayo Universidad Carlos III de Madrid con ensayo Vallecas

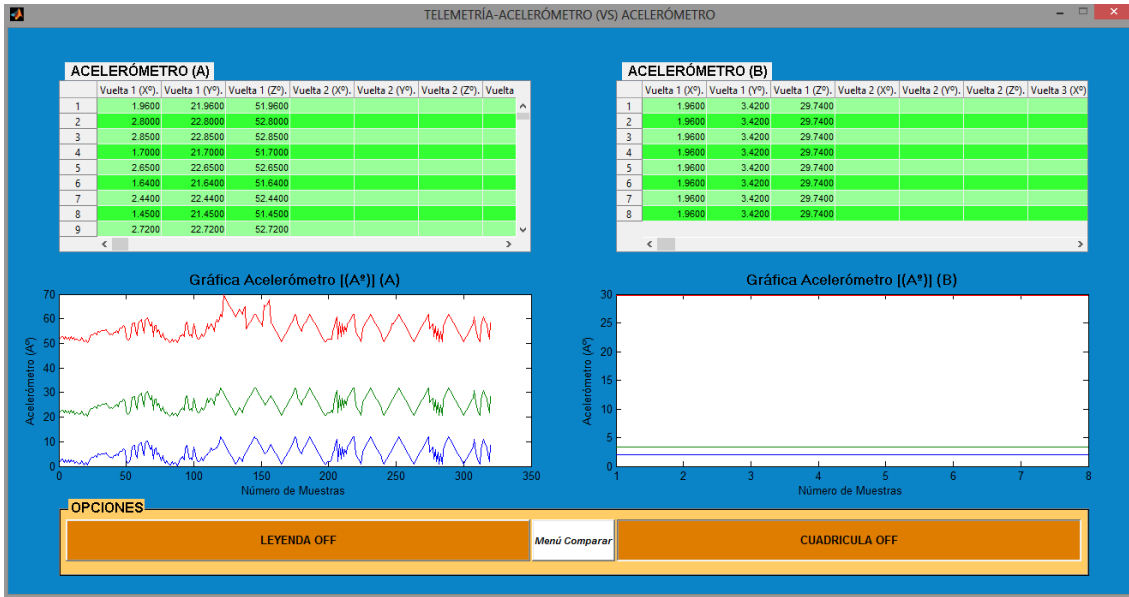


Figura 7.18. Comparación del sensor acelerómetro entre el ensayo Universidad Carlos III de Madrid y el ensayo Vallecas

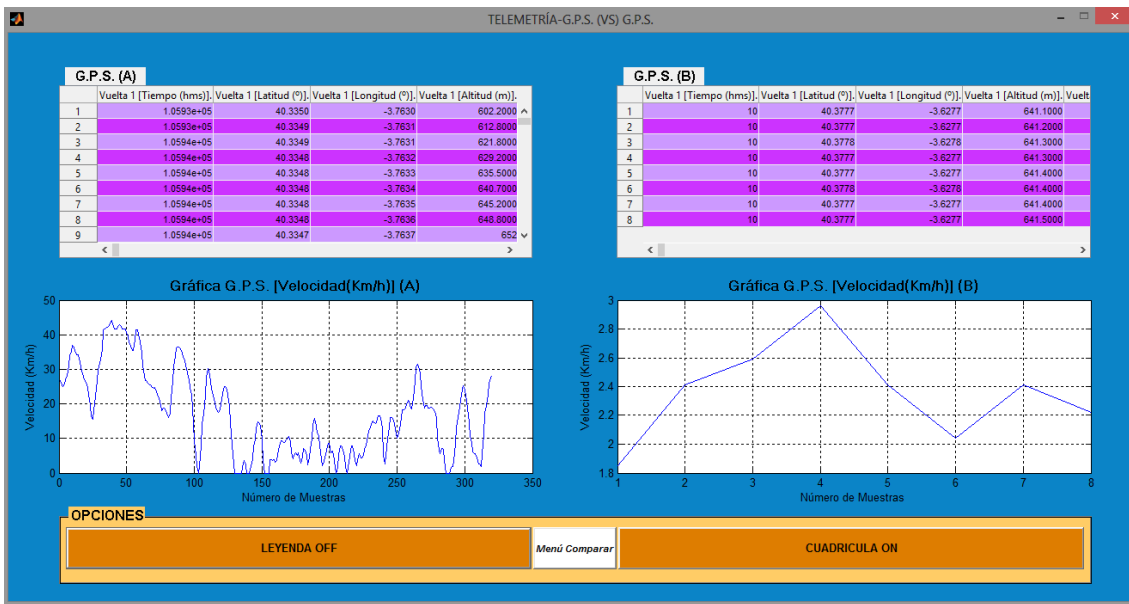


Figura 7.19. Comparación del sensor GPS entre el ensayo Universidad Carlos III de Madrid y el ensayo Vallecas

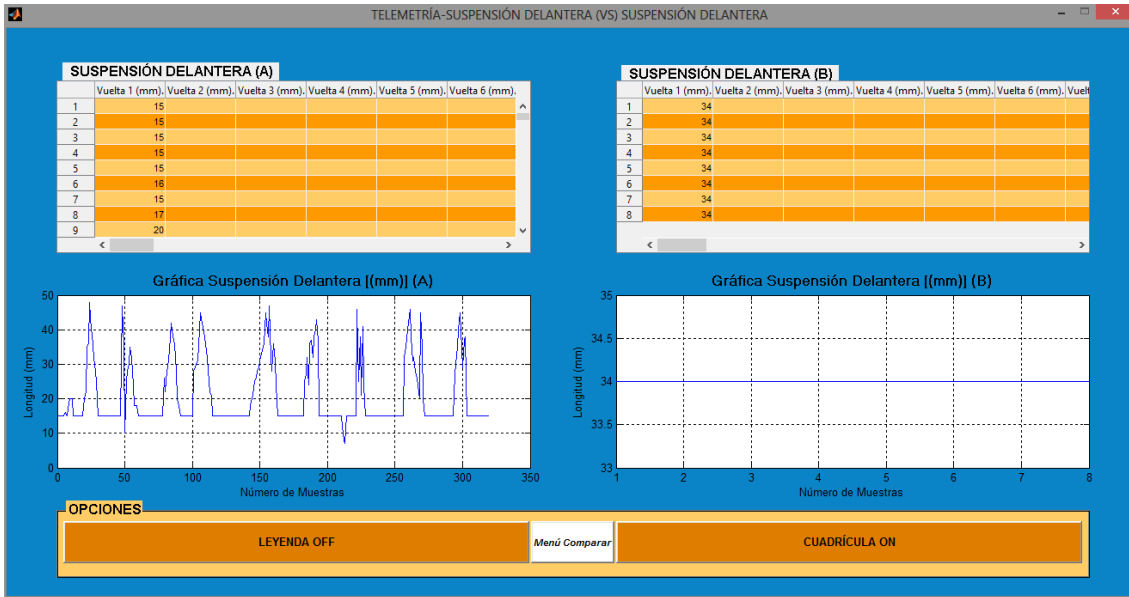


Figura 7.20. Comparación del sensor suspensión delantera entre el ensayo Universidad Carlos III de Madrid y el ensayo Vallecas

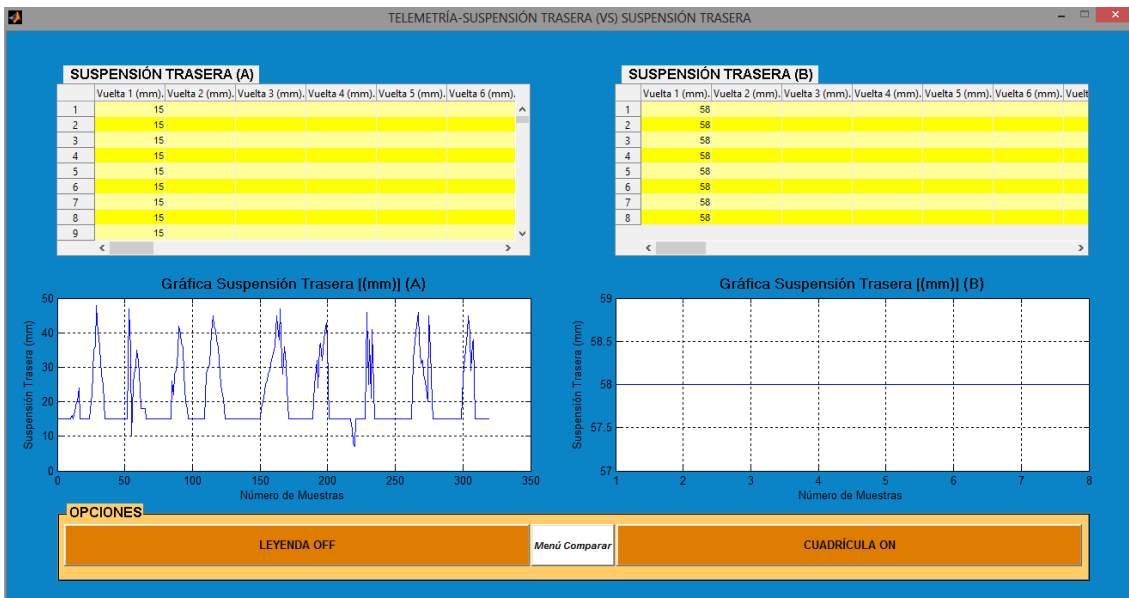


Figura 7.21. Comparación del sensor suspensión trasera entre el ensayo Universidad Carlos III de Madrid y el ensayo Vallecas

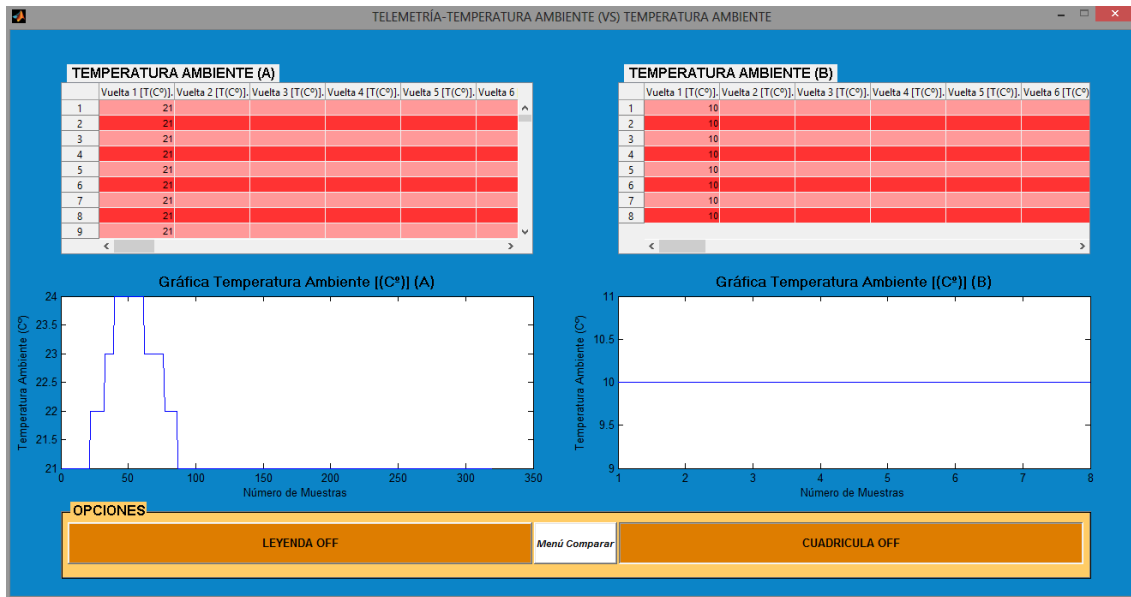


Figura 7.22. Comparación del sensor temperatura entre el ensayo Universidad Carlos III de Madrid y el ensayo Vallecas

Por último, se adjuntan los recorridos y resultados obtenidos a través de Google Earth tanto del ensayo Universidad Carlos III de Madrid como del ensayo Vallecas.



Figura 7.23. Recorrido del ensayo Universidad Carlos III de Madrid



Figura 7.24. Recorrido del ensayo Vallecas



Capítulo 8: CONCLUSIONES Y TRABAJOS FUTUROS

8.1. CONCLUSIONES

Llegados al final del presente trabajo de fin de grado, cabe destacar lo importante e interesante que es para la motivación del alumnado la posibilidad de acceder a realizar proyectos de este tipo. En "Diseño de la telemetría para una motocicleta de competición II" se experimenta la sensación de crear y desarrollar un trabajo útil para el mundo real, es decir, la preparación y los conocimientos, que se van adquiriendo a lo largo de la carrera universitaria, quedan palpables en la consecución de un proyecto que puede servir para mejorar y entender el comportamiento de una motocicleta de competición.

En este documento se puede afirmar que los objetivos propuestos para "Diseño de la telemetría para una motocicleta de competición II", se han cumplido de forma satisfactoria. Se ha explicado, de forma minuciosa, todo el proceso de creación y diseño del sistema de telemetría para la motocicleta del grupo MAQLAB/MotoStudent.

Se realizó un profundo estudio sobre el mundo de la telemetría. Se investigó desde como nacieron sus orígenes hasta la repercusión que ha tenido en nuestros días, logrando plasmar la evolución que ha sufrido la telemetría a lo largo de la historia.

Para la construcción del presente proyecto se seleccionaron los componentes más adecuados. Se utilizaron elementos dentro de las posibilidades económicas que se disponían, pero cuya respuesta fue de gran calidad. Otro punto importante es el desarrollo del sistema de telemetría. Se analizaron diferentes técnicas de comunicación para la adquisición y transferencia de los datos a tratar. Se diseñaron diferentes programas encargados de la lectura, procesamiento y monitorización de todos los datos adquiridos de la motocicleta de competición. Y como fruto de todo ello, se pudo diseñar una interfaz gráfica muy intuitiva capaz de habilitar al usuario el análisis de toda la información adquirida por la motocicleta.

Se consiguió realizar una aplicación de ordenador de todo el programa encargado de este sistema telemétrico, por lo que no sería necesario la utilización de ninguna plataforma software, como MATLAB, para la evaluación de los datos. Por lo que se consigue, aún más, abaratar el coste de este trabajo de fin de grado.

Cabe destacar, también, la consecución de un análisis más completo gracias a la conversión de los datos obtenidos a formato KML y a su posterior muestra por Google Earth, siendo capaces de mostrar y ubicar al usuario el recorrido trazado de un modo óptimo.

Se han adquirido elevados e importantes conocimientos sobre los microcontroladores y todo el entorno relacionado con Arduino. Del mismo modo se ha conseguido lo propio con el entorno de MATLAB, obteniendo un gran nivel de programación en cuanto a procesamiento de datos e interfaces gráficas de usuario.



Con los ensayos realizados, los resultados obtenidos y con la utilización de este sistema, se le dota al usuario de un aprendizaje muy completo sobre el análisis de datos. Se aprende y se sacan conclusiones sobre las aceleraciones, frenadas, maniobras y comportamientos mecánicos de un vehículo.

Mediante la creación de este trabajo de fin de grado, se ha podido obtener el primer sistema telemétrico propio para la Universidad Carlos III de Madrid. Datos telemétricos que sólo podían obtenerse con equipos profesionales de telemetría destinados a la competición, ya son accesibles para un público universitario y para cualquier tipo de público con interés y curiosidad por conocer más sobre este mundo. Debido a ello se consigue que pilotos novatos o amateurs puedan obtener información telemétrica sobre sus propios recorridos.

Sin duda las posibilidades que tiene la aplicación de la telemetría en el motociclismo son muchas, y es posible que con la creación de este proyecto se hayan sentado las bases para la mejora del mismo o para futuros proyectos relacionados con el mismo campo.

8.2. TRABAJOS FUTUROS

Si el presente sistema de telemetría tiene la capacidad de proporcionar un análisis completo sobre el comportamiento de una motocicleta de competición, las posibilidades que aún puede ofrecer son infinitas, pues con el tiempo necesario se pueden ir implementando de una forma paulatina.

En la finalización de este trabajo de fin de grado se han pensado en algunas mejoras y en algunas nuevas funcionalidades que, gracias a la flexibilidad que ofrece Arduino y MATLAB, se podrían llevar a cabo. A continuación se aportan algunas ideas:

- ❖ Explotación del navegador para internet proporcionado por MATLAB. Para este proyecto se llegó a programar una función para poder guardar los datos obtenidos de una sesión en la "nube de internet", es decir, poder almacenar pruebas realizadas en la cuenta de Google Drive que proporciona la Universidad Carlos III de Madrid a cada alumno. De este modo se conseguiría que cualquier persona que necesitara acceder a cualquier sesión guardada, lo pudiese hacer desde cualquier parte del mundo y sin la necesidad de realizarlo desde el mismo ordenador. Debido a la versión proporcionada para este proyecto no ha podido realizarse, ya que el navegador de MATLAB no soporta ciertas páginas o aplicaciones web de los navegadores de internet actuales. Por todo ello, la solución para poder lograr esta mejora sería la adquisición de una versión más actual del software MATLAB.
- ❖ Mejoras en todos los entorno gráficos de la interfaz relacionadas con el aspecto, ubicación de los elementos, introducción de videos y animaciones y todas las modificaciones necesarias para conseguir un espacio visual más intuitivo.
- ❖ Incorporación de una antena emisora y receptora para establecer la transferencia de datos, consiguiendo una comunicación no guiada. Cabe destacar que esta idea sería desde un punto de vista exterior a las normas de competición de MotoStudent, pues para competir con una motocicleta no está permitido la adquisición de datos en tiempo real.



- ❖ Acoplamiento de un módulo Bluetooth para Arduino que habilite el envío de toda la información sobre la motocicleta a dispositivos móviles.
- ❖ Acondicionamiento de un sensor de infrarrojos para el estudio de las revoluciones. De ese modo se podría visualizar mediante la interfaz gráfica de usuario la marcha engranada por la motocicleta.
- ❖ Obtención y visualización por pantalla de la fuerzas G sufridas por la motocicleta de competición mediante los datos generados por el acelerómetro, la distancia recorrida y la velocidad.
- ❖ Creación de un tacógrafo, en el cual se pueda evaluar y visualizar por pantalla toda la actividad generada por el motor de la motocicleta de competición. De este modo se puede determinar el tiempo, en el cual el motor está en marcha, a ralentí o un régimen de parada.
- ❖ Mediante la utilización de un económetro, realizar el estudio del consumo de combustible. Sería de gran utilidad poder visualizar por pantalla tanto la cantidad de combustible que queda, como los puntos del recorrido donde más se consume. Gracias a ello se podría aconsejar al piloto sobre su pilotaje para la administración de dicho combustible.

Las ideas mencionadas ayudarán a conseguir mayores objetivos y a progresar en el campo de la telemetría, pero sin embargo no son las únicas. El desarrollo de ideas es tan amplio que con seguridad, a lo largo del tiempo, aparecerán muchas más funcionalidades, creando un sistema de telemetría mucho más completo.



BIBLIOGRAFÍA

MATERIAL DE CONSULTA

- [1] Lequerica Ribas, Joan. *Arduino Práctico*. Anaya Multimedia, 2013. 400p. ISBN: 978-8441534193.
- [2] Blum, Jeremy. *ARDUINO A FONDO*. Anaya Multimedia, 2014. 352p. ISBN: 9788441536524.
- [3] Vázquez Alcocer, Pere-Pau; Marco Gómez Jordi; Martín Prat, Ángela; Albareda Molinera Xavier. *Programación en C++ para ingenieros*. Editorial Parainfo, 2006. 477p. ISBN: 9788497324854.
- [4] Moore, Holly. *MATLAB para ingenieros*. Edición: Primera-1ra, 2007. Formato: pdf. 628p.
- [5] Sizemore, Jim. *MATLAB for dummies*. Wiley, 2014. 432p. ISBN:978-1118820100.
- [6] Kurniawan, Agus. *Getting Started with Arduino and Matlab*. Lulu.com, 2012.
- [7] Olle, Eduard. *data box LÍNEAS MAESTRAS*. Ediciones IDA2-Impresión digital SL. 106p. ISBN: 978-84-611-6718-0.

ENLACES

- [8] <http://www.arduino.cc/> → Página oficial de Arduino.
- [9] <http://www.mathworks.com/> → Página oficial de Matlab.
- [10] <https://www.sparkfun.com/> → Página oficial Sparkfun.
- [11] http://www.gpsvisualizer.com/map_imput?form=googleearth → Enlace para visualizar los datos obtenidos mediante Google Earth.
- [12] <http://blog.bricogeek.com/> → Comunidad de conocimiento en electrónica.
- [13] <http://www.dailymotos.com/> → Entrevista a Ramón Aurín.
- [14] <http://www.motorpasionmoto.com/> → Blog más visitado en España sobre motociclismo.
- [15] <http://www.elgeniocobas.com/> → Página oficial de Antonio Cobas.
- [16] <http://www.seeedstudio.com> → Productos electrónicos.
- [17] <http://www.bottpower.com> → Tecnología electrónica.
- [18] <http://www.magnetimarelli.com> → Elementos y productos telemétricos.
- [19] <http://www.motogp.com> → Página oficial del mundial de motociclismo.



ANEXOS

ANEXO 1: MICROPROCESADOR ATMEL8U2

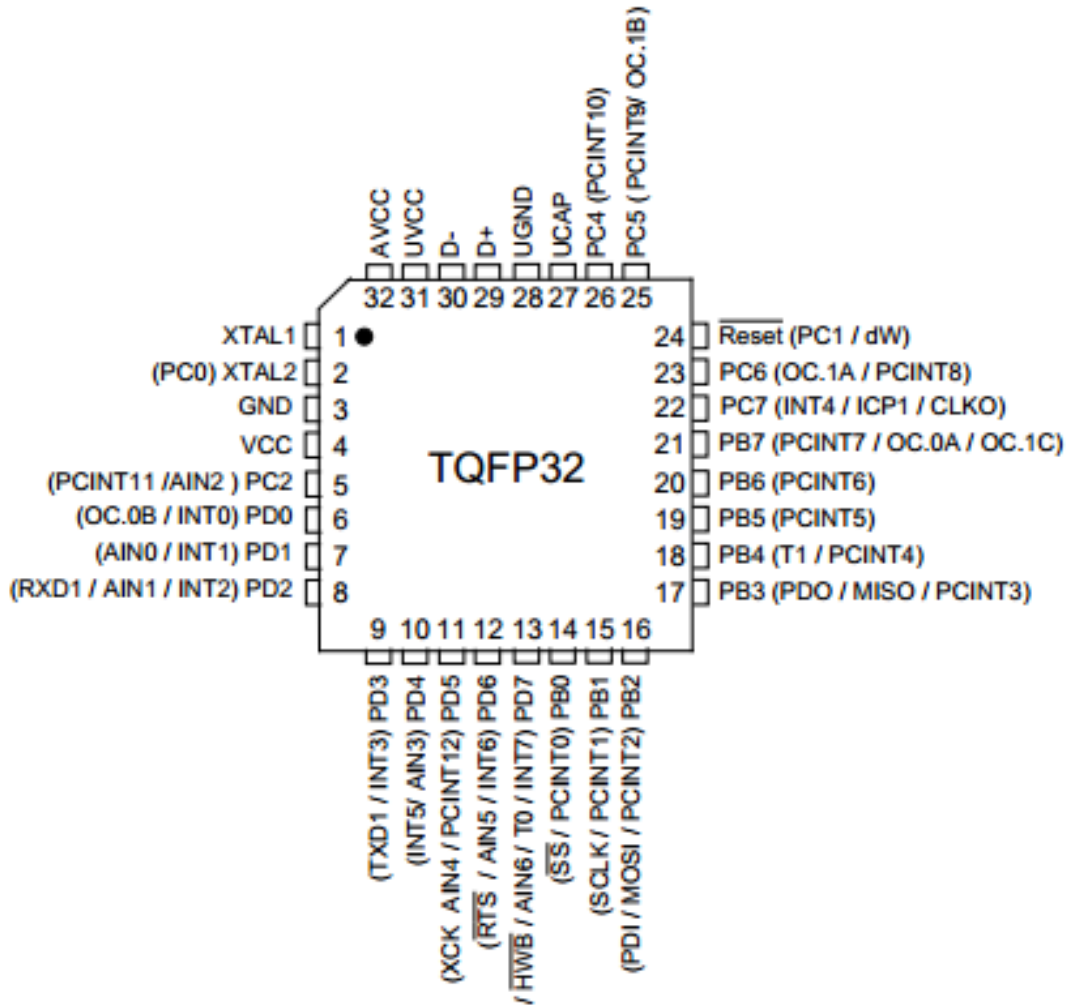


Figura 1. Anexo 1. Esquemático Microprocesador Atmel8U2



ANEXO 2: MICROPROCESADOR ATMEGA2560

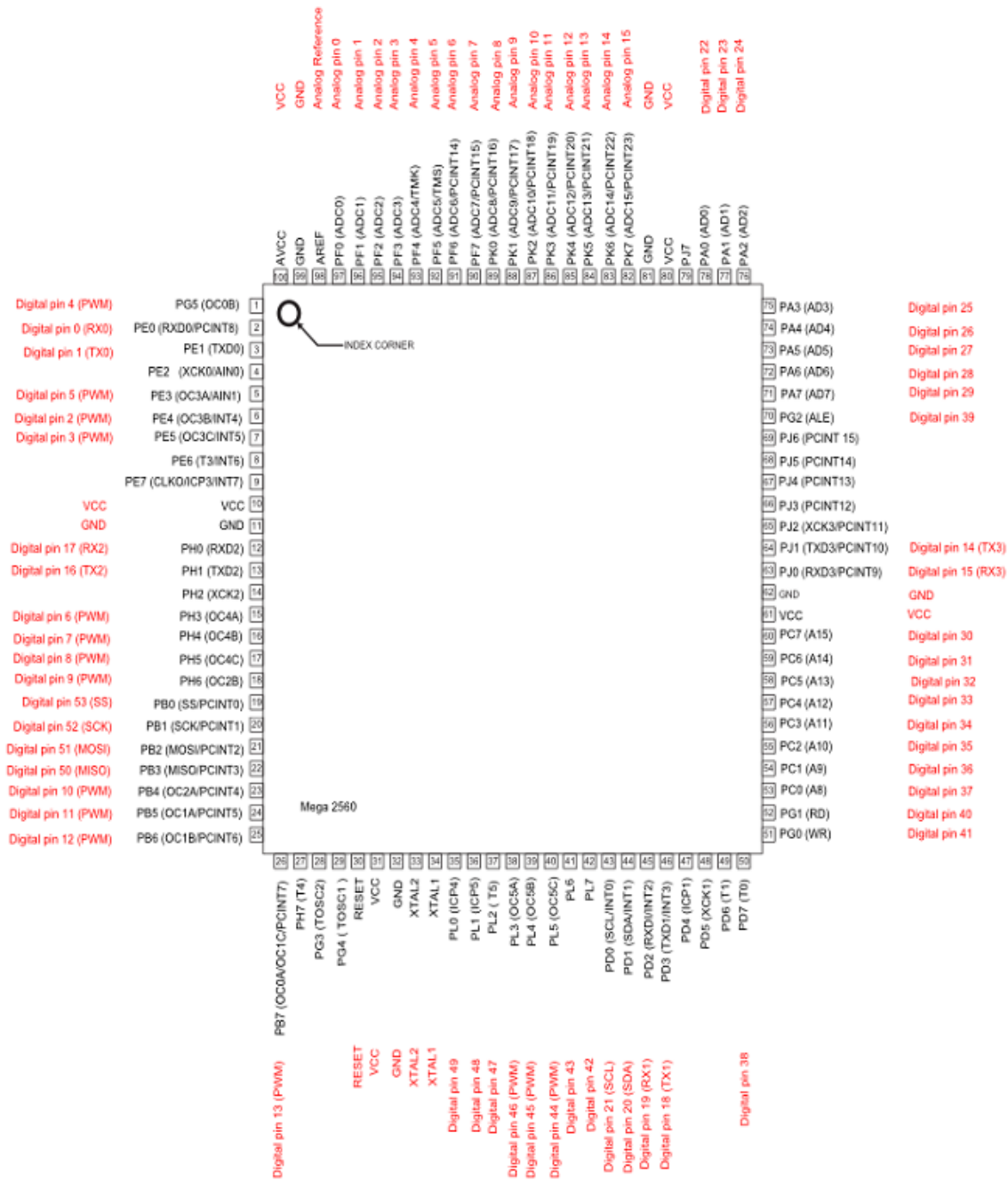


Figura 1. Anexo 2. Esquemático microprocesador ATMEGA2560



Arduino Mega 2560 PIN mapping table

Pin Number	Pin Name	Mapped Pin Name
1	PG5 (OC0B)	Digital pin 4 (PWM)
2	PE0 (RXD0/PCINT8)	Digital pin 0 (RX0)
3	PE1 (TXD0)	Digital pin 1 (TX0)
4	PE2 (XCK0/AIN0)	
5	PE3 (OC3A/AIN1)	Digital pin 5 (PWM)
6	PE4 (OC3B/INT4)	Digital pin 2 (PWM)
7	PE5 (OC3C/INT5)	Digital pin 3 (PWM)
8	PE6 (T3/INT6)	
9	PE7 (CLK0/ICP3/INT7)	
10	VCC	VCC
11	GND	GND
12	PH0 (RXD2)	Digital pin 17 (RX2)
13	PH1 (TXD2)	Digital pin 16 (TX2)
14	PH2 (XCK2)	
15	PH3 (OC4A)	Digital pin 6 (PWM)
16	PH4 (OC4B)	Digital pin 7 (PWM)
17	PH5 (OC4C)	Digital pin 8 (PWM)
18	PH6 (OC2B)	Digital pin 9 (PWM)
19	PB0 (SS/PCINT0)	Digital pin 53 (SS)
20	PB1 (SCK/PCINT1)	Digital pin 52 (SCK)
21	PB2 (MOSI/PCINT2)	Digital pin 51 (MOSI)
22	PB3 (MISO/PCINT3)	Digital pin 50 (MISO)
23	PB4 (OC2A/PCINT4)	Digital pin 10 (PWM)
24	PB5 (OC1A/PCINT5)	Digital pin 11 (PWM)
25	PB6 (OC1B/PCINT6)	Digital pin 12 (PWM)
26	PB7 (OC0A/OC1C/PCINT7)	Digital pin 13 (PWM)
27	PH7 (T4)	
28	PG3 (TOSC2)	
29	PG4 (TOSC1)	



30	RESET	RESET
31	VCC	VCC
32	GND	GND
33	XTAL2	XTAL2
34	XTAL1	XTAL1
35	PL0 (ICP4)	Digital pin 49
36	PL1 (ICP5)	Digital pin 48
37	PL2 (T5)	Digital pin 47
38	PL3 (OC5A)	Digital pin 46 (PWM)
39	PL4 (OC5B)	Digital pin 45 (PWM)
40	PL5 (OC5C)	Digital pin 44 (PWM)
41	PL6	Digital pin 43
42	PL7	Digital pin 42
43	PD0 (SCL/INT0)	Digital pin 21 (SCL)
44	PD1 (SDA/INT1)	Digital pin 20 (SDA)
45	PD2 (RXDI/INT2)	Digital pin 19 (RX1)
46	PD3 (TXD1/INT3)	Digital pin 18 (TX1)
47	PD4 (ICP1)	
48	PD5 (XCK1)	
49	PD6 (T1)	
50	PD7 (T0)	Digital pin 38
51	PG0 (WR)	Digital pin 41
52	PG1 (RD)	Digital pin 40
53	PC0 (A8)	Digital pin 37
54	PC1 (A9)	Digital pin 36
55	PC2 (A10)	Digital pin 35
56	PC3 (A11)	Digital pin 34
57	PC4 (A12)	Digital pin 33
58	PC5 (A13)	Digital pin 32
59	PC6 (A14)	Digital pin 31
60	PC7 (A15)	Digital pin 30
61	VCC	VCC
62	GND	GND
63	PJ0 (RXD3/PCINT9)	Digital pin 15 (RX3)
64	PJ1 (TXD3/PCINT10)	Digital pin 14 (TX3)



65	PJ2 (XCK3/PCINT11)	
66	PJ3 (PCINT12)	
67	PJ4 (PCINT13)	
68	PJ5 (PCINT14)	
69	PJ6 (PCINT 15)	
70	PG2 (ALE)	Digital pin 39
71	PA7 (AD7)	Digital pin 29
72	PA6 (AD6)	Digital pin 28
73	PA5 (AD5)	Digital pin 27
74	PA4 (AD4)	Digital pin 26
75	PA3 (AD3)	Digital pin 25
76	PA2 (AD2)	Digital pin 24
77	PA1 (AD1)	Digital pin 23
78	PA0 (AD0)	Digital pin 22
79	PJ7	
80	VCC	VCC
81	GND	GND
82	PK7 (ADC15/PCINT23)	Analog pin 15
83	PK6 (ADC14/PCINT22)	Analog pin 14
84	PK5 (ADC13/PCINT21)	Analog pin 13
85	PK4 (ADC12/PCINT20)	Analog pin 12
86	PK3 (ADC11/PCINT19)	Analog pin 11
87	PK2 (ADC10/PCINT18)	Analog pin 10
88	PK1 (ADC9/PCINT17)	Analog pin 9
89	PK0 (ADC8/PCINT16)	Analog pin 8
90	PF7 (ADC7)	Analog pin 7
91	PF6 (ADC6)	Analog pin 6
92	PF5 (ADC5/TMS)	Analog pin 5
93	PF4 (ADC4/TMK)	Analog pin 4
94	PF3 (ADC3)	Analog pin 3
95	PF2 (ADC2)	Analog pin 2
96	PF1 (ADC1)	Analog pin 1
97	PF0 (ADC0)	Analog pin 0
98	AREF	Analog Reference
99	GND	GND
100	AVCC	VCC



ANEXO 3: CÓDIGO ARDUINO

LECTURA TARJETA SD

```
// Leer datos de la tarjeta SD con Arduino

//Incluimos la librería SD

#include <SD.h>

// Guardamos en que entrada de arduino está conectado el pin CS del modulo

const int chipSelect = 53;

void setup(){

  // Configuramos el puerto serie para informar de fallos a través de él

  Serial.begin(9600);

  // El pin CS por defecto de la placa arduino debe ser configurado como salida

  // aunque no se use (10 en la mayoría de las placas, 53 en Arduino Mega).

  pinMode(53, OUTPUT);

  // Si ha habido error al leer la tarjeta informamos por el puerto serie.

  if (!SD.begin(chipSelect)){

    Serial.println("Error al leer la tarjeta.");

    return;

  }

  //Escribimos el programa dentro del setup para que solo se ejecute una vez.

  // Abrimos el archivo.

  File dataFile = SD.open("datalog.txt");

  // Si lo hemos podido abrir correctamente:

  if (dataFile) {

    // Mostramos un aviso de comienzo del txt

    // Mandamos sus datos por el puerto serie
```



```
while (dataFile.available()) {  
    Serial.write(dataFile.read());  
}  
  
// Cerramos el archivo  
dataFile.close();  
  
// Si no hemos conseguido abrir el archivo mandamos un error  
}else {  
    Serial.println("Error al abrir leeme.txt");  
}  
}  
  
//No escribimos nada dentro del loop  
void loop()  
{  
}
```



ANEXO 4: CÓDIGO MATLAB

FUNCIÓN DATOS SESIÓN

```
function s=datosSesion%(numero_muestras,vueltas)
%DATOSSESION Summary of this function goes here
% Detailed explanation goes here

clc;

%VARIABLES GLOBALES
clear global S; clear global ST; clear global STB;
clear global LAP; clear global LOP;
clear global VS; clear global VST;
clear global ALP; clear global ALST;
clear global TAS; clear global TAST;
clear global AS; clear global AST;
clear global SUSPDS; clear global SUSPDST;
clear global SUSPTS; clear global SUSPTST;
clear global XS; clear global YS; clear global ZS;
clear global C;
clear global lamin; clear global lamax;
clear global lomin; clear global lomax;
clear global almin;
clear global almax;
clear global mmuestras;

global S; global ST; global STB;
global GPSST;
global LAP; global LOP;
global ALP; global ALST;
global VS; global VST;
global TAS; global TAST;
global AS; global AST;
global SUSPDS; global SUSPDST;
global SUSPTS; global SUSPTST;
global XS; global YS; global ZS;
global C;
global latmeta; global lonmeta;
global numero_paquetesd;
global lamin; global lamax;
global lomin; global lomax;
global almin; global almax;
global mmuestras;
global salto;

%VARIABLES INTERNAS
int contmuestras;
contmuestras=0;
contvueltas=0;
salto=0;
double tas; double tap; double tast;
double xst; double yst; double zst;
double xp; double yp; double zp;
double xs; double ys; double zs;
double as; double ast;
double tst; double last; double lost; double vst; double alst;
double tp; double lap; double lop; double vp; double alp;
double ts; double las; double los; double vs; double als;
double suspdst; double susptst;
```



```

double suspdp; double susptp;
double suspds; double suspts;
c=1; a=1; at=1; b=1; bt=1; g=1; gt=1; fi=1;

%INICIALIZACIÓN DE EL PUERTO SERIAL QUE UTILIZAREMOS
delete(instrfind({'Port'}, {'COM5'}));
puerto_serial=serial('COM5');
puerto_serial.BaudRate = 9600;
warning('off', 'MATLAB:serial:fscanf:unsuccessfulRead');

%APERTURA DEL PUERTO SERIAL
fopen(puerto_serial);

%BUCLE ENCARGADO DE CALCULAR EL NÚMERO DE VUELTAS TOTALES DEL CIRCUITO
%Y DEL NÚMERO DE DATOS TOTALES DE LOS SENSORES POR CADA VUELTA
for f=1:numero_paquetesd

    %OBTENCIÓN DE MUESTRAS DE DATOS
    puerto = fscanf(puerto_serial, '%f');

    %ASOCIACIÓN DE VARIABLES CON LOS DATOS OBTENIDOS POR EL PUERTO
    tap(f,1)=puerto(1);
    xp(f,1)=puerto(2);
    yp(f,1)=puerto(3);
    zp(f,1)=puerto(4);
    tp(f,1)=puerto(5);
    lap(f,1)=puerto(6);
    lop(f,1)=puerto(7);
    vp(f,1)=puerto(8);
    alp(f,1)=puerto(9);
    suspdp(f,1)=puerto(10);
    susptp(f,1)=puerto(11);

    %CÁLCULO DE VUELTAS Y DE MUESTRAS DE LOS SENSORES POR CADA VUELTA
    latitud=puerto(6);
    longitud=puerto(7);

    if(latitud==latmeta && longitud==lonmeta)
        contvueltas = contvueltas+1;
        salto(f,1)=f;
    end

    k=puerto(1);

    if(k~=' ')
        contmuestras = contmuestras+1;
        if(latitud==latmeta && longitud==lonmeta)
            mmuestras(c,1) = contmuestras;
            c=c+1;
            contmuestras=0;
        end
    end
end

if(contmuestras~=0)
    mmuestras(c,1) = contmuestras;
end

```




```
if(contvueltas==0)
    contvueltas=1;
    mmuestras(c,1)=contmuestras;
end

C=c;

%BUCLE ENCARGADO DE ORDENAR, POR VUELTAS, TODOS LOS DATOS DE LA SESIÓN
for col=1:c
    for fil=1:mmuestras(col,1)

        %MATRICES DE DATOS DE CADA SENSOR
        tas(fil,col)=tap(fi,1);
        xs(fil,col)=xp(fi,1);
        ys(fil,col)=yp(fi,1);
        zs(fil,col)=zp(fi,1);
        ts(fil,col)=tp(fi,1);
        las(fil,col)=lap(fi,1);
        los(fil,col)=lop(fi,1);
        vs(fil,col)=vp(fi,1);
        als(fil,col)=alp(fi,1);
        suspds(fil,col)=suspdp(fi,1);
        suspts(fil,col)=susptp(fi,1);
        fi=fi+1;

        %MATRIZ DE DATOS (ORDENADOS) DE LA SESIÓN
        s(fil,a)=ts(fil,col);
        s(fil,a+1)=las(fil,col);
        s(fil,a+2)=los(fil,col);
        s(fil,a+3)=als(fil,col);
        s(fil,a+4)=vs(fil,col);
        s(fil,a+5)=tas(fil,col);
        s(fil,a+6)=suspds(fil,col);
        s(fil,a+7)=suspts(fil,col);
        s(fil,a+8)=xs(fil,col);
        s(fil,a+9)=ys(fil,col);
        s(fil,a+10)=zs(fil,col);

        %MATRIZ DE DATOS (ORDENANDOS) DEL ACELERÓMETRO DE LA SESIÓN
        as(fil,b)=xs(fil,col);
        as(fil,b+1)=ys(fil,col);
        as(fil,b+2)=zs(fil,col);

        %MATRIZ DE DATOS (ORDENADOS) DEL G.P.S. DE LA SESIÓN
        gpss(fil,g)=ts(fil,col);
        gpss(fil,g+1)=las(fil,col);
        gpss(fil,g+2)=tp(fi,1);
        gpss(fil,g+3)=ts(fil,col);
        gpss(fil,g+4)=ts(fil,col);
    end

    a=a+11;
    b=b+3;
    g=g+5;
end
```



```

%ASOCIACIÓN DE LAS MATRICES DE DATOS DE LOS SENSORES A NUEVAS
%VARIABLES PARA LA CONSTRUCCIÓN DE LA MATRIZ FINAL DE DATOS DE LA
%SESIÓN
tast=tas;
xst=xs;
yst=ys;
zst=zs;
tst=ts;
last=las;
lost=los;
vst=vs;
alst=als;
suspdst=suspbs;
susptst=suspts;

%CONSECUCIÓN DE LAS MATRICES DE DATOS DE LOS SENSORES QUE INDICAN EL
%FINAL DE LA RECOGIDA DE DATOS
tam1=size(tast);
tamc=tam1(:,2);
tamf=tam1(:,1);

for k=1:tamc
    for ff=1:tamf
        if ((tast(ff,k))==0&&(xst(ff,k))==0&&(yst(ff,k))==0&&...
            (zst(ff,k))==0&&(tst(ff,k))==0&&(last(ff,k))==0&&...
            (lost(ff,k))==0&&(vst(ff,k))==0&&(alst(ff,k))==0&&...
            (suspdst(ff,k))==0&&(susptst(ff,k))==0)
            g=1;
            tast(ff,k)=1/0;
            xst(ff,k)=1/0;
            yst(ff,k)=1/0;
            zst(ff,k)=1/0;
            tst(ff,k)=1/0;
            last(ff,k)=1/0;
            lost(ff,k)=1/0;
            vst(ff,k)=1/0;
            alst(ff,k)=1/0;
            suspdst(ff,k)=1/0;
            susptst(ff,k)=1/0;
        else
            g=0;
        end
    end
end

%BUCLE QUE CONSIGUE LA MATRIZ FINAL DE DATOS QUE SE MUESTRA EN LA
%SESIÓN
for col=1:tamc
    for fil=1:tamf

        %MATRIZ FINAL DE DATOS (TABLA SESIÓN)
        st(fil,at)=tst(fil,col);
        st(fil,at+1)=last(fil,col);
        st(fil,at+2)=lost(fil,col);
        st(fil,at+3)=alst(fil,col);
        st(fil,at+4)=vst(fil,col);
        st(fil,at+5)=tast(fil,col);
        st(fil,at+6)=suspdst(fil,col);
        st(fil,at+7)=susptst(fil,col);
        st(fil,at+8)=xst(fil,col);
        st(fil,at+9)=yst(fil,col);
    end
end

```



```

st(fil,at+10)=zst(fil,col);

%MATRIZ FINAL DE DATOS DEL ACELERÓMETRO.
ast(fil,bt)=xst(fil,col);
ast(fil,bt+1)=yst(fil,col);
ast(fil,bt+2)=zst(fil,col);

%MATRIZ FINAL DE DATOS DEL G.P.S.
gpsst(fil,gt)=tst(fil,col);
gpsst(fil,gt+1)=last(fil,col);
gpsst(fil,gt+2)=lost(fil,col);
gpsst(fil,gt+3)=alst(fil,col);
gpsst(fil,gt+4)=vst(fil,col);
end

at=at+11;
bt=bt+3;
gt=gt+5;
end

%LÍMITES MAXIMOS Y MINIMOS PARA LA GRÁFICA DEL CIRCUITO (SIN NECESIDAD
%DE INTRODUCIR ESTOS VALORES POR TECLADO)
lamax=max(max(lap));
lamin=min(min(lap));
lomax=max(max(lop));
lomin=min(min(lop));
almax=max(max(alp));
almin=min(min(alp));
lamax=lamax+0.0005;
lamin=lamin-0.0022;
lomax=lomax+0.0015;
lomin=lomin-0.0024;
almax=almax+108;
almin=almin-59;

%ASOCIACIÓN DE MATRICES A VARIABLES
S=s; ST=st; STB=st;
LAP=lap; LOP=lop;
VS=vs; VST=vst;
ALP=alp; ALST=alst;
TAS=tas; TAST=tast;
AS=as; AST=ast;
GPSST=gpsst;
SUSPDS=suspds; SUSPDST=suspdst;
SUSPTS=suspts; SUSPTST=susptst;
XS=xs; YS=ys; ZS=zs;

%CONTROL DE VARIABLES EN EL WORKSPACE
assignin('base','LAP',LAP);
assignin('base','LOP',LOP);
assignin('base','VS',VS);
assignin('base','ALP',ALP);
assignin('base','TAS',TAS);
assignin('base','AS',AS);
assignin('base','AST',AST);
assignin('base','ALST',ALST);
assignin('base','TAST',TAST);
assignin('base','XS',XS);
assignin('base','YS',YS);
assignin('base','ZS',ZS);

```



```
%CORTAMOS LA CONEXIÓN CON EL PUERTO SERIAL SIN ELIMINAR LOS DATOS
%OBTENIDOS
fclose(puerto_serial);
delete(puerto_serial);

end
```



FUNCIÓN APERTURA DE SESIÓN

```
function o= abrirS
%ABRIRS Summary of this function goes here
% Detailed explanation goes here

o=1;

%VARIABLES GLOBALES
clear LAOPEN; clear LOOPEN; clear ALOPEN; clear ALOPENI; clear TAOPEN;
clear AOPEN; clear SUSPDOPEN; clear SUSPTOPEN; clear ST; clear VOPEN;
clear GPSOPEN;

global TAOPEN; global AOPEN; global GPSOPEN; global VOPEN;
global SUSPDOPEN; global SUSPTOPEN; global lamin; global lamax;
global lomin; global lomax; global ST

%VARIABLES INTERNAS
tsc=1; lac=2; loc=3; alc=4; vc=5; tac=6; suspdc=7; susptc=8;
xc=9; yc=10; zc=11;
a=1; fi=1; g=1;

%APERTURA DE SESIÓN
uiopen('LOAD');

%OBTENCIÓN DE EL NÚMERO DE VUELTAS (COLUMNAS) DE CUALQUIER SESIÓN
%GUARDADA SIN NECESIDAD DE TENER QUE CARGAR UNA SESIÓN
size(ST);
tam1=size(ST);
tamc=tam1(:,2);
c=tamc/11;

    %BUCLE PARA OBTENER LOS DATOS DE CUALQUIER SESIÓN
if(c==1)
    taopeni=ST(:,tac);
    xopeni=ST(:,xc);
    yopeni=ST(:,yc);
    zopeni=ST(:,zc);
    aopeni=cat(2,xopeni,yopeni,zopeni);
    topeni=ST(:,tsc);
    laopen=ST(:,lac);
    loopen=ST(:,loc);
    alopen=ST(:,alc);
    vopeni=ST(:,vc);
    gpsopeni=cat(2,taopeni,laopen,loopen,alopen,vopeni);
    suspdopeni=ST(:,suspdc);
    susptopeni=ST(:,susptc);

elseif(c>1)
    for i=1:c
        tam2=size(ST(:,tac));
        tamf=(tam2(:,1));
        for f=1:tamf

            %MATRIZ SESIÓN
            taopeni(f,i)=ST(f,tac);
            topeni(f,i)=ST(f,tsc);
            laopeni(fi,1)=ST(f,lac); laopenii(f,i)=ST(f,lac);
            loopeni(fi,1)=ST(f,loc); loopenii(f,i)=ST(f,loc);
            alopeni(fi,1)=ST(f,alc); alopenii(f,i)=ST(f,alc);
            vopeni(f,i)=ST(f,vc);
            suspdopeni(f,i)=ST(f,suspdc);
            susptopeni(f,i)=ST(f,susptc);
```



```

xopeni (f, i) =ST (f, xc) ;
yopeni (f, i) =ST (f, yc) ;
zopeni (f, i) =ST (f, zc) ;

%MATRIZ ACELERÓMETRO
aopeni (f, a) =xopeni (f, i) ;
aopeni (f, a+1) =yopeni (f, i) ;
aopeni (f, a+2) =zopeni (f, i) ;

%MATRIZ G.P.S.
gpsopeni (f, g) =topeni (f, i) ;
gpsopeni (f, g+1) =laopenii (f, i) ;
gpsopeni (f, g+2) =loopenii (f, i) ;
gpsopeni (f, g+3) =alopenii (f, i) ;
gpsopeni (f, g+4) =vopeni (f, i) ;

if ((laopeni (fi, 1) ~= (1/0) && (loopeni (fi, 1) ~= (1/0) ...
    && (alopeni (f, 1) ~= (1/0) )
        laopen (fi, 1) =laopeni (fi, 1) ;
        loopen (fi, 1) =loopeni (fi, 1) ;
        alopen (fi, 1) =alopeni (fi, 1) ;
    end
    fi=fi+1;

end
lac=lac+11;
loc=loc+11;
alc=alc+11;
tac=tac+11;
vc=vc+11;
suspdc=suspdc+11;
susptc=susptc+11;
xc=xc+11;
yc=yc+11;
zc=zc+11;
a=a+3;
g=g+5;
end
end

%ASOCIACIÓN DE MATRICES A VARIABLES
global LAOPEN;
global LOOPEN;
global ALOPEN;
global ALOPENI;
LAOPEN=laopen;
LOOPEN=loopen;
ALOPEN=alopen;

if (c>1)
    ALOPENI=alopenii;
else
    ALOPENI=alopen;
end

GPSOPEN=gpsopeni;
VOPEN=vopeni;
TAOPEN=taopeni;
SUSPDOPEN=suspdopeni;
SUSPTOPEN=susptopeni;
AOPEN=aopeni;

```




```
%CONTROL DE VARIABLES EN EL WORKSPACE
assignin('base','TAOPEN',TAOPEN);
assignin('base','AOPEN',AOPEN);
assignin('base','LAOPEN',LAOPEN);
assignin('base','LOOPEN',LOOPEN);
assignin('base','ALOPEN',ALOPEN);
assignin('base','VOPEN',VOPEN);
assignin('base','SUSPDOPEN',SUSPDOPEN);
assignin('base','SUSPTOPEN',SUSPTOPEN);

%CONSECUCIÓN DE LOS LÍMITES DE CUALQUIER CIRCUITO DE CUALQUIER SESIÓN
%GUARDADA SIN NECESIDAD DE TENER QUE CARGAR UNA SESIÓN
lamax=max(max(LAOPEN));
lamin=min(min(LAOPEN));
lomax=max(max(LOOPEN));
lomin=min(min(LOOPEN));
almax=max(max(ALOPEN));
almin=min(min(ALOPEN));
lamax=lamax+0.0005;
lamin=lamin-0.0022;
lomax=lomax+0.0015;
lomin=lomin-0.0024;
almax=almax+108;
almin=almin-59;

end
```



FUNCIÓN APERTURA DE SESIÓN B

```

function ob = abrirSB
%ABRIRSB Summary of this function goes here
% Detailed explanation goes here

ob=1;

%VARIABLES GLOBALES
clear AOPENIB; clear TAOPENB; clear AOPENB; clear SUSPDOPENB;
clear SUSPTOPENB; clear STB; clear VOPENB; clear GPSOPENB;

global TAOPENB; global AOPENB; global GPSOPENB; global VOPENB;
global SUSPDOPENB; global SUSPTOPENB; global STB;

%VARIABLES INTERNAS
tscb=1; lacb=2; locb=3; alcb=4; vcb=5; tacb=6; suspdcb=7; susptcb=8;
xcb=9; ycb=10; zcb=11;
a=1; fi=1; g=1;

%APERTURA DE SESIÓN B
uiopen('LOAD');

%OBTENCIÓN DE EL NÚMERO DE VUELTAS (COLUMNAS) DE CUALQUIER SESIÓN
%GUARDADA SIN NECESIDAD DE TENER QUE CARGAR UNA SESIÓN
size(STB);
tam1=size(STB);
tamc=tam1(:,2);
c=tamc/11;

%BUCLE PARA OBTENER LOS DATOS DE CUALQUIER SESIÓN
if(c==1)
    taopeni=STB(:,tacb);
    xopeni=STB(:,xcb);
    yopeni=STB(:,ycb);
    zopeni=STB(:,zcb);
    aopeni=cat(2,xopeni,yopeni,zopeni);
    topeni=STB(:,tscb);
    laopen=STB(:,lacb);
    loopen=STB(:,locb);
    alopen=STB(:,alcb);
    vopeni=STB(:,vcb);
    gpsopeni=cat(2,taopeni,laopen,loopen,alopen,vopeni);
    suspdopeni=STB(:,suspdcb);
    susptopeni=STB(:,susptcb);

elseif(c>1)
    for i=1:c
        tam2=size(STB(:,tacb));
        tamf=(tam2(:,1));
        for f=1:tamf

            %MATRIZ SESIÓN
            taopeni(f,i)=STB(f,tacb);
            topeni(f,i)=STB(f,tscb);
            laopenii(f,i)=STB(f,lacb);
            loopenii(f,i)=STB(f,locb);
            alopeni(fi,1)=STB(f,alcb);alopenii(f,i)=STB(f,alcb);
            vopeni(f,i)=STB(f,vcb);
            suspdopeni(f,i)=STB(f,suspdcb);
            susptopeni(f,i)=STB(f,susptcb);
            xopeni(f,i)=STB(f,xcb);

```



```

yopeni (f, i) = STB (f, ycb);
zopeni (f, i) = STB (f, zcb);

%MATRIZ ACELERÓMETRO
aopeni (f, a) = xopeni (f, i);
aopeni (f, a+1) = yopeni (f, i);
aopeni (f, a+2) = zopeni (f, i);

%MATRIZ G.P.S.
gpsopeni (f, g) = topeni (f, i);
gpsopeni (f, g+1) = laopenii (f, i);
gpsopeni (f, g+2) = loopenii (f, i);
gpsopeni (f, g+3) = alopenii (f, i);
gpsopeni (f, g+4) = vopeni (f, i);
end

lacb=lacb+11;
locb=locb+11;
alcb=alcb+11;
tacb=tacb+11;
vcb=vcb+11;
suspdcb=suspdcb+11;
susptcb=susptcb+11;
xcb=xcb+11;
ycb=ycb+11;
zcb=zcb+11;
a=a+3;
g=g+5;
end
end

%ASOCIACIÓN DE MATRICES A VARIABLES
global ALOPENIB;

if (c>1)
    ALOPENIB=alopenii;
else
    ALOPENIB=alopen;
end

GPSOPENB=gpsopeni;
VOPENB=vopeni;
TAOPENB=taopeni;
SUSPDOPENB=suspdopeni;
SUSPTOPENB=susptopeni;
AOPENB=aopeni;

%CONTROL DE VARIABLES EN EL WORKSPACE
assignin ('base', 'TAOPEN', TAOPENB);
assignin ('base', 'AOPEN', AOPENB);
assignin ('base', 'VOPEN', VOPENB);
assignin ('base', 'SUSPDOPEN', SUSPDOPENB);
assignin ('base', 'SUSPTOPEN', SUSPTOPENB);
assignin ('base', 'GPSOPENB', GPSOPENB);
assignin ('base', 'ALOPENIB', ALOPENIB);

end

```



FUNCIÓN DATOS SENSOR TEMPERATURA AMBIENTE

```
function y=datostemperatura
%DATOSTEMPERATURA Summary of this function goes here
% Detailed explanation goes here

clc;

%VARIABLES GLOBALES
global C;
global mmuestras;

%INICIALIZACIÓN DE EL PUERTO SERIAL QUE UTILIZAREMOS
delete(instrfind({'Port'}, {'COM5'}));
puerto_serial=serial('COM5');
puerto_serial.BaudRate = 9600;
warning('off', 'MATLAB:serial:fscanf:unsuccessfulRead');

%APERTURA DEL PUERTO SERIAL
fopen(puerto_serial);

%CREACIÓN DE UNA VENTANA PARA LA GRÁFICA
figure('Name', 'Serial communication: Temperatura Ambiente');
title('SERIAL COMMUNICATION TEMPERATURA AMBIENTE');
xlabel('Numero de Muestras');
ylabel('Temperatura Ambiente (°C)');
grid on;
hold on;

%BUCLE ENCARGADO DE OBTENER LAS MUESTRAS DEL SENSOR TEMPERATURA
%DE UNA SESIÓN CARGADA (MUESTRAS DE ARDUINO) O GUARDADA
for c=1:C
    for f=1:mmuestras(c,1)
        ylim([5 35]);
        xlim([0 100]);

        %OBTENCIÓN DE MUESTRAS DE DATOS
        tempC = fscanf(puerto_serial, '%f');
        y(f,c) = tempC(1);

        %GRAFICAMOS LOS PUNTOS EXACTOS DE LOS DATOS OBTENIDOS DE
        %TEMPERATURA
        plot(f,y(f,c), 'Xr');
        drawnow;
    end

    %UNIÓN DE PUNTOS DE LOS DATOS OBTENIDOS
    plot(y, 'DisplayName', 'y', 'YDataSource', 'y');figure(gcf)
end

%CORTAMOS LA CONEXIÓN CON EL PUERTO SERIAL SIN ELIMINAR LOS DATOS
%OBTENIDOS
fclose(puerto_serial);
delete(puerto_serial);

end
```



FUNCIÓN DATOS SENSOR SUSPENSIÓN DELANTERA

```

function sud=datosSusDelantera
%DATOSSUSPDELANTERA Summary of this function goes here
% Detailed explanation goes here

clc;

%VARIABLES GLOBALES
global C;
global mmuestras;

%INICIALIZACIÓN DE EL PUERTO SERIAL QUE UTILIZAREMOS
delete(instrfind({'Port'}, {'COM5'}));
puerto_serial=serial('COM5');
puerto_serial.BaudRate = 9600;
warning('off', 'MATLAB:serial:fscanf:unsuccessfulRead');

%APERTURA DEL PUERTO SERIAL
fopen(puerto_serial);

%CREACIÓN DE UNA VENTANA PARA LA GRÁFICA
figure('Name', 'Serial communication: Suspensión Delantera');
title('SERIAL COMMUNICATION SUSPENSIÓN DELANTERA');
xlabel('Número de Muestras');
ylabel('Longitud (mm)');
grid on;
hold on;

%BUCLE ENCARGADO DE OBTENER LAS MUESTRAS DEL SENSOR SUSPENSIÓN
%DELANTERA DE UNA SESIÓN CARGADA (MUESTRAS DE ARDUINO) O GUARDADA
for c=1:C
    for f=1:mmuestras(c,1)
        ylim([0 80]);
        xlim([0 100]);

        %OBTENCIÓN DE MUESTRAS DE DATOS
        suspd = fscanf(puerto_serial, '%f');
        sud(f,c) = suspd(10);

        %GRAFICAMOS LOS PUNTOS EXACTOS DE LOS DATOS OBTENIDOS DE
        %S.DELANTERA
        plot(f,sud(f,c), 'Xr');
        drawnow;
    end

    %UNIÓN DE PUNTOS DE LOS DATOS OBTENIDOS
    plot(sud, 'DisplayName', 'sud', 'YDataSource', 'sud');figure(gcf)
end

%CORTAMOS LA CONEXIÓN CON EL PUERTO SERIAL SIN ELIMINAR LOS DATOS
%OBTENIDOS
fclose(puerto_serial);
delete(puerto_serial);

end

```



FUNCIÓN DATOS SENSOR SUSPENSIÓN TRASERA

```

%DATOSSUSPTRASERA Summary of this function goes here
% Detailed explanation goes here

clc;

%VARIABLES GLOBALES
global C;
global mmuestras;

%INICIALIZACIÓN DE EL PUERTO SERIAL QUE UTILIZAREMOS
delete(instrfind({'Port'}, {'COM5'}));
puerto_serial=serial('COM5');
puerto_serial.BaudRate = 9600;
warning('off', 'MATLAB:serial:fscanf:unsuccessfulRead');

%APERTURA DEL PUERTO SERIAL
fopen(puerto_serial);

%CREACIÓN DE UNA VENTANA PARA LA GRÁFICA
figure('Name','Serial communication: Suspensión Trasera');
title('SERIAL COMMUNICATION SUSPENSIÓN TRASERA');
xlabel('Número de Muestras');
ylabel('Longitud (mm)');
grid on;
hold on;

%BUCLE ENCARGADO DE OBTENER LAS MUESTRAS DEL SENSOR SUSPENSIÓN TRASERA
%DE UNA SESIÓN CARGADA (MUESTRAS DE ARDUINO) O GUARDADA
for c=1:C
    for f=1:mmuestras(c,1)
        ylim([0 80]);
        xlim([0 100]);

        %OBTENCIÓN DE MUESTRAS DE DATOS
        suspt = fscanf(puerto_serial,'%f');
        sut(f,c) = suspt(11);

        %GRAFICAMOS LOS PUNTOS EXACTOS DE LOS DATOS OBTENIDOS DE
        %S.TRASERA
        plot(f,sut(f,c),'Xr');
        drawnow;
    end

    %UNIÓN DE PUNTOS DE LOS DATOS OBTENIDOS
    plot(sut,'DisplayName','sut','YDataSource','sut');figure(gcf)
end

%CORTAMOS LA CONEXIÓN CON EL PUERTO SERIAL SIN ELIMINAR LOS DATOS
%OBTENIDOS
fclose(puerto_serial);
delete(puerto_serial);

end

```




FUNCIÓN DATOS SENSOR ACELERÓMETRO

```
function m=datosacelerometro
%DATOSACELEROMETRO Summary of this function goes here
% Detailed explanation goes here

clc;

%VARIABLES GLOBALES
global C;
global mmuestras;

%VARIABLES INTERNAS
double x;
double y;
double z;
a=1;

%INICIALIZACIÓN DE EL PUERTO SERIAL QUE UTILIZAREMOS
delete(instrfind({'Port'}, {'COM5'}));
puerto_serial=serial('COM5');
puerto_serial.BaudRate = 9600;
warning('off', 'MATLAB:serial:fscanf:unsuccessfulRead');

%APERTURA DEL PUERTO SERIAL
fopen(puerto_serial);

%CREACIÓN DE UNA VENTANA PARA LA GRÁFICA
figure('Name', 'Serial communication: Acelerómetro');
title('SERIAL COMMUNICATION ACELERÓMETRO');
xlabel('Número de Muestras');
ylabel('Angulo(°)');
grid on;
hold on;

%BUCLE ENCARGADO DE OBTENER LAS MUESTRAS DEL SENSOR ACELERÓMETRO DE
%UNA SESIÓN CARGADA (MUESTRAS DE ARDUINO) O GUARDADA
for c=1:C
    for f=1:mmuestras(c,1)
        ylim([0 380]);
        xlim([0 100]);

        %OBTENCIÓN DE MUESTRAS DE DATOS
        acel = fscanf(puerto_serial, '%f');
        x(f,c)=acel(2);
        y(f,c)=acel(3);
        z(f,c)=acel(4);

        %MATRIZ SENSOR ACELERÓMETRO ACELERÓMETRO
        m(f,a)=x(f,c);
        m(f,a+1)=y(f,c);
        m(f,a+2)=z(f,c);

        %GRAFICAMOS LOS PUNTOS EXACTOS DE LOS DATOS OBTENIDOS DE X,Y,Z
        plot(f,x(f,c), 'Xr');
        plot(f,y(f,c), 'Xb');
        plot(f,z(f,c), 'Xg');
        drawnow;
    end
end
```



```
%UNIÓN DE PUNTOS DE LOS DATOS OBTENIDOS
plot(x, 'DisplayName', 'x', 'YDataSource', 'x');figure(gcf)
plot(y, 'DisplayName', 'y', 'YDataSource', 'y');figure(gcf)
plot(z, 'DisplayName', 'z', 'YDataSource', 'z');figure(gcf)
a=a+3;
end

%CORTAMOS LA CONEXIÓN CON EL PUERTO SERIAL SIN ELIMINAR LOS DATOS
%OBTENIDOS
fclose(puerto_serial);
delete(puerto_serial);

end
```



FUNCIÓN DATOS SENSOR G.P.S.

```
function w=datosgps
%DATOSGPS Summary of this function goes here
% Detailed explanation goes here

clc;

%VARIABLES GLOBALES
global C;
global mmuestras;
global lamin;
global lamax;
global lomin;
global lomax;
global almin;
global almax;

%VARIABLES INTERNAS
double v;
double t;
double la;
double lo;
double al;
p=1;
b=1;

%INICIALIZACIÓN DE EL PUERTO SERIAL QUE UTILIZAREMOS
delete(instrfind({'Port'}, {'COM5'}));
puerto_serial=serial('COM5');
puerto_serial.BaudRate = 9600;
warning('off', 'MATLAB:serial:fscanf:unsuccessfulRead');

%APERTURA DEL PUERTO SERIAL
fopen(puerto_serial);

%CREACIÓN DE UNA VENTANA PARA LA GRÁFICA
figure('Name', 'Serial communication: G.P.S.');
```

```
%BUCLE ENCARGADO DE OBTENER LAS MUESTRAS DEL SENSOR G.P.S. DE UNA
%SESIÓN CARGADA (MUESTRAS DE ARDUINO) O GUARDADA
for c=1:C
    for f=1:mmuestras(c,1)

        %OBTENCIÓN DE MUESTRAS DE DATOS
        GPS = fscanf(puerto_serial, '%f');
        t(f,c)=GPS(5);
        la(f,c)=GPS(6);
        lo(f,c)=GPS(7);
        v(f,c)=GPS(8);
        al(f,c)=GPS(9);

        %MATRIZ CIRCUITO PARA REPRESENTAR LAS VARIABLES EN 3D Y 2D(V)
        s(f,p)=la(f,c);
        s(f,p+1)=lo(f,c);
        s(f,p+2)=al(f,c);

        %MATRIZ SENSOR G.P.S.
```



```

w(f,b)=t(f,c);
w(f,b+1)=la(f,c);
w(f,b+2)=lo(f,c);
w(f,b+3)=al(f,c);
w(f,b+4)=v(f,c);

%GRÁFICA CIRCUITO
subplot(3,1,1);
title('SERIAL COMMUNICATION G.P.S. ');
xlabel('Longitud Grados (°) ');
ylabel('Latitud Grados (°) ');
zlabel('Altitud (m) ');
grid on;
hold on;

%LÍMITES DEL CIRCUITO
zlim([almin almax]);
xlim([lomin lomax]);
ylim([lamin lamax]);

%GRAFICAMOS CIRCUITO PUNTO POR PUNTO EN 3D
plot3(lo(f,c),la(f,c),al(f,c),'Xb','Linewidth',1.5);
plot(lo,'DisplayName','lo','YDataSource','la','ZDataSource'...
,'al');figure(gcf)
rotate3d on;

%GRÁFICA VELOCIDAD
subplot(3,1,2);
xlabel('Número de Muestras');
ylabel('Velocidad (Km/h) ');
xlim([0 100]);
ylim([0 100]);
grid on;
hold on;

%GRAFICAMOS VELOCIDAD
plot(v,'DisplayName','v','YDataSource','v');figure(gcf)

%GRAFICAMOS CIRCUITO EN 2D
subplot(3,1,3);
title('CIRCUITO');
worldmap([lamin lamax],[lomin lomax]);
assignin('base','la',la);
assignin('base','lo',lo);
plotm(la,lo);
end

p=p+3;
b=b+5;
end

%CORTAMOS LA CONEXIÓN CON EL PUERTO SERIAL SIN ELIMINAR LOS DATOS
%OBTENIDOS
fclose(puerto_serial);
delete(puerto_serial);

end

```



FUNCIÓN GUIDE PRINCIPAL

```

function varargout = guidePrincipal(varargin)
% GUIDEPRINCIPAL M-file for guidePrincipal.fig
%   GUIDEPRINCIPAL, by itself, creates a new GUIDEPRINCIPAL or
%   raises the existing singleton*.
%
%   H = GUIDEPRINCIPAL returns the handle to a new GUIDEPRINCIPAL
%   or the handle to the existing singleton*.
%
%   GUIDEPRINCIPAL('CALLBACK',hObject,eventData,handles,...) calls
%   the local function named CALLBACK in GUIDEPRINCIPAL.M with the
%   given input arguments.
%
%   GUIDEPRINCIPAL('Property','Value',...) creates a new
%   GUIDEPRINCIPAL or raises the existing singleton*. Starting
%   from the left, property value pairs are applied to the GUI
%   before guidePrincipal_OpeningFcn gets called. An unrecognized
%   property name or invalid value makes property application stop.
%   All inputs are passed to guidePrincipal_OpeningFcn via
%   varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help guidePrincipal

% Last Modified by GUIDE v2.5 21-Aug-2014 17:00:09

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guidePrincipal_OpeningFcn, ...
                  'gui_OutputFcn',  @guidePrincipal_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before guidePrincipal is made visible.
function guidePrincipal_OpeningFcn(hObject, eventdata, handles,
varargout)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guidePrincipal (see VARARGIN)

```



```
% Choose default command line output for guidePrincipal
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guidePrincipal wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guidePrincipal_OutputFcn(hObject, eventdata,
handles)
% varargout    cell array for returning output args (see VARARGOUT);
% hObject     handle to figure
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%IMAGEN 1
axes(handles.GRFP1);
imagen = imread('motorland3.jpg');
axis off;
imshow(imagen);

%IMAGEN 2
axes(handles.GRFP2);
imagen = imread('uc3m2.jpg');
axis off;
imshow(imagen);

%IMAGEN 3
axes(handles.GRFP3);
imagen = imread('motostudent.jpg');
axis off;
imshow(imagen);

% --- Executes on button press in CargarSP.
function CargarSP_Callback(hObject, eventdata, handles)
% hObject     handle to CargarSP (see GCBO)
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global latmeta;
global lonmeta;
global numero_paquetesd;

%ASOCIACIÓN DE VALORES INTRODUCIDOS POR EL USUARIO A VARIABLES
latmeta=str2double(get(handles.latmeta,'string'));
lonmeta=str2double(get(handles.lonmeta,'string'));
numero_paquetesd=str2double(get(handles.numpaqd,'string'));

%LLAMADA A FUNCIÓN
s=datosSesion;
```



```

%VARIABLES GLOBALES
global OPENP; global OPENS;
global VOLVERS; global VOLVERSON; global VOLVERAC;
global VOLVERGPSC; global VOLVERSDC;
global VOLVERSTC; global VOLVERTAC;
global OPENCA; global OPENCB;
global OLA; global OCA;
global OLGP; global OCGP;
global OLSD; global OCSD;
global OLST; global OCST;
global OLTA; global OCTA;
global SAVEST;
global ONB;

%INICIALIZACIÓN DE VARIABLES GLOBALES
OPENP=0; OPENS=0;
VOLVERS=0; VOLVERSON=0; VOLVERAC=0;
VOLVERGPSC=0; VOLVERSDC=0;
VOLVERSTC=0; VOLVERTAC=0;
OPENCA=0; OPENCB=0;
OLA=0; OCA=0;
OLGP=0; OCGP=0;
OLSD=0; OCSD=0;
OLST=0; OCST=0;
OLTA=0; OCTA=0;
SAVEST=0;
ONB=0;

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
guideSesion;

% --- Executes on selection change in MenuAbrirP.
function MenuAbrirP_Callback(hObject, eventdata, handles)
% hObject    handle to MenuAbrirP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns MenuAbrirP
%        contents as cell array contents{get(hObject,'Value')} returns
%        selected item from MenuAbrirP

%ABRIMOS LOS DATOS DESDE EL EXPLORERADOR DE WINDOWS A PARTIR DE MATLAB
%Y A TRAVÉS DE INTERNET
%web https://drive.google.com/a/alumnos.uc3m.es/?tab=mo#my-drive
uiopen('LOAD');

% --- Executes during object creation, after setting all properties.
function MenuAbrirP_CreateFcn(hObject, eventdata, handles)
% hObject    handle to MenuAbrirP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
function edit1_Callback(hObject, eventdata, handles)

```




```

% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%         str2double(get(hObject,'String')) returns contents of edit1
%         as a double

% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function latmeta_Callback(hObject, eventdata, handles)
% hObject    handle to latmeta (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of latmeta as text
%         str2double(get(hObject,'String')) returns contents of latmeta
%         as a double

% --- Executes during object creation, after setting all properties.
function latmeta_CreateFcn(hObject, eventdata, handles)
% hObject    handle to latmeta (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function lonmeta_Callback(hObject, eventdata, handles)
% hObject    handle to lonmeta (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of lonmeta as text
%         str2double(get(hObject,'String')) returns contents of lonmeta
%         as a double

% --- Executes during object creation, after setting all properties.

```



```

function lonmeta_CreateFcn(hObject, eventdata, handles)
% hObject    handle to lonmeta (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in AbrirSP.
function AbrirSP_Callback(hObject, eventdata, handles)
% hObject    handle to AbrirSP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLE GLOBAL
global OPENP;

%LLAMADA A FUNCIÓN
abrirS;

%VARIABLES GLOBALES E INICIALIZACIÓN DE LAS MISMAS
OPENP=1;

global OPENS;
global VOLVERS; global VOLVERSON; global VOLVERAC;
global VOLVERGPSC; global VOLVERSDC;
global VOLVERSTC; global VOLVERTAC;
global OPENCA; global OPENCB;
global OLA; global OCA;
global OLGP; global OCGP;
global OLSD; global OCSD;
global OLST; global OCST;
global OLTA; global OCTA;
global SAVEST;
global ONB;

OPENS=0;
VOLVERS=0; VOLVERSON=0; VOLVERAC=0;
VOLVERGPSC=0; VOLVERSDC=0;
VOLVERSTC=0; VOLVERTAC=0;
OPENCA=0; OPENCB=0;
OLA=0; OCA=0;
OLGP=0; OCGP=0;
OLSD=0; OCSD=0;
OLST=0; OCST=0;
OLTA=0; OCTA=0;
SAVEST=1;
ONB=0;

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
guideSesion;

```



```
function numpaqd_Callback(hObject, eventdata, handles)
% hObject    handle to numpaqd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of numpaqd as text
%        str2double(get(hObject,'String')) returns contents of numpaqd
%        as a double

% --- Executes during object creation, after setting all properties.
function numpaqd_CreateFcn(hObject, eventdata, handles)
% hObject    handle to numpaqd (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),...
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in LIMPIAR.
function LIMPIAR_Callback(hObject, eventdata, handles)
% hObject    handle to LIMPIAR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
limpia=' ';
set(handles.numpaqd,'string',limpia);
set(handles.latmeta,'string',limpia);
set(handles.lonmeta,'string',limpia);
```



FUNCIÓN GUIDE SESIÓN

```

function varargout = guideSesion(varargin)
% GUIDESESSION M-file for guideSesion.fig
%   GUIDESESSION, by itself, creates a new GUIDESESSION or raises the
%   existing singleton*.
%
%   H = GUIDESESSION returns the handle to a new GUIDESESSION or the
%   handle to the existing singleton*.
%
%   GUIDESESSION('CALLBACK',hObject,eventData,handles,...) calls the
%   local function named CALLBACK in GUIDESESSION.M with the given
%   input arguments.
%
%   GUIDESESSION('Property','Value',...) creates a new GUIDESESSION
%   or raises the existing singleton*. Starting from the left,
%   property value pairs are applied to the GUI before
%   guideSesion_OpeningFcn gets called. An unrecognized property
%   name or invalid value makes property application stop. All
%   inputs are passed to guideSesion_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help guideSesion

% Last Modified by GUIDE v2.5 17-Jun-2014 17:36:57

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideSesion_OpeningFcn, ...
                  'gui_OutputFcn',  @guideSesion_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before guideSesion is made visible.
function guideSesion_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideSesion (see VARARGIN)

% Choose default command line output for guideSesion
handles.output = hObject;

```



```

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideSesion wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideSesion_OutputFcn(hObject, eventdata,
handles)
% varargout    cell array for returning output args (see VARARGOUT);
% hObject     handle to figure
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
global ST;
global OPENP; global OPENCA;
global LAOPEN; global LOOPEN;
global LAP; global LOP;
global VOLVERSON;
global lamin; global lamax;
global lomin; global lomax;

%CONTROL DE VARIABLES EN EL WORKSPACE
assignin('base','ST',ST);
assignin('base','LAOPEN',LAOPEN);
assignin('base','LOOPEN',LOOPEN);

%ABRIMOS SESIÓN GUARDADA O SESIÓN CARGADA
if((OPENP==1) || (OPENCA==1))

    %TABLA DATOS SESIÓN
    set(handles.SDA,'Data',ST);

    %GRÁFICA 1 (CIRCUITO 1)
    axes(findobj('type','axes','tag','SGC1'));
    worldmap([lamin lamax],[lomin lomax]);
    plotm(LAOPEN,LOOPEN);

    %GRÁFICA 2 (CIRCUITO 2)
    axes(findobj('type','axes','tag','SGC2'));
    worldmap([lamin lamax],[lomin lomax]);
    plotm(LAOPEN,LOOPEN);

elseif (VOLVERSON~=1)

    %CONTROL DE VARIABLES EN EL WORKSPACE
    assignin('base','LAS',LAP);
    assignin('base','LOS',LOP);

    %TABLA DATOS SESIÓN
    set(handles.SDA,'Data',ST);

    %GRÁFICA 1 (CIRCUITO 1)
    axes(findobj('type','axes','tag','SGC1'));
    worldmap([lamin lamax],[lomin lomax]);

```



```

plotm(LAP,LOP);

%GRÁFICA 2 (CIRCUITO 2)
axes(findobj('type','axes','tag','SGC2'));
worldmap([lamin lamax],[lomin lomax]);
plotm(LAP,LOP);
end

% --- Executes on button press in SACELEROMETRO.
function SACELEROMETRO_Callback(hObject, eventdata, handles)
% hObject    handle to SACELEROMETRO (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global OPENP;
global OPENS;
global OPENCA;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENS==1) || (OPENP==1) || (OPENCA==1))

    %LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
    guideSacelerometro;

else

    %LLAMADA A FUNCIÓN Y A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
    m=datosacelerometro;
    guideSacelerometro;
end

% --- Executes on button press in SSUSPD.
function SSUSPD_Callback(hObject, eventdata, handles)
% hObject    handle to SSUSPD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global OPENP;
global OPENS;
global OPENCA;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENS==1) || (OPENP==1) || (OPENCA==1))

    %LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
    guideSsuspdelantera;

else

    %LLAMADA A FUNCIÓN Y A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
    sud=datosSuspDelantera;
    guideSsuspdelantera;
end

```



```

% --- Executes on button press in SSUSPT.
function SSUSPT_Callback(hObject, eventdata, handles)
% hObject    handle to SSUSPT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global OPENP;
global OPENS;
global OPENCA;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENS==1) || (OPENP==1) || (OPENCA==1))

    %LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
    guideSsusptrasera;

else

    %LLAMADA A FUNCIÓN Y A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
    sut=datosSuspTrasera;
    guideSsusptrasera;
end

% --- Executes on button press in INICIO.
function INICIO_Callback(hObject, eventdata, handles)
% hObject    handle to INICIO (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global OPENP; global OPENS; global OPENCA;
global SAVEST;
global VOLVERS; global VOLVERAC; global VOLVERGPSC;
global VOLVERSDC; global VOLVERSTC; global VOLVERTAC;

%RETORNO AL MENÚ PRINCIPAL O PERMANECEMOS EN MENÚ SESIÓN
if((OPENP==1) || (OPENS==1) || (SAVEST==1))

    %PREGUNTA PARA EL USUARIO
    resp=questdlg('?DESEA ABANDONAR EL MENÚ SESIÓN?', 'INICIO...
    (2)', 'Si', 'No', 'No');

    if strcmp(resp, 'No')
        return;
    else
        OPENP=0;
        OPENS=0;
        OPENCA=0;
        SAVEST=0;

        %CERRAMOS VENTANAS DE COMPARATIVA ENTRE SENSORES
        if (VOLVERS==1)
            close('TELEMETRÍA-COMPARAR SESIÓN');
        end
        if (VOLVERAC==1)
            close('TELEMETRÍA-ACELERÓMETRO (VS) ACELERÓMETRO');
        end
    end
end

```




```

end
if (VOLVERGPSC==1)
    close('TELEMETRÍA-G.P.S. (VS) G.P.S.');
```

```

end
if (VOLVERSDC==1)
    close('TELEMETRÍA-SUSPENSIÓN DELANTERA (VS)...
    SUSPENSIÓN DELANTERA');
```

```

end
if (VOLVERSTC==1)
    close('TELEMETRÍA-SUSPENSIÓN TRASERA (VS)...
    SUSPENSIÓN TRASERA');
```

```

end
if (VOLVERTAC==1)
    close('TELEMETRÍA-TEMPERATURA AMBIENTE (VS)...
    TEMPERATURA AMBIENTE');
```

```

end

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)Y
%CIERRE DEL GUIDE TELEMETRÍA-SESIÓN
guidePrincipal;
close(handles.output);

end

else

%PREGUNTA PARA EL USUARIO
resp=questdlg('LOS DATOS DE LA SESIÓN ACTUAL, AL NO SER...
GUARDADOS, SE PERDERÁN. ¿DESEA CONTINUAR SIN GUARDAR LA...
SESIÓN?', 'INICIO (1)', 'Si', 'No', 'No');
```

```

if strcmp(resp, 'No')
    return;
else
    OPENP=0;
    OPENS=0;
    OPENCA=0;
    SAVEST=0;

    %CERRAMOS VENTANAS DE COMPARATIVA ENTRE SENSORES
    if (VOLVERS==1)
        close('TELEMETRÍA-COMPARAR SESIÓN');
```

```

    end
    if (VOLVERAC==1)
        close('TELEMETRÍA-ACELERÓMETRO (VS) ACELERÓMETRO');
```

```

    end
    if (VOLVERGPSC==1)
        close('TELEMETRÍA-G.P.S. (VS) G.P.S.');
```

```

    end
    if (VOLVERSDC==1)
        close('TELEMETRÍA-SUSPENSIÓN DELANTERA (VS) SUSPENSIÓN...
        DELANTERA');
```

```

    end
    if (VOLVERSTC==1)
        close('TELEMETRÍA-SUSPENSIÓN TRASERA (VS) SUSPENSIÓN...
        TRASERA');
```

```

    end
    if (VOLVERTAC==1)
        close('TELEMETRÍA-TEMPERATURA AMBIENTE (VS) TEMPERATURA...
        AMBIENTE');
```

```

    end
end

```



```

        %LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE
        %DEL GUIDE TELEMETRÍA-SESIÓN
        guidePrincipal;
        close(handles.output);
    end
end

% --- Executes on button press in CLIMA.
function CLIMA_Callback(~, eventdata, handles)
% hObject    handle to CLIMA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%APERTURA DE PÁGINA WEB DE INTERNET USANDO EL ENTORNO DE MATLAB
web 'www.eltiempo.es';

% --- Executes on button press in ABRIR.
function ABRIR_Callback(hObject, eventdata, handles)
% hObject    handle to ABRIR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global LAOPEN; global LOOPEN;
global lamin; global lamax;
global lomin; global lomax;
global OPENCA; global OPENP; global OPENS;
global ST;
global SAVEST;

%ABRIMOS SESIÓN GUARDADA
if((OPENP==1) || (OPENS==1) || (OPENCA==1) || (SAVEST==1))
    abrirS;
    OPENS=1;

    %TABLA DATOS SESIÓN
    set(handles.SDA, 'Data', ST);

    %GRÁFICA 1 (CIRCUITO 1)
    axes(findobj('type','axes','tag','SGC1'));
    worldmap([lamin lamax],[lomin lomax]);
    plotm(LAOPEN, LOOPEN);

else

    %PREGUNTA PARA EL USUARIO
    resp=questdlg('LOS DATOS DE LA SESIÓN ACTUAL, AL NO SER...
    GUARDADOS, SE PERDERÁN. ¿DESEA CONTINUAR SIN GUARDAR LA...
    SESIÓN?', 'ABRIR', 'Si', 'No', 'No');

    if strcmp(resp, 'No')
        return;
    else

        %LLAMADA A FUNCIÓN
        abrirS;
        OPENS=1;
    end
end

```



```

%TABLA DATOS SESIÓN
set(handles.SDA, 'Data', ST);

%GRÁFICA 1 (CIRCUITO 1)
axes(findobj('type','axes','tag','SGC1'));
worldmap([lamin lamax],[lomin lomax]);
plotm(LAOPEN, LOOPEN);
end
end

% --- Executes on button press in GUARDAR.
function GUARDAR_Callback(hObject, eventdata, handles)
% hObject    handle to GUARDAR (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global SAVEST;
global OPENS;
global OPENP;

%GUARDAMOS SESIÓN Y SESIÓN B O SEGUIMOS EN MENÚ SESIÓN
if((SAVEST==1) || ((OPENS==1) && (SAVEST==0) || (OPENP==1)))

    %AVISO INFORMATIVO AL USUARIO
    errordlg('LOS DATOS DE ESTA SESIÓN YA FUERON GUARDADOS.',...
    'AYUDA-GUARDAR (2)');

else
    uisave;

    %AVISO INFORMATIVO AL USUARIO
    resp=questdlg({'A CONTINUACIÓN, GUARDAREMOS DE NUEVO LOS DATOS...
    DE LA SESIÓN PARA CONTAR CON ELLOS EN CUALQUIER TIPO DE...
    COMPARATIVA, PARA ELLO:', ' ','- GUARDAREMOS LA SESIÓN,...
    UTILIZANDO EL MISMO NOMBRE ESCRITO ANTERIORMENTE PERO TERMINADO...
    EN ( B ) MAYÚSCULA.'}, 'AYUDA-GUARDAR (1)', 'Continuar',...
    'Continuar');

    if strcmp(resp, 'Continuar')
        uisave;
    end

    SAVEST=1;
end

% --- Executes on button press in COMP.
function COMP_Callback(hObject, eventdata, handles)
% hObject    handle to COMP (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
guideScompararsesion;

```



```
% --- Executes on button press in TEMPERATURA.
function TEMPERATURA_Callback(hObject, eventdata, handles)
% hObject    handle to TEMPERATURA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global OPENP;
global OPENS;
global OPENCA;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENS==1) || (OPENP==1) || (OPENCA==1))

    %LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
    guideStemperatura;

else

    %LLAMADA A FUNCIÓN Y A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
    y=datostemperatura;
    guideStemperatura;
end

% --- Executes on button press in SGPS.
function SGPS_Callback(hObject, eventdata, handles)
% hObject    handle to SGPS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global OPENP;
global OPENS;
global OPENCA;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENS==1) || (OPENP==1) || (OPENCA==1))

    %LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
    guideSgps;

else

    %LLAMADA A FUNCIÓN Y A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
    w=datosgps;
    guideSgps;
end

% --- Executes on button press in CIRCUITO2.
function CIRCUITO2_Callback(hObject, eventdata, handles)
% hObject    handle to CIRCUITO2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global LAOPEN; global LOOPEN;
global lamin; global lamax;
global lomín; global lomax;
```



```
%LLAMADA A FUNCIÓN
abrirS;

%GRÁFICA 2 (CIRCUITO 2)
axes(findobj('type','axes','tag','SGC2'));
worldmap([lamin lamax],[lomin lomax]);
plotm(LAOPEN,LOOPEN);
```



FUNCIÓN SENSOR TEMPERATURA AMBIENTE

```

function varargout = guideStemperatura(varargin)
% GUIDESTEMPERATURA M-file for guideStemperatura.fig
%   GUIDESTEMPERATURA, by itself, creates a new GUIDESTEMPERATURA
%   or raises the existing singleton*.
%
%   H = GUIDESTEMPERATURA returns the handle to a new
%   GUIDESTEMPERATURA or the handle to the existing singleton*.
%
%   GUIDESTEMPERATURA('CALLBACK',hObject,eventData,handles,...)
%   calls the local function named CALLBACK in GUIDESTEMPERATURA.M
%   with the given input arguments.
%
%   GUIDESTEMPERATURA('Property','Value',...) creates a new
%   GUIDESTEMPERATURA or raises the existing singleton*. Starting
%   from the left, property value pairs are applied to the GUI
%   before guideStemperatura_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideStemperatura_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help guideStemperatura

% Last Modified by GUIDE v2.5 27-Aug-2014 12:03:59

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideStemperatura_OpeningFcn,
                  ...
                  'gui_OutputFcn',  @guideStemperatura_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before guideStemperatura is made visible.
function guideStemperatura_OpeningFcn(hObject, eventdata, handles,
varargout)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideStemperatura (see
% VARARGIN)

```



```

% Choose default command line output for guideStemperatura
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideStemperatura wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideStemperatura_OutputFcn(hObject, eventdata,
handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%GRÁFICA 2 (IMAGEN 1)
axes(handles.STG2);
imagen = imread('lm35.jpg');
axis off;
imshow(imagen);

%PULSANDO BOTÓN TEMPERATURA(MENÚ SESIÓN)

%VARIABLES GLOBALES
global TAST;
global TAOPEN;
global OPENP;
global OPENS;
global OPENCA;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENS==1) || (OPENP==1) || (OPENCA==1))

    %TABLA 1 (DATOS GUARDADOS DEL SENSOR TEMPERATURA)
    set(handles.STDA, 'Data', TAOPEN);

    %GRÁFICA 1 (DATOS GUARDADOS DEL SENSOR TEMPERATURA)
    plot(handles.STG1, TAOPEN, 'DisplayName', 'OPENVUELTA', ...
'YDataSource', 'OPEN VUELTA'); figure(gcf)
    set(handles.STG1, 'xgrid', 'on', 'ygrid', 'on');
    xlabel(handles.STG1, 'Número de Muestras');
    ylabel(handles.STG1, 'Temperatura Ambiente (C°)');
    title(handles.STG1, 'Gráfica Temperatura Ambiente [(C°)]', ...
'FontSize', 14, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');

else

    %CIERRA LA GRÁFICA DE LA FUNCIÓN TEMPERATURA
    close('Serial communication: Temperatura Ambiente');

    %TABLA 1 (DATOS ACTUALES DEL SENSOR TEMPERATURA)
    set(handles.STDA, 'Data', TAST);

```




```

%GRÁFICA 1 (DATOS ACTUALES DEL SENSOR TEMPERATURA)
plot(handles.STG1,TAST,'DisplayName','VUELTA','YDataSource',...
'VUELTA');figure(gcf)
set(handles.STG1,'xgrid','on','ygrid','on');
xlabel(handles.STG1,'Número de Muestras');
ylabel(handles.STG1,'Temperatura Ambiente (C°)');
title(handles.STG1,'Gráfica Temperatura Ambiente [(C°)]',...
'FontSize',14,'FontName','MS Sans Serif','FontWeight','bold');
end

% --- Executes on button press in MS.
function MS_Callback(hObject, eventdata, handles)
% hObject    handle to MS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideSesion;
close(handles.output);

% --- Executes on button press in leyendaTA.
function leyendaTA_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaTA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaTA

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA (ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    legend(handles.STG1,'show');
    legend(handles.STG1,'boxoff');
    set(handles.leyendaTA,'String','LEYENDA ON');
else
    legend(handles.STG1,'off');
    set(handles.leyendaTA,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrículaTA.
function cuadrículaTA_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaTA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of cuadrículaTA

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRICULA (ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

```



```
if handles.o2==1
    grid(handles.STG1, 'off');
    set(handles.cuadriculaTA, 'String', 'CUADRICULA OFF');
else
    grid(handles.STG1, 'on');
    set(handles.cuadriculaTA, 'String', 'CUADRICULA ON');
end
```



FUNCIÓN GUIDE SENSOR SUSPENSIÓN DELANTERA

```

function varargout = guideSsuspdelantera(varargin)
% GUIDESSUSPDELANTERA M-file for guideSsuspdelantera.fig
%   GUIDESSUSPDELANTERA, by itself, creates a new
%   GUIDESSUSPDELANTERA or raises the existing singleton*.
%
%   H = GUIDESSUSPDELANTERA returns the handle to a new
%   GUIDESSUSPDELANTERA or the handle to the existing singleton*.
%
%   GUIDESSUSPDELANTERA('CALLBACK',hObject,eventData,handles,...)
%   calls the local function named CALLBACK in
%   GUIDESSUSPDELANTERA.M with the given input arguments.
%
%   GUIDESSUSPDELANTERA('Property','Value',...) creates a new
%   GUIDESSUSPDELANTERA or raises the existing singleton*.
%   Starting from the left, property value pairs are applied to the
%   GUI before guideSsuspdelantera_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideSsuspdelantera_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideSsuspdelantera

% Last Modified by GUIDE v2.5 27-Aug-2014 12:17:38

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideSsuspdelantera_OpeningFcn,
                  ...
                  'gui_OutputFcn',  @guideSsuspdelantera_OutputFcn,
                  ...
                  'gui_LayoutFcn',   [] , ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before guideSsuspdelantera is made visible.
function guideSsuspdelantera_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```



```

% varargin    command line arguments to guideSsuspdelantera (see
% VARARGIN)

% Choose default command line output for guideSsuspdelantera
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideSsuspdelantera wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideSsuspdelantera_OutputFcn(hObject, eventdata,
handles)
% varargout    cell array for returning output args (see VARARGOUT);
% hObject     handle to figure
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%GRÁFICA 2 (IMAGEN 1)
axes(handles.SSDG2);
imagen = imread('suspd.jpg');
axis off;
imshow(imagen);

%PULSANDO BOTÓN SUSPENSIÓN DELANTERA(MENÚ SESIÓN)

%VARIABLES GLOBALES
global SUSPDST;
global SUSPDOPEN;
global OPENP;
global OPENS;
global OPENCA;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENS==1) || (OPENP==1) || (OPENCA==1))

    %TABLA 1 (DATOS GUARDADOS DEL SENSOR SUSPENSIÓN DELANTERA)
    set(handles.SSDA, 'Data', SUSPDOPEN);

    %GRÁFICA 1 (DATOS GUARDADOS DEL SENSOR SUSPENSIÓN DELANTERA)
    plot(handles.SSDG1, SUSPDOPEN, 'DisplayName', 'OPEN VUELTA', ...
'YDataSource', 'OPEN VUELTA'); figure(gcf)
    set(handles.SSDG1, 'xgrid', 'on', 'ygrid', 'on');
    xlabel(handles.SSDG1, 'Número de Muestras');
    ylabel(handles.SSDG1, 'Longitud (mm)');
    title(handles.SSDG1, 'Gráfica Suspensión Delantera [(mm)]', ...
'FontSize', 14, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');

else

    %CIERRA LA GRÁFICA DE LA FUNCIÓN DEL SENSOR SUSPENSIÓN DELANTERA
    close('Serial communication: Suspensión Delantera');

```



```

%TABLA 1 (DATOS ACTUALES DEL SENSOR SUSPENSIÓN DELANTERA)
set(handles.SSDA, 'Data', SUSPDST);

%GRÁFICA 1 (DATOS ACTUALES DEL SENSOR SUSPENSIÓN DELANTERA)
plot(handles.SSDG1, SUSPDST, 'DisplayName', 'VUELTA', 'YDataSource'...
, 'VUELTA'); figure(gcf)
set(handles.SSDG1, 'xgrid', 'on', 'ygrid', 'on');
xlabel(handles.SSDG1, 'Número de Muestras');
ylabel(handles.SSDG1, 'Longitud (mm)');
title(handles.SSDG1, 'Gráfica Suspensión Delantera [(mm)]',...
'FontSize', 14, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');
end

% --- Executes on button press in leyendaSD.
function leyendaSD_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaSD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaSD

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA (ON,OFF)
o=get(hObject, 'Value');
handles.o=o;

if handles.o==1
    legend(handles.SSDG1, 'show');
    legend(handles.SSDG1, 'boxoff');
    set(handles.leyendaSD, 'String', 'LEYENDA ON');
else
    legend(handles.SSDG1, 'off');
    set(handles.leyendaSD, 'String', 'LEYENDA OFF');
end

% --- Executes on button press in cuadrículaSD.
function cuadrículaSD_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaSD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of cuadrículaSD

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA (ON,OFF)
o2=get(hObject, 'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SSDG1, 'off');
    set(handles.cuadrículaSD, 'String', 'CUADRÍCULA OFF');

else
    grid(handles.SSDG1, 'on');
    set(handles.cuadrículaSD, 'String', 'CUADRÍCULA ON');
end

```



```
% --- Executes on button press in MS.
function MS_Callback(hObject, eventdata, handles)
% hObject    handle to MS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideSesion;
close(handles.output);
```



FUNCIÓN GUIDE SENSOR SUSPENSIÓN TRASERA

```

function varargout = guideSsusptrasera(varargin)
% GUIDESSUSPTRASERA M-file for guideSsusptrasera.fig
%   GUIDESSUSPTRASERA, by itself, creates a new GUIDESSUSPTRASERA
%   or raises the existing singleton*.
%
%   H = GUIDESSUSPTRASERA returns the handle to a new
%   GUIDESSUSPTRASERA or the handle to the existing singleton*.
%
%   GUIDESSUSPTRASERA('CALLBACK',hObject,eventData,handles,...)
%   calls the local function named CALLBACK in GUIDESSUSPTRASERA.M
%   with the given input arguments.
%
%   GUIDESSUSPTRASERA('Property','Value',...) creates a new
%   GUIDESSUSPTRASERA or raises the existing singleton*. Starting
%   from the left, property value pairs are applied to the GUI
%   before guideSsusptrasera_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideSsusptrasera_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help guideSsusptrasera

% Last Modified by GUIDE v2.5 27-Aug-2014 12:50:10

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideSsusptrasera_OpeningFcn,
                  ...
                  'gui_OutputFcn',  @guideSsusptrasera_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before guideSsusptrasera is made visible.
function guideSsusptrasera_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideSsusptrasera (see
% VARARGIN)

```




```

% Choose default command line output for guideSsusptrasera
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideSsusptrasera wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideSsusptrasera_OutputFcn(hObject, eventdata,
handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%GRÁFICA 2 (IMAGEN 1)
axes(handles.SSTG2);
imagen = imread('suspt.jpg');
axis off;
imshow(imagen);

%PULSANDO BOTÓN SUSPENSIÓN TRASERA(MENÚ SESIÓN)

%VARIABLES GLOBALES
global SUSPTST;
global SUSPTOPEN;
global OPENP;
global OPENS;
global OPENCA;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENS==1) || (OPENP==1) || (OPENCA==1))

    %TABLA 1 (DATOS GUARDADOS DEL SENSOR SUSPENSIÓN TRASERA)
    set(handles.SSTDA, 'Data', SUSPTOPEN);

    %GRÁFICA 1 (DATOS GUARDADOS DEL SENSOR SUSPENSIÓN TRASERA)
    plot(handles.SSTG1, SUSPTOPEN, 'DisplayName', 'OPEN VUELTA', ...
'YDataSource', 'OPEN VUELTA');figure(gcf)
    set(handles.SSTG1, 'xgrid', 'on', 'ygrid', 'on');
    xlabel(handles.SSTG1, 'Número de Muestras');
    ylabel(handles.SSTG1, 'Suspensión Trasera (mm)');
    title(handles.SSTG1, 'Gráfica Suspensión Trasera [(mm)]', ...
'FontSize', 14, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');

else

    %CIERRA LA GRÁFICA DE LA FUNCIÓN DEL SENSOR SUSPENSIÓN TRASERA
    close('Serial communication: Suspensión Trasera');

    %TABLA 1 (DATOS ACTUALES DEL SENSOR SUSPENSIÓN TRASERA)
    set(handles.SSTDA, 'Data', SUSPTST);

```



```

%GRÁFICA 1 (DATOS ACTUALES DEL SENSOR SUSPENSIÓN TRASERA)
plot(handles.SSTG1,SUSPTST,'DisplayName','VUELTA','YDataSource'...
,'VUELTA');figure(gcf)
set(handles.SSTG1,'xgrid','on','ygrid','on');
xlabel(handles.SSTG1,'Número de Muestras');
ylabel(handles.SSTG1,'Suspensión Trasera (mm)');
title(handles.SSTG1,'Gráfica Suspensión Trasera [(mm)]',...
'FontSize',14,'FontName','MS Sans Serif','FontWeight','bold');
end

% --- Executes on button press in MS.
function MS_Callback(hObject, eventdata, handles)
% hObject    handle to MS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideSesion;
close(handles.output);

% --- Executes on button press in leyendaST.
function leyendaST_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaST (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaST

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA (ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    legend(handles.SSTG1,'show');
    legend(handles.SSTG1,'boxoff');
    set(handles.leyendaST,'String','LEYENDA ON');
else
    legend(handles.SSTG1,'off');
    set(handles.leyendaST,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrículaST.
function cuadrículaST_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaST (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of cuadrículaST

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA (ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SSTG1,'off');
    set(handles.cuadrículaST,'String','CUADRÍCULA OFF');

```



```
else
    grid(handles.SSTG1, 'on');
    set(handles.cuadrículaST, 'String', 'CUADRICULA ON');
end
```



FUNCIÓN GUIDE SENSOR ACELERÓMETRO

```

function varargout = guideSacelerometro(varargin)
% GUIDESACCELEROMETRO M-file for guideSacelerometro.fig
%   GUIDESACCELEROMETRO, by itself, creates a new GUIDESACCELEROMETRO
%   or raises the existing singleton*.
%
%   H = GUIDESACCELEROMETRO returns the handle to a new
%   GUIDESACCELEROMETRO or the handle to the existing singleton*.
%
%   GUIDESACCELEROMETRO('CALLBACK',hObject,eventData,handles,...)
%   calls the local function named CALLBACK in GUIDESACCELEROMETRO.M
%   with the given input arguments.
%
%   GUIDESACCELEROMETRO('Property','Value',...) creates a new
%   GUIDESACCELEROMETRO or raises the existing singleton*. Starting
%   from the left, property value pairs are applied to the GUI
%   before guideSacelerometro_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideSacelerometro_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideSacelerometro

% Last Modified by GUIDE v2.5 27-Aug-2014 13:25:56

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideSacelerometro_OpeningFcn,
                  ...
                  'gui_OutputFcn',  @guideSacelerometro_OutputFcn,
                  ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before guideSacelerometro is made visible.
function guideSacelerometro_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```



```

% varargin    command line arguments to guideSacerometro (see
% VARARGIN)

% Choose default command line output for guideSacerometro
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideSacerometro wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideSacerometro_OutputFcn(hObject, eventdata,
handles)
% varargout    cell array for returning output args (see VARARGOUT);
% hObject     handle to figure
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%GRÁFICA 2 (IMAGEN 1)
axes(handles.SAG2);
imagen = imread('adx1335.jpg');
axis off;
imshow(imagen);

%PULSANDO BOTÓN ACELERÓMETRO (MENÚ SESIÓN)

%VARIABLES GLOBALES
global AST;
global AOPEN;
global OPENP;
global OPENS;
global OPENCA;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENS==1) || (OPENP==1) || (OPENCA==1))

    %TABLA 1 (DATOS GUARDADOS DEL SENSOR ACELERÓMETRO)
    set(handles.SADA, 'Data', AOPEN);

    %GRÁFICA 1 (DATOS GUARDADOS DEL SENSOR ACELERÓMETRO)
    plot(handles.SAG1, AOPEN, 'DisplayName', 'OPEN VUELTA', ...
'YDataSource', 'OPEN VUELTA'); figure(gcf)
    set(handles.SAG1, 'xgrid', 'on', 'ygrid', 'on');
    xlabel(handles.SAG1, 'Número de Muestras');
    ylabel(handles.SAG1, 'Acelerómetro (A°)');
    title(handles.SAG1, 'Gráfica Acelerómetro [(A°)]', ...
'FontSize', 14, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');

else

    %CIERRA LA GRÁFICA DE LA FUNCIÓN DEL SENSOR ACELERÓMETRO
    close('Serial communication: Acelerómetro');

```



```

%TABLA 1 (DATOS ACTUALES DEL SENSOR ACELERÓMETRO)
set(handles.SADA, 'Data', AST);

%GRÁFICA 1 (DATOS ACTUALES DEL SENSOR ACELERÓMETRO)
plot(handles.SAG1, AST, 'DisplayName', 'VUELTA', 'YDataSource', ...
'VUELTA'); figure(gcf)
set(handles.SAG1, 'xgrid', 'on', 'ygrid', 'on');
xlabel(handles.SAG1, 'Número de Muestras');
ylabel(handles.SAG1, 'Acelerómetro (A°)');
title(handles.SAG1, 'Gráfica Acelerómetro [(A°)]', ...
'FontSize', 14, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');
end

% --- Executes on button press in leyendaA.
function leyendaA_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaA

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA (ON,OFF)
o=get(hObject, 'Value');
handles.o=o;

if handles.o==1
    legend(handles.SAG1, 'show');
    legend(handles.SAG1, 'boxoff');
    set(handles.leyendaA, 'String', 'LEYENDA ON');
else
    legend(handles.SAG1, 'off');
    set(handles.leyendaA, 'String', 'LEYENDA OFF');
end

% --- Executes on button press in cuadrículaA.
function cuadrículaA_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of cuadrículaA

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA (ON,OFF)
o2=get(hObject, 'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SAG1, 'off');
    set(handles.cuadrículaA, 'String', 'CUADRÍCULA OFF');
else
    grid(handles.SAG1, 'on');
    set(handles.cuadrículaA, 'String', 'CUADRÍCULA ON');
end

```



```
% --- Executes on button press in MS.
function MS_Callback(hObject, eventdata, handles)
% hObject    handle to MS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideSesion;
close(handles.output);
```




FUNCIÓN GUIDE SENSOR G.P.S.

```

function varargout = guideSgps(varargin)
% GUIDESGPS M-file for guideSgps.fig
%   GUIDESGPS, by itself, creates a new GUIDESGPS or raises the
%   existing singleton*.
%
%   H = GUIDESGPS returns the handle to a new GUIDESGPS or the
%   handle to the existing singleton*.
%
%   GUIDESGPS('CALLBACK',hObject,eventData,handles,...) calls the
%   local function named CALLBACK in GUIDESGPS.M with the given
%   input arguments.
%
%   GUIDESGPS('Property','Value',...) creates a new GUIDESGPS or
%   raises the existing singleton*. Starting from the left,
%   property value pairs are applied to the GUI before
%   guideSgps_OpeningFcn gets called. An unrecognized property
%   name or invalid value makes property application stop. All
%   inputs are passed to guideSgps_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help guideSgps

% Last Modified by GUIDE v2.5 27-Aug-2014 17:47:01

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideSgps_OpeningFcn, ...
                  'gui_OutputFcn',  @guideSgps_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before guideSgps is made visible.
function guideSgps_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideSgps (see VARARGIN)

% Choose default command line output for guideSgps
handles.output = hObject;

```



```

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideSgps wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideSgps_OutputFcn(hObject, eventdata, handles)
% varargout cell array for returning output args (see VARARGOUT);
% hObject handle to figure
% eventdata reserved - to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%GRÁFICA 4 (IMAGEN 1)
axes(handles.SGG4);
imagen = imread('venusgps.jpg');
axis off;
imshow(imagen);

%PULSANDO BOTÓN G.P.S. (MENÚ SESIÓN)

%VARIABLES GLOBALES
global GPSST; global GPSOPEN;
global VST; global VOPEN;
global ALST; global ALOPENI;
global LAP; global LOP; global ALP;
global LAOPEN; global LOOPEN; global ALOPEN;
global OPENP; global OPENS; global OPENCA;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENS==1) || (OPENP==1) || (OPENCA==1))

    %TABLA 1 (DATOS GUARDADOS DEL SENSOR G.P.S.)
    set(handles.SGDA, 'Data', GPSOPEN);

    %GRÁFICA 1 (VELOCIDAD GUARDADA DEL SENSOR G.P.S.)
    plot(handles.SGG1, VOPEN, 'DisplayName', 'OPEN VUELTA', ...
        'YDataSource', 'OPEN VUELTA'); figure(gcf)
    set(handles.SGG1, 'xgrid', 'on', 'ygrid', 'on');
    ylabel(handles.SGG1, 'Velocidad (Km/h).');
    title(handles.SGG1, 'Gráfica G.P.S. [Velocidad(Km/h)].', ...
        'FontSize', 14, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');

    %GRÁFICA 2 (ALTITUD GUARDADA DEL SENSOR G.P.S.)
    plot(handles.SGG2, ALOPENI, 'DisplayName', 'OPEN VUELTA', ...
        'YDataSource', 'VUELTA'); figure(gcf)
    set(handles.SGG2, 'xgrid', 'on', 'ygrid', 'on');
    ylabel(handles.SGG2, 'Altitud (m)');
    title(handles.SGG2, 'Gráfica G.P.S. [Altitud(m)].', ...
        'FontSize', 14, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');

    %GRÁFICA 3 (RECORRIDO GUARDADO DEL SENSOR G.P.S.)
    axes(findobj('type', 'axes', 'tag', 'SGG3'));
    plot3(LAOPEN, LOOPEN, ALOPEN);
    xlabel(handles.SGG3, 'Longitud Grados(°)');
    ylabel(handles.SGG3, 'Latitud Grados(°)');

```



```

zlabel(handles.SGG3,'Altitud (m)');
title(handles.SGG3,'Gráfica G.P.S. [Recorrido3D]',...
'FontSize',14,'FontName','MS Sans Serif','FontWeight','bold');

else

%CIERRA LA GRÁFICA DE LA FUNCIÓN G.P.S.
close('Serial communication: G.P.S.');
```

%TABLA 1 (DATOS ACTUALES DEL G.P.S.)

```

set(handles.SGDA,'Data',GPSST);

%GRÁFICA 1 (VELOCIDAD ACTUAL DEL SENSOR G.P.S.)
plot(handles.SGG1,VST,'DisplayName','VUELTA','YDataSource',...
'VUELTA');figure(gcf)
set(handles.SGG1,'xgrid','on','ygrid','on');
ylabel(handles.SGG1,'Velocidad (Km/h)');
title(handles.SGG1,'Gráfica G.P.S. [Velocidad(Km/h)]',...
'FontSize',14,'FontName','MS Sans Serif','FontWeight','bold');
```

%GRÁFICA 2 (ALTITUD ACTUAL DEL SENSOR G.P.S.)

```

plot(handles.SGG2,ALST,'DisplayName','VUELTA','YDataSource',...
'VUELTA');figure(gcf)
set(handles.SGG2,'xgrid','on','ygrid','on');
ylabel(handles.SGG2,'Altitud (m)');
title(handles.SGG2,'GráficaG.P.S. [Altitud(m)]',...
'FontSize',14,'FontName','MS Sans Serif','FontWeight','bold');
```

%GRÁFICA 3 (RECORRIDO ACTUAL DEL SENSOR G.P.S.)

```

axes(findobj('type','axes','tag','SGG3'));
plot3(LOP,LAP,ALP);
xlabel(handles.SGG3,'Longitud Grados(°)');
ylabel(handles.SGG3,'Latitud Grados(°)');
zlabel(handles.SGG3,'Altitud (m)');
title(handles.SGG3,'Gráfica G.P.S. [Recorrido3D]',...
'FontSize',14,'FontName','MS Sans Serif','FontWeight','bold');
```

end

% --- Executes on button press in leyendaGPS.

```

function leyendaGPS_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaGPS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaGPS

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA (ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    legend(handles.SGG1,'show');
    legend(handles.SGG1,'boxoff');
    legend(handles.SGG2,'show');
    legend(handles.SGG2,'boxoff');
    set(handles.leyendaGPS,'String','LEYENDA ON');
else
    legend(handles.SGG1,'off');
    legend(handles.SGG2,'off');
```



```

        set(handles.leyendaGPS, 'String', 'LEYENDA OFF');
end

% --- Executes on button press in cuadrículaGPS.
function cuadrículaGPS_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaGPS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of cuadrículaGPS

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA(ON,OFF)
o2=get(hObject, 'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SGG1, 'off');
    grid(handles.SGG2, 'off');
    set(handles.cuadrículaGPS, 'String', 'CUADRÍCULA OFF');
else
    grid(handles.SGG1, 'on');
    grid(handles.SGG2, 'on');
    set(handles.cuadrículaGPS, 'String', 'CUADRÍCULA ON');
end

% --- Executes on button press in MS.
function MS_Callback(hObject, eventdata, handles)
% hObject    handle to MS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideSesion;
close(handles.output);

% --- Executes on button press in TRES DGPS.
function TRES DGPS_Callback(hObject, eventdata, handles)
% hObject    handle to TRES DGPS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of TRES DGPS

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN 3D(ON,OFF)
o3=get(hObject, 'Value');
handles.o3=o3;

if handles.o3==1
    rotate3d(handles.SGG3, 'on');
    set(handles.TRES DGPS, 'String', '3D ON');
else
    rotate3d(handles.SGG3, 'off');
    set(handles.TRES DGPS, 'String', '3D OFF');
end
end

```



FUNCIÓN GUIDE COMPARAR SESIÓN

```

function varargout = guideScompararsesion(varargin)
% GUIDESCOMPARARSESION M-file for guideScompararsesion.fig
%   GUIDESCOMPARARSESION, by itself, creates a new
%   GUIDESCOMPARARSESION or raises the existing singleton*.
%
%   H = GUIDESCOMPARARSESION returns the handle to a new
%   GUIDESCOMPARARSESION or the handle to the existing singleton*.
%
%   GUIDESCOMPARARSESION('CALLBACK',hObject,eventData,handles,...)
%   calls the local function named CALLBACK in
%   GUIDESCOMPARARSESION.M with the given input arguments.
%
%   GUIDESCOMPARARSESION('Property','Value',...) creates a new
%   GUIDESCOMPARARSESION or raises the existing singleton*.Starting
%   from the left, property value pairs are applied to the GUI
%   before guideScompararsesion_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideScompararsesion_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
guideScompararsesion

% Last Modified by GUIDE v2.5 06-Jun-2014 14:57:36

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideScompararsesion_OpeningFcn,
                  ...
                  'gui_OutputFcn',  @guideScompararsesion_OutputFcn,
                  ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before guideScompararsesion is made visible.
function guideScompararsesion_OpeningFcn(hObject, eventdata, handles,
varargout)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

```



```

% varargin    command line arguments to guideScompararsesion (see
% VARARGIN)

% Choose default command line output for guideScompararsesion
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideScompararsesion wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideScompararsesion_OutputFcn(hObject,
eventdata, handles)
% varargout    cell array for returning output args (see VARARGOUT);
% hObject      handle to figure
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%PULSANDO BOTÓN COMPARAR SESIÓN(MENÚ SESIÓN)

%VARIABLES GLOBALES
global OPENP;
global OPENS;
global OPENCB;
global ST;

%APERTURA DE SESIÓN CARGADA
if((OPENCB~=1) || (OPENCB==1) || (OPENS==1) || (OPENP==1))

    %TABLA DATOS SESIÓN
    set(handles.SCSA, 'Data', ST);
end

% --- Executes on button press in AbrirSB.
function AbrirSB_Callback(hObject, eventdata, handles)
% hObject      handle to AbrirSB (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global STB;
global OPENCB;
global ONB;

%ASOCIACIÓN DE VALORES A VARIABLES
OPENCB=1;
ONB=1;

%LLAMADA A FUNCIÓN
abrirSB;

```



```
%TABLA DATOS SESIÓN B
set(handles.SCSB, 'Data', STB);

% --- Executes on button press in MS.
function MS_Callback(hObject, eventdata, handles)
% hObject    handle to MS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global OPENS;
global OPENCA;
global VOLVERS;
global SAVEST;
global VOLVERSON;

%ASOCIACIÓN DE VALORES A VARIABLES
VOLVERSON=1;
VOLVERS=1;

%VUELTA AL MENÚ SESIÓN
if(( (OPENCA==1) && (SAVEST==0) && (OPENS~=1) ))

    %LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE
    %DEL ACTUAL GUIDE
    guideSesion;

elseif(( (OPENCA==1) && (SAVEST==0) && (OPENS==1) ))

    %LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE
    %DEL ACTUAL GUIDE
    guideSesion;

else

    %LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE
    %DEL ACTUAL GUIDE
    guideSesion;
end

% --- Executes on button press in AbrirSA.
function AbrirSA_Callback(hObject, eventdata, handles)
% hObject    handle to AbrirSA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global ST;
global OPENCA;
global SAVEST;
global OPENP;
global OPENS;
```




```

%APERTURA DE SESIÓN
if ((OPENP==1) || (OPENS==1) || (OPENCA==1) || (SAVEST==1))
    OPENCA=1;

    %LLAMADA A FUNCIÓN
    abrirS;

    %TABLA DATOS SESIÓN
    set(handles.SCSA, 'Data', ST);

else

    %PREGUNTA PARA EL USUARIO
    resp=questdlg({'LOS DATOS DE LA SESIÓN CARGADA, AL NO SER...
    GUARDADOS, SE PERDERÁN.', ' ', '¿DESEA CONTINUAR SIN GUARDAR...
    SESIÓN?'}, 'INFORMACIÓN-SESIÓN', 'Si', 'No', 'No');

    if strcmp(resp, 'No')
        helpdlg({'1º-PARA GUARDAR LA SESIÓN CARGADA, PULSAR (MENÚ...
        SESIÓN) Y DESPUÉS (GUARDAR', ' ', ' SESIÓN).', ' ', '...',
        '2º-PARA SEGUIR COMPARANDO LA SESIÓN CARGADA, CONTINUAR...
        EN COMPARAR-SESIÓN.'}, 'AYUDA-GUARDAR-COMPARAR');
        return;
    else
        OPENCA=1;

        %LLAMADA A FUNCIÓN
        abrirS;

        %AVISO INFORMATIVO AL USUARIO
        helpdlg({'1º-LOS DATOS DE LA SESIÓN CARGADA SE PERDIERON,...
        SI QUIERE RECUPERARLOS, ', ' ', 'PULSAR (MENÚ SESIÓN),...
        PULSAR (MENÚ PRINCIPAL) Y VOLVER A CARGAR LA SESIÓN.', ' ', '...',
        '2º-PARA SEGUIR COMPARANDO LA SESIÓN ELEGIDA, CONTINUAR...
        EN COMPARAR-SESIÓN.'}, 'AYUDA-RECUPERAR-COMPARAR');

        %TABLA DATOS SESIÓN
        set(handles.SCSA, 'Data', ST);
    end
end

% --- Executes on selection change in MenuVariablesB.
function MenuVariablesB_Callback(hObject, eventdata, handles)
% hObject    handle to MenuVariablesB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
% MenuVariablesB contents as cell array
%    contents{get(hObject,'Value')} returns selected item from
% MenuVariablesB

%MENÚ COMPARAR VARIABLES SESIÓN(B)
opcion=get(handles.MenuVariablesB, 'Value');
set(handles.opVB, 'String', opcion);

```



```

switch opcion
    case 1 %Altitud (VS) Temperatura Ambiente
        guideCvariableBaltitudtempambiente;
    case 2 %Suspensión Delantera (VS) Suspensión Trasera
        guideCvariableBsuspDsuspT;
    case 3 %Suspensión Delantera (VS) Altitud
        guideCvariableBsuspDaltitud;
    case 4 %Suspensión Trasera (VS) Altitud
        guideCvariableBsuspTaltitud;
    case 5 %Velocidad (VS) Altitud
        guideCvariableBvelocidadAltitud;
    case 6 %Velocidad (VS) Suspensión Delantera
        guideCvariableBvelocidadSuspD;
    case 7 %Velocidad (VS) Suspensión Trasera
        guideCvariableBvelocidadSuspT;
    case 8 %Velocidad (VS) Temperatura Ambiente
        guideCvariableBvelocidadtempambiente;
end

% --- Executes during object creation, after setting all properties.
function MenuVariablesB_CreateFcn(hObject, eventdata, handles)
% hObject    handle to MenuVariablesB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function opVB_Callback(hObject, eventdata, handles)
% hObject    handle to opVB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of opVB as text
%        str2double(get(hObject,'String')) returns contents of opVB as
%        a double

% --- Executes during object creation, after setting all properties.
function opVB_CreateFcn(hObject, eventdata, handles)
% hObject    handle to opVB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```



```

% --- Executes on selection change in MenuVariablesA.
function MenuVariablesA_Callback(hObject, eventdata, handles)
% hObject    handle to MenuVariablesA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
% MenuVariablesA contents as cell array
%          contents{get(hObject,'Value')} returns selected item from
% MenuVariablesA

%MENÚ VARIABLES SESIÓN(A)
opcion=get(handles.MenuVariablesA,'Value');
set(handles.opVA,'String',opcion);

switch opcion
case 1 %Altitud (VS) Temperatura Ambiente
    guideCvariableAaltitudtempambiente;
case 2 %Suspensión Delantera (VS) Suspensión Trasera
    guideCvariableAsuspDsuspT;
case 3 %Suspensión Delantera (VS) Altitud
    guideCvariableAsuspDaltitud;
case 4 %Suspensión Trasera (VS) Altitud
    guideCvariableAsuspTaltitud;
case 5 %Velocidad (VS) Altitud
    guideCvariableAvelocidadaltitud;
case 6 %Velocidad (VS) Suspensión Delantera
    guideCvariableAvelocidadsuspD;
case 7 %Velocidad (VS) Suspensión Trasera
    guideCvariableAvelocidadsuspT;
case 8 %Velocidad (VS) Temperatura Ambiente
    guideCvariableAvelocidadtempambiente;
end

% --- Executes during object creation, after setting all properties.
function MenuVariablesA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to MenuVariablesA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function opVA_Callback(hObject, eventdata, handles)
% hObject    handle to opVA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of opVA as text
%        str2double(get(hObject,'String')) returns contents of opVA as
% a double

```



```

% --- Executes during object creation, after setting all properties.
function opVA_CreateFcn(hObject, eventdata, handles)
% hObject    handle to opVA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in MenuSensores.
function MenuSensores_Callback(hObject, eventdata, handles)
% hObject    handle to MenuSensores (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
% MenuSensores contents as cell array
%       contents{get(hObject,'Value')} returns selected item from
% MenuSensores

%MENÚ COMPARAR SENSORES SESIÓN(A) VS SESIÓN(B)
opcion=get(handles.MenuSensores,'Value');
set(handles.opS,'String',opcion);

switch opcion
    case 1 %Acelerómetro A (VS) Acelerómetro B
        guideCsensoracelerometro;
    case 2 %G.P.S. A (VS) G.P.S. B
        guideCsensorgps;
    case 3 %Suspensión Delantera A (VS) Suspensión Trasera B
        guideCsensorsuspdelantera;
    case 4 %Suspensión Trasera A (VS) Suspensión Trasera B
        guideCsensorsusptrasera;
    case 5 %Temperatura Ambiente A (VS) Temperatura Ambiente B
        guideCsensortempambiente;
end

% --- Executes during object creation, after setting all properties.
function MenuSensores_CreateFcn(hObject, eventdata, handles)
% hObject    handle to MenuSensores (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

```



```
function opS_Callback(hObject, eventdata, handles)
% hObject    handle to opS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of opS as text
%        str2double(get(hObject,'String')) returns contents of opS as
% a double

% --- Executes during object creation, after setting all properties.
function opS_CreateFcn(hObject, eventdata, handles)
% hObject    handle to opS (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```



FUNCIÓN GUIDE COMPARAR SENSORES TEMPERATURA AMBIENTE

```

function varargout = guideCsensortempambiente(varargin)
% GUIDECSENSORTEMPAMBIENTE M-file for guideCsensortempambiente.fig
%   GUIDECSENSORTEMPAMBIENTE, by itself, creates a new
%   GUIDECSENSORTEMPAMBIENTE or raises the existing singleton*.
%
%   H = GUIDECSENSORTEMPAMBIENTE returns the handle to a new
%   GUIDECSENSORTEMPAMBIENTE or the handle to the existing
%   singleton*.
%
%   GUIDECSENSORTEMPAMBIENTE('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECSENSORTEMPAMBIENTE.M with the given input arguments.
%
%   GUIDECSENSORTEMPAMBIENTE('Property','Value',...) creates a new
%   GUIDECSENSORTEMPAMBIENTE or raises the existing singleton*.
%   Starting from the left, property value pairs are applied to the
%   GUI before guideCsensortempambiente_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCsensortempambiente_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCsensortempambiente

% Last Modified by GUIDE v2.5 28-Aug-2014 18:56:56

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCsensortempambiente_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCsensortempambiente_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

```



```

% --- Executes just before guideCsensortempambiente is made visible.
function guideCsensortempambiente_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin    command line arguments to guideCsensortempambiente (see
% VARARGIN)

% Choose default command line output for guideCsensortempambiente
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCsensortempambiente wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCsensortempambiente_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
global TAST; global TAOPEN; global TAOPENB;
global OPENCA; global OPENCB; global OPENP; global OPENS;
global OLTA; global OCTA;
global ONB;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENCA==1) || ((OPENS==1) && (OPENCA==0)) || ((OPENP==1) && (OPENCA==0)))

    %TABLA A (DATOS GUARDADOS DEL SENSOR TEMPERATURA SESIÓN A)
    set(handles.SCTATA, 'Data', TAOPEN);

    %GRÁFICA A (DATOS GUARDADOS DEL SENSOR TEMPERATURA SESIÓN A)
    plot(handles.SCTAGA, TAOPEN, 'DisplayName', 'OPEN(A) ...
VUELTA', 'YDataSource', 'OPEN(A) VUELTA'); figure(gcf)
    set(handles.SCTAGA, 'xgrid', 'on', 'ygrid', 'on');
    xlabel(handles.SCTAGA, 'Número de Muestras');
    ylabel(handles.SCTAGA, 'Temperatura Ambiente (C°)');
    title(handles.SCTAGA, 'Gráfica Temperatura Ambiente [(C°)] (A)', ...
'FontSize', 12, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');

elseif((OPENCA==0))

    %TABLA A (DATOS ACTUALES DEL SENSOR TEMPERATURA SESIÓN A)
    set(handles.SCTATA, 'Data', TAST);

```




```

%GRÁFICA A (DATOS ACTUALES DEL SENSOR TEMPERATURA SESIÓN A)
plot(handles.SCTAGA,TAST,'DisplayName','(A)VUELTA',...
'YDataSource','(A)VUELTA');figure(gcf)
set(handles.SCTAGA,'xgrid','on','ygrid','on');
xlabel(handles.SCTAGA,'Número de Muestras');
ylabel(handles.SCTAGA,'Temperatura Ambiente (C°)');
title(handles.SCTAGA,'Gráfica Temperatura Ambiente [(C°)] (A)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

if((OPENCB==1) && (ONB==1))

%TABLA B (DATOS GUARDADOS DEL SENSOR TEMPERATURA SESIÓN B)
set(handles.SCTATB,'Data',TAOPENB);

%GRÁFICA B (DATOS GUARDADOS DEL SENSOR TEMPERATURA SESIÓN B)
plot(handles.SCTAGB,TAOPENB,'DisplayName','OPEN(B)VUELTA',...
'YDataSource','OPEN(B)VUELTA');figure(gcf)
set(handles.SCTAGB,'xgrid','on','ygrid','on');
xlabel(handles.SCTAGB,'Número de Muestras');
ylabel(handles.SCTAGB,'Temperatura Ambiente (C°)');
title(handles.SCTAGB,'Gráfica Temperatura Ambiente [(C°)] (B)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

%APARECE LA PANTALLA CON EL ESTADO QUE TENÍA LA LEYENDA Y LA
%CUADRÍCULA ANTERIORMENTE
if(OLTA==1)
    legend(handles.SCTAGA,'show');
    legend(handles.SCTAGA,'boxoff');
    legend(handles.SCTAGB,'show');
    legend(handles.SCTAGB,'boxoff');
end

if(OCTA==1)
    grid(handles.SCTAGA,'off');
    grid(handles.SCTAGB,'off');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global VOLVERTAC;
VOLVERTAC=1;

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
guideScompararsesion;

```



```
% --- Executes on button press in leyendaCTA.
function leyendaCTA_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCTA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCTA

%VARIABLES GLOBALES
global OLTa;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    legend(handles.SCTAGA,'show');
    legend(handles.SCTAGA,'boxoff');
    legend(handles.SCTAGB,'show');
    legend(handles.SCTAGB,'boxoff');
    set(handles.leyendaCTA,'String','LEYENDA ON');
else
    legend(handles.SCTAGA,'off');
    legend(handles.SCTAGB,'off');
    set(handles.leyendaCTA,'String','LEYENDA OFF');
end

OLTa=o;

% --- Executes on button press in cuadrículaCTA.
function cuadrículaCTA_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaCTA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of cuadrículaCTA

%VARIABLES GLOBALES
global OCTa;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA(ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCTAGA,'off');
    grid(handles.SCTAGB,'off');
    set(handles.cuadrículaCTA,'String','CUADRÍCULA OFF');
else
    grid(handles.SCTAGA,'on');
    grid(handles.SCTAGB,'on');
    set(handles.cuadrículaCTA,'String','CUADRÍCULA ON');
end

OCTa=o2;
```



FUNCIÓN GUIDE COMPARAR SENSORES SUSPENSIÓN DELANTERA

```

function varargout = guideCsensorsuspdelantera(varargin)
% GUIDECSENSORSUSPDELANTERA M-file for guideCsensorsuspdelantera.fig
%   GUIDECSENSORSUSPDELANTERA, by itself, creates a new
%   GUIDECSENSORSUSPDELANTERA or raises the existing singleton*.
%
%   H = GUIDECSENSORSUSPDELANTERA returns the handle to a new
%   GUIDECSENSORSUSPDELANTERA or the handle to the existing
%   singleton*.
%
%   GUIDECSENSORSUSPDELANTERA('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECSENSORSUSPDELANTERA.M with the given input arguments.
%
%   GUIDECSENSORSUSPDELANTERA('Property','Value',...) creates a new
%   GUIDECSENSORSUSPDELANTERA or raises the existing singleton*.
%   Starting from the left, property value pairs are applied to the
%   GUI before guideCsensorsuspdelantera_OpeningFcn gets called.
%   An unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCsensorsuspdelantera_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCsensorsuspdelantera

% Last Modified by GUIDE v2.5 02-Sep-2014 19:52:59

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCsensorsuspdelantera_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCsensorsuspdelantera_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before guideCsensorsuspdelantera is made visible.
function guideCsensorsuspdelantera_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB

```



```

% handles      structure with handles and user data (see GUIDATA)
% varargin     command line arguments to guideCsensorsuspdelantera (see
% VARARGIN)

% Choose default command line output for guideCsensorsuspdelantera
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCsensorsuspdelantera wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCsensorsuspdelantera_OutputFcn(hObject,
eventdata, handles)
% varargout    cell array for returning output args (see VARARGOUT);
% hObject     handle to figure
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
global SUSPDST; global SUSPDOPEN; global SUSPDOPENB;
global OPENCA; global OPENCB; global OPENP; global OPENS;
global OLSD; global OCSD;
global ONB;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENCA==1) || ((OPENS==1) && (OPENCA==0)) || ((OPENP==1) && (OPENCA==0)))

    %TABLA A (DATOS GUARDADOS DEL SENSOR SUSPENSIÓN DELANTERA
    %SESIÓN A)
    set(handles.SCSDTA, 'Data', SUSPDOPEN);

    %GRÁFICA A (DATOS GUARDADOS DEL SENSOR SUSPENSIÓN DELANTERA
    %SESIÓN A)
    plot(handles.SCSDGA, SUSPDOPEN, 'DisplayName', 'OPEN (A)VUELTA', ...
'YDataSource', 'OPEN(A) VUELTA'); figure(gcf)
    set(handles.SCSDGA, 'xgrid', 'on', 'ygrid', 'on');
    xlabel(handles.SCSDGA, 'Número de Muestras');
    ylabel(handles.SCSDGA, 'Longitud (mm)');
    title(handles.SCSDGA, 'Gráfica Suspensión Delantera [(mm)] (A)', ...
'FontSize', 12, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');

elseif((OPENCA==0))

    %TABLA A (DATOS ACTUALES DEL SENSOR SUSPENSIÓN DELANTERA SESIÓN A)
    set(handles.SCSDTA, 'Data', SUSPDST);

    %GRÁFICA A (DATOS ACTUALES DEL SENSOR SUSPENSIÓN DELANTERA
    %SESIÓN A)
    plot(handles.SCSDGA, SUSPDST, 'DisplayName', 'VUELTA', ...
'YDataSource', 'VUELTA'); figure(gcf)
    set(handles.SCSDGA, 'xgrid', 'on', 'ygrid', 'on');

```



```

xlabel(handles.SCSDDGA,'Número de Muestras');
ylabel(handles.SCSDDGA,'Longitud (mm)');
title(handles.SCSDDGA,'Gráfica Suspensión Delantera [(mm)] (A)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

if((OPENCB==1) && (ONB==1))

%TABLA B (DATOS GUARDADOS DEL SENSOR SUSPENSIÓN DELANTERA
%SESIÓN B)
set(handles.SCSDDTB,'Data',SUSPDOPENB);

%GRÁFICA B (DATOS GUARDADOS DEL SENSOR SUSPENSIÓN DELANTERA
%SESIÓN B)
plot(handles.SCSDDGB,SUSPDOPENB,'DisplayName','OPEN(B) VUELTA',...
'YDataSource','OPEN(B) VUELTA');figure(gcf)
set(handles.SCSDDGB,'xgrid','on','ygrid','on');
xlabel(handles.SCSDDGB,'Número de Muestras');
ylabel(handles.SCSDDGB,'Longitud (mm)');
title(handles.SCSDDGB,'Gráfica Suspensión Delantera [(mm)] (B)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

%APARECE LA PANTALLA CON EL ESTADO QUE TENÍA LA LEYENDA Y LA
%CUADRÍCULA ANTERIORMENTE
if(OLSD==1)
legend(handles.SCSDDGA,'show');
legend(handles.SCSDDGA,'boxoff');
legend(handles.SCSDDGB,'show');
legend(handles.SCSDDGB,'boxoff');
end

if(OCSD==1)
grid(handles.SCSDDGA,'off');
grid(handles.SCSDDGB,'off');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global VOLVERSDC;
VOLVERSDC=1;

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
guideScompararsesion;

% --- Executes on button press in leyendaCSD.
function leyendaCSD_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCSD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCSD

```



```
%VARIABLES GLOBALES
global OLSD;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA (ON, OFF)
o=get(hObject, 'Value');
handles.o=o;

if handles.o==1
    legend(handles.SCSDGA, 'show');
    legend(handles.SCSDGA, 'boxoff');
    legend(handles.SCSDGB, 'show');
    legend(handles.SCSDGB, 'boxoff');
    set(handles.leyendaCSD, 'String', 'LEYENDA ON');
else
    legend(handles.SCSDGA, 'off');
    legend(handles.SCSDGB, 'off');
    set(handles.leyendaCSD, 'String', 'LEYENDA OFF');
end

OLSD=o;

% --- Executes on button press in cuadrículaCSD.
function cuadrículaCSD_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaCSD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of cuadrículaCSD

%VARIABLES GLOBALES
global OCSD;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA (ON, OFF)
o2=get(hObject, 'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCSDGA, 'off');
    grid(handles.SCSDGB, 'off');
    set(handles.cuadrículaCSD, 'String', 'CUADRÍCULA OFF');
else
    grid(handles.SCSDGA, 'on');
    grid(handles.SCSDGB, 'on');
    set(handles.cuadrículaCSD, 'String', 'CUADRÍCULA ON');
end

OCSD=o2;
```



FUNCIÓN GUIDE COMPARAR SENSORES SUSPENSIÓN TRASERA

```

function varargout = guideCsensorsusptrasera(varargin)
% GUIDECSENSORSUSPTRASERA M-file for guideCsensorsusptrasera.fig
%   GUIDECSENSORSUSPTRASERA, by itself, creates a new
%   GUIDECSENSORSUSPTRASERA or raises the existing singleton*.
%
%   H = GUIDECSENSORSUSPTRASERA returns the handle to a new
%   GUIDECSENSORSUSPTRASERA or the handle to the existing
%   singleton*.
%
%   GUIDECSENSORSUSPTRASERA('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECSENSORSUSPTRASERA.M with the given input arguments.
%
%   GUIDECSENSORSUSPTRASERA('Property','Value',...) creates a new
%   GUIDECSENSORSUSPTRASERA or raises the existing singleton*.
%   Starting from the left, property value pairs are applied to the
%   GUI before guideCsensorsusptrasera_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCsensorsusptrasera_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCsensorsusptrasera

% Last Modified by GUIDE v2.5 29-Aug-2014 12:19:10

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCsensorsusptrasera_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCsensorsusptrasera_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

```




```

% --- Executes just before guideCsensorsusptrasera is made visible.
function guideCsensorsusptrasera_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideCsensorsusptrasera (see
% VARARGIN)

% Choose default command line output for guideCsensorsusptrasera
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCsensorsusptrasera wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCsensorsusptrasera_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
global SUSPTST; global SUSPTOPEN; global SUSPTOPENB;
global OPENCA; global OPENCB; global OPENP; global OPENS;
global OLST; global OCST;
global ONB;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENCA==1) || ((OPENS==1) && (OPENCA==0)) || ((OPENP==1) && (OPENCA==0)))

    %TABLA A (DATOS GUARDADOS DEL SENSOR SUSPENSIÓN TRASERA SESIÓN A)
    set(handles.SCSTTA, 'Data', SUSPTOPEN);

    %GRÁFICA A (DATOS GUARDADOS DEL SENSOR SUSPENSIÓN TRASERA
    %SESIÓN A)
    plot(handles.SCSTGA, SUSPTOPEN, 'DisplayName', 'OPEN(A) VUELTA', ...
'YDataSource', 'OPEN(A) VUELTA'); figure(gcf)
    set(handles.SCSTGA, 'xgrid', 'on', 'ygrid', 'on');
    xlabel(handles.SCSTGA, 'Número de Muestras');
    ylabel(handles.SCSTGA, 'Suspensión Trasera (mm)');
    title(handles.SCSTGA, 'Gráfica Suspensión Trasera [(mm)] (A)', ...
'FontSize', 12, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');

elseif((OPENCA==0))

    %TABLA A (DATOS ACTUALES DEL SENSOR SUSPENSIÓN TRASERA SESIÓN A)
    set(handles.SCSTTA, 'Data', SUSPTST);

```



```

%GRÁFICA A (DATOS ACTUALES DEL SENSOR SUSPENSIÓN TRASERA SESIÓN A)
plot(handles.SCSTGA,SUSPTST,'DisplayName','VUELTA',...
'YDataSource','VUELTA');figure(gcf)
set(handles.SCSTGA,'xgrid','on','ygrid','on');
xlabel(handles.SCSTGA,'Número de Muestras');
ylabel(handles.SCSTGA,'Suspensión Trasera (mm)');
title(handles.SCSTGA,'Gráfica Suspensión Trasera [(mm)] (A)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

if((OPENCB==1) && (ONB==1))

%TABLA B (DATOS GUARDADOS DEL SENSOR SUSPENSIÓN TRASERA SESIÓN B)
set(handles.SCSTTB,'Data',SUSPTOPENB);

%GRÁFICA B (DATOS GUARDADOS DEL SENSOR SUSPENSIÓN TRASERA
%SESIÓN B)
plot(handles.SCSTGB,SUSPTOPENB,'DisplayName','OPEN(B) VUELTA',...
'YDataSource','OPEN(B) VUELTA');figure(gcf)
set(handles.SCSTGB,'xgrid','on','ygrid','on');
xlabel(handles.SCSTGB,'Número de Muestras');
ylabel(handles.SCSTGB,'Suspensión Trasera (mm)');
title(handles.SCSTGB,'Gráfica Suspensión Trasera [(mm)] (B)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

%APARECE LA PANTALLA CON EL ESTADO QUE TENÍA LA LEYENDA Y LA
%CUADRÍCULA ANTERIORMENTE
if(OLST==1)
    legend(handles.SCSTGA,'show');
    legend(handles.SCSTGA,'boxoff');
    legend(handles.SCSTGB,'show');
    legend(handles.SCSTGB,'boxoff');
end

if(OCST==1)
    grid(handles.SCSTGA,'off');
    grid(handles.SCSTGB,'off');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global VOLVERSTC;
VOLVERSTC=1;

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
guideScompararsesion;

```



```
% --- Executes on button press in leyendaCST.
function leyendaCST_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCST (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCST

%VARIABLES GLOBALES
global OLST;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    legend(handles.SCSTGA,'show');
    legend(handles.SCSTGA,'boxoff');
    legend(handles.SCSTGB,'show');
    legend(handles.SCSTGB,'boxoff');
    set(handles.leyendaCST,'String','LEYENDA ON');
else
    legend(handles.SCSTGA,'off');
    legend(handles.SCSTGB,'off');
    set(handles.leyendaCST,'String','LEYENDA OFF');
end

OLST=o;

% --- Executes on button press in cuadrículaCST.
function cuadrículaCST_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaCST (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of cuadrículaCST

%VARIABLES GLOBALES
global OCST;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA(ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCSTGA,'off');
    grid(handles.SCSTGB,'off');
    set(handles.cuadrículaCST,'String','CUADRICULA OFF');
else
    grid(handles.SCSTGA,'on');
    grid(handles.SCSTGB,'on');
    set(handles.cuadrículaCST,'String','CUADRICULA ON');
end

OCST=o2;
```



FUNCIÓN GUIDE COMPARAR SENSORES ACELERÓMETRO

```

function varargout = guideCsensoracelerometro(varargin)
% GUIDECSENSORACELEROMETRO M-file for guideCsensoracelerometro.fig
%   GUIDECSENSORACELEROMETRO, by itself, creates a new
%   GUIDECSENSORACELEROMETRO or raises the existing singleton*.
%
%   H = GUIDECSENSORACELEROMETRO returns the handle to a new
%   GUIDECSENSORACELEROMETRO or the handle to the existing
%   singleton*.
%
%   GUIDECSENSORACELEROMETRO('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECSENSORACELEROMETRO.M with the given input arguments.
%
%   GUIDECSENSORACELEROMETRO('Property','Value',...) creates a new
%   GUIDECSENSORACELEROMETRO or raises the existing singleton*.
%   Starting from the left, property value pairs are applied to the
%   GUI before guideCsensoracelerometro_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCsensoracelerometro_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCsensoracelerometro

% Last Modified by GUIDE v2.5 02-Sep-2014 19:51:36

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCsensoracelerometro_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCsensoracelerometro_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

```



```

% --- Executes just before guideCsensoracelerometro is made visible.
function guideCsensoracelerometro_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideCsensoracelerometro (see
% VARARGIN)

% Choose default command line output for guideCsensoracelerometro
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCsensoracelerometro wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCsensoracelerometro_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
global AST; global AOPEN; global AOPENB;
global OPENCA; global OPENCB; global OPENP; global OPENS;
global OLA; global OCA;
global ONB;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENCA==1) || ((OPENS==1) && (OPENCA==0)) || ((OPENP==1) && (OPENCA==0)))

    %TABLA A (DATOS GUARDADOS DEL SENSOR ACELERÓMETRO SESIÓN A)
    set(handles.SCATA, 'Data', AOPEN);

    %GRÁFICA A (DATOS GUARDADOS DEL SENSOR ACELERÓMETRO SESIÓN A)
    plot(handles.SCAGA, AOPEN, 'DisplayName', 'OPEN(A) VUELTA', ...
        'YDataSource', 'OPEN(A) VUELTA'); figure(gcf)
    set(handles.SCAGA, 'xgrid', 'on', 'ygrid', 'on');
    xlabel(handles.SCAGA, 'Número de Muestras');
    ylabel(handles.SCAGA, 'Acelerómetro (A°)');
    title(handles.SCAGA, 'Gráfica Acelerómetro [(A°)] (A)', ...
        'FontSize', 12, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');

elseif((OPENCA==0))

    %TABLA A (DATOS ACTUALES DEL SENSOR ACELERÓMETRO SESIÓN A)
    set(handles.SCATA, 'Data', AST);

```



```

%GRÁFICA A (DATOS ACTUALES DEL SENSOR ACELERÓMETRO SESIÓN A)
plot(handles.SCAGA,AST,'DisplayName','VUELTA',...
'YDataSource','VUELTA');figure(gcf)
set(handles.SCAGA,'xgrid','on','ygrid','on');
xlabel(handles.SCAGA,'Número de Muestras');
ylabel(handles.SCAGA,'Acelerómetro (A°)');
title(handles.SCAGA,'Gráfica Acelerómetro [(A°)] (A)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

if((OPENCB==1) && (ONB==1))

%TABLA B (DATOS GUARDADOS DEL SENSOR ACELERÓMETRO SESIÓN B)
set(handles.SCATB,'Data',AOPENB);

%GRÁFICA B (DATOS DEL SENSOR ACELERÓMETRO SESIÓN B)
plot(handles.SCAGB,AOPENB,'DisplayName','OPEN(B) VUELTA',...
'YDataSource','OPEN(B) VUELTA');figure(gcf)
set(handles.SCAGB,'xgrid','on','ygrid','on');
xlabel(handles.SCAGB,'Número de Muestras');
ylabel(handles.SCAGB,'Acelerómetro (A°)');
title(handles.SCAGB,'Gráfica Acelerómetro [(A°)] (B)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

%APARECE LA PANTALLA CON EL ESTADO QUE TENÍA LA LEYENDA Y LA
%CUADRÍCULA ANTERIORMENTE
if(OLA==1)
    legend(handles.SCAGA,'show');
    legend(handles.SCAGA,'boxoff');
    legend(handles.SCAGB,'show');
    legend(handles.SCAGB,'boxoff');
end

if(OCA==1)
    grid(handles.SCAGA,'off');
    grid(handles.SCAGB,'off');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global VOLVERAC;
VOLVERAC=1;

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
guideScompararsesion;

```



```
% --- Executes on button press in leyendaCA.
function leyendaCA_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCA

%VARIABLES GLOBALES
global OLA;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    legend(handles.SCAGA,'show');
    legend(handles.SCAGA,'boxoff');
    legend(handles.SCAGB,'show');
    legend(handles.SCAGB,'boxoff');
    set(handles.leyendaCA,'String','LEYENDA ON');
else
    legend(handles.SCAGA,'off');
    legend(handles.SCAGB,'off');
    set(handles.leyendaCA,'String','LEYENDA OFF');
end

OLA=o;

% --- Executes on button press in cuadrículaCA.
function cuadrículaCA_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaCA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of cuadrículaCA

%VARIABLES GLOBALES
global OCA;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA(ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCAGA,'off');
    grid(handles.SCAGB,'off');
    set(handles.cuadrículaCA,'String','CUADRÍCULA OFF');
else
    grid(handles.SCAGA,'on');
    grid(handles.SCAGB,'on');
    set(handles.cuadrículaCA,'String','CUADRÍCULA ON');
end

OCA=o2;
```



FUNCIÓN GUIDE COMPARAR SENSORES G.P.S.

```

function varargout = guideCsensorgps(varargin)
% GUIDECSENSORGPS M-file for guideCsensorgps.fig
%   GUIDECSENSORGPS, by itself, creates a new GUIDECSENSORGPS or
%   raises the existing singleton*.
%
%   H = GUIDECSENSORGPS returns the handle to a new GUIDECSENSORGPS
%   or the handle to the existing singleton*.
%
%   GUIDECSENSORGPS('CALLBACK',hObject,eventData,handles,...) calls
%   the local function named CALLBACK in GUIDECSENSORGPS.M with the
%   given input arguments.
%
%   GUIDECSENSORGPS('Property','Value',...) creates a new
%   GUIDECSENSORGPS or raises the existing singleton*. Starting
%   from the left, property value pairs are applied to the GUI
%   before guideCsensorgps_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCsensorgps_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help guideCsensorgps

% Last Modified by GUIDE v2.5 02-Sep-2014 19:52:27

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCsensorgps_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCsensorgps_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before guideCsensorgps is made visible.
function guideCsensorgps_OpeningFcn(hObject, eventdata, handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideCsensorgps (see VARARGIN)

% Choose default command line output for guideCsensorgps
handles.output = hObject;

```




```

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCsensorsgps wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCsensorsgps_OutputFcn(hObject, eventdata,
handles)
% varargout    cell array for returning output args (see VARARGOUT);
% hObject     handle to figure
% eventdata   reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
global GPSST; global GPSOPEN; global GPSOPENB;
global VST; global VOPEN; global VOPENB;
global OPENCA; global OPENCB; global OPENP; global OPENS;
global OLGP; global OCGP;
global ONB;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENCA==1) || ((OPENS==1) && (OPENCA==0)) || ((OPENP==1) && (OPENCA==0)))

    %TABLA A (DATOS GUARDADOS DEL SENSOR G.P.S. SESIÓN A)
    set(handles.SCGTA, 'Data', GPSOPEN);

    %GRÁFICA A (DATOS GUARDADOS DEL SENSOR G.P.S. SESIÓN A)
    plot(handles.SCGGA, VOPEN, 'DisplayName', 'OPEN(A) VUELTA', ...
'YDataSource', 'OPEN(A) VUELTA'); figure(gcf)
set(handles.SCGGA, 'xgrid', 'on', 'ygrid', 'on');
xlabel(handles.SCGGA, 'Número de Muestras');
ylabel(handles.SCGGA, 'Velocidad (Km/h)');
title(handles.SCGGA, 'Gráfica G.P.S. [Velocidad(Km/h)] (A)', ...
'FontSize', 12, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');

elseif((OPENCA==0))

    %TABLA A (DATOS ACTUALES DEL SENSOR G.P.S. SESIÓN A)
    set(handles.SCGTA, 'Data', GPSST);

    %GRÁFICA A (DATOS ACTUALES DEL SENSOR G.P.S. SESIÓN A)
    plot(handles.SCGGA, VST, 'DisplayName', 'VUELTA', ...
'YDataSource', 'VUELTA'); figure(gcf)
set(handles.SCGGA, 'xgrid', 'on', 'ygrid', 'on');
xlabel(handles.SCGGA, 'Número de Muestras');
ylabel(handles.SCGGA, 'Velocidad (Km/h)');
title(handles.SCGGA, 'Gráfica G.P.S. [Velocidad(Km/h)] (A)', ...
'FontSize', 12, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');
end

if((OPENCB==1) && (ONB==1))

    %TABLA B (DATOS GUARDADOS DEL SENSOR G.P.S. SESIÓN B)
    set(handles.SCGTB, 'Data', GPSOPENB);

```



```

%GRÁFICA B (DATOS GUARDADOS DEL SENSOR G.P.S. SESIÓN B)
plot(handles.SCGGB,VOOPENB,'DisplayName','OPEN(B) VUELTA',...
'YDataSource','OPEN(B) VUELTA');figure(gcf)
set(handles.SCGGB,'xgrid','on','ygrid','on');
xlabel(handles.SCGGB,'Número de Muestras');
ylabel(handles.SCGGB,'Velocidad (Km/h)');
title(handles.SCGGB,'Gráfica G.P.S. [Velocidad(Km/h)] (B)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

%APARECE LA PANTALLA CON EL ESTADO QUE TENÍA LA LEYENDA Y LA
%CUADRÍCULA ANTERIORMENTE
if (OLGP==1)
    legend(handles.SCGGA,'show');
    legend(handles.SCGGA,'boxoff');
    legend(handles.SCGGB,'show');
    legend(handles.SCGGB,'boxoff');
end

if (OCGP==1)
    grid(handles.SCGGA,'off');
    grid(handles.SCGGB,'off');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%VARIABLES GLOBALES
global VOLVERGPSC;
VOLVERGPSC=1;

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA)
guideScompararsesion;

% --- Executes on button press in leyendaCG.
function leyendaCG_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCG (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCG

%VARIABLES GLOBALES
global OLGP;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    legend(handles.SCGGA,'show');
    legend(handles.SCGGA,'boxoff');
    legend(handles.SCGGB,'show');
    legend(handles.SCGGB,'boxoff');
    set(handles.leyendaCG,'String','LEYENDA ON');

```



```
else
    legend(handles.SCGGA, 'off');
    legend(handles.SCGGB, 'off');
    set(handles.leyendaCG, 'String', 'LEYENDA OFF');
end

OLGP=o;

% --- Executes on button press in cuadrículaCG.
function cuadrículaCG_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaCG (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of cuadrículaCG

%VARIABLES GLOBALES
global OCGP;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA(ON,OFF)
o2=get(hObject, 'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCGGA, 'off');
    grid(handles.SCGGB, 'off');
    set(handles.cuadrículaCG, 'String', 'CUADRÍCULA OFF');
else
    grid(handles.SCGGA, 'on');
    grid(handles.SCGGB, 'on');
    set(handles.cuadrículaCG, 'String', 'CUADRÍCULA ON');
end

OCGP=o2;
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (A)

ALTITUD-TEMPERATURA AMBIENTE

```

function varargout = guideCvariableAaltitudtempambiente(varargin)
% GUIDECVARIABLEAALTITUDTEMPAMBIENTE M-file for
% guideCvariableAaltitudtempambiente.fig
%
%   GUIDECVARIABLEAALTITUDTEMPAMBIENTE, by itself, creates a new
%   GUIDECVARIABLEAALTITUDTEMPAMBIENTE or raises the existing
%   singleton*.
%
%   H = GUIDECVARIABLEAALTITUDTEMPAMBIENTE returns the handle to a
%   new GUIDECVARIABLEAALTITUDTEMPAMBIENTE or the handle to the
%   existing singleton*.
%
%   GUIDECVARIABLEAALTITUDTEMPAMBIENTE('CALLBACK',hObject,
%   eventData, handles,...) calls the local function named CALLBACK
%   in GUIDECVARIABLEAALTITUDTEMPAMBIENTE.M with the given input
%   arguments.
%
%   GUIDECVARIABLEAALTITUDTEMPAMBIENTE('Property','Value',...)
%   creates a new GUIDECVARIABLEAALTITUDTEMPAMBIENTE or raises the
%   existing singleton*. Starting from the left, property value
%   pairs are applied to the GUI before
%   guideCvariableAaltitudtempambiente_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableAaltitudtempambiente_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableAaltitudtempambiente

% Last Modified by GUIDE v2.5 03-Sep-2014 17:48:10

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCvariableAaltitudtempambiente_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCvariableAaltitudtempambiente_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

```



```

% --- Executes just before guideCvariableAaltitudtempambiente is made
% visible.
function guideCvariableAaltitudtempambiente_OpeningFcn(hObject,
eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to
% guideCvariableAaltitudtempambiente (see VARARGIN)

% Choose default command line output for
% guideCvariableAaltitudtempambiente
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableAaltitudtempambiente wait for user
% response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout =
guideCvariableAaltitudtempambiente_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global ALST; global ALOPENI;
global TAST; global TAOPEN;
global OPENCA; global OPENS; global OPENP;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if((OPENCA==1) || ((OPENS==1) && (OPENCA==0)) || ((OPENP==1) && (OPENCA==0)))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (A): SENSOR G.P.S. (ALTITUD) Y
    %SENSOR TEMPERATURA AMBIENTE)
    set(handles.SCVAALT1, 'Data', ALOPENI);
    set(handles.SCVATAT2, 'Data', TAOPEN);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (A): ALTITUD (VS)
    %TEMPERATURA AMBIENTE)
    size(ALOPENI);
    tam1=size(ALOPENI);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,ALOPENI,x,TAOPEN);
    set(H1, 'DisplayName', 'OPEN ALTITUD VUELTA', 'YDataSource', ...
    'OPEN ALTITUD VUELTA', 'LineStyle', '-');

```



```

set (H2, 'DisplayName', 'OPEN TEMPERATURA VUELTA', 'YDataSource', ...
'OPEN TEMPERATURA VUELTA', 'LineStyle', '--');
set (get (AX (1), 'Ylabel'), 'String', 'Altitud (m)');
set (get (AX (2), 'Ylabel'), 'String', 'Temperatura Ambiente [T(C°)]');
set (handles.SCVAAALTAG1, 'xgrid', 'on', 'ygrid', 'on');
xlabel (handles.SCVAAALTAG1, 'Número de Muestras');
title (handles.SCVAAALTAG1, 'Gráfica Variables (A):...
Altitud (m) (VS) Temperatura Ambiente [T(C°)]', ...
'FontSize', 12, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');

elseif ((OPENCA==0))

%TABLA 1 Y 2 (DATOS ACTUALES SESIÓN (A): SENSOR G.P.S. (ALTITUD) Y
%SENSOR TEMPERATURA AMBIENTE)
set (handles.SCVAAALT1, 'Data', ALST);
set (handles.SCVATAT2, 'Data', TAST);

%GRÁFICA 1 (VARIABLES ACTUALES (A): ALTITUD (VS) TEMPERATURA
%AMBIENTE)
size (ALST);
tam1=size (ALST);
tamx=tam1 (:, 1);
x=1:tamx;
[AX, H1, H2] = plotyy (x, ALST, x, TAST);
set (H1, 'DisplayName', 'ALTITUD VUELTA', 'YDataSource', ...
'ALTITUD VUELTA', 'LineStyle', '-');
set (H2, 'DisplayName', 'TEMPERATURA VUELTA', 'YDataSource', ...
'TEMPERATURA VUELTA', 'LineStyle', '--');
set (get (AX (1), 'Ylabel'), 'String', 'Altitud (m)');
set (get (AX (2), 'Ylabel'), 'String', 'Temperatura Ambiente [T(C°)]');
set (handles.SCVAAALTAG1, 'xgrid', 'on', 'ygrid', 'on');
xlabel (handles.SCVAAALTAG1, 'Número de Muestras');
title (handles.SCVAAALTAG1, 'Gráfica Variables (A):...
Altitud (m) (VS) Temperatura Ambiente [T(C°)]', ...
'FontSize', 12, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject      handle to MC (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close (handles.output);

% --- Executes on button press in leyendaCVAALTvSTA.
function leyendaCVAALTvSTA_Callback(hObject, eventdata, handles)
% hObject      handle to leyendaCVAALTvSTA (see GCBO)
% eventdata    reserved - to be defined in a future version of MATLAB
% handles      structure with handles and user data (see GUIDATA)

% Hint: get(hObject, 'Value') returns toggle state of leyendaCVAALTvSTA

```



```
%VARIABLES GLOBALES
global H1;
global H2;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set(legend(H1));
    set(legend(H2));
    legend(handles.SCVAALTAG1,'boxoff');
    set(handles.leyendaCVAALTVsTA,'String','LEYENDA ON');
else
    legend(handles.SCVAALTAG1,'off');
    delete(legend(H2));
    set(handles.leyendaCVAALTVsTA,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrículaCVAALTVsTA.
function cuadrículaCVAALTVsTA_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaCVAALTVsTA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% cuadrículaCVAALTVsTA

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA(ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVAALTAG1,'off');
    set(handles.cuadrículaCVAALTVsTA,'String','CUADRÍCULA OFF');
else
    grid(handles.SCVAALTAG1,'on');
    set(handles.cuadrículaCVAALTVsTA,'String','CUADRÍCULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (A) SUSPENSIÓN DELANTERA-ALTITUD

```

function varargout = guideCvariableAsuspDaltitud(varargin)
% GUIDECVARIABLEASUSPDALTITUD M-file for
% guideCvariableAsuspDaltitud.fig
%
%   GUIDECVARIABLEASUSPDALTITUD, by itself, creates a new
%   GUIDECVARIABLEASUSPDALTITUD or raises the existing singleton*.
%
%   H = GUIDECVARIABLEASUSPDALTITUD returns the handle to a new
%   GUIDECVARIABLEASUSPDALTITUD or the handle to the existing
%   singleton*.
%
%   GUIDECVARIABLEASUSPDALTITUD('CALLBACK',hObject,
%   eventData,handles,...) calls the local function named CALLBACK
%   in GUIDECVARIABLEASUSPDALTITUD.M with the given input
%   arguments.
%
%   GUIDECVARIABLEASUSPDALTITUD('Property','Value',...) creates a
%   new GUIDECVARIABLEASUSPDALTITUD or raises the existing
%   singleton*. Starting from the left, property value pairs are
%   applied to the GUI before
%   guideCvariableAsuspDaltitud_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableAsuspDaltitud_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableAsuspDaltitud

% Last Modified by GUIDE v2.5 03-Sep-2014 13:08:28

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCvariableAsuspDaltitud_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCvariableAsuspDaltitud_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

```




```

% --- Executes just before guideCvariableAsuspDaltitud is made
% visible.
function guideCvariableAsuspDaltitud_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideCvariableAsuspDaltitud
% (see VARARGIN)

% Choose default command line output for guideCvariableAsuspDaltitud
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableAsuspDaltitud wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCvariableAsuspDaltitud_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global SUSPDST; global SUSPDOPEN;
global ALST; global ALOPENI;
global OPENCA; global OPENS; global OPENP;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if ((OPENCA==1) || ((OPENS==1) && (OPENCA==0)) || ((OPENP==1) && (OPENCA==0)))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (A): SENSOR SUSPENSIÓN
    %DELANTERA Y SENSOR G.P.S. (ALTITUD))
    set(handles.SCVASDT12, 'Data', SUSPDOPEN);
    set(handles.SCVAALT2, 'Data', ALOPENI);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (A): SUSPENSIÓN DELANTERA
    %(VS) ALTITUD.)
    size(SUSPDOPEN);
    tam1=size(SUSPDOPEN);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,SUSPDOPEN,x,ALOPENI);
    set(H1, 'DisplayName', 'OPEN SUSP.DELANTERA VUELTA', ...
'YDataSource', 'OPEN SUSP.DELANTERA VUELTA', 'LineStyle', '-');
    set(H2, 'DisplayName', 'OPEN ALTITUD VUELTA', 'YDataSource', ...
'OPEN ALTITUD VUELTA', 'LineStyle', '--');

```



```

set(get(Ax(1),'Ylabel'),'String','Longitud (mm)');
set(get(Ax(2),'Ylabel'),'String','Altitud (m)');
set(handles.SCVASDALG1,'xgrid','on','ygrid','on');
xlabel(handles.SCVASDALG1,'Número de Muestras');
title(handles.SCVASDALG1,'Gráfica Variables (A):...
Suspensión Delantera (mm) (VS) Altitud (m)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');

elseif((OPENCA==0))

%TABLA 1 Y 2 (DATOS ACTUALES SESIÓN (A): SENSOR SUSPENSIÓN
%DELANTERA Y SENSOR G.P.S. (ALTITUD))
set(handles.SCVASDT12,'Data',SUSPDST);
set(handles.SCVAAALT2,'Data',ALST);

%GRÁFICA 1 (VARIABLES ACTUALES SESIÓN (A): SUSPENSIÓN DELANTERA
%(VS) ALTITUD)
size(SUSPDST);
tam1=size(SUSPDST);
tamx=tam1(:,1);
x=1:tamx;
[AX,H1,H2] = plotyy(x,SUSPDST,x,ALST);
set(H1,'DisplayName','SUSP.DELANTERA VUELTA',...
'YDataSource','SUSP.DELANTERA VUELTA','LineStyle','-');
set(H2,'DisplayName','ALTITUD VUELTA','YDataSource',...
'ALTITUD VUELTA','LineStyle','--');
set(get(Ax(1),'Ylabel'),'String','Longitud (mm)');
set(get(Ax(2),'Ylabel'),'String','Altitud (m)');
set(handles.SCVASDALG1,'xgrid','on','ygrid','on');
xlabel(handles.SCVASDALG1,'Número de Muestras');
title(handles.SCVASDALG1,'Gráfica Variables (A):...
Suspensión Delantera (mm) (VS) Altitud (m)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close(handles.output);

% --- Executes on button press in leyendaCVASDvsALT.
function leyendaCVASDvsALT_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVASDvsALT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVASDvsALT

%VARIABLES GLOBALES
global H1;
global H2;

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set(legend(H1));
    set(legend(H2));
    legend(handles.SCVASDALG1,'boxoff');
    set(handles.leyendaCVASDvsALT,'String','LEYENDA ON');
else
    legend(handles.SCVASDALG1,'off');
    delete(legend(H2));
    set(handles.leyendaCVASDvsALT,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrículaCVASDvsALT.
function cuadrículaCVASDvsALT_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaCVASDvsALT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% cuadrículaCVASDvsALT

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA(ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVASDALG1,'off');
    set(handles.cuadrículaCVASDvsALT,'String','CUADRÍCULA OFF');
else
    grid(handles.SCVASDALG1,'on');
    set(handles.cuadrículaCVASDvsALT,'String','CUADRÍCULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (A)

SUSPENSIÓN DELANTERA-SUSPENSIÓN TRASERA

```
function varargout = guideCvariableAsuspDsuspT(varargin)
% GUIDECVARIABLEASUSPDSUSPT M-file for guideCvariableAsuspDsuspT.fig
%   GUIDECVARIABLEASUSPDSUSPT, by itself, creates a new
%   GUIDECVARIABLEASUSPDSUSPT or raises the existing singleton*.
%
%   H = GUIDECVARIABLEASUSPDSUSPT returns the handle to a new
%   GUIDECVARIABLEASUSPDSUSPT or the handle to the existing
%   singleton*.
%
%   GUIDECVARIABLEASUSPDSUSPT('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECVARIABLEASUSPDSUSPT.M with the given input arguments.
%
%   GUIDECVARIABLEASUSPDSUSPT('Property','Value',...) creates a new
%   GUIDECVARIABLEASUSPDSUSPT or raises the existing singleton*.
%   Starting from the left, property value pairs are applied to the
%   GUI before guideCvariableAsuspDsuspT_OpeningFcn gets called.
%   An unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableAsuspDsuspT_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableAsuspDsuspT

% Last Modified by GUIDE v2.5 02-Sep-2014 20:32:38

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCvariableAsuspDsuspT_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCvariableAsuspDsuspT_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```



```

% --- Executes just before guideCvariableAsuspDsuspT is made visible.
function guideCvariableAsuspDsuspT_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideCvariableAsuspDsuspT (see
% VARARGIN)

% Choose default command line output for guideCvariableAsuspDsuspT
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableAsuspDsuspT wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCvariableAsuspDsuspT_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global SUSPDST; global SUSPTST; global SUSPDOPEN; global SUSPTOPEN;
global OPENCA; global OPENS; global OPENP;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if ((OPENCA==1) || ((OPENS==1) && (OPENCA==0)) || ((OPENP==1) && (OPENCA==0)))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (A): SENSOR SUSPENSIÓN
    %DELANTERA Y SENSOR SUSPENSIÓN TRASERA)
    set(handles.SCVASDT1, 'Data', SUSPDOPEN);
    set(handles.SCVASTT2, 'Data', SUSPTOPEN);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (A): SUSPENSIÓN DELANTERA
    %(VS) SUSPENSIÓN TRASERA)
    size(SUSPDOPEN);
    tam1=size(SUSPDOPEN);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,SUSPDOPEN,x,SUSPTOPEN);
    set(H1, 'DisplayName', 'OPEN SUSP.DELANTERA VUELTA', ...
    'YDataSource', 'OPEN SUSP.DELANTERA VUELTA', 'LineStyle', '-');
    set(H2, 'DisplayName', 'OPEN SUSP.TRASERA VUELTA', ...
    'YDataSource', 'OPEN SUSP.TRASERA VUELTA', 'LineStyle', '--');
    set(get(AX(1), 'Ylabel'), 'String', 'Longitud (mm)');
    set(get(AX(2), 'Ylabel'), 'String', 'Longitud (mm)');

```



```

set(handles.SCVASDSTG1,'xgrid','on','ygrid','on');
xlabel(handles.SCVASDSTG1,'Número de Muestras');
title(handles.SCVASDSTG1,'Gráfica Variables (A):...
Suspensión Delantera (mm) (VS) Suspensión Trasera (mm)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');

elseif((OPENCA==0))

%TABLA 1 Y 2 (DATOS ACTUALES SESIÓN(A): SENSOR SUSPENSIÓN
%DELANTERA Y SENSOR SUSPENSIÓN TRASERA)
set(handles.SCVASDT1,'Data',SUSPDST);
set(handles.SCVASTT2,'Data',SUSPTST);

%GRÁFICA 1 (VARIABLES ACTUALES SESIÓN (A): SUSPENSIÓN DELANTERA
%(VS) SUSPENSIÓN TRASERA)
size(SUSPDST);
tam1=size(SUSPDST);
tamx=tam1(:,1);
x=1:tamx;
[AX,H1,H2] = plotyy(x,SUSPDST,x,SUSPTST);
set(H1,'DisplayName','SUSP.DELANTERA VUELTA',...
'YDataSource','SUSP.DELANTERA VUELTA','LineStyle','-');
set(H2,'DisplayName','SUSP.TRASERA VUELTA',...
'YDataSource','SUSP.TRASERA VUELTA','LineStyle','--');
set(get(AX(1),'Ylabel'),'String','Longitud (mm)');
set(get(AX(2),'Ylabel'),'String','Longitud (mm)');
set(handles.SCVASDSTG1,'xgrid','on','ygrid','on');
xlabel(handles.SCVASDSTG1,'Número de Muestras');
title(handles.SCVASDSTG1,'Gráfica Variables (A):...
Suspensión Delantera (mm) (VS) Suspensión Trasera (mm)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close(handles.output);

% --- Executes on button press in leyendaCVASDvsST.
function leyendaCVASDvsST_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVASDvsST (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVASDvsST

%VARIABLES GLOBALES
global H1;
global H2;

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set(legend(H1));
    set(legend(H2));
    legend(handles.SCVASDSTG1,'boxoff');
    set(handles.leyendaCVASDvsST,'String','LEYENDA ON');
else
    legend(handles.SCVASDSTG1,'off');
    delete(legend(H2));
    set(handles.leyendaCVASDvsST,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrículaCVASDvsST.
function cuadrículaCVASDvsST_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaCVASDvsST (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% cuadrículaCVASDvsST

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA(ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVASDSTG1,'off');
    set(handles.cuadrículaCVASDvsST,'String','CUADRÍCULA OFF');
else
    grid(handles.SCVASDSTG1,'on');
    set(handles.cuadrículaCVASDvsST,'String','CUADRÍCULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (A)

SUSPENSIÓN TRASERA-ALTITUD

```

function varargout = guideCvariableAsuspTaltitud(varargin)
% GUIDECVARIABLEASUSPTALTITUD M-file for
% guideCvariableAsuspTaltitud.fig
%
%   GUIDECVARIABLEASUSPTALTITUD, by itself, creates a new
%   GUIDECVARIABLEASUSPTALTITUD or raises the existing singleton*.
%
%   H = GUIDECVARIABLEASUSPTALTITUD returns the handle to a new
%   GUIDECVARIABLEASUSPTALTITUD or the handle to the existing
%   singleton*.
%
%   GUIDECVARIABLEASUSPTALTITUD('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECVARIABLEASUSPTALTITUD.M with the given input arguments.
%
%   GUIDECVARIABLEASUSPTALTITUD('Property','Value',...) creates a
%   new GUIDECVARIABLEASUSPTALTITUD or raises the existing
%   singleton*. Starting from the left, property value pairs are
%   applied to the GUI before
%   guideCvariableAsuspTaltitud_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableAsuspTaltitud_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableAsuspTaltitud

% Last Modified by GUIDE v2.5 03-Sep-2014 14:01:59

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCvariableAsuspTaltitud_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCvariableAsuspTaltitud_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

```




```

% --- Executes just before guideCvariableAsuspTaltitud is made
% visible.
function guideCvariableAsuspTaltitud_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideCvariableAsuspTaltitud
% (see VARARGIN)

% Choose default command line output for guideCvariableAsuspTaltitud
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableAsuspTaltitud wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCvariableAsuspTaltitud_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global SUSPTST; global SUSPTOPEN;
global ALST; global ALOPENI;
global OPENCA; global OPENS; global OPENP;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if ((OPENCA==1) || ((OPENS==1) && (OPENCA==0)) || ((OPENP==1) && (OPENCA==0)))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (A): SENSOR SUSPENSIÓN
    %TRASERA Y SENSOR G.P.S. (ALTITUD))
    set(handles.SCVASTT1, 'Data', SUSPTOPEN);
    set(handles.SCVAALT22, 'Data', ALOPENI);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (A): SUSPENSIÓN TRASERA
    %(VS) ALTITUD)
    size(SUSPTOPEN);
    tam1=size(SUSPTOPEN);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,SUSPTOPEN,x,ALOPENI);
    set(H1, 'DisplayName', 'OPEN SUSP. TRASERA VUELTA', ...
        'YDataSource', 'OPEN SUSP. TRASERA VUELTA', 'LineStyle', '-');
    set(H2, 'DisplayName', 'OPEN ALTITUD VUELTA', ...
        'YDataSource', 'OPEN ALTITUD VUELTA', 'LineStyle', '--');

```



```

set(get(AX(1),'Ylabel'),'String','Longitud (mm)');
set(get(AX(2),'Ylabel'),'String','Altitud (m)');
set(handles.SCVASTALG1,'xgrid','on','ygrid','on');
xlabel(handles.SCVASTALG1,'Número de Muestras');
title(handles.SCVASTALG1,'Gráfica Variables (A):...
Suspensión Trasera (mm) (VS) Altitud (m)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');

elseif((OPENCA==0))

%TABLA 1 Y 2 (DATOS ACTUALES SESIÓN (A): SENSOR SUSPENSIÓN TRASERA
%Y SENSOR G.P.S.(ALTITUD)
set(handles.SCVASTT1,'Data',SUSPTST);
set(handles.SCVAAALT22,'Data',ALST);

%GRÁFICA 1 (VARIABLES ACTUALES SESIÓN (A): SUSPENSIÓN TRASERA (VS)
%ALTITUD)
size(SUSPTST);
tam1=size(SUSPTST);
tamx=tam1(:,1);
x=1:tamx;
[AX,H1,H2] = plotyy(x,SUSPTST,x,ALST);
set(H1,'DisplayName','SUSP.TRASERA VUELTA',...
'YDataSource','SUSP.TRASERA VUELTA','LineStyle','-');
set(H2,'DisplayName','ALTITUD VUELTA',...
'YDataSource','ALTITUD VUELTA','LineStyle','--');
set(get(AX(1),'Ylabel'),'String','Longitud (mm)');
set(get(AX(2),'Ylabel'),'String','Altitud (m)');
set(handles.SCVASTALG1,'xgrid','on','ygrid','on');
xlabel(handles.SCVASTALG1,'Número de Muestras');
title(handles.SCVASTALG1,'Gráfica Variables (A):...
Suspensión Trasera (mm) (VS) Altitud (m)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close(handles.output);

% --- Executes on button press in leyendaCVASTvsALT.
function leyendaCVASTvsALT_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVASTvsALT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVASTvsALT

%VARIABLES GLOBALES
global H1;
global H2;

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set(legend(H1));
    set(legend(H2));
    legend(handles.SCVASTALG1,'boxoff');
    set(handles.leyendaCVASTvsALT,'String','LEYENDA ON');
else
    legend(handles.SCVASTALG1,'off');
    delete(legend(H2));
    set(handles.leyendaCVASTvsALT,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrriculaCVASTvsALT.
function cuadrriculaCVASTvsALT_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrriculaCVASTvsALT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% cuadrriculaCVASTvsALT

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA(ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVASTALG1,'off');
    set(handles.cuadrriculaCVASTvsALT,'String','CUADRICULA OFF');
else
    grid(handles.SCVASTALG1,'on');
    set(handles.cuadrriculaCVASTvsALT,'String','CUADRICULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (A) VELOCIDAD-ALTITUD

```

function varargout = guideCvariableAvelocidadAltitud(varargin)
% GUIDECVARIABLEAVELOCIDADALTITUD M-file for
% guideCvariableAvelocidadAltitud.fig
%
%   GUIDECVARIABLEAVELOCIDADALTITUD, by itself, creates a new
%   GUIDECVARIABLEAVELOCIDADALTITUD or raises the existing
%   singleton*.
%
%   H = GUIDECVARIABLEAVELOCIDADALTITUD returns the handle to a new
%   GUIDECVARIABLEAVELOCIDADALTITUD or the handle to the existing
%   singleton*.
%
%   GUIDECVARIABLEAVELOCIDADALTITUD('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECVARIABLEAVELOCIDADALTITUD.M with the given input
%   arguments.
%
%   GUIDECVARIABLEAVELOCIDADALTITUD('Property','Value',...) creates
%   a new GUIDECVARIABLEAVELOCIDADALTITUD or raises the existing
%   singleton*. Starting from the left, property value pairs are
%   applied to the GUI before
%   guideCvariableAvelocidadAltitud_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableAvelocidadAltitud_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableAvelocidadAltitud

% Last Modified by GUIDE v2.5 03-Sep-2014 13:03:04

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn',
@guideCvariableAvelocidadAltitud_OpeningFcn, ...
                  'gui_OutputFcn',
@guideCvariableAvelocidadAltitud_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
% --- Executes just before guideCvariableAvelocidadAltitud is made
% visible.

```



```

function guideCvariableAvelocidadAltitud_OpeningFcn(hObject,
eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideCvariableAvelocidadAltitud
% (see VARARGIN)

% Choose default command line output for
% guideCvariableAvelocidadAltitud
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableAvelocidadAltitud wait for user response
% (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout =
guideCvariableAvelocidadAltitud_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global ALST; global ALOPENI;
global VST; global VOPEN;
global OPENCA; global OPENS; global OPENP;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if ((OPENCA==1) || ((OPENS==1) && (OPENCA==0)) || ((OPENP==1) && (OPENCA==0)))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (A): SENSOR G.P.S. (VELOCIDAD)
    %Y SENSOR G.P.S. (ALTITUD))
    set(handles.SCVAVT1, 'Data', VOPEN);
    set(handles.SCVAAALT223, 'Data', ALOPENI);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (A): VELOCIDAD (VS)
    %ALTITUD)
    size(VOPEN);
    tam1=size(VOPEN);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,VOPEN,x,ALOPENI);
    set(H1, 'DisplayName', 'OPEN VELOCIDAD VUELTA', ...
        'YDataSource', 'OPEN VELOCIDAD VUELTA', 'LineStyle', '-');
    set(H2, 'DisplayName', 'OPEN ALTITUD VUELTA', ...
        'YDataSource', 'OPEN ALTITUD VUELTA', 'LineStyle', '--');
    set(get(AX(1), 'Ylabel'), 'String', 'Velocidad (Km/h)');

```



```

set(get(Ax(2), 'Ylabel'), 'String', 'Altitud (m)');
set(handles.SCVAVAlG1, 'xgrid', 'on', 'ygrid', 'on');
xlabel(handles.SCVAVAlG1, 'Número de Muestras');
title(handles.SCVAVAlG1, 'Gráfica Variables (A):...
Velocidad (Km/h) (VS) Altitud (m)',...
'FontSize', 12, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');

elseif((OPENCA==0))

%TABLA 1 Y 2 (DATOS ACTUALES SESIÓN (A): SENSOR G.P.S.(VELOCIDAD)
%Y SENSOR G.P.S.(ALTITUD))
set(handles.SCVAVT1, 'Data', VST);
set(handles.SCVAAALT223, 'Data', ALST);

%GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (A): VELOCIDAD (VS)
%ALTITUD)
size(VST);
tam1=size(VST);
tamx=tam1(:,1);
x=1:tamx;
[AX,H1,H2] = plotyy(x,VST,x,ALST);
set(H1, 'DisplayName', 'VELOCIDAD VUELTA',...
'YDataSource', 'VELOCIDAD VUELTA', 'LineStyle', '-');
set(H2, 'DisplayName', 'ALTITUD VUELTA',...
'YDataSource', 'ALTITUD VUELTA', 'LineStyle', '--');
set(get(Ax(1), 'Ylabel'), 'String', 'Velocidad (Km/h)');
set(get(Ax(2), 'Ylabel'), 'String', 'Altitud (m)');
set(handles.SCVAVAlG1, 'xgrid', 'on', 'ygrid', 'on');
xlabel(handles.SCVAVAlG1, 'Número de Muestras');
title(handles.SCVAVAlG1, 'Gráfica Variables (A):...
Velocidad (Km/h) (VS) Altitud (m)',...
'FontSize', 12, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close(handles.output);

% --- Executes on button press in leyendaCVAVvsALT.
function leyendaCVAVvsALT_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVAVvsALT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVAVvsALT

%VARIABLES GLOBALES
global H1;
global H2;

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set(legend(H1));
    set(legend(H2));
    legend(handles.SCVAVG1,'boxoff');
    set(handles.leyendaCVAVvsALT,'String','LEYENDA ON');
else
    legend(handles.SCVAVG1,'off');
    delete(legend(H2));
    set(handles.leyendaCVAVvsALT,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrículaCVAVvsALT.
function cuadrículaCVAVvsALT_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaCVAVvsALT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% cuadrículaCVAVvsALT

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA(ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVAVG1,'off');
    set(handles.cuadrículaCVAVvsALT,'String','CUADRÍCULA OFF');
else
    grid(handles.SCVAVG1,'on');
    set(handles.cuadrículaCVAVvsALT,'String','CUADRÍCULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (A)

VELOCIDAD-SUSPENSIÓN DELANTERA

```

function varargout = guideCvariableAvelocidadSuspD(varargin)
% GUIDECVARIABLEAVELOCIDADXSUSPD M-file for
% guideCvariableAvelocidadSuspD.fig
%
%   GUIDECVARIABLEAVELOCIDADXSUSPD, by itself, creates a new
%   GUIDECVARIABLEAVELOCIDADXSUSPD or raises the existing
%   singleton*.
%
%   H = GUIDECVARIABLEAVELOCIDADXSUSPD returns the handle to a new
%   GUIDECVARIABLEAVELOCIDADXSUSPD or the handle to the existing
%   singleton*.
%
%   GUIDECVARIABLEAVELOCIDADXSUSPD('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECVARIABLEAVELOCIDADXSUSPD.M with the given input arguments.
%
%   GUIDECVARIABLEAVELOCIDADXSUSPD('Property','Value',...) creates a
%   new GUIDECVARIABLEAVELOCIDADXSUSPD or raises the existing
%   singleton*. Starting from the left, property value pairs are
%   applied to the GUI before
%   guideCvariableAvelocidadSuspD_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableAvelocidadSuspD_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableAvelocidadSuspD

% Last Modified by GUIDE v2.5 03-Sep-2014 11:08:50

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCvariableAvelocidadSuspD_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCvariableAvelocidadSuspD_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

```




```

% --- Executes just before guideCvariableAvelocidadSuspD is made
% visible.
function guideCvariableAvelocidadSuspD_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideCvariableAvelocidadSuspD
%(see VARARGIN)

% Choose default command line output for guideCvariableAvelocidadSuspD
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableAvelocidadSuspD wait for user response
% (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCvariableAvelocidadSuspD_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global VST; global VOPEN;
global SUSPDST; global SUSPDOPEN;
global OPENCA; global OPENS; global OPENP;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if ((OPENCA==1) || ((OPENS==1) && (OPENCA==0)) || ((OPENP==1) && (OPENCA==0)))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (A): SENSOR G.P.S. (VELOCIDAD)
    %Y SENSOR SUSPENSIÓN DELANTERA)
    set(handles.SCVAVT12, 'Data', VOPEN);
    set(handles.SCVASDT2, 'Data', SUSPDOPEN);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (A): VELOCIDAD (VS)
    %SUSPENSIÓN DELANTERA)
    size(VOPEN);
    tam1=size(VOPEN);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,VOPEN,x,SUSPDOPEN);
    set(H1, 'DisplayName', 'OPEN VELOCIDAD VUELTA',...
        'YDataSource', 'OPEN VELOCIDAD VUELTA', 'LineStyle', '-');
    set(H2, 'DisplayName', 'OPEN SUSP.DELANTERA VUELTA',...
        'YDataSource', 'OPEN SUSP.DELANTERA VUELTA', 'LineStyle', '--');

```



```

set(get(Ax(1),'Ylabel'),'String','Velocidad (Km/h)');
set(get(Ax(2),'Ylabel'),'String','Longitud (mm)');
set(handles.SCVAVSDG1,'xgrid','on','ygrid','on');
xlabel(handles.SCVAVSDG1,'Número de Muestras');
title(handles.SCVAVSDG1,'Gráfica Variables (A):...
Velocidad (Km/h) (VS) Suspensión Delantera (mm)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');

elseif((OPENCA==0))

%TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (A): SENSOR G.P.S.(VELOCIDAD)
%Y SENSOR SUSPENSIÓN DELANTERA)
set(handles.SCVAVT12,'Data',VST);
set(handles.SCVASDT2,'Data',SUSPDST);

%GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (A): VELOCIDAD (VS)
%SUSPENSIÓN DELANTERA)
size(VST);
tam1=size(VST);
tamx=tam1(:,1);
x=1:tamx;
[AX,H1,H2] = plotyy(x,VST,x,SUSPDST);
set(H1,'DisplayName','VELOCIDAD VUELTA',...
'YDataSource','VELOCIDAD VUELTA','LineStyle','-');
set(H2,'DisplayName','SUSP.DELANTERA VUELTA',...
'YDataSource','SUSP.DELANTERA VUELTA','LineStyle','--');
set(get(Ax(1),'Ylabel'),'String','Velocidad (Km/h)');
set(get(Ax(2),'Ylabel'),'String','Longitud (mm)');
set(handles.SCVAVSDG1,'xgrid','on','ygrid','on');
xlabel(handles.SCVAVSDG1,'Número de Muestras');
title(handles.SCVAVSDG1,'Gráfica Variables (A):...
Velocidad (Km/h) (VS) Suspensión Delantera (mm)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close(handles.output);

% --- Executes on button press in leyendaCVAVvsSD.
function leyendaCVAVvsSD_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVAVvsSD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVAVvsSD

%VARIABLES GLOBALES
global H1;
global H2;

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set(legend(H1));
    set(legend(H2));
    legend(handles.SCVAVSDG1,'boxoff');
    set(handles.leyendaCVAVvsSD,'String','LEYENDA ON');
else
    legend(handles.SCVAVSDG1,'off');
    delete(legend(H2));
    set(handles.leyendaCVAVvsSD,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrículaCVAVvsSD.
function cuadrículaCVAVvsSD_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaCVAVvsSD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% cuadrículaCVAVvsSD

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA(ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVAVSDG1,'off');
    set(handles.cuadrículaCVAVvsSD,'String','CUADRÍCULA OFF');
else
    grid(handles.SCVAVSDG1,'on');
    set(handles.cuadrículaCVAVvsSD,'String','CUADRÍCULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (A)

VELOCIDAD-SUSPENSIÓN TRASERA

```

function varargout = guideCvariableAvelocidadSuspT(varargin)
% GUIDECVARIABLEAVELOCIDADXSUSPT M-file for
% guideCvariableAvelocidadSuspT.fig
%
%   GUIDECVARIABLEAVELOCIDADXSUSPT, by itself, creates a new
%   GUIDECVARIABLEAVELOCIDADXSUSPT or raises the existing
%   singleton*.
%
%   H = GUIDECVARIABLEAVELOCIDADXSUSPT returns the handle to a new
%   GUIDECVARIABLEAVELOCIDADXSUSPT or the handle to the existing
%   singleton*.
%
%   GUIDECVARIABLEAVELOCIDADXSUSPT('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECVARIABLEAVELOCIDADXSUSPT.M with the given input arguments.
%
%   GUIDECVARIABLEAVELOCIDADXSUSPT('Property','Value',...) creates a
%   new GUIDECVARIABLEAVELOCIDADXSUSPT or raises the existing
%   singleton*. Starting from the left, property value pairs are
%   applied to the GUI before
%   guideCvariableAvelocidadSuspT_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableAvelocidadSuspT_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableAvelocidadSuspT

% Last Modified by GUIDE v2.5 03-Sep-2014 12:30:38

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCvariableAvelocidadSuspT_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCvariableAvelocidadSuspT_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

```



```

% --- Executes just before guideCvariableAvelocidadSuspT is made
% visible.
function guideCvariableAvelocidadSuspT_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin    command line arguments to guideCvariableAvelocidadSuspT
% (see VARARGIN)

% Choose default command line output for guideCvariableAvelocidadSuspT
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableAvelocidadSuspT wait for user response
% (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCvariableAvelocidadSuspT_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global VST; global VOPEN;
global SUSPTST; global SUSPTOPEN;
global OPENCA; global OPENS; global OPENP;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if ((OPENCA==1) || ((OPENS==1) && (OPENCA==0)) || ((OPENP==1) && (OPENCA==0)))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (A): SENSOR G.P.S. (VELOCIDAD)
    %Y SENSOR SUSPENSIÓN TRASERA)
    set(handles.SCVAVT123, 'Data', VOPEN);
    set(handles.SCVASTT22, 'Data', SUSPTOPEN);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (A): VELOCIDAD (VS)
    %SUSPENSIÓN TRASERA)
    size(VOPEN);
    tam1=size(VOPEN);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,VOPEN,x,SUSPTOPEN);
    set(H1, 'DisplayName', 'OPEN VELOCIDAD VUELTA',...
        'YDataSource', 'OPEN VELOCIDAD VUELTA', 'LineStyle', '-');
    set(H2, 'DisplayName', 'OPEN SUSP. TRASERA VUELTA',...
        'YDataSource', 'OPEN SUSP. TRASERA VUELTA', 'LineStyle', '--');

```



```

set(get(Ax(1),'Ylabel'),'String','Velocidad (Km/h)');
set(get(Ax(2),'Ylabel'),'String','Longitud (mm)');
set(handles.SCVAVSTG1,'xgrid','on','ygrid','on');
xlabel(handles.SCVAVSTG1,'Número de Muestras');
title(handles.SCVAVSTG1,'Gráfica Variables (A):...
Velocidad (Km/h) (VS) Suspensión Trasera (mm)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');

elseif((OPENCA==0))

%TABLA 1 Y 2 ( DATOS ACTUALES SESIÓN (A): SENSOR G.P.S. (VELOCIDAD)
%Y SENSOR SUSPENSIÓN TRASERA)
set(handles.SCVAVT123,'Data',VST);
set(handles.SCVASTT22,'Data',SUSPTST);

%GRÁFICA 1 (VARIABLES ACTUALES SESIÓN (A): VELOCIDAD (VS)
%SUSPENSIÓN TRASERA)
size(VST);
tam1=size(VST);
tamx=tam1(:,1);
x=1:tamx;
[AX,H1,H2] = plotyy(x,VST,x,SUSPTST);
set(H1,'DisplayName','VELOCIDAD VUELTA',...
'YDataSource','VELOCIDAD VUELTA','LineStyle','-');
set(H2,'DisplayName','SUSP.TRASERA VUELTA',...
'YDataSource','SUSP.TRASERA VUELTA','LineStyle','--');
set(get(Ax(1),'Ylabel'),'String','Velocidad (Km/h)');
set(get(Ax(2),'Ylabel'),'String','Longitud (mm)');
set(handles.SCVAVSTG1,'xgrid','on','ygrid','on');
xlabel(handles.SCVAVSTG1,'Número de Muestras');
title(handles.SCVAVSTG1,'Gráfica Variables (A):...
Velocidad (Km/h) (VS) Suspensión Trasera (mm)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close(handles.output);

% --- Executes on button press in leyendaCVAVvsST.
function leyendaCVAVvsST_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVAVvsST (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVAVvsST

%VARIABLES GLOBALES
global H1
global H2

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set(legend(H1));
    set(legend(H2));
    legend(handles.SCVAVSTG1,'boxoff');
    set(handles.leyendaCVAVvsST,'String','LEYENDA ON');
else
    legend(handles.SCVAVSTG1,'off');
    delete(legend(H2));
    set(handles.leyendaCVAVvsST,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrículaCVAVvsST.
function cuadrículaCVAVvsST_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaCVAVvsST (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% cuadrículaCVAVvsST

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA(ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVAVSTG1,'off');
    set(handles.cuadrículaCVAVvsST,'String','CUADRICULA OFF');
else
    grid(handles.SCVAVSTG1,'on');
    set(handles.cuadrículaCVAVvsST,'String','CUADRICULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (A) VELOCIDAD-TEMPERATURA AMBIENTE

```
function varargout = guideCvariableAvelocidadtempambiente(varargin)
% GUIDECVARIABLEAVELOCIDADTEMPAMBIENTE M-file for
% guideCvariableAvelocidadtempambiente.fig
%
%   GUIDECVARIABLEAVELOCIDADTEMPAMBIENTE, by itself, creates a new
%   GUIDECVARIABLEAVELOCIDADTEMPAMBIENTE or raises the existing
%   singleton*.
%
%   H = GUIDECVARIABLEAVELOCIDADTEMPAMBIENTE returns the handle to
%   a new GUIDECVARIABLEAVELOCIDADTEMPAMBIENTE or the handle to the
%   existing singleton*.
%
%   %GUIDECVARIABLEAVELOCIDADTEMPAMBIENTE('CALLBACK',hObject,
%   eventData,handles,...) calls the local function named CALLBACK
%   in GUIDECVARIABLEAVELOCIDADTEMPAMBIENTE.M with the given input
%   arguments.
%
%   GUIDECVARIABLEAVELOCIDADTEMPAMBIENTE('Property','Value',...)
%   creates a new GUIDECVARIABLEAVELOCIDADTEMPAMBIENTE or raises
%   the existing singleton*. Starting from the left, property
%   value pairs are applied to the GUI before
%   guideCvariableAvelocidadtempambiente_OpeningFcn gets called.
%   An unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableAvelocidadtempambiente_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableAvelocidadtempambiente

% Last Modified by GUIDE v2.5 03-Sep-2014 12:30:53

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCvariableAvelocidadtempambiente_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCvariableAvelocidadtempambiente_OutputFcn, ...
                  'gui_LayoutFcn',   [] , ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargin
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```




```

% --- Executes just before guideCvariableAvelocidadtempambiente is
% made visible.
function guideCvariableAvelocidadtempambiente_OpeningFcn(hObject,
eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to
% guideCvariableAvelocidadtempambiente (see VARARGIN)

% Choose default command line output for
% guideCvariableAvelocidadtempambiente
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableAvelocidadtempambiente wait for user
% response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout =
guideCvariableAvelocidadtempambiente_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global VST; global VOPEN;
global TAST; global TAOPEN;
global OPENCA; global OPENS; global OPENP;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA O DE SESIÓN CARGADA
if( (OPENCA==1) || ( (OPENS==1) && (OPENCA==0) ) || ( (OPENP==1) && (OPENCA==0) ) )

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (A): SENSOR G.P.S. (VELOCIDAD)
    %Y SENSOR TEMPERATURA AMBIENTE)
    set(handles.SCVAVT1234, 'Data', VOPEN);
    set(handles.SCVATAT22, 'Data', TAOPEN);

    %GRÁFICA 1 (VAIABLES GUARDADAS SESIÓN (A): VELOCIDAD (VS)
    %TEMPERATURA AMBIENTE)
    size(VOPEN);
    tam1=size(VOPEN);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,VOPEN,x,TAOPEN);
    set(H1, 'DisplayName', 'OPEN VELOCIDAD VUELTA', ...
        'YDataSource', 'OPEN VELOCIDAD VUELTA', 'LineStyle', '-');

```



```

set (H2, 'DisplayName', 'OPEN TEMPERATURA VUELTA', ...
'YDataSource', 'OPEN TEMPERATURA VUELTA', 'LineStyle', '--');
set (get (AX (1), 'Ylabel'), 'String', 'Velocidad (Km/h)');
set (get (AX (2), 'Ylabel'), 'String', 'Temperatura Ambiente [T(C°)]');
set (handles.SCVAVTAG1, 'xgrid', 'on', 'ygrid', 'on');
xlabel (handles.SCVAVTAG1, 'Número de Muestras');
title (handles.SCVAVTAG1, 'Gráfica Variables (A):...
Velocidad (Km/h) (VS) Temperatura Ambiente [T(C°)]', ...
'FontSize', 12, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');

elseif ((OPENCA==0))

%TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (A): SENSOR G.P.S. (VELOCIDAD)
%Y SENSOR TEMPERATURA AMBIENTE)
set (handles.SCVAVT1234, 'Data', VST);
set (handles.SCVATAT22, 'Data', TAST);

%GRÁFICA 1 (VAIABLES GUARDADAS SESIÓN (A): VELOCIDAD (VS)
%TEMPERATURA AMBIENTE)
size (VST);
tam1=size (VST);
tamx=tam1 (:, 1);
x=1:tamx;
[AX, H1, H2] = plotyy (x, VST, x, TAST);
set (H1, 'DisplayName', 'VELOCIDAD VUELTA', ...
'YDataSource', 'VELOCIDAD VUELTA', 'LineStyle', '-');
set (H2, 'DisplayName', 'SUSP.TRASERA VUELTA', ...
'YDataSource', 'SUSP.TRASERA VUELTA', 'LineStyle', '--');
set (get (AX (1), 'Ylabel'), 'String', 'Velocidad (Km/h)');
set (get (AX (2), 'Ylabel'), 'String', 'Temperatura Ambiente [T(C°)]');
set (handles.SCVAVTAG1, 'xgrid', 'on', 'ygrid', 'on');
xlabel (handles.SCVAVTAG1, 'Número de Muestras');
title (handles.SCVAVTAG1, 'Gráfica Variables (A):...
Velocidad (Km/h) (VS) Temperatura Ambiente [T(C°)]', ...
'FontSize', 12, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close (handles.output);

% --- Executes on button press in leyendaCVAVvsTA.
function leyendaCVAVvsTA_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVAVvsTA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVAVvsTA

%VARIABLES GLOBALES
global H1;
global H2;

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set(legend(H1));
    set(legend(H2));
    legend(handles.SCVAVTAG1,'boxoff');
    set(handles.leyendaCVAVvsTA,'String','LEYENDA ON');
else
    legend(handles.SCVAVTAG1,'off');
    delete(legend(H2));
    set(handles.leyendaCVAVvsTA,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrículaCVAVvsTA.
function cuadrículaCVAVvsTA_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrículaCVAVvsTA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% cuadrículaCVAVvsTA

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA(ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVAVTAG1,'off');
    set(handles.cuadrículaCVAVvsTA,'String','CUADRÍCULA OFF');
else
    grid(handles.SCVAVTAG1,'on');
    set(handles.cuadrículaCVAVvsTA,'String','CUADRÍCULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (B)

ALTITUD-TEMPERATURA AMBIENTE

```
function varargout = guideCvariableBaltitudtempambiente(varargin)
% GUIDECVARIABLEBALTITUDTEMPAMBIENTE M-file for
% guideCvariableBaltitudtempambiente.fig
%
%   GUIDECVARIABLEBALTITUDTEMPAMBIENTE, by itself, creates a new
%   GUIDECVARIABLEBALTITUDTEMPAMBIENTE or raises the existing
%   singleton*.
%
%   H = GUIDECVARIABLEBALTITUDTEMPAMBIENTE returns the handle to a
%   new GUIDECVARIABLEBALTITUDTEMPAMBIENTE or the handle to the
%   existing singleton*.
%
%   GUIDECVARIABLEBALTITUDTEMPAMBIENTE('CALLBACK',hObject,
%   eventData,handles,...) calls the local function named CALLBACK
%   in GUIDECVARIABLEBALTITUDTEMPAMBIENTE.M with the given input
%   arguments.
%
%   GUIDECVARIABLEBALTITUDTEMPAMBIENTE('Property','Value',...)
%   creates a new GUIDECVARIABLEBALTITUDTEMPAMBIENTE or raises the
%   existing singleton*. Starting from the left, property value
%   pairs are applied to the GUI before
%   guideCvariableBaltitudtempambiente_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableBaltitudtempambiente_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableBaltitudtempambiente

% Last Modified by GUIDE v2.5 03-Sep-2014 18:03:50

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCvariableBaltitudtempambiente_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCvariableBaltitudtempambiente_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```



```

% --- Executes just before guideCvariableBaltitudtempambiente is made
% visible.
function guideCvariableBaltitudtempambiente_OpeningFcn(hObject,
eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to
% guideCvariableBaltitudtempambiente (see VARARGIN)

% Choose default command line output for
% guideCvariableBaltitudtempambiente
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableBaltitudtempambiente wait for user
% response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout =
guideCvariableBaltitudtempambiente_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global OPENCB;
global ONB;
global ALOPENIB;
global TAOPENB;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA
if ((OPENCB==1) && (ONB==1))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (B): SENSOR G.P.S. (ALTITUD) Y
    %SENSOR TEMPERATURA AMBIENTE)
    set(handles.SCVBALT1, 'Data', ALOPENIB);
    set(handles.SCVBTAT2, 'Data', TAOPENB);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (B): ALTITUD (VS)
    %TEMPERATURA AMBIENTE)
    size(ALOPENIB);
    tam1=size(ALOPENIB);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,ALOPENIB,x,TAOPENB);
    set(H1, 'DisplayName', 'OPENB ALTITUD VUELTA', ...

```



```

'YDataSource','OPENB ALTITUD VUELTA','LineStyle','-');
set(H2,'DisplayName','OPENB TEMPERATURA VUELTA',...
'YDataSource','OPENB TEMPERATURA VUELTA','LineStyle','-');
set(get(AX(1),'Ylabel'),'String','Altitud (m)');
set(get(AX(2),'Ylabel'),'String','Temperatura Ambiente [T(C°)]');
set(handles.SCVBALTAG1,'xgrid','on','ygrid','on');
xlabel(handles.SCVBALTAG1,'Número de Muestras');
title(handles.SCVBALTAG1,'Gráfica Variables (B):...
Altitud (m) (VS) Temperatura Ambiente [T(C°)]',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close(handles.output);

% --- Executes on button press in leyendaCVBALTVsTA.
function leyendaCVBALTVsTA_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVBALTVsTA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVBALTVsTA

%VARIABLES GLOBALES
global H1;
global H2;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set(legend(H1));
    set(legend(H2));
    legend(handles.SCVBALTAG1,'boxoff');
    set(handles.leyendaCVBALTVsTA,'String','LEYENDA ON');
else
    legend(handles.SCVBALTAG1,'off');
    delete(legend(H2));
    set(handles.leyendaCVBALTVsTA,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrriculaCVBALTVsTA.
function cuadrriculaCVBALTVsTA_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrriculaCVBALTVsTA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
%cuadrriculaCVBALTVsTA

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA (ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVBALTAG1,'off');
    set(handles.cuadrículaCVBALTVsTA,'String','CUADRICULA OFF');
else
    grid(handles.SCVBALTAG1,'on');
    set(handles.cuadrículaCVBALTVsTA,'String','CUADRICULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (B) SUSPENSIÓN DELANTERA-ALTITUD

```

function varargout = guideCvariableBsuspDaltitud(varargin)
% GUIDECVARIABLEBSUSPDALTITUD M-file for
% guideCvariableBsuspDaltitud.fig
%
%   GUIDECVARIABLEBSUSPDALTITUD, by itself, creates a new
%   GUIDECVARIABLEBSUSPDALTITUD or raises the existing singleton*.
%
%   H = GUIDECVARIABLEBSUSPDALTITUD returns the handle to a new
%   GUIDECVARIABLEBSUSPDALTITUD or the handle to the existing
%   singleton*.
%
%   GUIDECVARIABLEBSUSPDALTITUD('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECVARIABLEBSUSPDALTITUD.M with the given input arguments.
%
%   GUIDECVARIABLEBSUSPDALTITUD('Property','Value',...) creates a
%   new GUIDECVARIABLEBSUSPDALTITUD or raises the existing
%   singleton*. Starting from the left, property value pairs are
%   applied to the GUI before
%   guideCvariableBsuspDaltitud_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableBsuspDaltitud_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableBsuspDaltitud

% Last Modified by GUIDE v2.5 03-Sep-2014 18:33:41

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCvariableBsuspDaltitud_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCvariableBsuspDaltitud_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

```




```

% --- Executes just before guideCvariableBsuspDaltitud is made
% visible.
function guideCvariableBsuspDaltitud_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideCvariableBsuspDaltitud
% (see VARARGIN)

% Choose default command line output for guideCvariableBsuspDaltitud
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableBsuspDaltitud wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCvariableBsuspDaltitud_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global OPENCB;
global ONB;
global SUSPDOPENB;
global ALOPENIB;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA
if((OPENCB==1) && (ONB==1))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (B): SENSOR SUSPENSIÓN
    %DELANTERA Y SENSOR G.P.S. (ALTITUD))
    set(handles.SCVBSDT12, 'Data', SUSPDOPENB);
    set(handles.SCVBALT2, 'Data', ALOPENIB);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (A): SUSPENSIÓN DELANTERA
    %(VS) ALTITUD)
    size(SUSPDOPENB);
    tam1=size(SUSPDOPENB);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,SUSPDOPENB,x,ALOPENIB);
    set(H1, 'DisplayName', 'OPENB SUSP.DELANTERA VUELTA',...
'YDataSource', 'OPENB SUSP.DELANTERA VUELTA', 'LineStyle', '-');
    set(H2, 'DisplayName', 'OPENB ALTITUD VUELTA',...

```



```

'YDataSource','OPENB ALTITUD VUELTA','LineStyle','--');
set(get(AX(1),'Ylabel'),'String','Longitud (mm)');
set(get(AX(2),'Ylabel'),'String','Altitud (m)');
set(handles.SCVBSDALG1,'xgrid','on','ygrid','on');
xlabel(handles.SCVBSDALG1,'Número de Muestras');
title(handles.SCVBSDALG1,'Gráfica Variables (B):...
Suspensión Delantera (mm) (VS) Altitud (m)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close(handles.output);

% --- Executes on button press in leyendaCVBSDvsALT.
function leyendaCVBSDvsALT_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVBSDvsALT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVBSDvsALT

%VARIABLES GLOBALES
global H1;
global H2;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set(legend(H1));
    set(legend(H2));
    legend(handles.SCVBSDALG1,'boxoff');
    set(handles.leyendaCVBSDvsALT,'String','LEYENDA ON');
else
    legend(handles.SCVBSDALG1,'off');
    delete(legend(H2));
    set(handles.leyendaCVBSDvsALT,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrriculaCVBSDvsALT.
function cuadrriculaCVBSDvsALT_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrriculaCVBSDvsALT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% cuadrriculaCVBSDvsALT

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA (ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVBSDALG1,'off');
    set(handles.cuadriculaCVBSDvsALT,'String','CUADRICULA OFF');
else
    grid(handles.SCVBSDALG1,'on');
    set(handles.cuadriculaCVBSDvsALT,'String','CUADRICULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (B)

SUSPENSIÓN DELANTERA-SUSPENSIÓN TRASERA

```

function varargout = guideCvariableBsuspDsuspT(varargin)
% GUIDECVARIABLEBSUSPDSUSPT M-file for guideCvariableBsuspDsuspT.fig
%   GUIDECVARIABLEBSUSPDSUSPT, by itself, creates a new
%   GUIDECVARIABLEBSUSPDSUSPT or raises the existing singleton*.
%
%   H = GUIDECVARIABLEBSUSPDSUSPT returns the handle to a new
%   GUIDECVARIABLEBSUSPDSUSPT or the handle to the existing
%   singleton*.
%
%   GUIDECVARIABLEBSUSPDSUSPT('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECVARIABLEBSUSPDSUSPT.M with the given input arguments.
%
%   GUIDECVARIABLEBSUSPDSUSPT('Property','Value',...) creates a new
%   GUIDECVARIABLEBSUSPDSUSPT or raises the existing singleton*.
%   Starting from the left, property value pairs are applied to the
%   GUI before guideCvariableBsuspDsuspT_OpeningFcn gets called.
%   An unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableBsuspDsuspT_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableBsuspDsuspT

% Last Modified by GUIDE v2.5 03-Sep-2014 20:01:45

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCvariableBsuspDsuspT_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCvariableBsuspDsuspT_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

```



```

% --- Executes just before guideCvariableBsuspDsuspT is made visible.
function guideCvariableBsuspDsuspT_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideCvariableBsuspDsuspT (see
% VARARGIN)

% Choose default command line output for guideCvariableBsuspDsuspT
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableBsuspDsuspT wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCvariableBsuspDsuspT_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global OPENCB;
global ONB;
global SUSPDOPENB;
global SUSPTOPENB;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA
if ((OPENCB==1) && (ONB==1))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (B): SENSOR SUSPENSIÓN
    %DELANTERA Y SENSOR SUSPENSIÓN TRASERA)
    set(handles.SCVBSDT1, 'Data', SUSPDOPENB);
    set(handles.SCVBSTT2, 'Data', SUSPTOPENB);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (B): SUSPENSIÓN DELANTERA
    %(VS) SUSPENSIÓN TRASERA)
    size(SUSPDOPENB);
    tam1=size(SUSPDOPENB);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,SUSPDOPENB,x,SUSPTOPENB);
    set(H1, 'DisplayName', 'OPENB SUSP.DELANTERA VUELTA',...
    'YDataSource', 'OPENB SUSP.DELANTERA VUELTA', 'LineStyle', '-');
    set(H2, 'DisplayName', 'OPENB SUSP.TRASERA VUELTA',...
    'YDataSource', 'OPENB SUSP.TRASERA VUELTA', 'LineStyle', '--');

```



```

set (get (AX (1), 'Ylabel'), 'String', 'Longitud (mm)');
set (get (AX (2), 'Ylabel'), 'String', 'Longitud (mm)');
set (handles.SCVBSDSTG1, 'xgrid', 'on', 'ygrid', 'on');
xlabel (handles.SCVBSDSTG1, 'Número de Muestras');
title (handles.SCVBSDSTG1, 'Gráfica Variables (B):...
Suspensión Delantera (mm) (VS) Suspensión Trasera (mm)', ...
'FontSize', 12, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close(handles.output);

% --- Executes on button press in leyendaCVBSDvsST.
function leyendaCVBSDvsST_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVBSDvsST (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVBSDvsST

%VARIABLES GLOBALES
global H1;
global H2;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject, 'Value');
handles.o=o;

if handles.o==1
    set (legend (H1));
    set (legend (H2));
    legend (handles.SCVBSDSTG1, 'boxoff');
    set (handles.leyendaCVBSDvsST, 'String', 'LEYENDA ON');
else
    legend (handles.SCVBSDSTG1, 'off');
    delete (legend (H2));
    set (handles.leyendaCVBSDvsST, 'String', 'LEYENDA OFF');
end

% --- Executes on button press in cuadrriculaCVBSDvsST.
function cuadrriculaCVBSDvsST_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrriculaCVBSDvsST (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% cuadrriculaCVBSDvsST

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA (ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVBSDSTG1,'off');
    set(handles.cuadrículaCVBSDvsST,'String','CUADRICULA OFF');
else
    grid(handles.SCVBSDSTG1,'on');
    set(handles.cuadrículaCVBSDvsST,'String','CUADRICULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (B) SUSPENSIÓN TRASERA-ALTITUD

```
function varargout = guideCvariableBsuspTaltitud(varargin)
% GUIDECVARIABLEBSUSPTALTITUD M-file for
% guideCvariableBsuspTaltitud.fig
%
%   GUIDECVARIABLEBSUSPTALTITUD, by itself, creates a new
%   GUIDECVARIABLEBSUSPTALTITUD or raises the existing singleton*.
%
%   H = GUIDECVARIABLEBSUSPTALTITUD returns the handle to a new
%   GUIDECVARIABLEBSUSPTALTITUD or the handle to the existing
%   singleton*.
%
%   GUIDECVARIABLEBSUSPTALTITUD('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECVARIABLEBSUSPTALTITUD.M with the given input arguments.
%
%   GUIDECVARIABLEBSUSPTALTITUD('Property','Value',...) creates a
%   new GUIDECVARIABLEBSUSPTALTITUD or raises the existing
%   singleton*. Starting from the left, property value pairs are
%   applied to the GUI before
%   guideCvariableBsuspTaltitud_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableBsuspTaltitud_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableBsuspTaltitud

% Last Modified by GUIDE v2.5 04-Sep-2014 20:16:28

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCvariableBsuspTaltitud_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCvariableBsuspTaltitud_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```




```

% --- Executes just before guideCvariableBsuspTaltitud is made
% visible.
function guideCvariableBsuspTaltitud_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideCvariableBsuspTaltitud
% (see VARARGIN)

% Choose default command line output for guideCvariableBsuspTaltitud
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableBsuspTaltitud wait for user response (see
% UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCvariableBsuspTaltitud_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global OPENCB;
global ONB;
global SUSPTOPENB;
global ALOPENIB;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA
if((OPENCB==1) && (ONB==1))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (B): SENSOR SUSPENSIÓN
    %TRASERA Y SENSOR G.P.S. (ALTITUD))
    set(handles.SCVBSTT1, 'Data', SUSPTOPENB);
    set(handles.SCVBALT22, 'Data', ALOPENIB);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (B): SUSPENSIÓN TRASERA
    %(VS) ALTITUD)
    size(SUSPTOPENB);
    tam1=size(SUSPTOPENB);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,SUSPTOPENB,x,ALOPENIB);
    set(H1, 'DisplayName', 'OPENB SUSP.TRASERA VUELTA',...
        'YDataSource', 'OPENB SUSP.TRASERA VUELTA', 'LineStyle', '-');
    set(H2, 'DisplayName', 'OPENB ALTITUD VUELTA',...

```



```

'YDataSource','OPENB ALTITUD VUELTA','LineStyle','--');
set(get(AX(1),'Ylabel'),'String','Longitud (mm)');
set(get(AX(2),'Ylabel'),'String','Altitud (m)');
set(handles.SCVBSTALG1,'xgrid','on','ygrid','on');
xlabel(handles.SCVBSTALG1,'Número de Muestras');
title(handles.SCVBSTALG1,'Gráfica Variables (B):...
Suspensión Trasera (mm) (VS) Altitud (m)',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close(handles.output);

% --- Executes on button press in leyendaCVBSTvsALT.
function leyendaCVBSTvsALT_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVBSTvsALT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVBSTvsALT

%VARIABLES GLOBALES
global H1;
global H2;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set(legend(H1));
    set(legend(H2));
    legend(handles.SCVBSTALG1,'boxoff');
    set(handles.leyendaCVBSTvsALT,'String','LEYENDA ON');
else
    legend(handles.SCVBSTALG1,'off');
    delete(legend(H2));
    set(handles.leyendaCVBSTvsALT,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrriculaCVBSTvsALT.
function cuadrriculaCVBSTvsALT_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrriculaCVBSTvsALT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% cuadrriculaCVBSTvsALT

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA (ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVBSTALG1,'off');
    set(handles.cuadriculaCVBSTvsALT,'String','CUADRICULA OFF');
else
    grid(handles.SCVBSTALG1,'on');
    set(handles.cuadriculaCVBSTvsALT,'String','CUADRICULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (B) VELOCIDAD-ALTITUD

```
function varargout = guideCvariableBvelocidadAltitud(varargin)
% GUIDECVARIABLEBVELOCIDADALTITUD M-file for
% guideCvariableBvelocidadAltitud.fig
%
%   GUIDECVARIABLEBVELOCIDADALTITUD, by itself, creates a new
%   GUIDECVARIABLEBVELOCIDADALTITUD or raises the existing
%   singleton*.
%
%   H = GUIDECVARIABLEBVELOCIDADALTITUD returns the handle to a new
%   GUIDECVARIABLEBVELOCIDADALTITUD or the handle to the existing
%   singleton*.
%
%   GUIDECVARIABLEBVELOCIDADALTITUD('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECVARIABLEBVELOCIDADALTITUD.M with the given input
%   arguments.
%
%   GUIDECVARIABLEBVELOCIDADALTITUD('Property','Value',...) creates
%   a new GUIDECVARIABLEBVELOCIDADALTITUD or raises the existing
%   singleton*. Starting from the left, property value pairs are
%   applied to the GUI before
%   guideCvariableBvelocidadAltitud_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableBvelocidadAltitud_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableBvelocidadAltitud

% Last Modified by GUIDE v2.5 03-Sep-2014 20:08:40

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn',
@guideCvariableBvelocidadAltitud_OpeningFcn, ...
                  'gui_OutputFcn',
@guideCvariableBvelocidadAltitud_OutputFcn, ...
                  'gui_LayoutFcn', [] , ...
                  'gui_Callback', []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```



```

% --- Executes just before guideCvariableBvelocidadAltitud is made
% visible.
function guideCvariableBvelocidadAltitud_OpeningFcn(hObject,
eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin    command line arguments to guideCvariableBvelocidadAltitud
% (see VARARGIN)

% Choose default command line output for
% guideCvariableBvelocidadAltitud
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableBvelocidadAltitud wait for user response
% (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout =
guideCvariableBvelocidadAltitud_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global OPENCB;
global ONB;
global VOPENB;
global ALOPENIB;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA
if ((OPENCB==1) && (ONB==1))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (B): SENSOR G.P.S. (VELOCIDAD)
    %Y SENSOR G.P.S. (ALTITUD))
    set(handles.SCVBVT1, 'Data', VOPENB);
    set(handles.SCVBALT223, 'Data', ALOPENIB);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (B): VELOCIDAD (VS)
    %ALTITUD)
    size(VOPENB);
    tam1=size(VOPENB);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,VOPENB,x,ALOPENIB);
    set(H1, 'DisplayName', 'OPENB VELOCIDAD VUELTA',...
        'YDataSource', 'OPENB VELOCIDAD VUELTA', 'LineStyle', '-');

```



```

set (H2, 'DisplayName', 'OPENB ALTITUD VUELTA', ...
'YDataSource', 'OPENB ALTITUD VUELTA', 'LineStyle', '--');
set (get (AX (1), 'Ylabel'), 'String', 'Velocidad (Km/h)');
set (get (AX (2), 'Ylabel'), 'String', 'Altitud (m)');
set (handles.SCVBVALG1, 'xgrid', 'on', 'ygrid', 'on');
xlabel (handles.SCVBVALG1, 'Número de Muestras');
title (handles.SCVBVALG1, 'Gráfica Variables (B):...
Velocidad (Km/h) (VS) Altitud (m)', ...
'FontSize', 12, 'FontName', 'MS Sans Serif', 'FontWeight', 'bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close(handles.output);

% --- Executes on button press in leyendaCVBVvsALT.
function leyendaCVBVvsALT_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVBVvsALT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVBVvsALT

%VARAIBLES GLOBALES
global H1;
global H2;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set (legend(H1));
    set (legend(H2));
    legend(handles.SCVBVALG1, 'boxoff');
    set(handles.leyendaCVBVvsALT, 'String', 'LEYENDA ON');
else
    legend(handles.SCVBVALG1, 'off');
    delete (legend(H2));
    set(handles.leyendaCVBVvsALT, 'String', 'LEYENDA OFF');
end

% --- Executes on button press in cuadrriculaCVBVvsALT.
function cuadrriculaCVBVvsALT_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrriculaCVBVvsALT (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% cuadrriculaCVBVvsALT

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA (ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVBVALG1,'off');
    set(handles.cuadrículaCVBVvsALT,'String','CUADRICULA OFF');
else
    grid(handles.SCVBVALG1,'on');
    set(handles.cuadrículaCVBVvsALT,'String','CUADRICULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (B)

VELOCIDAD-SUSPENSIÓN DELANTERA

```

function varargout = guideCvariableBvelocidadSuspD(varargin)
% GUIDECVARIABLEBVELOCIDADSPD M-file for
% guideCvariableBvelocidadSuspD.fig
%
%   GUIDECVARIABLEBVELOCIDADSPD, by itself, creates a new
%   GUIDECVARIABLEBVELOCIDADSPD or raises the existing
%   singleton*.
%
%   H = GUIDECVARIABLEBVELOCIDADSPD returns the handle to a new
%   GUIDECVARIABLEBVELOCIDADSPD or the handle to the existing
%   singleton*.
%
%   GUIDECVARIABLEBVELOCIDADSPD('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECVARIABLEBVELOCIDADSPD.M with the given input arguments.
%
%   GUIDECVARIABLEBVELOCIDADSPD('Property','Value',...) creates a
%   new GUIDECVARIABLEBVELOCIDADSPD or raises the existing
%   singleton*. Starting from the left, property value pairs are
%   applied to the GUI before
%   guideCvariableBvelocidadSuspD_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableBvelocidadSuspD_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableBvelocidadSuspD

% Last Modified by GUIDE v2.5 03-Sep-2014 20:18:54

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCvariableBvelocidadSuspD_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCvariableBvelocidadSuspD_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',    []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

```




```

% --- Executes just before guideCvariableBvelocidadSuspD is made
% visible.
function guideCvariableBvelocidadSuspD_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideCvariableBvelocidadSuspD
% (see VARARGIN)

% Choose default command line output for guideCvariableBvelocidadSuspD
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableBvelocidadSuspD wait for user response
% (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCvariableBvelocidadSuspD_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global OPENCB;
global ONB;
global VOPENB;
global SUSPDOPENB;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA
if ((OPENCB==1) && (ONB==1))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (B): SENSOR G.P.S. (VELOCIDAD)
    %Y SENSOR SUSPENSIÓN DELANTERA)
    set(handles.SCVBVT12, 'Data', VOPENB);
    set(handles.SCVBSDT2, 'Data', SUSPDOPENB);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (B): VELOCIDAD (VS)
    %SUSPENSIÓN DELANTERA)
    size(VOPENB);
    tam1=size(VOPENB);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,VOPENB,x,SUSPDOPENB);
    set(H1, 'DisplayName', 'OPENB VELOCIDAD VUELTA',...
        'YDataSource', 'OPENB VELOCIDAD VUELTA', 'LineStyle', '-');
    set(H2, 'DisplayName', 'OPENB SUSP.DELANTERA VUELTA',...

```



```

    'YDataSource','OPENB SUSP.DELANTERA VUELTA','LineStyle','--');
    set(get(AX(1),'Ylabel'),'String','Velocidad (Km/h)');
    set(get(AX(2),'Ylabel'),'String','Longitud (mm)');
    set(handles.SCVBVSDG1,'xgrid','on','ygrid','on');
    xlabel(handles.SCVBVSDG1,'Número de Muestras');
    title(handles.SCVBVSDG1,'Gráfica Variables (B):...
    Velocidad (Km/h) (VS) Suspensión Delantera (mm)',...
    'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close(handles.output);

% --- Executes on button press in leyendaCVBVvsSD.
function leyendaCVBVvsSD_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVBVvsSD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVBVvsSD

%VARIABLES GLOBALES
global H1;
global H2;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set(legend(H1));
    set(legend(H2));
    legend(handles.SCVBVSDG1,'boxoff');
    set(handles.leyendaCVBVvsSD,'String','LEYENDA ON');
else
    legend(handles.SCVBVSDG1,'off');
    delete(legend(H2));
    set(handles.leyendaCVBVvsSD,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrriculaCVBVvsSD.
function cuadrriculaCVBVvsSD_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrriculaCVBVvsSD (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% cuadrriculaCVBVvsSD

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA (ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVBVSDG1,'off');
    set(handles.cuadrículaCVBVvsSD,'String','CUADRICULA OFF');
else
    grid(handles.SCVBVSDG1,'on');
    set(handles.cuadrículaCVBVvsSD,'String','CUADRICULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (B)

VELOCIDAD-SUSPENSIÓN TRASERA

```

function varargout = guideCvariableBvelocidadSuspT(varargin)
% GUIDECVARIABLEBVELOCIDADXSUSPT M-file for
% guideCvariableBvelocidadSuspT.fig
%
%   GUIDECVARIABLEBVELOCIDADXSUSPT, by itself, creates a new
%   GUIDECVARIABLEBVELOCIDADXSUSPT or raises the existing
%   singleton*.
%
%   H = GUIDECVARIABLEBVELOCIDADXSUSPT returns the handle to a new
%   GUIDECVARIABLEBVELOCIDADXSUSPT or the handle to
%   the existing singleton*.
%
%   GUIDECVARIABLEBVELOCIDADXSUSPT('CALLBACK',hObject,eventData,
%   handles,...) calls the local function named CALLBACK in
%   GUIDECVARIABLEBVELOCIDADXSUSPT.M with the given input arguments.
%
%   GUIDECVARIABLEBVELOCIDADXSUSPT('Property','Value',...) creates a
%   new GUIDECVARIABLEBVELOCIDADXSUSPT or raises the existing
%   singleton*. Starting from the left, property value pairs are
%   applied to the GUI before
%   guideCvariableBvelocidadSuspT_OpeningFcn gets called. An
%   unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableBvelocidadSuspT_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableBvelocidadSuspT

% Last Modified by GUIDE v2.5 03-Sep-2014 20:32:37

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCvariableBvelocidadSuspT_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCvariableBvelocidadSuspT_OutputFcn, ...
                  'gui_LayoutFcn',  [], ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

```



```

% --- Executes just before guideCvariableBvelocidadSuspT is made
% visible.
function guideCvariableBvelocidadSuspT_OpeningFcn(hObject, eventdata,
handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to guideCvariableBvelocidadSuspT
% (see VARARGIN)

% Choose default command line output for guideCvariableBvelocidadSuspT
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableBvelocidadSuspT wait for user response
% (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = guideCvariableBvelocidadSuspT_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global OPENCB;
global ONB;
global VOPENB;
global SUSPTOPENB;
global H1; global H2,

%APERTURA DE SESIÓN GUARDADA
if ((OPENCB==1) && (ONB==1))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (B): SENSOR G.P.S. (VELOCIDAD)
    %Y SENSOR SUSPENSIÓN TRASERA)
    set(handles.SCVBVT123, 'Data', VOPENB);
    set(handles.SCVBSTT22, 'Data', SUSPTOPENB);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (B): VELOCIDAD (VS)
    %SUSPENSIÓN TRASERA)
    size(VOPENB);
    tam1=size(VOPENB);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,VOPENB,x,SUSPTOPENB);
    set(H1, 'DisplayName', 'OPENB VELOCIDAD VUELTA',...
        'YDataSource', 'OPENB VELOCIDAD VUELTA', 'LineStyle', '-');
    set(H2, 'DisplayName', 'OPENB SUSP. TRASERA VUELTA',...

```



```

    'YDataSource','OPENB SUSP.TRASERA VUELTA','LineStyle','--');
    set(get(AX(1),'Ylabel'),'String','Velocidad (Km/h)');
    set(get(AX(2),'Ylabel'),'String','Longitud (mm)');
    set(handles.SCVBVSTG1,'xgrid','on','ygrid','on');
    xlabel(handles.SCVBVSTG1,'Número de Muestras');
    title(handles.SCVBVSTG1,'Gráfica Variables (B):...
    Velocidad (Km/h) (VS) Suspensión Trasera (mm)',...
    'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close(handles.output);

% --- Executes on button press in leyendaCVBVvsST.
function leyendaCVBVvsST_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVBVvsST (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVBVvsST

%VARIABLES GLOBALES
global H1;
global H2;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set(legend(H1));
    set(legend(H2));
    legend(handles.SCVBVSTG1,'boxoff');
    set(handles.leyendaCVBVvsST,'String','LEYENDA ON');
else
    legend(handles.SCVBVSTG1,'off');
    delete(legend(H2));
    set(handles.leyendaCVBVvsST,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrriculaCVBVvsST.
function cuadrriculaCVBVvsST_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrriculaCVBVvsST (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
% cuadrriculaCVBVvsST

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA (ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVBVSTG1,'off');
    set(handles.cuadrículaCVBVvsST,'String','CUADRICULA OFF');
else
    grid(handles.SCVBVSTG1,'on');
    set(handles.cuadrículaCVBVvsST,'String','CUADRICULA ON');
end
```



FUNCIÓN GUIDE COMPARAR VARIABLES SESIÓN (B) VELOCIDAD-TEMPERATURA AMBIENTE

```

function varargout = guideCvariableBvelocidadtempambiente(varargin)
% GUIDECVARIABLEBVELOCIDADTEMPAMBIENTE M-file for
% guideCvariableBvelocidadtempambiente.fig
%
%   GUIDECVARIABLEBVELOCIDADTEMPAMBIENTE, by itself, creates a new
%   GUIDECVARIABLEBVELOCIDADTEMPAMBIENTE or raises the existing
%   singleton*.
%
%   H = GUIDECVARIABLEBVELOCIDADTEMPAMBIENTE returns the handle to
%   a new GUIDECVARIABLEBVELOCIDADTEMPAMBIENTE or the handle to the
%   existing singleton*.
%
%   GUIDECVARIABLEBVELOCIDADTEMPAMBIENTE('CALLBACK',hObject,
%   eventData,handles,...) calls the local function named CALLBACK
%   in GUIDECVARIABLEBVELOCIDADTEMPAMBIENTE.M with the given input
%   arguments.
%
%   GUIDECVARIABLEBVELOCIDADTEMPAMBIENTE('Property','Value',...)
%   creates a new GUIDECVARIABLEBVELOCIDADTEMPAMBIENTE or raises
%   the existing singleton*. Starting from the left, property
%   value pairs are applied to the GUI before
%   guideCvariableBvelocidadtempambiente_OpeningFcn gets called.
%   An unrecognized property name or invalid value makes property
%   application stop. All inputs are passed to
%   guideCvariableBvelocidadtempambiente_OpeningFcn via varargin.
%
%   *See GUI Options on GUIDE's Tools menu. Choose "GUI allows
%   only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
% guideCvariableBvelocidadtempambiente

% Last Modified by GUIDE v2.5 04-Sep-2014 20:43:48

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                  'gui_Singleton',  gui_Singleton, ...
                  'gui_OpeningFcn', @guideCvariableBvelocidadtempambiente_OpeningFcn, ...
                  'gui_OutputFcn',  @guideCvariableBvelocidadtempambiente_OutputFcn, ...
                  'gui_LayoutFcn',  [] , ...
                  'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

```




```

% --- Executes just before guideCvariableBvelocidadtempambiente is
% made visible.
function guideCvariableBvelocidadtempambiente_OpeningFcn(hObject,
eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
% varargin    command line arguments to
% guideCvariableBvelocidadtempambiente (see VARARGIN)

% Choose default command line output for
% guideCvariableBvelocidadtempambiente
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes guideCvariableBvelocidadtempambiente wait for user
% response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout =
guideCvariableBvelocidadtempambiente_OutputFcn(hObject, eventdata,
handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

%VARIABLES GLOBALES
clear global H1; clear global H2;

global OPENCB;
global ONB;
global VOPENB;
global TAOPENB;
global H1; global H2;

%APERTURA DE SESIÓN GUARDADA
if ((OPENCB==1) && (ONB==1))

    %TABLA 1 Y 2 (DATOS GUARDADOS SESIÓN (B): SENSOR G.P.S.(VELOCIDAD)
    %Y SENSOR TEMPERATURA AMBIENTE)
    set(handles.SCVBVT1234, 'Data', VOPENB);
    set(handles.SCVBTAT22, 'Data', TAOPENB);

    %GRÁFICA 1 (VARIABLES GUARDADAS SESIÓN (B): VELOCIDAD (VS)
    %TEMPERATURA AMBIENTE)
    size(VOPENB);
    tam1=size(VOPENB);
    tamx=tam1(:,1);
    x=1:tamx;
    [AX,H1,H2] = plotyy(x,VOPENB,x,TAOPENB);
    set(H1, 'DisplayName', 'OPENB VELOCIDAD VUELTA', ...

```



```

'YDataSource','OPENB VELOCIDAD VUELTA','LineStyle','-');
set(H2,'DisplayName','OPENB TEMPERATURA VUELTA',...
'YDataSource','OPENB TEMPERATURA VUELTA','LineStyle','-');
set(get(AX(1),'Ylabel'),'String','Velocidad (Km/h)');
set(get(AX(2),'Ylabel'),'String','Temperatura Ambiente [T(C°)]');
set(handles.SCVBVTAG1,'xgrid','on','ygrid','on');
xlabel(handles.SCVBVTAG1,'Número de Muestras');
title(handles.SCVBVTAG1,'Gráfica Variables (B):...
Velocidad (Km/h) (VS) Temperatura Ambiente [T(C°)]',...
'FontSize',12,'FontName','MS Sans Serif','FontWeight','bold');
end

% --- Executes on button press in MC.
function MC_Callback(hObject, eventdata, handles)
% hObject    handle to MC (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%LLAMADA A SU CORRESPONDIENTE GUIDE (INTERFAZ GRÁFICA) Y CIERRE DEL
%ACTUAL GUIDE
guideScompararsesion;
close(handles.output);

% --- Executes on button press in leyendaCVBVvsTA.
function leyendaCVBVvsTA_Callback(hObject, eventdata, handles)
% hObject    handle to leyendaCVBVvsTA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of leyendaCVBVvsTA

%VARIABLES GLOBALES
global H1;
global H2;

%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN LEYENDA(ON,OFF)
o=get(hObject,'Value');
handles.o=o;

if handles.o==1
    set(legend(H1));
    set(legend(H2));
    legend(handles.SCVBVTAG1,'boxoff');
    set(handles.leyendaCVBVvsTA,'String','LEYENDA ON');
else
    legend(handles.SCVBVTAG1,'off');
    delete(legend(H2));
    set(handles.leyendaCVBVvsTA,'String','LEYENDA OFF');
end

% --- Executes on button press in cuadrriculaCVBVvsTA.
function cuadrriculaCVBVvsTA_Callback(hObject, eventdata, handles)
% hObject    handle to cuadrriculaCVBVvsTA (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
%cuadrriculaCVBVvsTA

```



```
%EVALUACIÓN SOBRE EL ESTADO DEL BOTÓN CUADRÍCULA (ON,OFF)
o2=get(hObject,'Value');
handles.o2=o2;

if handles.o2==1
    grid(handles.SCVBVTAG1,'off');
    set(handles.cuadriculaCVBVvsTA,'String','CUADRICULA OFF');
else
    grid(handles.SCVBVTAG1,'on');
    set(handles.cuadriculaCVBVvsTA,'String','CUADRICULA ON');
end
```

