Sistema de Monitorización de Turbinas de Gas Aeronáuticas

Departamento Ingeniería de Sistemas y Automática

Escuela Técnica Superior de Ingenieros
Universidad de Sevilla

Mayo de 2007

María Rodríguez Barquero

AGRADECIMIENTOS

Me gustaría agradecer a mi tutor en Inglaterra, el profesor Peter Fleming, por darme la oportunidad de hacer mi proyecto final de carrera con el grupo *Rolls-Royce UTC* del departamento *Automatic Control and Systems Engineering* de la Universidad de Sheffield. Ha sido un honor para mi.

Gracias a Andrew Mills, investigador del grupo y también mi tutor, por su atención y sus valiosos consejos. Siempre dispuesto a ayudarme en lo necesario y a animarme para que siguiera trabajando duro, ha sido para mi un gran apoyo. Por supuesto, gracias también por su paciencia en mi lucha con el idioma.

También me gustaría mencionar al profesor Haydn Thompson cuya ayuda ha resultado muy útil durante el desarrollo del proyecto. Haydn me proporcionó el *hardware* necesario a la vez que contribuyó en el proyecto con sus ideas.

Gracias a todos los miembros del laboratorio B20 del grupo *Rolls-Royce UTC*. El buen ambiente de trabajo y su amable ayuda hicieron agradable el tiempo que trabajé con ellos.

Me gustaría agradecer a mi tutor en España, el profesor Eduardo Fernández Camacho, por permitirme llevar a cabo mi proyecto en Sheffield. He mejorado mi inglés a la vez que he vivido una importante experiencia en mi vida. También me dio la oportunidad de aprender sobre un tema desconocido anteriormente para mi y que me ha resultado muy interesante. Además pude compartir todo esto con unos maravillosos compañeros en el departamento y con toda la gente que la experiencia me ha permitido conocer.

Por último, pero no por ello menos importante, me gustaría agradecer a mi familia por su cariño y su infinito apoyo. En el mismo sentido, agradecer a los que siempre han estado ahí, aún en la distancia.

ÍNDICE

1. Introducción....1
   1.1. Situación....1
   1.2. Descripción del Proyecto....2
   1.3. Motivación del Proyecto....2
   1.4. Monitorización de Fallos....3
   1.5. Estructura del Proyecto....4
2. Turbinas de Gas....5
   2.1. Introducción a las Turbinas de Gas....5
   2.2. Funcionamiento de una Turbina de Gas....8
3. Detección y Aislamiento de Fallos en Turbinas de Gas....13
   3.1. Detección y Aislamiento de Fallos....13
   3.2. Método para Diagnóstico de Fallos....14
   3.3. Consideraciones Mecánicas....15
   3.4. Variables a Medir....15
   3.5. Monitorización de la Vibración....16
4. Tecnología *Wireless*....17
   4.1. Introducción....17
   4.2. Comunicaciones Inalámbricas en la Industria Aeronáutica....18
   4.3. Comunicación Inalámbrica en el Ámbito Industrial....19
   4.4. Protocolos y Arquitecturas....22
      4.4.1. Modelo OSI....22
      4.4.2. *WLAN*....24
   4.5. Algunos Estándares Inalámbricos....25
5. Equipo Utilizado....28
   5.1. Breve descripción del Equipo Utilizado....28
   5.2. Sistema Inalámbrico....29
      5.2.1. Problema....33
      5.2.2. Pruebas Realizadas....34
         5.2.2.1. Prueba 1....34
         5.2.2.2. Prueba 2....35
         5.2.2.3. Prueba 3....36
         5.2.2.4. Prueba 4....37

5.2.2.5. Prueba 5....38
5.2.2.6. Otras Observaciones....39
5.2.3. Solución Adoptada por el Fabricante....39
5.3. Bus CAN....39
5.3.1. Principales Características de CAN....41
5.3.2. Proceso de Transmisión de Datos....41
5.3.3. Especificación CAN 2.0A y CAN 2.0B. Trama de Datos....42
6. Descripción de la Aplicación....44
6.1. *Software*....44
6.2. Desarrollo de la Aplicación....45
6.3. Paso a Paso....46
6.3.1. *Write – Read*....46
6.3.2. Solo *Read* con *Excel*....51
6.3.3. Aplicación Definitiva....55
7. Análisis de los Datos Recogidos....64
7.1. Introducción....64
7.2. Representación de los Datos Recogidos....65
7.3. Descripción de las Plataformas Utilizadas....67
7.4. Análisis de Datos....70
7.4.1. Recogidos por Sensores Cableados....70
7.4.2. Recogidos por Sensores Inalámbricos....75
7.4.3. Comparación....81
8. Conclusiones y Trabajo Futuro....88
8.1. Conclusiones....88
8.2. Trabajo Futuro....89

Capítulo 1

Introducción

1.1. Situación

Este trabajo se ha desarrollado en colaboración con la Universidad de Sheffield, en el departamento *Automatic Control and Systems Engineering*. En 1993, se estableció un Centro Universitario de Tecnología (UTC, *University Technology Centre*) en dicho departamento, promovido por la empresa *Rolls-Royce*. Se trata de un centro que desarrolla diversos programas de investigación en cooperación con *Rolls-Royce* teniendo como objetivo investigar en los campos tecnológicos de interés para la empresa en el contexto de la automática.

Rolls-Royce obtiene diversos beneficios al haber establecido este grupo:

- Tiene acceso a tecnología punta y especialistas apropiados para el negocio.

- Maximiza la efectividad en la transferencia de la tecnología incluyendo la captación de nuevos empleados.

- Maximiza el aprovechamiento de su inversión en investigación.

La universidad a su vez se beneficia de tener un acceso directo a la industria desde el punto de vista de impulsar tanto los avances tecnológicos desarrollados como a sus estudiantes.

Dentro del grupo, en el departamento se establecen a su vez subgrupos que se centran en distintos campos de la automática:

- *Architectures and Hardware*: diseño de arquitecturas y análisis de nuevas tecnologías.

- *Advanced Control Law Strategies*: estrategias de control para nuevas máquinas, mejorando el desarrollo y la eficiencia del diseño.

- *Prognostics and Aftermarket*: detección de fallos en fabricación o en servicio y gestión.

El trabajo que aquí se presenta se engloba en el ultimo subgrupo *Prognostics and Aftermarket*. En este subgrupo se trata de maximizar el ciclo de vida de las máquinas mediante la detección de fallos. Para ello se usan diversas tecnologías, se almacenan históricos de datos y se cuenta con expertos en el análisis de los mismos.

1.2. Descripción del Proyecto

El objetivo del presente proyecto de final de carrera es el desarrollo de una aplicación informática para recibir, descodificar, transformar, almacenar y analizar datos tomados por sensores inalámbricos instalados en turbinas de gas de aviones.

El trabajo se engloba en un amplio campo que en inglés llaman *Health Monitoring*. Se trata de monitorizar y automatizar todo el proceso de observación de un motor para comprobar su correcto funcionamiento. De esta forma, se puede detectar cualquier anomalía o variación de las condiciones nominales de funcionamiento, pudiendo detectar así todo tipo de fallos.

En este documento se analizará el desarrollo de dicha aplicación así como las diversas pruebas realizadas para comprobar su correcto funcionamiento, tanto en simulación como con motores rotativos reales instalados en plataformas de prueba.

1.3. Motivación del Proyecto

Un problema de interés para el mantenimiento de motores de aviones es la detección automática, clasificación y predicción de fallos críticos en los componentes de las turbinas de gas. La monitorización automática ofrece la posibilidad de reducir sustancialmente el coste de mantenimiento, a la vez que puede contribuir en salvar vidas.

La detección y el aislamiento de fallos en turbinas de gas son cruciales, y pueden considerarse como una parte integral dentro de la seguridad en la aviación. La capacidad de detectar fallos cuando éstos están empezando a desarrollarse puede evitar situaciones potencialmente peligrosas.

1.4. Monitorización de Fallos

Esta monitorización, en inglés *Health Monitoring*, se está llevando a cabo para ser utilizada principalmente en tres campos:

- Dentro del proceso de fabricación de turbinas de gas. Existe una última etapa en la que se llevan a cabo una serie de pruebas para comprobar si el funcionamiento es el deseado. Es en esta etapa en la que es posible monitorizar el estado de la turbina y de sus componentes tratando de detectar posibles fallos. De esta forma, sería posible arreglar la turbina o el componente en cuestión antes de que el daño fuera irreparable y hubiera que reemplazarla por completo.

- Una vez con la turbina en operación y mientras el avión está en servicio, es posible monitorizar el estado de las turbinas y sus componentes desde un PC en tierra. De esta forma, sería posible detectar ciertas variaciones y anomalías en los datos recibidos, pudiendo actuar antes de llegar a daños irreparables, contribuyendo de esta forma en la seguridad.

- La monitorización se está desarrollando teniendo como aplicación principal la reducción de costes de mantenimiento, lo cual sería posible mediante la habilidad de detectar posibles fallos tan pronto como se desarrollan y de planificar las fechas de mantenimiento más eficientemente.

1.5. Estructura del Proyecto

Este trabajo está dividido en ocho capítulos. El presente capítulo es el primero y trata de introducir y justificar el motivo de la realización del mismo.

Los capítulos 2, 3 y 4 tratan de introducir al lector en el tema en el que se trabaja. En el capítulo dos se da una visión general a lo que son las turbinas de gas, por qué se usan en los aviones y el funcionamiento de las mismas. En el capítulo tres se trata la detección y el aislamiento de fallos de una manera general y en particular en lo que a las turbinas de gas se refiere. Se explican de esta manera los métodos que existen para llevar a cabo dichas tareas y se plantea qué variables sería necesario monitorizar en el caso de las turbinas de

gas para poder tratarlos. Por último, en el capítulo cuatro se hace un repaso al estado del arte actual en el que se encuentra la tecnología *wireless*. Se resaltan las ventajas que ésta podría proporcionar al usarla en la aviación y se detallan algunos de los protocolos existentes en el mercado.

En los capítulos 5, 6 y 7 se desarrolla el trabajo llevado a cabo durante este proyecto. El capítulo cinco explica la tecnología utilizada así como ciertos problemas encontrados durante la utilización del equipo del que se disponía. En el capítulo seis se describe detalladamente la aplicación desarrollada y en el siete se presentan las distintas pruebas realizadas en laboratorio y los posteriores análisis realizados de los datos recogidos.

En el capítulo ocho se recogen las conclusiones y posibles ramas de trabajo futuro. Estas recomendaciones para el futuro pueden tratarse como continuación y mejora de algunos puntos de este proyecto.

Al final se añaden diversos apéndices en los que se recogen los distintos programas desarrollados durante este trabajo.

Capítulo 2

Turbinas de Gas

2.1. Introducción a las Turbinas de Gas

Las turbinas de gas modernas no son mas que un avance de lo que fue el primer modelo de reactor desarrollado por *Frank Whittle* y *Hans Joachin Pabst Von Ohation*, los cuales trabajaron de manera independiente a principios de la década 1930.

Como en todo motor térmico, al expandir un fluido a alta presión y alta temperatura se obtiene un trabajo. Cuando dicho motor térmico es una turbina de gas el fluido que circula es un gas no condensable. A la turbina hay que añadirle los elementos necesarios para obtener la alta presión y la alta temperatura indicadas anteriormente.

Así, el proceso que se lleva a cabo en un reactor puede resumirse como un flujo de aire que entra por un compresor, para aumentar la presión, pasa por una cámara de combustión, para aumentar la temperatura, y después por una turbina, donde finalmente el aire es impulsado hacia atrás para obtener la potencia en el reactor.

Se trata de un motor térmico de combustión interna, por ello es una máquina mucho más compacta que si se tratara de un motor de combustión externa. Siendo ésta una de las principales ventajas de los motores de combustión interna respecto a los de combustión externa.

Las turbinas de gas han sido aplicadas a vehículos pero en la actualidad solo existe algún proyecto, como el Volvo ECC (híbrido eléctrico-turbina de gas).

Entre los problemas que dificultan su aplicación en automoción, se encuentra que aceptan mal los arranques y las paradas y les cuesta mucho cambiar de régimen (las turbinas de gas son muy lentas acelerando). De hecho, el funcionamiento habitual de las turbinas de gas es siempre al mismo régimen y las variaciones de demanda de potencia se hacen manteniendo el régimen y variando el par (fuerza de giro) generado.

La aplicación más común de estas máquinas es la propulsión de aviones a reacción, aprovechando así la compacidad del motor de combustión interna que tan necesaria es en aviación. De dichas máquinas de propulsión de aviones a reacción derivan las turbinas utilizadas en las centrales termoeléctricas para generación de energía eléctrica.

Respecto a la aplicación en aviación, cabe destacar que el uso de las turbinas de gas se refiere a la propulsión de los aviones. Es decir, la energía obtenida en la turbina se usa para obtener el empuje necesario para propulsar el avión y no otras fuerzas necesarias que quedan fuera de estudio en este trabajo.

El concepto básico del funcionamiento del motor de un avión está en la capacidad para producir un empuje que propulse el avión mediante el impulso de una gran masa de aire hacia detrás (*Rolls-Royce plc, 1986a*). Esto sucede porque la maquina está diseñada para acelerar una corriente de aire a alta velocidad para generar el empuje para propulsar el avión.

Figura 2.1. Mecanismo de Propulsión de un Reactor (Rolls-Royce plc 1986).

El proceso se basa en la tercera ley de Newton o la ley de acción-reacción según la cual “Con toda acción ocurre siempre una reacción igual y contraria; las acciones mutuas de dos cuerpos son iguales y dirigidas en sentidos opuestos”. El aire a presión atmosférica se inyecta por la parte delantera del motor, se comprime, se calienta y se expulsa por la parte trasera de la máquina para generar el empuje necesario para acelerar el avión.

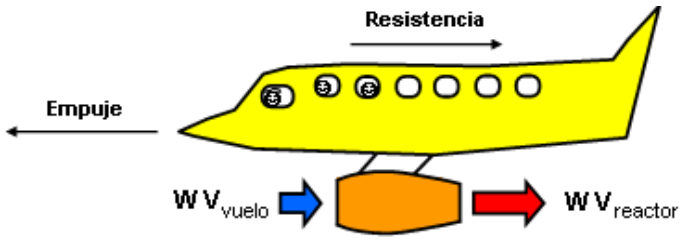


Figura 2.2. Propulsión en un avión.

El empuje viene dado por:

$$Empuje = W \cdot \left(V_{vuelo} - V_{reactor}\right)$$

Siendo el *Empuje* la fuerza de presión neta que se obtiene en la estructura, obtenido como el producto de la masa de aire, *W*, y la variación de velocidad en el reactor.

Cuando el avión está acelerando el *Empuje* es mayor que la *Resistencia*. Sin embargo, cuando lleva una velocidad constante el *Empuje* y la *Resistencia* tienen el mismo valor.

La forma más económica de propulsar un avión de pasajeros es una turbina de gas ya que proporcionan el impulso necesario a baja velocidad. Se trata de trabajar al máximo rendimiento aunque sea necesario sacrificar el empuje obtenido. Sin embargo, en aviones de combate se busca el máximo empuje sacrificando así el rendimiento de la máquina.

Un ejemplo reciente de turbinas de gas usadas en aviones de pasajeros es el motor *Trent 800* desarrollado por *Rolls-Royce*, el cual se representa en la Figura 2.3 y se usa para propulsar el *Boeing 777*.

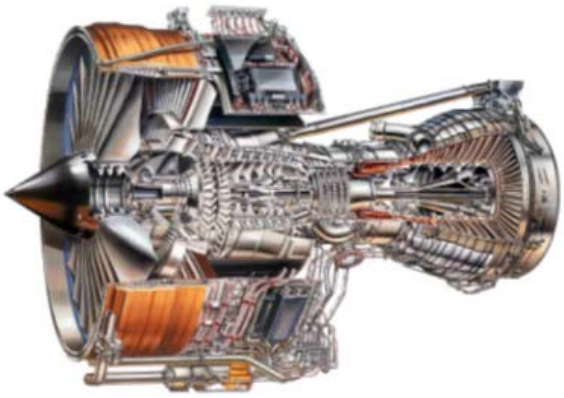


Figura 2.3. Rolls-Royce Trent 800.

2.2. Funcionamiento de una Turbina de Gas

Una turbina de gas es un motor térmico que desarrolla trabajo al expandir un gas. Se puede considerar un motor de combustión interna. Está compuesta por un compresor, una o varias cámaras de combustión y la turbina de gas propiamente dicha.

Se trata de un ciclo abierto. Existe una entrada continua de aire fresco de la atmósfera, aportándose la energía por medio de la combustión de un combustible en el seno de dicho aire. Los productos de esta combustión se expanden seguidamente en la turbina, siendo descargados a la atmósfera.

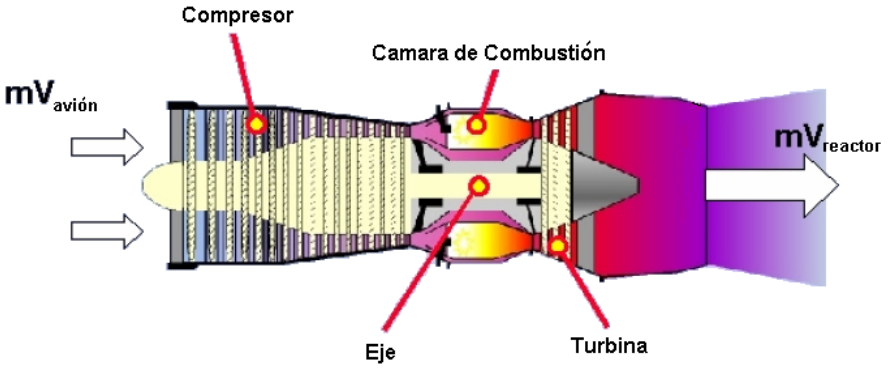


Figura 2.4. Turbina de Gas.

El ciclo termodinámico del gas en estas turbinas corresponde al ciclo *Brayton*. De manera ideal, el ciclo *Brayton* consiste en una compresión isentrópica, combustión a presión constante y finaliza con una expansión isentrópica volviendo a la presión de partida.

Más detalladamente, el ciclo de trabajo de una turbina de gas puede dividirse en cuatro fases: inducción, compresión, combustión y expansión. Cada una de ellas representada por una línea diferente en el diagrama (ver Figura 2.5):

- Línea 4-1: representa la inducción, fase en la que el aire es conducido al interior de la turbina de gas.

- Línea 1-2: representa la compresión, fase en la que se comprime el aire incrementando su presión.

- Línea 2-3: representa la combustión, fase en la que se quema combustible en el seno del aire comprimido a presión constante para elevar la temperatura del fluido.

- Línea 3-4: representa la expansión, fase en la que la mezcla caliente es expandida al pasar por la turbina, proporcionando el impulso y saliendo por la boquilla trasera.

El ciclo descrito es ideal y se aleja bastante del real cuantitativa y cualitativamente.

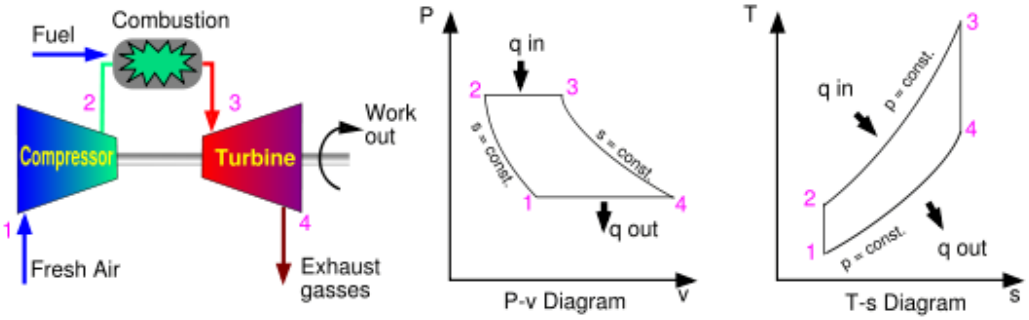


Figura 2.5. Ciclo de Brayton Ideal.

En la práctica, la fricción y las turbulencias, entre otros múltiples factores, dan lugar a:

- Compresión no isentrópica: en el compresor se obtiene una temperatura más alta de la ideal. Aunque el proceso de compresión es adiabático.

- Combustión con pérdida de presión. Además hay pérdida de rendimiento en la cámara de combustión debido a que la combustión no es completa, debido a defectos en la formación de la mezcla y al equilibrio químico, y a que hay pérdidas de calor a través de las paredes de la cámara.

- Expansión no isentrópica: aunque en la turbina la temperatura baja lo suficiente como para que el compresor no se dañe, la presión asociada es mayor, con lo cual la expansión disponible disminuye, disminuyendo así la cantidad de trabajo útil obtenido. Aunque el proceso de expansión, al igual que el de compresión, es adiabático.

- La presión disminuye en la entrada de aire y en la cámara de combustión hay gases de combustión que también aumentan de temperatura: esto disminuye la cantidad de trabajo útil obtenido.

Como en todos los motores térmicos, altas temperaturas de combustión dan lugar a una mayor eficiencia. El factor limitante en este sentido son los materiales de construcción de la maquina para que pudieran soportar las altas temperaturas y la presión. Es por eso que se trata de mantener las distintas partes de la turbina a no muy altas temperaturas, frecuentemente usando una

parte del mismo aire fresco atmosférico que entra para refrigerar. Existen mecanismos para conseguir que la temperatura suba por encima del límite impuesto por el material. Entre ellos: recubrimiento cerámico en los álabes, refrigerar por aire, agujerear los álabes, etc.

Existen diversas formas de mejorar el rendimiento de una instalación de turbina de gas. Algunas de ellas se basan en las mejoras en el diseño de la turbina, cámara de combustión y demás elementos auxiliares de la instalación. Otras, sin embargo, se refieren a aspectos termodinámicos del ciclo.

Haciendo referencia a los aspectos termodinámicos del ciclo:

- La mayoría de las turbinas tratan de reutilizar el calor a la salida de la turbina ya que, de otro modo, seria energía perdida. Los recuperadores son intercambiadores de calor que calientan el aire comprimido a la salida del compresor justo antes de la combustión. De esta manera, la energía necesaria a aportar en la combustión es menor. Se trata de un ciclo regenerativo. Para que la transmisión de calor sea posible es necesario disponer de un gradiente de temperaturas suficientemente elevado que compense el empleo de un intercambiador de calor.

  El problema de la regeneración es el aumento del peso al tener un elemento más, el intercambiador. Es por ello que esta técnica para mejorar el rendimiento de la instalación no se usa en aviación, en la que el peso es un factor determinante.

- Se puede enfriar el aire a la entrada del compresor para tener que realizar menos trabajo en la compresión.

- Ciclo compuesto con expansión escalonada: se trata de expandir un poco, calentar el producto de dicha expansión y volver a expandir tras el calentamiento.

- Ciclo compuesto regenerativo: se trata de comprimir el aire, enfriar dicho aire comprimido, volver a comprimir, calentar el aire comprimido antes de la combustión para tener que aportar menos energía en la combustión, expandir un poco los productos resultantes de la combustión, calentar y volver expandir la mezcla recalentada.

- Ciclos combinados: se trata de aumentar el rendimiento económico de la planta mediante la utilización de plantas basadas en la combinación de dos ciclos termodinámicos que trabajan con el mismo o distinto fluido. Con esta disposición se intenta aumentar la temperatura termodinámica media de aportación de calor y/o disminuir la de cesión de calor. En el caso particular de las turbinas de gas se combina el ciclo con una turbina de vapor. De esta forma, la aportación de energía en la turbina de gas se

hace a alta temperatura y la cesión de calor en la turbina de vapor se realiza a temperatura baja (relativamente próxima a la del ambiente). Se trata de usar la temperatura de los gases de escape de la turbina de gas para producir vapor. La temperatura de los gases de escape de las turbinas de gas modernas es suficientemente alta como para producir un vapor de razonable calidad.

- Así mismo, en cogeneración se reutiliza el calor perdido para calentar agua.

Mecánicamente las turbinas de gas son considerablemente menos complejas que los pistones de combustión interna. Las turbinas más simples pueden tener una sola parte móvil (ver Figura 2.5), sin contar el sistema de inyección del combustible en la cámara de combustión.

Cuando se requiere que la turbina de gas funcione en condiciones de velocidad y carga fijas, como sucede en las centrales de punta, resultará adecuado un montaje en un solo eje. En este caso carecen de importancia la flexibilidad de funcionamiento, es decir la rapidez con que la máquina se adapta por sí misma a las variaciones de carga y de régimen, y el rendimiento a cargas parciales. Una ventaja evidente de este montaje es su elevada inercia, consecuencia del arrastre del compresor, pues disminuye así el peligro de que se alcancen velocidades excesivas en el caso de una eventual pérdida de carga eléctrica.

Sin embargo, cuando es de capital importancia una gran flexibilidad de funcionamiento, como sucede en aplicaciones de automoción, ferroviarias y marinas, es aconsejable el uso de una “turbina de potencia” o “libre” independiente mecánicamente del resto del conjunto. En este tipo de montaje en dos ejes, la turbina de alta presión mueve el compresor actuando la combinación de ambos como “generador de gas” para la turbina de potencia de baja presión. Los montajes en dos ejes se emplean también en grupos grandes para la generación de energía eléctrica, diseñándose la turbina de potencia para que gire a la velocidad del alternador sin necesidad de usar una reductora. Otra ventaja, aunque de menor importancia, es que el motor de arranque puede ser de menor tamaño al tener que mover solo el generador de gas. Existe, sin embargo, el peligro potencial de que un corte de la carga eléctrica puede hacer que la turbina de potencia se sobrerrevolucione por lo cual será necesario disponer de un adecuado sistema de control.

Como regla general, cuanto más pequeña es la máquina, mayor tendrá que ser la velocidad de rotación del eje para mantener la velocidad del diámetro exterior del rotor. Dicha velocidad del diámetro exterior del rotor determina la máxima presión que se puede obtener, independientemente del tamaño de la máquina. Así, los reactores operan alrededor de 10.000 rpm y las microturbinas alrededor de 100.000 rpm.

Desde el punto de vista de un sistema de control, una turbina de gas puede dividirse en tres grandes partes: el motor, los accesorios y el controlador electrónico. El motor consiste en el compresor, la cámara de combustión, turbina y carcasas. Los accesorios se encuentran alrededor del motor y entre ellos se puede nombrar el sistema de inyección de combustible en la cámara de combustión. El controlador electrónico permite interrelacionar la turbina de gas con el resto de los sistemas de control del avión, pudiendo controlar el empuje y proporcionando información acerca del estado de la turbina de gas, pudiendo detectar fallos y anomalías en la misma.

Capítulo 3

Detección y Aislamiento de Fallos en Turbinas de Gas

3.1. Detección y Aislamiento de Fallos

En muchas aplicaciones industriales, especialmente en aquellas en las que la seguridad es un factor crítico, ser capaces de detectar un fallo es un indicador crucial de la continuidad del proceso. Por ejemplo, en la industria de la aviación, procesos químicos, fabricación del acero, etc.

A la vez que la complejidad de los modernos sistemas de control y el sofisticado desarrollo de los algoritmos de control aumentan, aspectos como la disponibilidad, eficiencia, coste, fiabilidad, seguridad de operación y protección del medioambiente van ganando en importancia. Conseguir detectar fallos cuando éstos se están empezando a desarrollar puede contribuir a evitar averías en los sistemas, abortar misiones y retrasos innecesarios que pueden llegar a costar grandes cantidades de dinero.

El término fallo puede definirse como un cambio inesperado en el funcionamiento del sistema, a causa de deterioro o de una operación anormal. Los modernos sistemas de detección de fallos intentan minimizar las actividades de mantenimiento no programadas mediante la prevención y el aislamiento.

Un sistema de diagnóstico de fallos es un sistema monitorizado para detectar fallos tan pronto como se estén desarrollando y localizarlos en el sistema para prevenir serias consecuencias.

Las tareas que normalmente se engloban en un sistema de monitorización consisten en los siguientes puntos (*Kurz et al., 2001*), de los cuales los dos primeros suelen englobarse como uno solo llamado Diagnóstico de Fallos:

1. Detección de Fallos: detectar las condiciones de operación.

2. Aislamiento de Fallos: localizar el fallo.

3. Identificación de Fallos: identificar el tamaño y el tipo de fallo.

El Diagnóstico de Fallos contribuye de manera importante en el sistema de control de fallos. Sirve para detener o retener una aplicación concreta en caso de fallo, mientras se procede a tomar la decisión adecuada para reanudar el funcionamiento correcto acorde con el fallo. Por ejemplo, en el sistema de control de vuelo para mantener la seguridad del avión en condiciones de fallos.

3.2. Métodos para Diagnóstico de Fallos

Tradicionalmente el Diagnóstico de Fallos se ha enfocado desde dos puntos de vista diferentes, atendiendo al momento en el que se realiza dicho diagnostico:

- Después del incidente: Mediante este método se trata de detectar fallos una vez que el incidente ya ha ocurrido. Con ello se intenta analizar la degradación gradual antes de que el fallo ocurriera. El ligero deterioro causado por la degradación crecerá y éste puede producir el fallo del sistema si no se detecta pronto. Ese ligero deterioro puede encontrarse dentro de las tolerancias, pero habría que prestarle una mayor atención y tener en cuenta la posibilidad de que crezca hasta convertirse en un fallo importante. Esto es útil para realzar ciertos componentes en el mantenimiento lo cual puede llevar a una reducción del coste.

- Método *online*: Se trata de monitorizar cualquier desviación de las condiciones normales de operación de los parámetros de la planta. Si se detecta alguna condición fuera de lo normal se procederá a actuar, tomando las decisiones necesarias mientras la planta continúa en funcionamiento. Si fuera necesario se podría detener la planta o reconfigurarla para que pueda seguir en funcionamiento con las nuevas condiciones de operación. Así mismo, se podría detener por completo si se considera que dichas condiciones llevarán a daños mayores.

La aplicación desarrollada en este proyecto se centra en el método online. Se trata pues de una aplicación orientada a la recogida, tratamiento y análisis de

datos para monitorizar el estado de las turbinas de gas cuando el avión se encuentra en operación.

3.3. Consideraciones Mecánicas

Como se ha detallado en el capítulo anterior, una turbina de gas está formada por varios componentes. La degradación de cualquiera de esos componentes afectará al rendimiento del sistema completo, aunque se produzca en uno de los componentes en particular. Principalmente, la degradación en las turbinas de gas suele ser una degradación mecánica, como cambios en las superficies de los álabes debido a la erosión, efectos que pueden ocasionar cambios de hermeticidad o del tamaño de las holguras, efectos de flujos parásitos, etc.

En particular, los componentes aerodinámicos como el compresor, las turbinas, el eje, bombas, etc., tienen que operar en ambientes que terminarán degradando su funcionamiento. Analizando los datos de cada componente por separado se pueden anticipar los efectos de la degradación y separarlos de posibles fallos. Se puede conseguir así un diagnóstico robusto del comportamiento del motor.

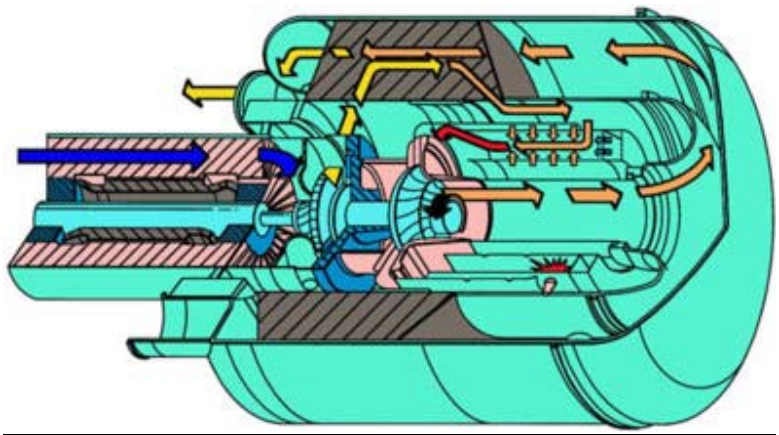


Figura 3.1. Turbina de Gas.

3.4. Variables a medir

Las variables que suelen medirse en los motores son: temperaturas y presiones de los gases en su recorrido por el motor, velocidad del rotor, flujo de combustible, posición de las válvulas reguladoras, posición de las boquillas, posición del estator, propiedades del aceite, vibración y tiempo que lleva el motor trabajando. Para el diagnóstico de la vibración han resultado ser buenos

indicadores del estado de los motores las señales promediadas y la vibración de cada componente. Así encontramos una razón para pensar que la variable a medir sea la vibración.

Los fallos mecánicos más comunes, incluyendo desequilibrio y desalineación, se originan en el rotor y provocan cambios en el eje de vibración. Esta vibración puede extenderse a la carcasa pero esto dependerá del ratio de transferencia de la vibración que tenga el motor. Por experiencia, se ha demostrado que la transmisión de la vibración no es constante a lo largo del rotor y que puede variar debido a la naturaleza de la fuente de vibración y a la velocidad de la máquina. Esto hace que sea complicado monitorizar el estado de los rotores y que, por tanto, el sistema de monitorización sea complejo. Esta complejidad hace que el sistema sea más susceptible a falsas alarmas y/o pérdida de detección de un verdadero fallo en el motor.

Para lograr una información fiable del motor el sistema de monitorización tiene que ser simple. La medida directa de la vibración del rotor puede jugar un papel importante en la búsqueda de la fiabilidad de la información ya que puede permitir que el sistema de monitorización sea más simple. Si se pueden obtener medidas directas, éstas no dependerán del ratio de transferencia de la vibración que tenga el motor, el cual rara vez es constante.

3.5. Monitorización de la Vibración

La monitorización de la vibración trata de identificar condiciones peligrosas a todas las velocidades de operación del motor evitando los daños secundarios causados por la degradación de los componentes del motor.

La monitorización de la vibración se divide en dos partes: una a bordo y otra en tierra. La parte que está a bordo compara las amplitudes de vibración medidas en partes específicas del motor con límites predefinidos (absolutos y relativos). Cuando la amplitud de la vibración rebasa el límite, se manda un mensaje de aviso a la cabina del piloto, el sistema de monitorización o el sistema de mantenimiento. La parte en tierra procesa los datos de la parte de a bordo ayudándose de unos algoritmos más sofisticados y modelos. De esta manera, se monitoriza la tendencia de la vibración proporcionando avisos de anomalías cuando éstas están desarrollándose, como pueden ser: desequilibrio, desgaste y rozamiento.

Los datos de vibración se recogen desde la parte de a bordo mediante acelerómetros que pueden ir montados en la carcasa del motor, bridas o engranajes. Estos acelerómetros típicamente cubren varios rangos de frecuencias.

Capítulo 4

Tecnología *Wireless*

4.1. Introducción

Hasta hace poco, toda la tecnología utilizada tanto en el ámbito industrial como en cualquier otro, ha sido básicamente cableada. Pero dichas tecnologías están siendo sustituidas por las inalámbricas.

Existen diferentes tecnologías que utilizamos cada día dependiendo de la aplicación a la que estén destinadas. No sólo el teléfono móvil se ha convertido en imprescindible, para muy corto alcance se está imponiendo el uso de Bluetooth, por ejemplo. Existen multitud de dispositivos como PDAs, manos libres, vehículos, etc. que ya disponen de esta tecnología. Por otro lado, si queremos disponer de comunicaciones en un ámbito local sin necesidad de cables, existen diferentes tecnologías como las redes *WiFi*. En definitiva, se descubren las ventajas de los entornos inalámbricos y aparecerán en un futuro próximo nuevas aplicaciones, incluso algunas aún no imaginadas.

Una vez introducido este tipo de tecnologías en la sociedad, comienzan a aparecer sistemas y servicios basados en tecnologías inalámbricas, mejorándose los procedimientos que tradicionalmente requerían una interacción directa con el ser humano y pueden ahora realizarse de forma distribuida por medio de sistemas gestores inteligentes. Concretamente, desde hace algunos años, han comenzado a emerger las “Redes de Sensores”. Los sensores son fuentes de información tan variados como lo son las medidas que realizan. Los hay de temperatura, de luminosidad, de presión, de humedad, de velocidad, de aceleración, de presencia, de volumen y un sin fin de magnitudes que se nos ocurran. Si a estos sensores que nos reportan información valiosa para nuestras

vidas, les añadimos la capacidad de comunicación inalámbrica y la posibilidad de formación de redes *ad-hoc*, obtenemos las "Redes de Sensores Inalámbricos", que están teniendo un auge cada vez mayor debido principalmente a la multitud de aplicaciones que se están desarrollando.

Encontramos estas aplicaciones sin más que mirar a nuestro alrededor: la contaminación de una ciudad se mide con unos sensores de polución, de CO_2, ozono, etc. que, envían los datos en tiempo real y sin necesidad de ningún tipo de interacción por parte humana a un centro de control, y cuando ésta sobrepasa ciertos niveles, se generan unas alarmas para que se tomen las medidas oportunas. En los edificios existen sensores de presencia, que se conectan con el centro de control y emiten alarmas cuando detectan presencias en momentos en que no debería haber nadie. También esta información procedente de los sensores puede ser procesada y provocar una acción correctora. Así por ejemplo en una bodega, con sensores de humedad repartidos por toda la planta, cuando se detecta alguna zona en la que la humedad supera cierto umbral se encienden unos ventiladores en caso de exceso de humedad o se encienden unos microaspersores en caso de defecto de la misma.

Tender cables y líneas de datos en plantas muy extensas o sustituir cadenas de arrastre debido al desgaste no son tareas sencillas ni económicas.

En estos casos, los sistemas inalámbricos pueden ser una alternativa muy rentable ya que reducen los tiempos de mantenimiento y aumentan la disponibilidad de las instalaciones, con el consiguiente aumento de la productividad. Otro aspecto importante de estos sistemas es la máxima fiabilidad de la transmisión de señales y datos, que es condición fundamental para el uso industrial.

En definitiva, las aplicaciones de este tipo de redes son múltiples, desde aplicaciones de seguimiento, de seguridad, de salud, de gestión, etc.

4.2. Comunicaciones Inalámbricas en la Industria Aeronáutica

En la actualidad algunos aviones usan sensores que miden vibración en sus turbinas de gas, pero ninguno de ellos usa sensores inalámbricos. El principal inconveniente de este tipo de sistemas radica en las restricciones que impone el cableado. Entre ellas se pueden citar:

- Complejidad en la instalación.

- Debido a esta complejidad se necesita un mayor tiempo para la instalación.

- Complejidad en el mantenimiento porque hay mas fuentes de error.

- Peso, lo cual es muy desfavorable en la aviación.

- Número de sensores limitado por el cableado, de esta manera se tiene menos resolución.

- Al tener menos resolución los resultados pueden ser menos fiables.

A pesar de ello, la información obtenida a través de los sensores sigue siendo de gran valor, pudiéndose aprovechar para diversos usos, entre los cuales destacamos principalmente dos de ellos:

- Avisar de una situación crítica al piloto o al sistema de control en tierra.

- O, lo que es más útil, para analizar la información recabada y los posibles cambios en la vibración. De esta manera, cabe la posibilidad de predecir posibles fallos y actuar a tiempo.

4.3. Comunicación Inalámbrica en el Ámbito Industrial

En los últimos años se ha producido un revolucionario desarrollo de sistemas de comunicación inalámbrica en los más diversos campos. Empresas analistas del mercado tecnológico indican que “la tecnología sin hilos tendrá un alto impacto en el mercado industrial en los próximos cinco años”. Así, impulsados por el rápido desarrollo de la tecnología, los dispositivos inalámbricos se están introduciendo gradualmente en el ámbito industrial. Pero, ¿cuál es el especial atractivo de la tecnología inalámbrica para aplicaciones y qué espera alcanzar la industria utilizando esta tecnología?

La respuesta es simple. La tecnología sin hilos tiene, principalmente, tres grandes ventajas respecto de las tecnologías tradicionales:

- Reduce costos de implementación, gracias a la facilidad y sencillez con que se montan estos equipos. Además, su ingeniería es mucho más simple y se reduce notablemente el costo de los materiales.

- Aumenta la productividad por la introducción del concepto de movilidad, flexibilidad y rápido acceso a las redes de comunicación.

- Permite desarrollar nuevos servicios, de marcado valor agregado, como por ejemplo los interfaces hombre-maquina portátiles, que facilita diagnósticos, controles e intervenciones sobre el proceso, al mismo

tiempo que brindan la posibilidad de monitoreo y seguimiento por parte del cliente.

Estas ventajas se explotan completamente en los nuevos equipos que están desarrollando los grandes proveedores tecnológicos. Hoy día, existe disponibilidad de todo tipo de sensores inalámbricos, tanto inductivos como capacitivos, que nos permiten realizar todos los tipos de aplicaciones tradicionales, con posibilidades de acceder a lugares y aplicaciones que hasta ahora eran muy difíciles o simplemente imposibles. Así por ejemplo, hoy día, se puede utilizar un equipo inalámbrico para configurar un dispositivo de terreno o para comprobar las características de funcionamiento de un motor en condiciones reales de la aplicación. Pero nadie había podido, hasta ahora, solucionar la compleja tarea de control, a circuito cerrado, con sistema de sensores inalámbricos y sin baterías. No obstante, ya existen elementos probados en el mercado, en diferentes aplicaciones, que nos demuestran que esto es posible.

A menudo, en las instalaciones técnicas de procesos, los datos de sensores solo pueden captarse con un coste elevado, ya que debe puentearse una distancia grande o un área de acceso difícil. Además, en plantas como, por ejemplo de la industria del cemento, en los dispositivos del suministro y captación de agua así como en las centrales eléctricas falta a menudo la infraestructura de comunicación para enlazar participantes adicionales dentro del margen de medidas de ampliación. Dado que las instalaciones crecen con el tiempo, aparecen a menudo problemas con líneas defectuosas. Utilizando sistemas inalámbricos pueden evitarse instalaciones de cables de alto coste así como fallos y costes de mantenimiento.

Así, hoy en día, podemos poner ejemplos de aplicaciones tan variados como:

- Instalación de escenarios, bastidores de teatros y cines en los que el montaje y desmontaje tiene que ser rápido.

- Instalaciones móviles o de materia prima, en las que los cambios de emplazamiento son muy corrientes.

- Ampliación de instalaciones existentes.

- Captación de sensores ampliamente distribuidos, como puede suceder en una depuradora.

- Instalaciones temporales, como por ejemplo instalaciones eléctricas necesarias durante el proceso de construcción de una instalación. Los sistemas inalámbricos aceleran el proceso de montaje y desmontaje a la vez que evitan el problema del desgaste de los conectores debido a la continua conexión y desconexión.

- Es típica la necesidad de poner en marcha una máquina o instalación grande o realizar trabajos de mantenimiento, siendo deseable el poder efectuar los trabajos libremente entre las unidades funcionales individuales sin molestias de poner cables intermedios. Esto es posible con sistemas *wireless* que permiten acoplar equipos terminales como PCs portátiles o PDAs. De esta manera, se tiene acceso en todo momento y en todos los lugares a todos los datos importantes, lo que supone un aumento de la disponibilidad de la instalación y, como resultado, también de la productividad.

- Muchas soluciones de visualización ya funcionan en terminales de operación móviles. A través de los equipos radiogobernados, el usuario puede acceder a las informaciones necesarias así como hacer el retorno *online* de datos y resultados de operaciones.

- Reducción del coste de mantenimiento del fabricante. Teniendo en cuenta que las máquinas se venden en todo el mundo, el fabricante puede controlar sus instalaciones y mantenerlas a distancia a través de un equipo inalámbrico.

- Especialmente la automatización de fábricas está caracterizada mediante partes de máquina continuamente en movimiento. El mejor ejemplo, son los robots que se emplean en todas las aplicaciones industriales para la preparación de productos. No obstante, lo que se mueve también se desgasta de forma rápida, así pues deben calcularse los correspondientes costes para piezas de repuesto y los costes debidos a los tiempos de paro.

  Alta movilidad es la condición fundamental para los dispositivos técnicos de transporte como, por ejemplo los sistemas de transporte sin conductor, ya que tienen que adaptarse flexiblemente a cada situación de pedido. Los equipos inalámbricos garantizan una comunicaron rápida y económica.

- En la industria del automóvil, las cadenas de arrastre y los anillos rozantes pueden sustituirse por procedimientos de transmisión inalámbrica que pueden montarse directamente en los robots.

- Los monocarriles aéreos están equipados con una gran cantidad de controles descentralizados y los sistemas inalámbricos aportan un máximo de movilidad.

Cada aplicación exige sus condiciones individuales en cuanto a la comunicación radiogobernada. No obstante, a todas las aplicaciones les es común que los componentes para la transmisión vía radio utilizados tienen que

ser igualmente sencillos en su operación, seguros y fiables al igual que los equipos previstos con cables.

La condición más importante para el uso de las tecnologías de radio en aplicaciones industriales es que deben funcionar también bajo las condiciones agresivas de forma tan robusta y fiable como una comunicación vía cable. En la comunicación inalámbrica los datos se transmiten por medio de ondas electromagnéticas utilizando el espacio libre como medio de transmisión que no se tiene a disposición de forma exclusiva. Por eso, la comunicación vía radio está expuesta a inducciones parásitas y campos parásitos que pueden causar perturbaciones en la transmisión. Además, pueden aparecer reflexiones, *fading*, interferencias y oscurecimientos.

A pesar de las acciones descritas, existen muchos protocolos desarrollados, como por ejemplo *Trusted Wireless*, *Bluetooth* o *WLAN 802.11*, que debido a su funcionamiento especial emiten sin interferencias. Esto se comprueba mediante extensas pruebas en la práctica.

De esta manera, se pueden obtener sistemas de arquitectura reducida, simplificando la instalación y el mantenimiento por el uso de la tecnología *wireless*. Por lo tanto, ésta es la tecnología que se utilizará, inicialmente en los países desarrollados, en todos los ámbitos de la industria productiva.

4.4. Protocolos y Arquitecturas

4.4.1. Modelo OSI

Para entender adecuadamente este apartado sería conveniente introducir el modelo OSI (*Open Systems Interconnection*). Se trata de un modelo de comunicaciones basado en una propuesta desarrollada por la organización de estándares internacional (ISO), por lo que también se le conoce como modelo ISO - OSI.

Este modelo tiene como función la de definir la forma en que se comunican los sistemas abiertos de telecomunicaciones, es decir, los sistemas que se comunican con otros sistemas.

El modelo de referencia consiste en 7 capas. Estas capas se visualizan generalmente como un conjunto de bloques apilados o "*stack of blocks*", por lo que en inglés a esto se le conoce como el "*OSI Protocol Stack*".

La Figura 4.1 muestra un esquema de dicho modelo de referencia y sus capas.

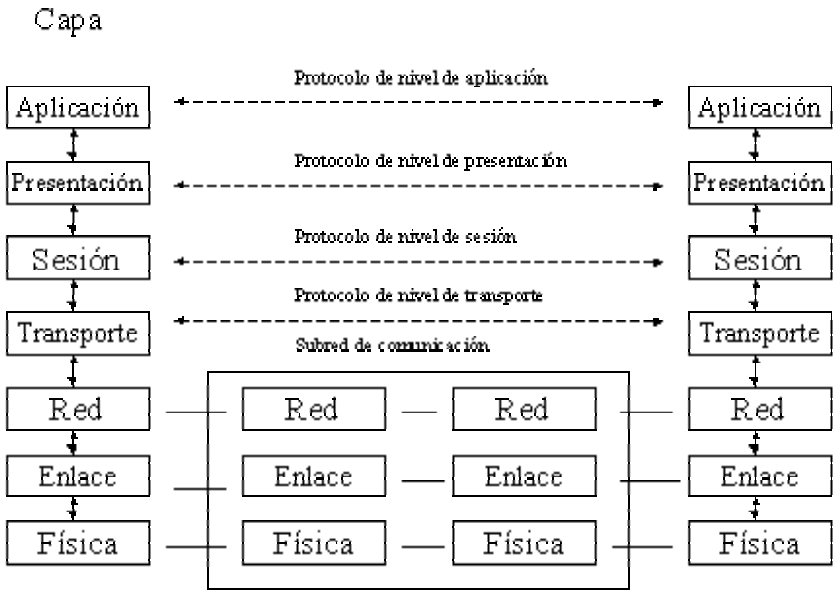


Figura 4.1. Arquitectura de Red Basada en el Modelo OSI.

En la Figura 4.2 se muestra una tabla que da una breve descripción de las capas del modelo de referencia OSI y las funciones de cada una de ellas.

Capa de Aplicación	Es el nivel último de la capa, el que aloja el programa de red que interactúa con el usuario.
Capa de Presentación	Maneja los datos de la aplicación y los acomoda en un formato que pueda ser transmitido en una red.
Capa de Sesión	Establece conexiones lógicas entre puntos de la red.
Capa de Transporte	Maneja la entrega entre un punto y otro de la red de los mensajes de una sesión.
Capa de Red	Maneja destinos, rutas, congestión en rutas, alternativas de enrutamiento, etc.
Capa de Enlace de Datos	Entrega los datos entre un nodo y otro en un enlace de red.
Capa Física	Define la conexión física de la red.

Figura 4.2. Breve Descripción de las Capas del Modelo OSI.

Algunas características interesantes a resaltar son:

- En este modelo, solo las capas que tengan otra capa equivalente en el nodo remoto podrán comunicarse, esto es, solo las capas que son iguales entre sí se comunican entre sí.

- El protocolo de cada capa se interesa solo por la información de dicha capa y no por la de las demás.

- La información se pasa a las capas de abajo hasta que se llega a la red. En el nodo remoto, la información es entonces pasada hacia arriba hasta que llega a la aplicación correspondiente. Cada capa confía en que las demás harán su trabajo, una capa no se interesa por el funcionamiento de las demás, lo único que le es de interés es la forma en como los datos serán pasados hacia arriba o hacia abajo.

- Se logra esto, empaquetando y desempaquetando información en los mensajes que se van a enviar.

4.4.2. *WLAN*

El concepto de *WLAN* (*Wireless Local Area Network*) se corresponde con un sistema de comunicaciones de datos flexible utilizado como alternativa a las redes cableadas. Este tipo de redes se diferencia de las convencionales principalmente en la capa física y en la capa de enlace de datos, según el modelo de referencia OSI.

La capa Física (PHY) indica cómo son enviados los bits de una estación a otra. La capa de Enlace de Datos (MAC) se encarga de describir cómo se empaquetan y verifican los bits de manera que no tengan errores. Las demás capas se encargan de los protocolos, de los puentes, encaminadores o puertas de enlace que se utilizan para conectarse.

Los dos métodos que se emplean para reemplazar la capa física en una red inalámbrica son la transmisión de Radio Frecuencia y la Luz Infrarroja.

Los sistemas por infrarrojos según el ángulo de apertura con que se emite la información, pueden clasificarse en:

- Sistemas de corta apertura, también denominados de rayo dirigido o de línea de visión (*LOS, line of sight*).

- Sistemas de gran apertura, también denominados reflejados o difusos.

Por otra parte, las comunicaciones inalámbricas que utilizan radiofrecuencia pueden clasificarse en:

- Sistemas de banda estrecha (*narrow band*) o de frecuencia dedicada. Este tipo trabaja de una forma similar a las ondas de una estación de

radio. Esta señal puede atravesar paredes por lo que puede alcanzar una red bastante amplia. Sin embargo, tienen problemas con las reflexiones que sufren las ondas de radio, para establecer esto hay que evitar las posibles interferencias.

- Sistemas basados en espectro disperso o extendido (*spread spectrum*). La FCC (Comisión Federal de Comunicaciones) a partir de 1985 permitió la operación sin licencia de dispositivos que utilicen 1 watio de energía o menos, en tres bandas de frecuencias: 902 a 928 MHz, 2400 a 2483.5 MHz y 5725 a 5850 MHz.

El desarrollo de protocolos y arquitecturas para el diseño de redes de sensores inalámbricos se ha convertido en un campo de investigación muy importante en los últimos años. El desarrollo de estas tecnologías ha venido de la mano de nuevos estándares inalámbricos de comunicación. Dentro de este nuevo escenario se prevé un futuro en el que minúsculos sensores monitoricen el medio continuamente y reporten su información a los nodos próximos y a una estación base central, formando una arquitectura computacional en rejilla (*grid*).

El número de situaciones en los que este tipo de tecnología es susceptible de aplicación es enorme, siendo algunos de los ejemplos típicos: domótica y control inteligente de edificios, medicina, agricultura, sistemas de control industrial, etc. Los requerimientos de estos sistemas están mayormente encaminados a situaciones donde no es necesario un gran ancho de banda, pero el posicionamiento de los nodos restringe su consumo de potencia, ya que muchas veces deberán estar alimentados por baterías.

4.4.3. Algunos Estándares Inalámbricos

Los estándares inalámbricos más extendidos hoy en día son IEEE 802.11, *Bluetooth* (o IEEE 802.15.1) y IEEE 802.15.4. Cada uno de ellos nació con un objetivo distinto y por lo tanto tiene un rango de aplicación y unas características diferentes.

El estándar WLAN 802.11 está desarrollado por el Instituto de Ingeniería Eléctrica y Electrónica IEEE 802.11. Describe las normas a seguir por cualquier fabricante de dispositivos *Wireless* para que puedan ser compatibles entre sí. El 802.11 es una red local inalámbrica que usa la transmisión por radio en la banda 2.4 GHz, o infrarroja, con regímenes binarios de 1 a 2 Mbit/s. El método de acceso al medio es mediante escucha pero sin detección de colisión, que se conoce como DFWMAC (*Distributed Foundation Wireless MAC*).

Los estándares más importantes son:

- IEEE 802.11a: hasta 54 Mbps de ancho de banda disponible, trabajando en la frecuencia de 5 GHz.

- IEEE 802.11b: hasta 11 Mbps. Éste es el más usual y el más utilizado, trabajando en la frecuencia de 2.4 GHz.

- IEEE 802.11g: futuro estándar hasta 54 Mbps, trabajando en la frecuencia de 2.4 GHz como IEEE 802.11a.

Al ser un estándar mundial, muchos fabricantes de hardware están creando equipos *Wireless* para poder conectar ordenadores, y van mucho más allá, utilizando *Wireless* para otras aplicaciones como pueden ser: servidores de impresión o cámaras *Web*.

Bluetooth es el nombre común de la especificación industrial IEEE 802.15.1, que define un estándar global de comunicación inalámbrica que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia segura, globalmente y sin licencia de corto rango. Los principales objetivos que se quiere conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles y fijos.

- Eliminar cables y conectores entre éstos.

- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre nuestros equipos personales.

La especificación de *Bluetooth* define un canal de comunicación de como máximo 720 kb/s (1 Mbps de capacidad bruta) con rango óptimo de 10 metros. Se podrían alcanzar alrededor de los 100 metros (similar a Wi-Fi) usando repetidores. Para mejorar la comunicación es recomendable que nada físico, como por ejemplo una pared, se interponga.

Existe un protocolo de comunicación inalámbrico muy similar al *Bluetooth* llamado *ZigBee*. Se usa principalmente en el ámbito de la domotica debido a su bajo consumo, su sistema de comunicaciones vía radio y su fácil integración. Las características que lo diferencian del *Bluetooth* son:

- Una red *ZigBee* puede constar de un máximo de 255 nodos, frente a los 8 máximos de una red *Bluetooth*.

- Menor consumo eléctrico que el ya de por sí bajo del *Bluetooth*.

- Ancho de banda 250 kbps, mientras que *Bluetooth* tiene 1 Mbps.

- Debido al ancho de banda de cada uno, uno es más apropiado que otro para ciertas aplicaciones. Por ejemplo, mientras que el *Bluetooth* se usa para aplicaciones como el *Wireless USB*, los teléfonos móviles y la informática casera, el ancho de banda del *ZigBee* se hace insuficiente para estas tareas, desviándolo para usos como los controles remotos, los productos dependientes de la batería, los sensores médicos y en artículos de juguetería, en los cuales la transferencia de datos es menor.

Resumiendo, en los estándares recogidos bajo la norma 802.11 se busca la interconexión de computadores con unas necesidades de ancho de banda elevadas de hasta 54 Mbps para el 802.11g.

El caso de *Bluetooth* está orientado hacia la implementación de redes inalámbricas de dispositivos periféricos con un ancho de banda menor, de hasta 1 Mbps. Se trata de la tecnología preferida para integrar componentes en redes de bus de campo o redes industriales *Ethernet*.

El problema que tienen estos dos estándares, 802.11 y *Bluetooth*, para convertirse en una tecnología usable a nivel de redes de sensores es la complejidad del protocolo y la rigidez de la topología de la red, así como el alto consumo de potencia de los *transceivers*.

Por su parte el estándar IEEE 802.15.4 está encaminado hacia el desarrollo de redes inalámbricas de baja velocidad, bajo coste y bajo consumo de potencia (LRWPAN). Este estándar está orientado a aplicaciones donde la velocidad de transferencia no es muy alta, pero permite que los nodos de la red puedan ser alimentados mediante baterías y puedan funcionar sin ser recargados durante años. Es por lo tanto el estándar que actualmente mejor se adapta a los requisitos impuestos para el desarrollo de redes de sensores.

Sin embargo, en el trabajo que aquí se presenta se ha utilizado una red de sensores inalámbricos con un protocolo de comunicación específico. No se trata de un protocolo estándar, sino de un protocolo desarrollado específicamente por el fabricante para comunicar los sensores con los receptores. No se conocen los detalles del funcionamiento de dicho protocolo, pero se sabe que trabaja a una frecuencia de 435 MHz.

Pensando en ahorrar en coste, tratamos de adaptar un *hardware* ya existente y diseñado para trabajar en automoción, a la aplicación en aviación que aquí se expone. Se trata de un tipo de sensores bastante sencillos pero que, según lo estudiado, contienen los requisitos necesarios que impone una red de estas características y es una tecnología perfectamente adaptable a dichas necesidades. Se comprueba así que, a veces, aunque el *hardware* fue

desarrollado para una aplicación concreta, el diseño puede ser utilizado en otras situaciones.

Capítulo 5

Equipo Utilizado

5.1. Breve descripción del equipo utilizado

En este capítulo se describirán en detalle cada una de las partes del equipo utilizado en este trabajo. Se trata de un sistema inalámbrico de sensores realizado con componentes comerciales de bajo coste.

El prototipo del que se parte está compuesto por elementos emisores, sensores, y receptores. Se alimentan mediante batería y directamente de la red eléctrica, respectivamente.

Para la activación y la desactivación de los sensores es necesario el uso de un dispositivo de disparo. Éste es activado a través del usuario.

Los sensores se encargan de tomar medidas y los receptores realizan las funciones básicas de proceso de datos recibidos. De esta forma, los receptores actúan de puente entre los sensores y el bus utilizado, que en este caso se trata de un bus CAN.

En el capítulo anterior se comentó que se ha tratado de adaptar un *hardware* existente de bajo coste a la aplicación en aviación que aquí se estudia. Dicho *hardware* fue diseñado para su uso en automovilismo, de ahí el uso del bus CAN, que es el más común en la industria de automoción.

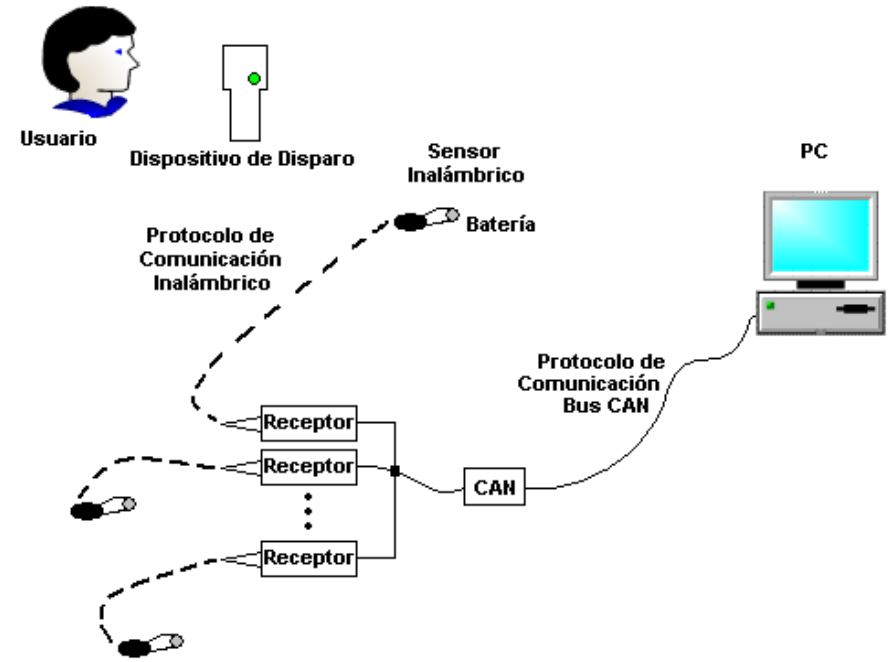


Figura 5.1. Esquema del Equipo Utilizado.

5.2. Sistema Inalámbrico

Se ha trabajado con sensores inalámbricos que están alimentados por baterías de 3 Voltios, consumiendo alrededor de unos 36 mW. En la Figura 5.2 se presenta una imagen real de uno de los sensores y la batería utilizada para alimentarlo.

Figura 5.2. Sensor Inalámbrico y Batería.

Estos sensores se engloban dentro de Sistemas Micro-Electro Mecánicos, mejor conocidos como *Micro Electro-Mechanical Systems* o MEMS. MEMS se refieren a la tecnología electromecánica micrométrica y sus productos. Esta tecnología puede ser implementada usando una gran cantidad de materiales diferentes y técnicas de fabricación. La elección depende del dispositivo que se fabrica y del sector del mercado en el que tiene que operar.

Entre los materiales utilizados para el desarrollo de aparatos MEMS los más comunes son los siguientes:

- Silicio: es el material utilizado para crear la mayoría de los circuitos integrados en la actualidad. La disponibilidad de materiales de alta calidad y la capacidad de incorporar funcionalidad electrónica hacen del silicio un material atractivo para una amplia variedad de aplicaciones en MEMS. Además tiene ventajas significativas debido a las propiedades del material. En forma de cristal simple, la silicio es un material casi perfectamente elástico, lo que significa que cuando se flexiona, idealmente, no hay histéresis y por lo tanto no hay disipación de energía. Esto hace que el silicio sea muy fiable si sufre a fatiga y puede estar en servicio entre billones y trillones de ciclos sin romperse. Las técnicas básicas de fabricación de dispositivos MEMS basados en silicio son deposición de capas de material, diseñando dichas capas mediante impresión y después grabando para producir las formas requeridas.

- Polímeros: A pesar de las ventajas que supone el uso de la silicio, producir silicio en forma cristalina es todavía bastante complicado y relativamente caro. Sin embargo los polímeros pueden ser producidos en grandes cantidades con una gran variedad de características en los materiales.

- Metales: Aunque los metales no tienen las ventajas mecánicas que proporciona la silicio, cuando se usan dentro de sus limitaciones, los metales pueden ser bastante fiables. Algunos de los metales que más se usan son: oro, níquel, aluminio, cromo, titanio, platino y plata.

En este trabajo se han usado sensores desarrollados con silicio. Estos sensores son capaces de dar medidas de vibración porque en su interior llevan distintas capas de silicio y un mecanismo electrónico. Explicando lo que un sensor lleva en su interior de una manera simple, se trata de una masa cuadrada sujetada por cuatro resortes. Al vibrar, los resortes se encogen y se estiran. De esta manera, se crean corrientes eléctricas diferentes dependiendo de la dirección en la que estén vibrando.

Es importante destacar que los sensores utilizados miden la vibración en las tres direcciones del espacio X, Y, Z, pero con distinta resolución. Así, la resolución en las direcciones X, Y es mayor que en la dirección Z, ya que los

resortes que sujetan la masa están alineados con las direcciones X, Y. De esta manera, se consigue más movilidad y, por tanto, más resolución en dichas direcciones. Será más sencillo de entender con el esquema representado en la Figura 5.3.

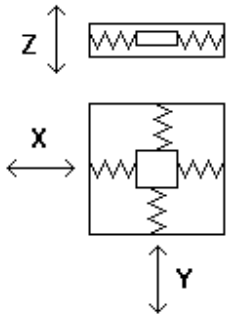


Figura 5.3. Esquema del Acelerómetro del Sensor.

Para la activación de los sensores se cuenta con un dispositivo de disparo. Básicamente, al pulsar el botón de dicho dispositivo se crea una onda electromagnética que hace saltar una chispa en el interior del sensor activando así dichas corrientes eléctricas. En la Figura 5.4 se presenta una imagen real de dicho dispositivo de disparo.

Figura 5.4. Dispositivo de Disparo.

Cada sensor tiene un identificador. Éste está indicado en el sensor mediante una etiqueta, como puede observarse en la Figura 5.2, y en los datos que manda dicho sensor en uno de sus *bytes*.

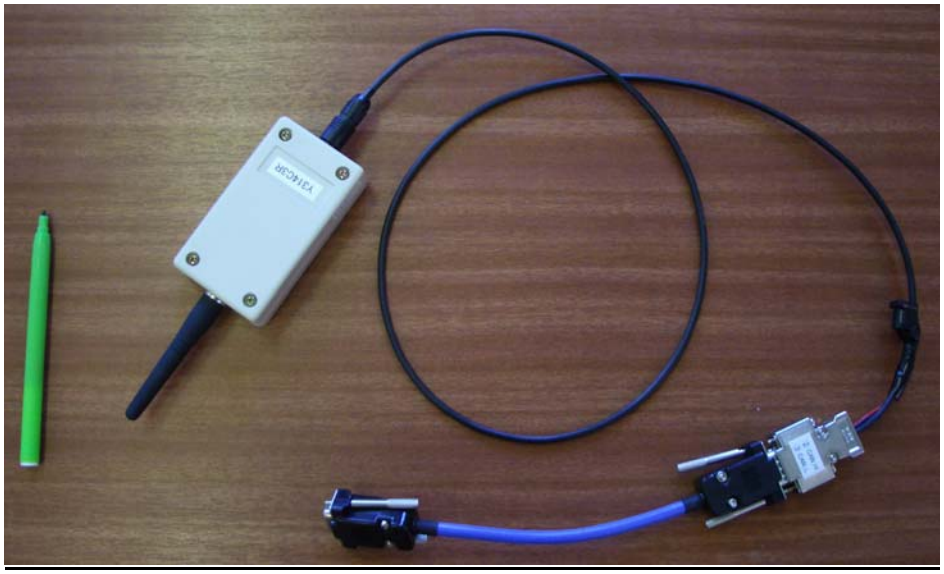


Figura 5.5. Receptor con su Cable Conector.

A su vez, cada receptor tiene asociado otro identificador. Dicho identificador está indicado en el receptor mediante una etiqueta, como puede apreciarse en la Figura 5.5, y aparece reflejado en los mensajes recibidos mediante el identificador del mensaje del bus CAN.

Los sensores se instalan en las turbinas de gas y recogen los datos. Estos datos son transmitidos a los receptores mediante un protocolo de comunicaron inalámbrico específicamente desarrollado por el fabricante para este producto. Por tanto, no se trata de un protocolo estándar de comunicación, como se comentó en el capítulo anterior.

Los receptores transforman los datos recibidos de manera inalámbrica de los sensores, para convertirlos en un tipo de información entendible por el bus CAN. Más específicamente, los sensores mandan la información dividida en dos mensajes, el primero de longitud siete *bytes* y el segundo de longitud cinco *bytes*, tal y como se indica en la Figura 5.6.

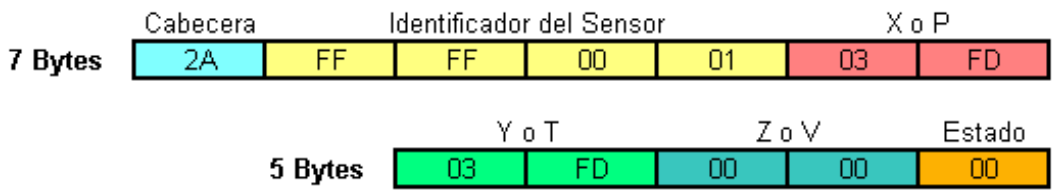


Figura 5.6. Mensajes de los Sensores.

Los sensores utilizados nos proporcionan medidas de:

- Vibración en las tres direcciones del espacio X, Y, Z.
- Temperatura, T.
- Presión, P.
- Voltaje de la batería, V.

Dependiendo del estado, los datos recibidos son de vibración, cuando el estado es 0, o presión, temperatura y voltaje, cuando el estado es 1.

Desde un principio se desarrolló la aplicación acorde con el tipo de datos recibidos a través de los receptores y el bus CAN. Pero cuando se procedió a probar dicha aplicación con múltiples sensores y receptores, se detectaron ciertas anomalías en el *hardware*. En los apartados siguientes se describe el problema encontrado, se muestran las distintas pruebas realizadas y se explica la solución adoptada por el fabricante.

5.2.1. Problema

En la aplicación desarrollada es deseable recibir datos de distintos sensores simultáneamente. Se esperaba que cada receptor estuviera íntimamente relacionado con un solo sensor. De esta manera, se esperaba que cada receptor fuera capaz de leer los datos de un solo sensor. Sin embargo, cuando se procedió a realizar pruebas con múltiples sensores, los datos recibidos eran bastante aleatorios. Mediante las pruebas realizadas se obtuvieron ciertas conclusiones.

5.2.2. Pruebas Realizadas

5.2.2.1. Prueba 1

En esta prueba se utilizaron los siguientes receptores y sensores, ambos indicados mediante su identificador. También se describe la situación del equipo:

→ Receptores: ID 16, 17 y ID 18, 19.

→ Sensores: 4 y 5.

→ Situación: Todo separado en la misma mesa.

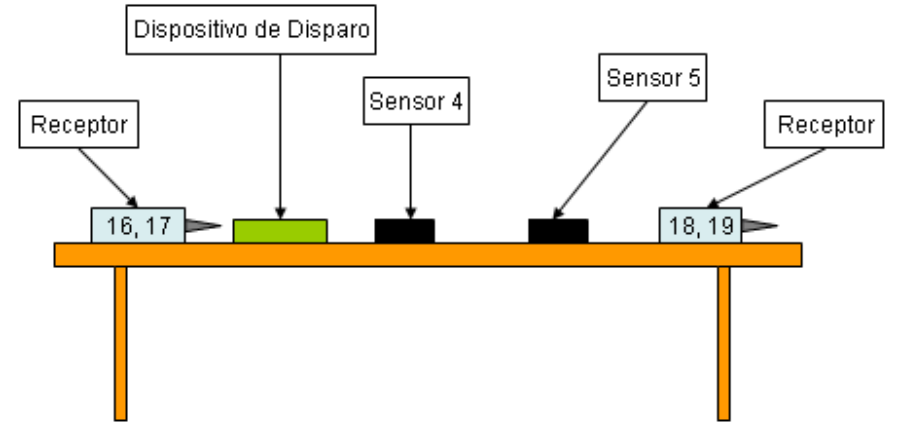


Figura 5.7. Esquema Prueba 1.

Al activar los sensores mediante el dispositivo de disparo, ambos receptores leían los datos de un mismo sensor. Al pulsar de nuevo el botón del dispositivo de disparo, ambos receptores pasaban a leer los datos del otro sensor.

Tras esta prueba se concluyó que era posible que cada vez que se activaba el dispositivo de disparo se estuviera encendiendo un sensor y apagando el otro, alternándose en cada activación el sensor que estaba activado. El problema es que no hay ninguna manera de comprobar cuando un sensor está activo, solo cuando se están recibiendo datos de dicho sensor.

5.2.2.2. Prueba 2

En esta prueba se utilizaron los siguientes receptores y sensores, ambos indicados mediante su identificador. También se describe la situación del equipo:

→ Receptores: ID 12, 13 y ID 14, 15.

→ Sensores: 2 y 3.

→ Situación: Receptores separados y cada sensor encima de uno de los receptores, todo sobre la misma mesa.

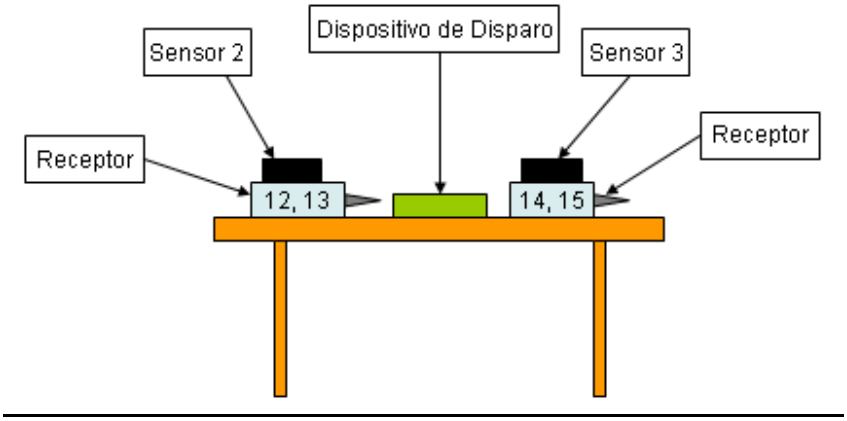


Figura 5.8. Esquema Prueba 2.

La primera medida en esta prueba fue retirar las baterías de los sensores para asegurarse de que éstos estarían desactivados. Una vez colocadas las baterías de nuevo, se activaron los sensores con solo pulsar una vez el botón del dispositivo de disparo. En esta situación cada receptor leía el sensor que estaba colocado encima del mismo receptor.

Si se cambiaba la posición de los sensores, es decir, si se movía el sensor 2 y se ponía encima del receptor 14,15 y, al mismo tiempo, se movía el sensor 3 y se colocaba encima del sensor 12,13 (ver Figura 5.8), los receptores comenzaban a leer los datos del sensor colocado encima del mismo receptor. Finalmente, los sensores fueron desactivados con solo pulsar una vez el botón del dispositivo de disparo.

Mediante esta prueba se concluyó que era posible que los receptores estuvieran leyendo datos de aquel sensor cuya señal era más intensa.

5.2.2.3. Prueba 3

En esta prueba se utilizaron los siguientes receptores y sensores, ambos indicados mediante su identificador. También se describe la situación del equipo:

→ Receptores: ID 12, 13 y ID 14, 15.

→ Sensores: 2 y 3.

→ Situación: Receptores separados y cada sensor encima de uno de los receptores, todo sobre la misma mesa.

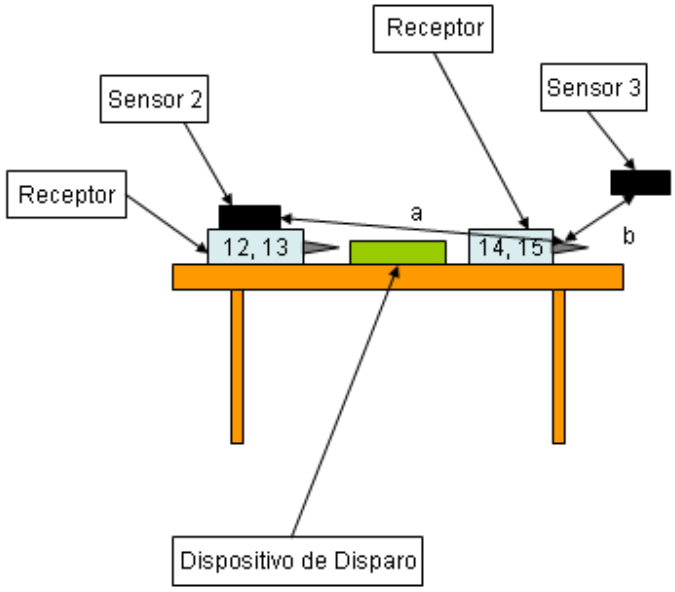


Figura 5.9. Esquema Prueba 3.

Se trata de una prueba bastante similar a la anterior. En este caso, en vez de intercambiar la posición de los sensores lo que se hizo es separar uno de ellos del receptor, como se muestra en la Figura 5.9. En esta situación, ambos receptores leían datos del sensor cuya posición no había sido modificada, en la Figura 5.9 el sensor 2.

Se observó que sucedía incluso cuando la distancia entre el sensor movido, sensor 3, y su receptor, receptor 14,15, era menor que la distancia entre el sensor movido, sensor 3, y el otro receptor, receptor 12,13. Es decir, ocurría incluso cuando b < a (ver Figura 5.9).

Esta prueba contradice la idea de que cada receptor lee el sensor cuya señal es más intensa.

5.2.2.4. Prueba 4

En esta prueba se utilizaron los siguientes receptores y sensores, ambos indicados mediante su identificador. También se describe la situación del equipo:

→ Receptores: ID 12, 13 y ID 14, 15.

→ Sensores: 2 y 3.

→ Situación: Receptores separados sobre la misma mesa. Sensores físicamente separados entre ellos y de los receptores.

El primer paso en esta prueba fue activar el sensor 2 mediante el dispositivo de disparo (ver Figura 5.10). En esta situación ambos receptores empezaron a leer el sensor que estaba activado, en este caso el sensor 2.

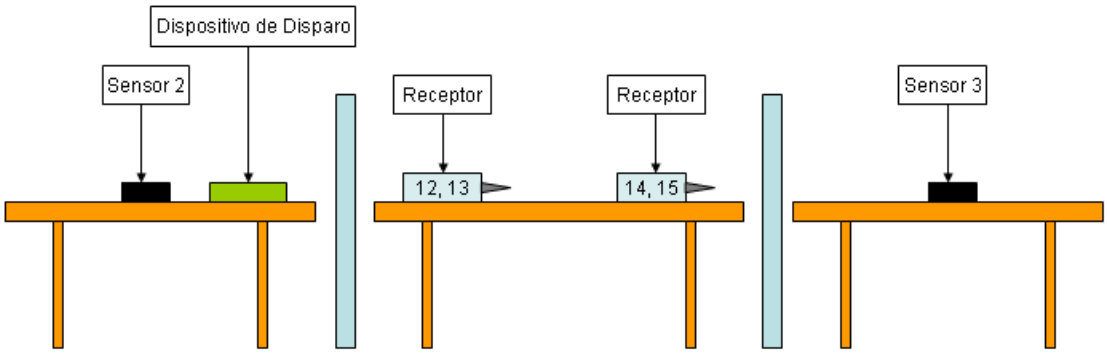


Figura 5.10. Esquema Prueba 4. Activación del Sensor 2.

Después se activó el sensor 3 mediante el dispositivo de disparo (ver Figura 5.11).

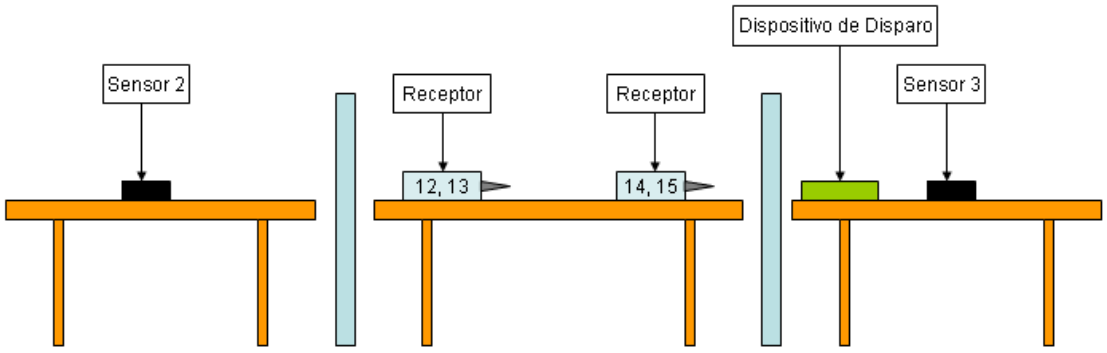


Figura 5.11. Esquema Prueba 4. Activación del Sensor 3.

En este momento cada receptor comenzó a leer el sensor que se encontraba mas cerca.

Esta prueba vuelve a apoyar la idea de que cada receptor lee aquel sensor cuya señal es más intensa.

5.2.2.5. Prueba 5

En esta prueba se utilizaron los siguientes receptores y sensores, ambos indicados mediante su identificador. También se describe la situación del equipo:

→ Receptores: ID 12, 13 y ID 14, 15.

→ Sensores: 2 y 3.

→ Situación: Receptores juntos sobre la misma mesa. Sensores físicamente separados entre ellos y de los receptores.

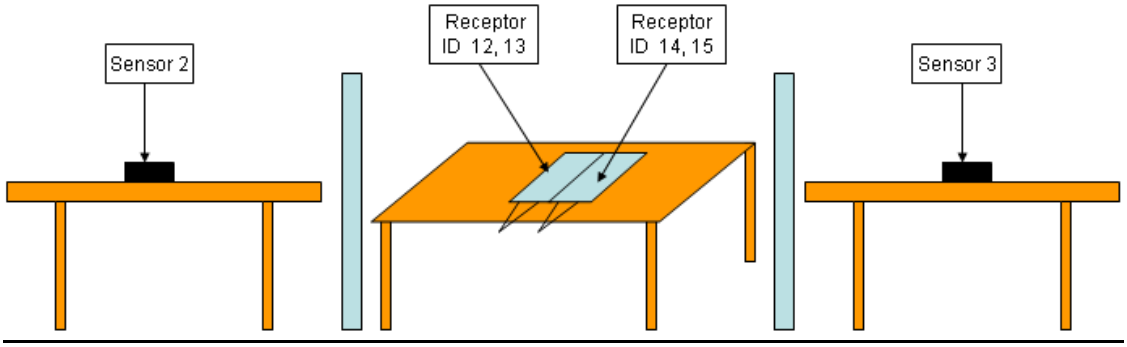


Figura 5.12. Esquema Prueba 5.

En esta situación ambos receptores leían siempre al mismo sensor, alternando entre el sensor 2 y el 3 cada vez que se pulsaba el botón del dispositivo de disparo. Sucedía incluso tras retirar las baterías y volverlas a colocar para asegurar la desactivación de ambos sensores.

5.2.2.6. Otras Observaciones

Si dos sensores están trabajando y uno de ellos tiene la batería más descargada que el otro, los receptores tendían a leer aquel sensor con la batería más cargada. Esto provocaba un problema porque es imposible conseguir que todas las baterías se encuentren al mismo nivel, además de que carece de sentido tener que preocuparse de eso. En la práctica, podrían plantearse múltiples situaciones en las que no fuera posible.

5.2.3. Solución Adoptada por el Fabricante

Todas estas pruebas fueron enviadas al fabricante. Tras el estudio de las mismas se recibió una respuesta obvia, y es que para la anterior aplicación en automoción estos problemas no se habían encontrado debido a que la distancia entre los sensores era siempre suficiente para no tener dificultades.

Analizaron nuestras necesidades y finalmente se recibieron nuevos dispositivos. Básicamente, es el mismo *hardware*, con el mismo aspecto físico y con el mismo tipo de datos, pero de alguna manera cada sensor esta ligado a cada receptor. Así, cada receptor es capaz de leer sólo y únicamente los datos del sensor asociado. De esta manera, se asegura que no se están leyendo datos repetidos a la vez que se evitan interferencias entre los datos y el problema de las baterías.

5.3. Bus CAN

CAN es un protocolo de comunicaciones desarrollado por la firma alemana *Robert Bosch GmbH*, basado en una topología de bus para la transmisión de mensajes en ambientes distribuidos, además ofrece una solución a la gestión de la comunicación entre múltiples unidades de centrales de proceso.

Como se ha comentado anteriormente, CAN fue desarrollado, inicialmente para aplicaciones en los automóviles y, por lo tanto, la plataforma del protocolo es resultado de las necesidades existentes en el área automotriz, pero rápidamente despertó una creciente atención en el área de control y automatización industrial.

El bus ofrece altas prestaciones contra errores de transmisión, ruidos electromagnéticos y fallos internos en los dispositivos conectados al bus, características imprescindibles para sistemas críticos y soluciones con altos requisitos de fiabilidad.

Actualmente el bus CAN está estandarizado bajo el ISO-11898 y gestionado por la Asociación de Fabricantes y Usuarios *CAN-in-Automation (CiA)* situada en Alemania.

El protocolo de comunicaciones CAN proporciona los siguientes beneficios:

- Es un protocolo de comunicaciones normalizado, con lo que se simplifica y economiza la tarea de comunicar subsistemas de diferentes fabricantes sobre una red común o bus.

- El procesador anfitrión *(host)* delega la carga de comunicaciones a un periférico inteligente, por lo tanto el procesador anfitrión dispone de mayor tiempo para ejecutar sus propias tareas.

- Al ser una red multiplexada, reduce considerablemente el cableado y elimina las conexiones punto a punto.

Entre sus fortalezas, el bus CAN considera una arquitectura multimaestra capaz de proveer características de respuesta en tiempo real y tolerancia a fallos en la recepción de mensajes y mal funcionamiento de los nodos. En cuanto a la detección y manejo de errores, un controlador CAN cuenta con la capacidad de detectar y manejar los errores que surjan en una red. Todo error detectado por un nodo, se notifica inmediatamente al resto de los nodos.

Se muestra a continuación, en la Figura 5.13, una imagen real del bus CAN utilizado en este trabajo.

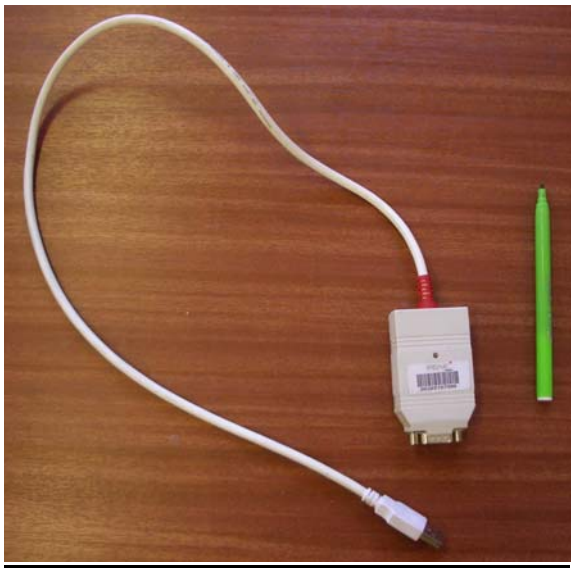


Figura 5.13. Bus CAN.

5.3.1. Principales Características de CAN

CAN se basa en el modelo productor/consumidor, el cual es un concepto, o paradigma de comunicaciones de datos, que describe una relación entre un productor y uno o más consumidores. CAN es un protocolo orientado a mensajes, es decir, la información que se va a intercambiar se descompone en mensajes, a los cuales se les asigna un identificador y se encapsulan en tramas para su transmisión. Cada mensaje tiene un identificador único dentro

de la red, con el cual los nodos deciden aceptar o no dicho mensaje. Dentro de sus principales características se encuentran:

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

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

5.3.2. Proceso de Transmisión de Datos

El proceso de transmisión de datos se desarrolla siguiendo un ciclo de varias fases:

Suministro de datos: Una unidad de mando recibe información de los sensores que tiene asociados. Su microprocesador pasa la información al controlador donde es gestionada y acondicionada para a su vez ser pasada al trasmisor-receptor donde se transforma en señales eléctricas.

Transmisión de datos: El controlador de dicha unidad transfiere los datos y su identificador junto con la petición de inicio de transmisión, asumiendo la responsabilidad de que el mensaje sea correctamente trasmitido a todas las unidades de mando asociadas. Para trasmitir el mensaje ha tenido que encontrar el bus libre, y en caso de colisión con otra unidad de mando intentando trasmitir simultáneamente, tener una prioridad mayor. A partir del

momento en que esto ocurre, el resto de unidades de mando se convierten en receptoras.

Recepción del mensaje: Cuando la totalidad de las unidades de mando reciben el mensaje, verifican el identificador para determinar si el mensaje va a ser utilizado por ellas. Las unidades de mando que necesiten los datos del mensaje lo procesan, si no lo necesitan, el mensaje es ignorado.

Estas medidas hacen que las probabilidades de error en la emisión y recepción de mensajes sean muy bajas, por lo que es un sistema extraordinariamente seguro.

El planteamiento del bus CAN, como puede deducirse, permite disminuir notablemente el cableado en el automóvil, puesto que si una unidad de mando dispone de una información, como por ejemplo, la temperatura del motor, esta puede ser utilizada por el resto de unidades de mando sin que sea necesario que cada una de ellas reciba la información de dicho sensor.

Otra ventaja obvia es que las funciones pueden ser repartidas entre distintas unidades de mando, y que incrementar las funciones de las mismas no presupone un coste adicional excesivo.

5.3.3. Especificación CAN 2.0A y CAN 2.0B. Trama de Datos.

Hay que referir la existencia de dos versiones del protocolo CAN, la estándar (2.0A) y la extendida (2.0B), ligada ésta última a la necesidad de mayores prestaciones.

Como ya se ha comentado anteriormente los mensajes transmitidos desde cualquier nodo en una red CAN no contienen la dirección del nodo emisor ni la del nodo receptor. En vez de esto, los mensajes contienen una etiqueta identificativa, única en toda la red, que realiza esa función. Estos identificadores determinan la prioridad del mensaje. El mensaje de mayor prioridad gana el acceso al bus, mientras que los mensajes de menor prioridad se retransmitirán automáticamente en los siguientes ciclos de bus. Como consecuencia de esto, varios nodos pueden recibir y actuar simultáneamente sobre el mismo mensaje.

Esta estructura de los mensajes ofrece a la red una gran flexibilidad y posibilidad de expansión, ya que nuevos nodos pueden ser añadidos a la red sin la necesidad de hacer ningún cambio en el *hardware* ni en el *software* existente.

Las tramas de los mensajes son los elementos básicos de transmisión y van de un nodo emisor a uno o varios nodos receptores. Hay dos protocolos de comunicación: el estándar, que soporta mensajes con identificadores de 11 bits, y el extendido, que soporta 29 bits (ver Figura 5.14). El mensaje está dividido en siete campos diferentes, cada uno de ellos con una función específica.

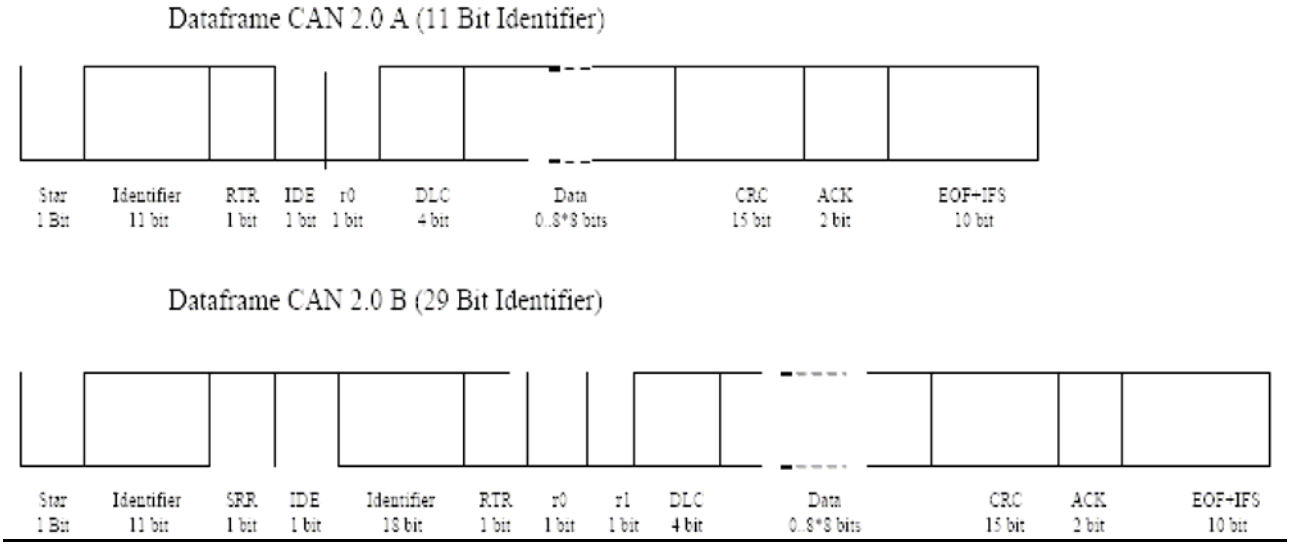


Figura 5.14. Trama de Datos CAN.

Capítulo 6

Descripción de la aplicación

6.1. *Software*

Para el desarrollo de la aplicación se ha usado el *software Visual Basic*. Se trata de un lenguaje de programación cuya primera versión fue desarrollada con la intención de simplificar la programación utilizando un ambiente de desarrollo completamente gráfico que facilitara la creación de interfaces gráficas y, en cierta medida, también la programación misma.

La facilidad del lenguaje permite crear aplicaciones para *Windows* en un tiempo razonable. En otras palabras, permite un desarrollo eficaz y muy poca inversión en tiempo, menos que con cualquier otro lenguaje.

El motivo por el cual se eligió dicho *software* es porque se trata de hacer un prototipo y con *Visual Basic* se consigue una manera rápida y sencilla de desarrollarlo, a la vez que se asegura la posibilidad de interactuar con el bus CAN, lo cual era imprescindible en este trabajo por el *hardware* utilizado.

Se representa la información recibida en tiempo real y se va almacenando para tener la posibilidad de un posterior análisis si es necesario. Para ese posterior análisis se ha usado el software *Matlab*, desarrollando un programa capaz de importar los datos y representarlos.

Matlab también ha sido utilizado en un análisis posterior más profundo. Se tratan datos recogidos por los sensores en plataformas reales mediante la aplicación en *Visual Basic*.

Todos estos análisis realizados en *Matlab* se explican en detalle en el capítulo siguiente.

6.2. Desarrollo de la Aplicación

Durante el desarrollo de la aplicación se han realizado muchas versiones de la misma. En este documento se explicará en detalle la versión definitiva, pero también se comentarán algunos detalles de las versiones anteriores.

En un principio se partió del desarrollo de dos aplicaciones, una para mandar datos y otra para recibirlos. Como ayuda, se usó una aplicación más simple previamente desarrollada. Sirvió para empezar a entender el tipo de datos que se pueden mandar a través del bus CAN, consiguiendo así analizar el formato de los datos. De esta manera, se consiguió familiarizarse con las funciones específicas para acceder al bus CAN y se comenzó a esbozar lo que sería la aplicación real.

El siguiente paso fue pasar al uso de una única aplicación para recibir datos, usando los sensores reales como emisores de datos. Se trabajó con un solo sensor y un solo receptor y se adaptó la aplicación de lectura para descodificar los datos reales que los sensores mandaban. El almacenamiento de los datos una vez abortada la aplicación y la representación final de todos los datos recogidos, se realizó mediante la exportación de todos los datos desde *Visual Basic* a *Excel*.

Al pasar a usar múltiples sensores se encontraron las dificultades analizadas en el capítulo anterior. Hubo una fase de pruebas y análisis de las anomalías encontradas hasta deducir que el *hardware* no trabajaba como se esperaba. Una vez que el fabricante suministró los nuevos sensores ligados a los receptores se pudo adaptar la aplicación a la multiplicidad de emisores de datos.

Uno de los problemas de mayor índole encontrado durante el desarrollo de la aplicación fue el hecho de tener una gran cantidad de datos y la necesidad de almacenarlos para un posterior análisis de los mismos. Se barajaron diversas posibilidades entre ellas la exportación de los mismos a *Excel*, pero la eficiencia de la exportación era muy baja debido al tiempo necesario. Además *Excel* imponía una restricción más, el número de datos a almacenar estaba limitado por la longitud de las hojas de *Excel*, es decir, por el número de celdas de una hoja *Excel*. Finalmente se decidió exportarlos en ficheros de texto y desarrollar un programa en *Matlab* que los analizara.

6.3. Paso a Paso

6.3.1. *Write - Read*

Se comenzó trabajando con dos ordenadores, de los cuales, uno de ellos simulaba estar recogiendo datos y mandándoselos al otro, es decir, realizaba la función de los sensores.

En la Figura 6.1 se muestra una imagen de la situación. En este caso, es el ordenador portátil el que simula realizar la función de los sensores. Se conectaron los ordenadores mediante dos buses CAN que también se pueden ver en la Figura 6.1.

Figura 6.1. Imagen Real. Etapa Write - Read.

El objetivo era comenzar a esbozar la aplicación de lectura y familiarizarse con las funciones de acceso al bus CAN. Para ello se usó una aplicación previamente desarrollada *PCANView*. Es un programa de fácil manejo para supervisar el flujo de datos a través de un bus CAN.

En la Figura 6.2 se representa la pantalla de inicio, con todas las opciones de conexión que presenta. Los parámetros a destacar son:

- La velocidad de transferencia medida en baudios, o *Baud Rate*.
- El tipo de mensaje que se desea transferir/recibir que puede ser estándar, *Standard*, o extendido, *Extended*.

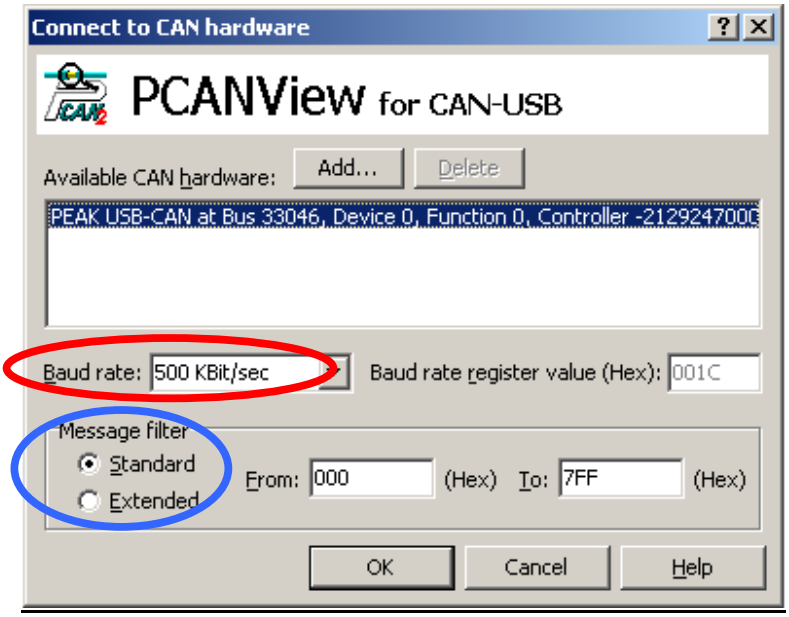


Figura 6.2. Aplicación para Manejo del Bus CAN.

En la Figura 6.3 se representa dicha aplicación en funcionamiento. Por tanto, en dicha figura se observan los parámetros representados en tiempo real cuando la aplicación está en funcionamiento:

- El identificador del mensaje CAN recibido/transmitido en hexadecimal. Puede variar de 0 a 7FFh para identificadores de 11 *bits* y de 0 a 7FFFFFFFh para identificadores de 29 *bits* (especificación 2.0B de CAN). En el caso del mensaje recibido, dicho identificador coincide con el del receptor cuando proviene del sistema inalámbrico.

- La longitud de los mensajes recibidos/transmitidos. Puede variar de 0 a 8. Los recibidos a través del sistema inalámbrico, como ya se comentó en el capítulo anterior, son dos, el primero de longitud siete *bytes* y el segundo de longitud cinco *bytes*.

- El mensaje recibido/transmitido en sí, con la información en hexadecimal.

- El periodo de tiempo entre los dos últimos mensajes recibidos/transmitidos con el mismo identificador.

- Número de mensajes recibidos/transmitidos con el mismo identificador desde la última vez que el usuario reseteó la información.

- Los dos últimos campos representados son para tramas remotas, que en nuestra aplicación no serán utilizadas.

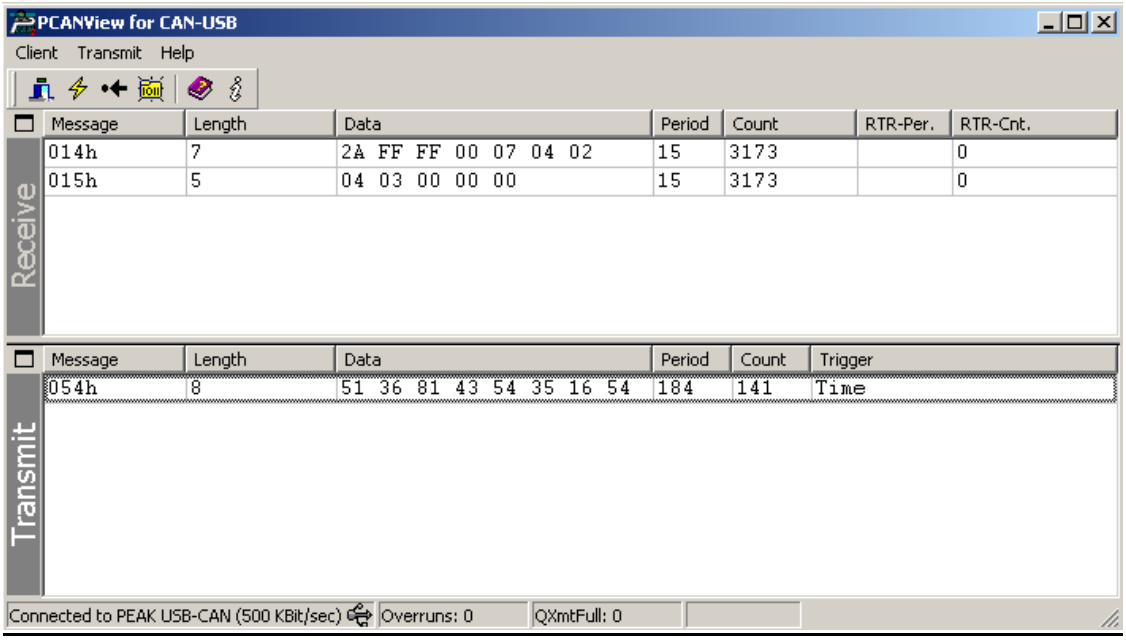


Figura 6.3. Aplicación para el Manejo del Bus CAN en Funcionamiento.

Las aplicaciones desarrolladas para este trabajo se muestran en las Figuras 6.4 y 6.5.

En la Figura 6.4 se representa la aplicación de escritura. Básicamente, consta de un botón para inicializar el bus, al que se ha denominado *CAN_Init*, uno para escribir mensajes, llamado *CAN_Write* y otro para cerrar el bus, al que se ha nombrado *CAN_Close*.

Para definir el mensaje a transmitir se tienen varios cuadros de texto:

- Identificador del mensaje.
- Longitud del mensaje.
- El mensaje en sí, en esta aplicación escrito en decimal.

En caso de intentar transmitir un mensaje con alguno de estos campos en blanco, la aplicación muestra un mensaje para indicar que falta información necesaria para realizar la operación de envío. Cuando esto pasa, no se realiza la transmisión sino que se le permite al usuario rellenar los campos vacíos antes de solicitar una nueva transmisión.

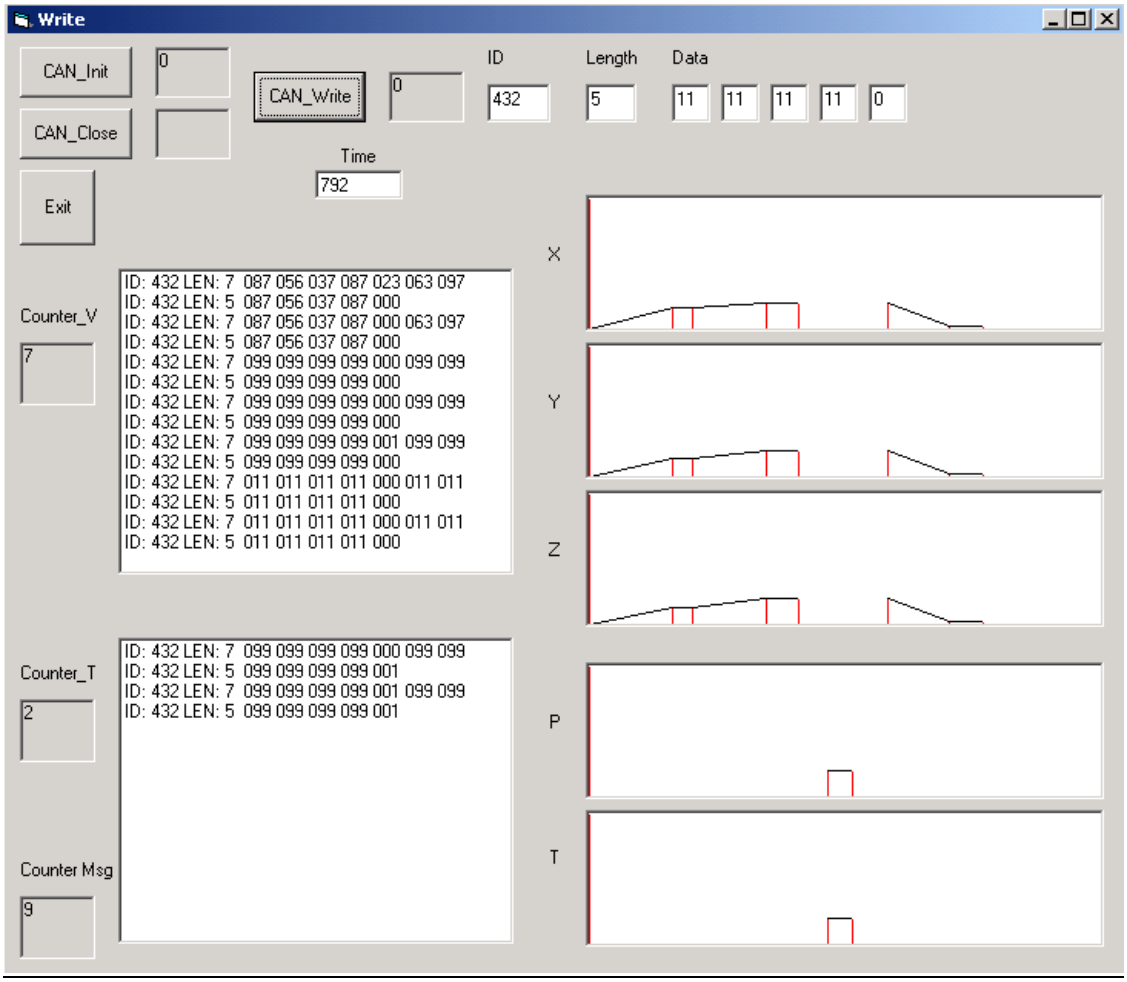


Figura 6.4. Aplicación Write.

El último *byte* del mensaje de longitud 5 tiene que ser un 0 o un 1 para indicar el tipo de datos que se están enviando. Si esto no es así, el dato enviado se ignora y no se recoge.

Los mensajes enviados se van representando en tiempo real en las graficas proporcionadas. Se destaca el valor del dato recogido mediante la vertical en rojo. Se almacena en una variable auxiliar el tipo de dato recibido justo antes y en otra el tipo de dato a representar en este momento. Si se reciben dos datos del mismo tipo de manera consecutiva, se representan los puntos unidos por una línea recta. Si, por el contrario, el dato recibido no es del mismo tipo que el anterior, se representará por un punto aislado, dejando un hueco en los instantes anteriores en dicha gráfica. De esta forma, se pretende indicar que durante ese periodo de tiempo no se recibieron datos de ese tipo. Esto se observa claramente en la Figura 6.4 en los mensajes de vibración, quedando un hueco en las gráficas cuando se han estado recibiendo datos de presión y temperatura.

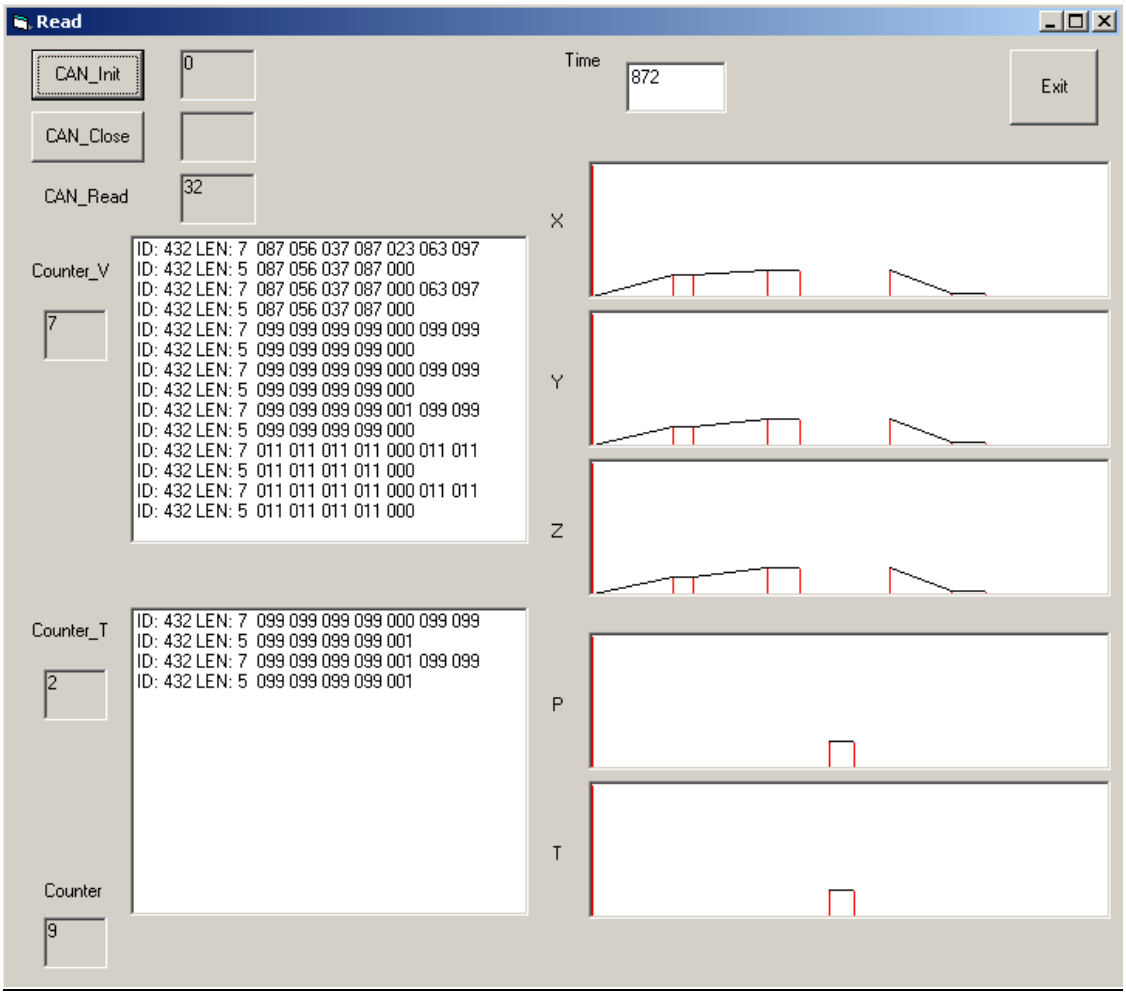


Figura 6.5. Aplicación Read.

Se puede observar también, que en esta aplicación no se representa el voltaje. Esto se debe a que al principio del desarrollo se desconocía que los sensores proporcionaran esa medida.

Se tienen también dos listas en las que se escriben los mensajes transmitidos, una para los mensajes de vibración y otra para los de presión y temperatura. Ambas con unos contadores para registrar el número de mensajes transmitidos hasta el momento.

Se representa a su vez el tiempo que lleva la aplicación en funcionamiento. Y se dispone de un botón de salida inmediata en caso de que se quiera abortar la aplicación, *Exit*, aunque se obtendría el mismo resultado cerrando la ventana de una manera directa.

En la Figura 6.5 se representa la aplicación de lectura. Se observa que es prácticamente igual que la anterior. La única diferencia está en que

desaparecen los cuadros de texto para definir el mensaje a transmitir y aparece la información que proporciona la función de lectura, para analizar si ha habido algún tipo de error al intentar extraer un nuevo mensaje de la cola.

Para la lectura de los mensajes se usa un temporizador que saca un mensaje de la cola cada 10 milisegundos. Se tiene seguridad de que se recogen todos los mensajes porque la frecuencia de recepción es de 100 Hz aproximadamente.

6.3.2. Solo *Read* con *Excel*

En esta etapa se usó una única aplicación para recibir datos y se usaron los sensores reales como emisores de datos en vez de simularlos a través de otro ordenador. En un principio, se trabajó con un solo sensor y un solo receptor, teniendo como objetivo adaptar la aplicación de lectura para descodificar los datos reales que los sensores mandaban. Ya se tenía conocimiento de que los sensores mandaban datos del estado del voltaje de la batería, así que también se añadió esta parte a la aplicación.

Posteriormente, se pasó a trabajar con múltiples sensores encontrando los problemas ya comentados acerca del funcionamiento del *hardware*. Cuando el fabricante mandó los nuevos sensores se adaptó esta misma aplicación a la multiplicidad. El almacenamiento de la información y la representación final de los datos recogidos se realizó en *Excel*.

En este apartado solo es relevante comentar el almacenamiento y las representaciones en *Excel* por ser la única parte eliminada de la aplicación real, la cual se presenta en detalle en el apartado siguiente.

El aspecto de la interfaz gráfica del usuario es muy similar al definitivo que se muestra posteriormente en la Figura 6.14. La única diferencia es que se dispone de un botón adicional, *Excel*. Cuando el usuario pulsa dicho botón la aplicación es interrumpida para exportar toda la información recogida hasta el momento a *Excel* y representar las gráficas de cada variable.

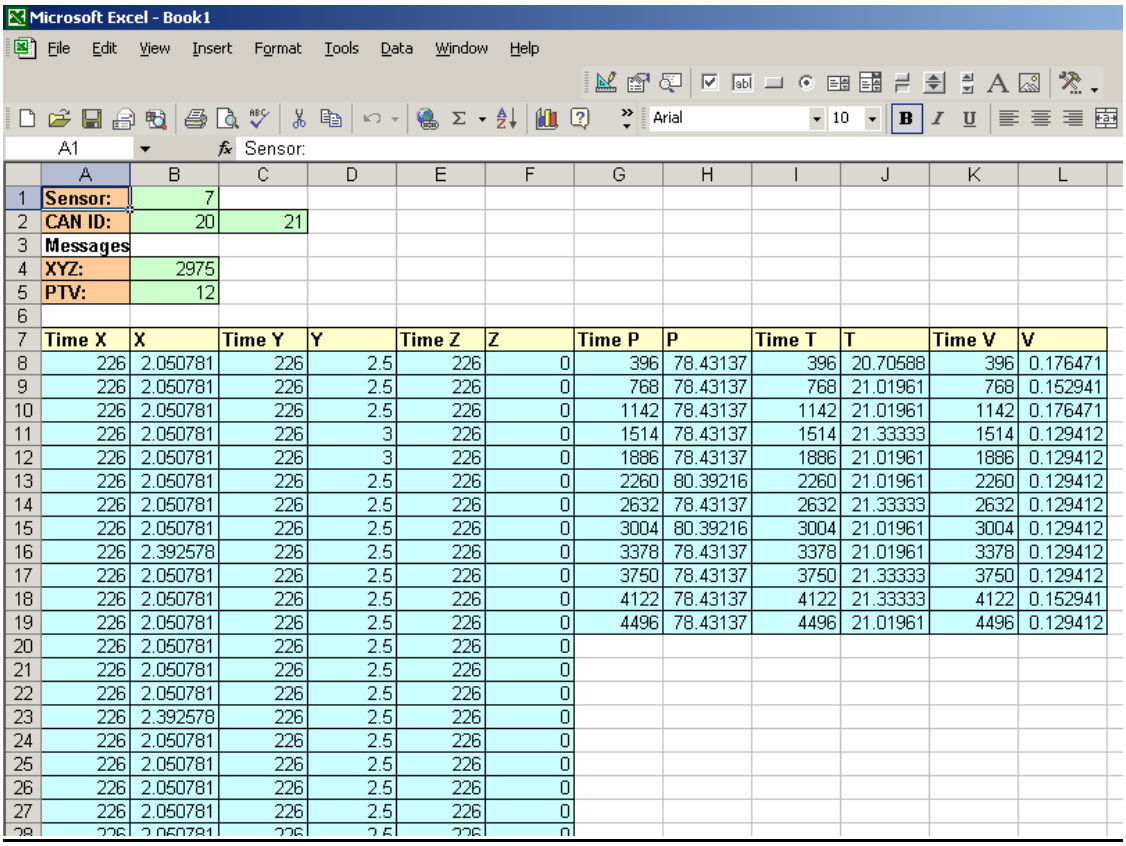
Se abre un nuevo libro en *Excel* y se crean para cada sensor con el que se está trabajando una hoja y seis gráficas. En la hoja, cuyo aspecto se muestra en la Figura 6.6, se recoge toda la información acerca del sensor en cuestión:

- Identificador del sensor.

- Identificador del receptor asociado, que coincide con el de los mensajes CAN en los que se reciben sus datos.

- Número total de mensajes recibidos de cada tipo: vibración en las tres direcciones del espacio, XYZ, y presión, temperatura y voltaje, PTV.

- Listado de todos los datos recibidos de este sensor durante la ejecución de la aplicación hasta el momento en que se solicitó la información en *Excel*.



Microsoft Excel - Book1

File Edit View Insert Format Tools Data Window Help

A1 Sensor:

	A	B	C	D	E	F	G	H	I	J	K	L
1	**Sensor:**	7										
2	**CAN ID:**	20	21									
3	**Messages**											
4	**XYZ:**	2975										
5	**PTV:**	12										
6												
7	**Time X**	**X**	**Time Y**	**Y**	**Time Z**	**Z**	**Time P**	**P**	**Time T**	**T**	**Time V**	**V**
8	226	2.050781	226	2.5	226	0	396	78.43137	396	20.70588	396	0.176471
9	226	2.050781	226	2.5	226	0	768	78.43137	768	21.01961	768	0.152941
10	226	2.050781	226	2.5	226	0	1142	78.43137	1142	21.01961	1142	0.176471
11	226	2.050781	226	3	226	0	1514	78.43137	1514	21.33333	1514	0.129412
12	226	2.050781	226	3	226	0	1886	78.43137	1886	21.01961	1886	0.129412
13	226	2.050781	226	2.5	226	0	2260	80.39216	2260	21.01961	2260	0.129412
14	226	2.050781	226	2.5	226	0	2632	78.43137	2632	21.33333	2632	0.129412
15	226	2.050781	226	2.5	226	0	3004	80.39216	3004	21.01961	3004	0.129412
16	226	2.392578	226	2.5	226	0	3378	78.43137	3378	21.01961	3378	0.129412
17	226	2.050781	226	2.5	226	0	3750	78.43137	3750	21.33333	3750	0.129412
18	226	2.050781	226	2.5	226	0	4122	78.43137	4122	21.33333	4122	0.152941
19	226	2.050781	226	2.5	226	0	4496	78.43137	4496	21.01961	4496	0.129412
20	226	2.050781	226	2.5	226	0						
21	226	2.050781	226	2.5	226	0						
22	226	2.050781	226	2.5	226	0						
23	226	2.392578	226	2.5	226	0						
24	226	2.050781	226	2.5	226	0						
25	226	2.050781	226	2.5	226	0						
26	226	2.050781	226	2.5	226	0						
27	226	2.050781	226	2.5	226	0						

Figura 6.6. Hoja Excel del Libro Creado por la Aplicación.

Las seis gráficas creadas para cada sensor, contienen la representación de cada una de las variables: vibración en las tres direcciones del espacio X, Y, Z, presión, temperatura y voltaje de la batería. Dichas representaciones contendrán todos los datos recogidos hasta el momento, es decir, los exportados a *Excel* y recogidos en la hoja anterior.

A continuación en las Figuras 6.7 – 6.12, se muestran como ejemplo las representaciones para un solo sensor, refiriéndose al ejemplo mostrado en la Figura 6.6. Se muestran primero las gráficas de vibración en las tres direcciones del espacio X, Y, Z, y a continuación las correspondientes a

presión, temperatura y voltaje de la batería. Todos estos datos fueron recogidos moviendo los sensores de manera manual.

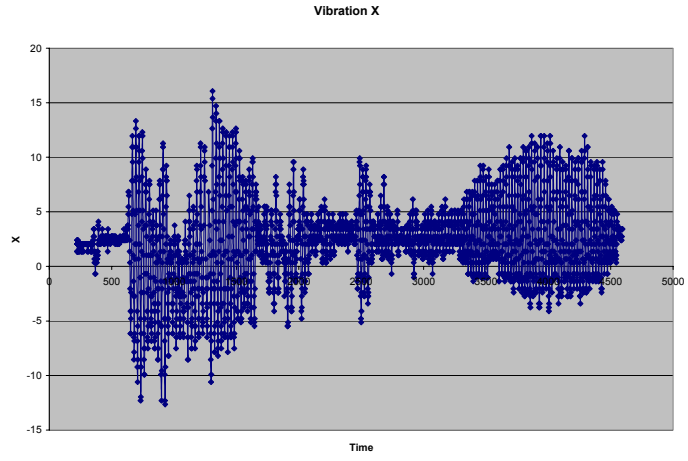


Figura 6.7. Vibración en la Dirección X.

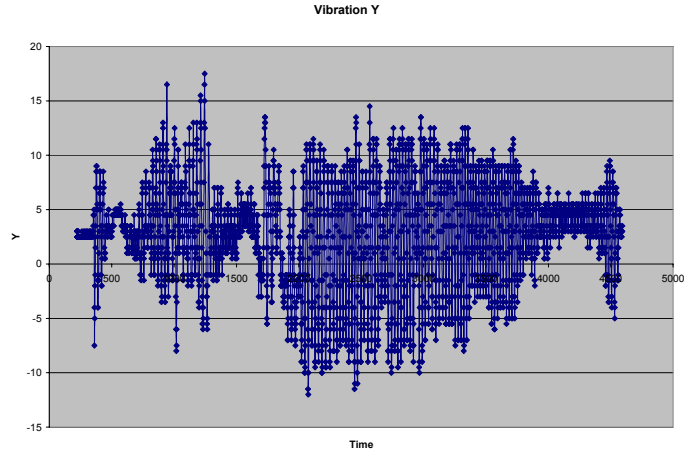


Figura 6.8. Vibración en la Dirección Y.

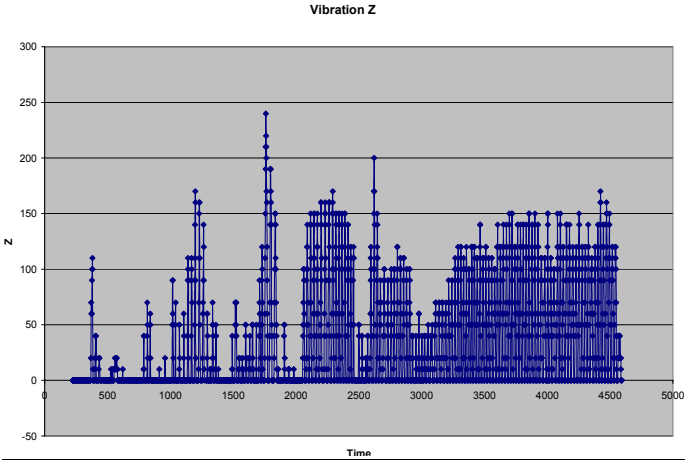


Figura 6.9. Vibración en la Dirección Z.

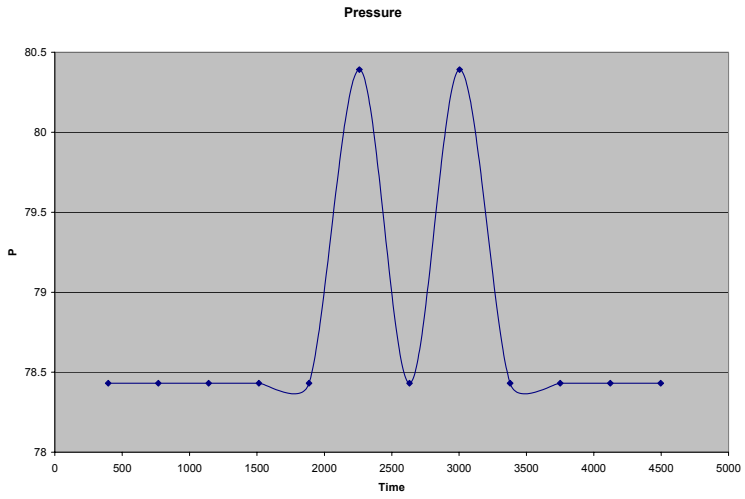


Figura 6.10. Presión.

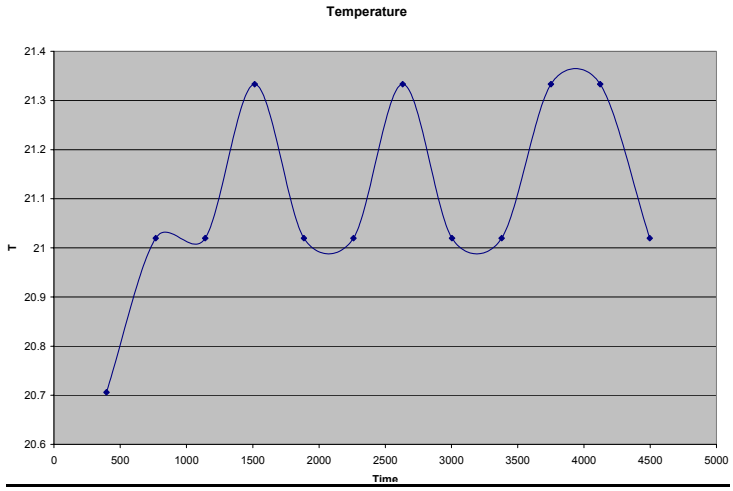


Figura 6.11. Temperatura.

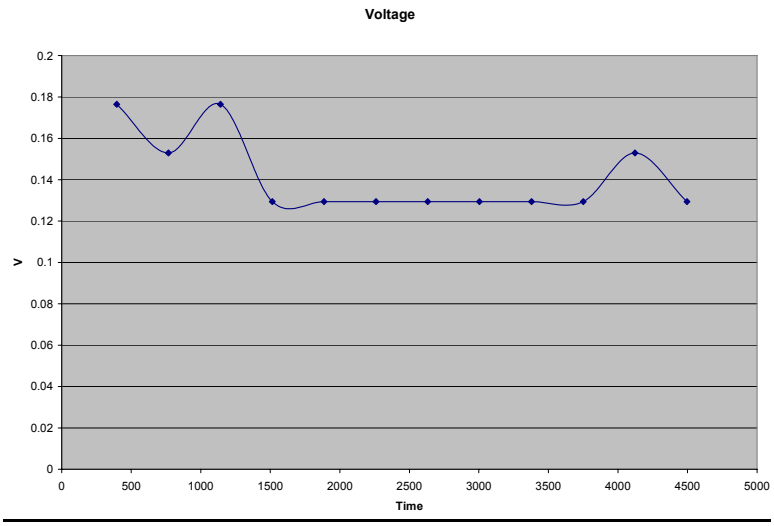


Figura 6.12. Voltaje de la Batería.

Todo lo obtenido era útil pero muy ineficiente. Trabajando con un solo sensor, la exportación de todos los datos a *Excel* y la representación de dichos datos en las seis gráficas tardaba alrededor del doble del tiempo que llevaba la aplicación ejecutándose cuando se pulsaba el botón. Al trabajar con múltiples sensores se necesitaba aún más tiempo. Es por este motivo, por lo que se decidió eliminar esta parte de la aplicación y recurrir a otra solución.

6.3.3. Aplicación Definitiva

En este apartado se procederá a la descripción detallada de la aplicación en su versión definitiva. Aquellas partes ya comentadas por ser comunes a las versiones anteriores se resumirán de una manera breve.

Para explicar en detalle la aplicación definitiva será útil la visualización en forma esquemática de los pasos más relevantes que se realizan durante la ejecución de la misma (ver Figura 6.13). Se representa mediante un recuadro en línea de puntos lo que se considera el bucle principal de la aplicación, en el que se llevan a cabo los pasos más importantes.

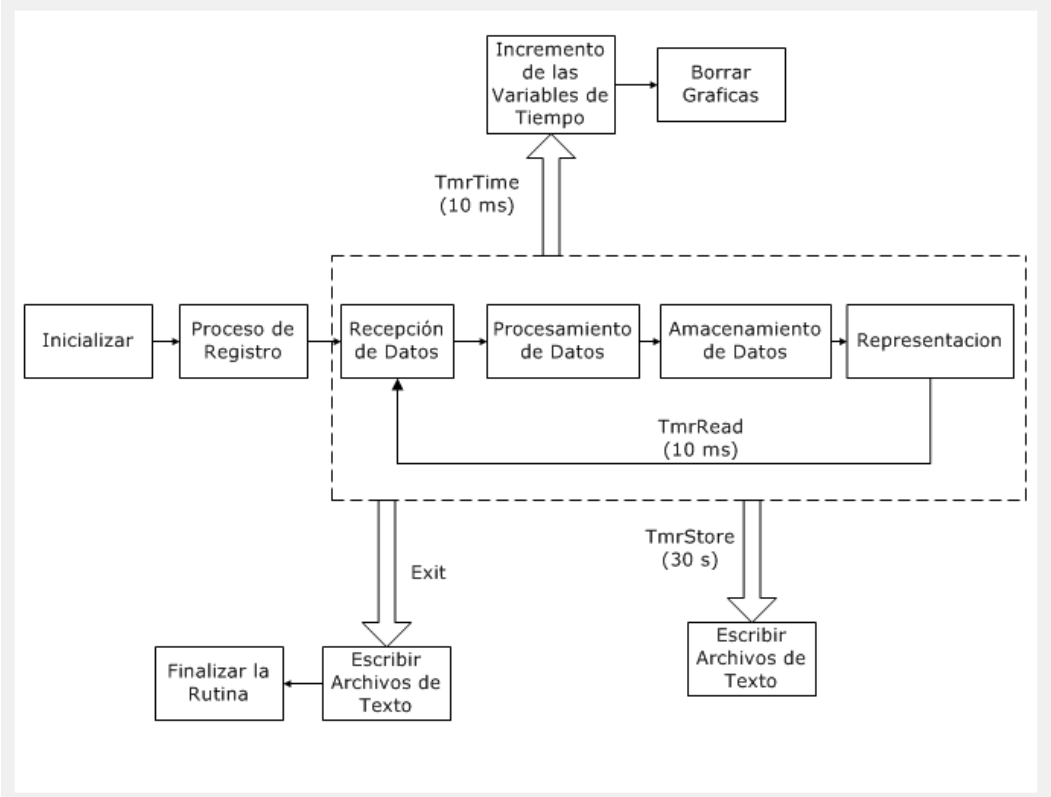


Figura 6.13. Esquema General de la Aplicación.

A su vez resultará útil tener presente el aspecto definitivo de la interfaz gráfica del usuario, el cual se presenta en la Figura 6.14. Se trata de una imagen real obtenida durante las pruebas de laboratorio que se comentarán en el capítulo siguiente.

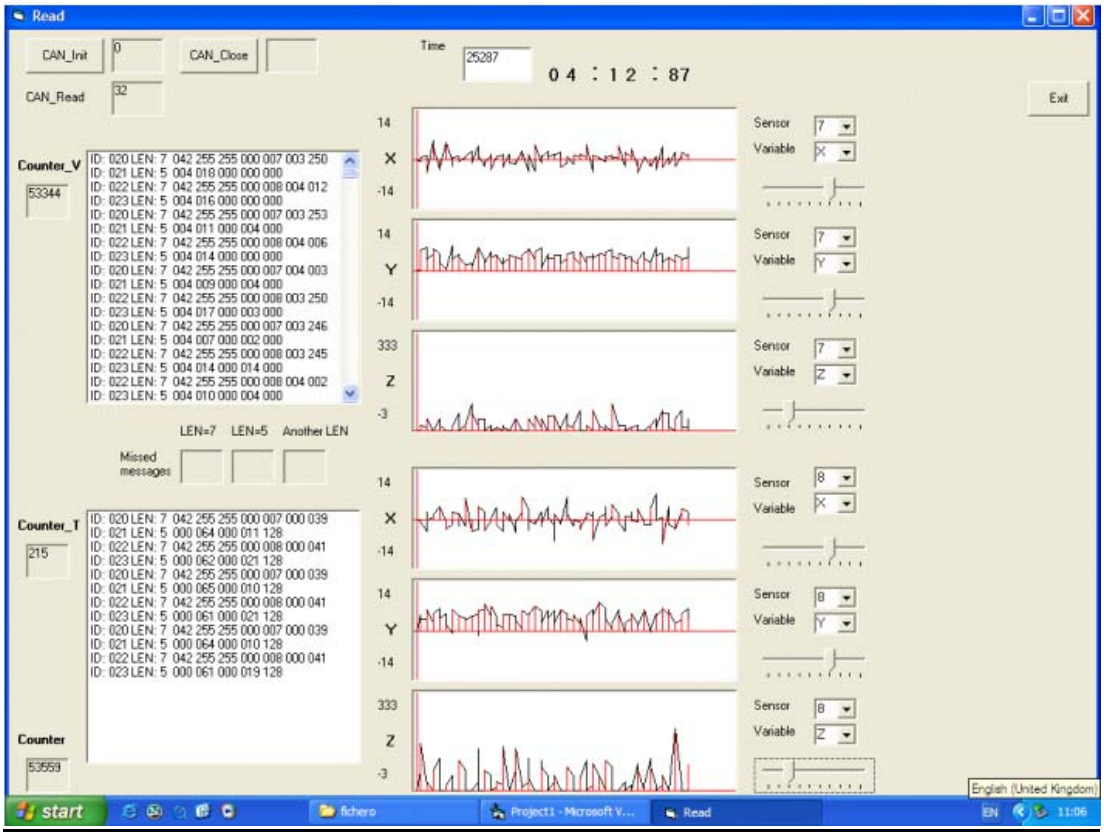


Figura 6.14. Interfaz Gráfica del Usuario.

Al ejecutar la aplicación, lo primero que se hace es preguntar al usuario el número de sensores con los que se va a trabajar. El cuadro de diálogo que aparece se muestra en la Figura 6.15:

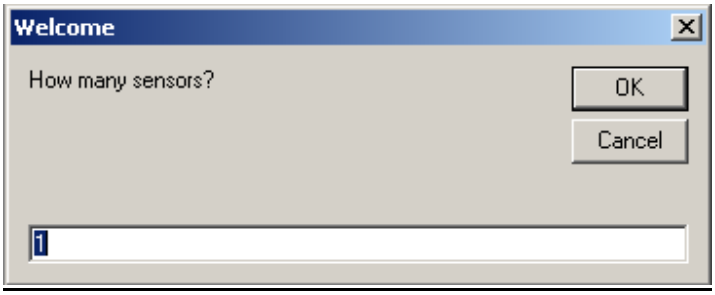


Figura 6.15. Inicio de la Aplicación.

Con esta información la aplicación crea una estructura de datos para cada sensor. En cada una de estas estructuras serán almacenados los parámetros indicados a continuación:

- Identificador del receptor asociado al sensor en cuestión. Se sabe que este valor coincide con el identificador de los mensajes CAN que mandará dicho sensor.

- Identificador del sensor.

- Variable que almacenará el tipo de dato que se está recibiendo en cada momento, es decir, mensaje de vibración o de presión, temperatura y voltaje de la batería.

- Variable que almacena la misma información que la anterior pero refiriéndose al mensaje previo recibido. Será útil para la representación de dichos valores como se indicó anteriormente en este capítulo.

- Contadores de mensajes de cada tipo recibidos desde este sensor, de los mensajes totales recibidos y de los mensajes perdidos.

- Variables auxiliares para almacenar los datos y representarlos en las listas que se observan en la parte izquierda de la interfaz gráfica del usuario (ver Figura 6.14).

- Variable para indicar que se recibe un mensaje de longitud 7 *bytes* y se está a la espera del mensaje de longitud 5 *bytes* asociado.

- Variables de almacenamiento de los datos recibidos desde este sensor.

Tras esto, es necesario que el usuario inicialice el bus CAN para poder recibir datos de los sensores a través del botón *CAN_Ini* (ver Figura 6.14).

Con esto la aplicación se encuentra en proceso de registro. Se trata de registrar los identificadores de todos los sensores que están trabajando. Para ello, se espera a recibir un mensaje, leer el identificador del sensor que lo envía y registrar dicho sensor. En la interfaz gráfica del usuario se comenzará representando la información recibida por el primer sensor registrado, es decir, se dispone de seis gráficas en las que se comienza representando los datos de vibración en las direcciones X, Y, Z, la presión, la temperatura y el voltaje de la batería del primer sensor registrado. Posteriormente qué variables y de qué sensor representar será totalmente flexible y será elección del usuario.

El proceso de registro durará hasta que la aplicación haya registrado todos los sensores que se esperan según lo indicado por el usuario. Pero es necesario destacar que se irán almacenando los datos de los sensores tan pronto como hayan sido registrados, aunque el proceso de registro no se haya finalizado aún.

Como se observa en el esquema de la Figura 6.13, tras el registro del sensor se entra en el bucle principal de la aplicación, que va desde la recepción de un dato hasta la representación del mismo, en caso de que el usuario lo haya indicado en la interfaz gráfica. Se presenta en la Figura 6.16 un diagrama de flujo para seguir sin dificultad las operaciones realizadas en dicho bucle principal incluyendo el proceso de registro ya comentado.

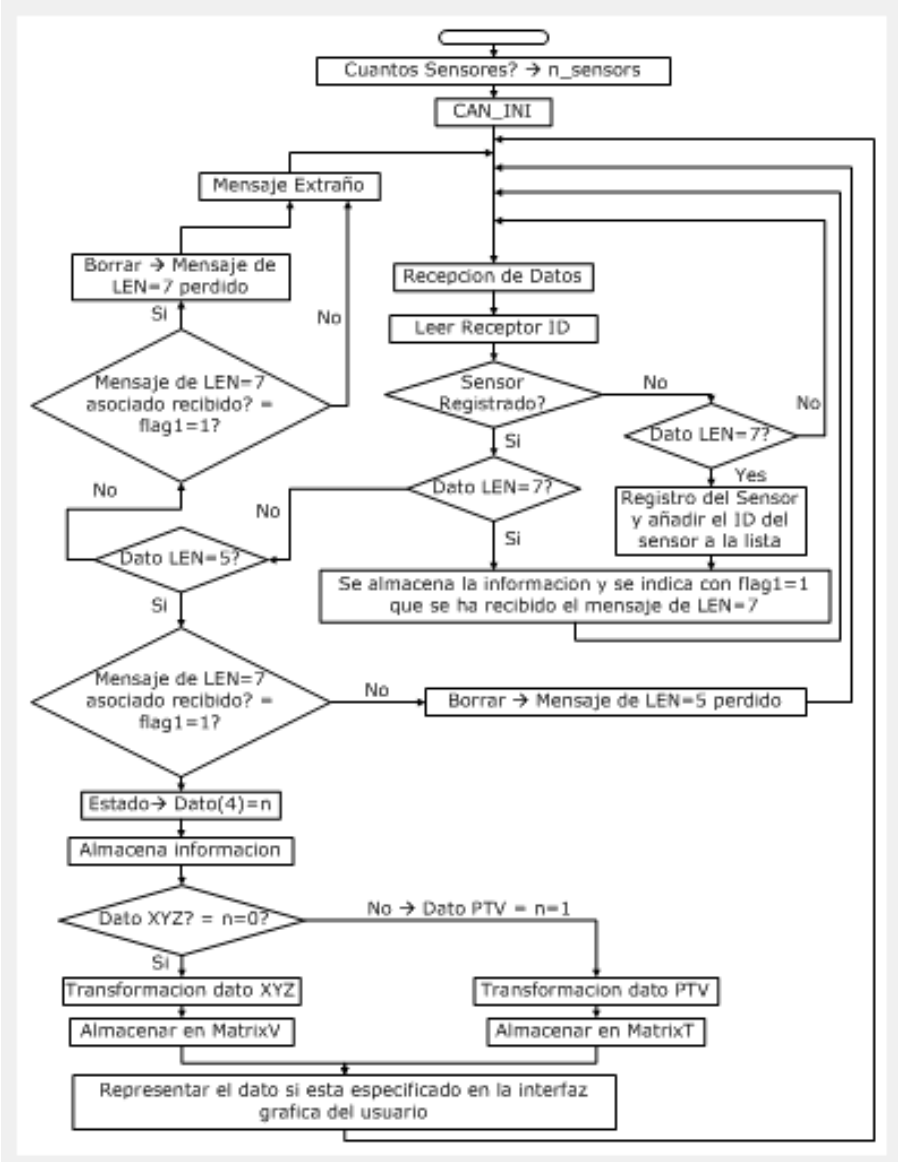


Figura 6.16. Bucle Principal de la Aplicación.

Se comentó en el capítulo anterior que los sensores mandan la información dividida en dos mensajes, tal y como se muestra en la Figura 6.17.

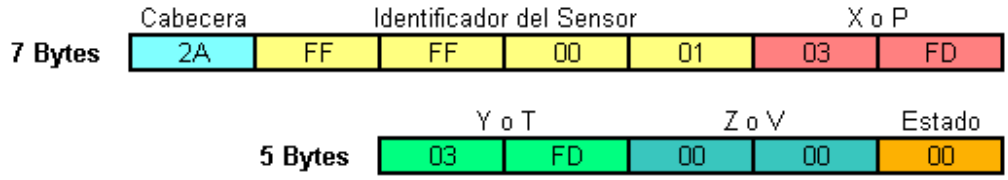


Figura 6.17. Mensajes de los Sensores.

Para descodificar dicha información se lleva a cabo el siguiente procedimiento (ver Figura 6.16):

- Se espera a recibir un mensaje y se lee el identificador del sensor. Si es de longitud 7 *bytes* y no ha sido registrado anteriormente, se registra añadiendo el número del sensor en la lista de sensores de la interfaz gráfica del usuario. Si ya había sido registrado se omite el proceso de registro. Se indica que se ha recibido un mensaje de longitud 7 *bytes* y se almacena la información recibida en dicho mensaje. En este punto todavía no se tiene conocimiento del tipo de dato recibido ya que el estado nos indicará el tipo de dato recibido y, como se observa en la Figura 6.17, éste viene almacenado en el último *byte* del mensaje de longitud 5.

  Cabe destacar que, aunque no se indica específicamente en el esquema del bucle principal de la aplicación que se presenta en la Figura 6.16, una vez que la aplicación ha registrado tantos sensores como esperaba, según lo indicado en un principio por el usuario, deja de preguntarse si el mensaje recibido proviene de un sensor registrado o no.

- Una vez recibido el mensaje de longitud 7 *bytes*, se observa si el siguiente mensaje es de longitud 5 *bytes*. Si no lo es, se elimina la información guardada porque no se considera un mensaje válido a menos que los mensajes de longitud 7 y 5 *bytes* sean recibidos de manera consecutiva. También se pierden los mensajes de longitud 5 *bytes* que se reciben sin ningún mensaje previo de longitud 7 *bytes* o los mensajes de cualquier otra longitud.

  Se tienen contadores de mensajes perdidos, los cuales se añadieron teniendo como objetivo observar la cantidad de información que se pierde. Sin embargo, se ha observado que, en este sentido, la información es recabada de una manera muy robusta sin apenas

ninguna pérdida. Solo se recogen mensajes perdidos algunas veces al arrancar o cuando se detienen los sensores y se vuelven a poner en marcha a través del dispositivo de disparo.

- Una vez que se reciben ambos mensajes de manera consecutiva, se averigua qué tipo de mensaje se ha recibido, almacenándose en una variable llamada "n" (como se observa en la Figura 6.16):

  - Estado = 0, mensaje de vibración en las tres direcciones del espacio X, Y, Z.

  - Estado = 1, mensaje de presión, temperatura y voltaje de la batería.

- Es importante la diferenciación anterior porque los mensajes de vibración están almacenados en dos *bytes* mientras que los de presión, temperatura y voltaje de la batería están almacenados en solo un *byte*.

  El valor a transformar cuando el dato viene almacenado en dos *bytes* es:

$$valor = byte(2) \cdot 256 + byte(1)$$

  Cuando el dato está almacenado en un solo *byte*, el valor a transformar es directamente el recogido en ese *byte*:

$$valor = byte(1)$$

  Además la transformación de cada tipo de dato es diferente teniendo en cuenta las escalas y las unidades de lo que se está midiendo en cada caso. Se presentan a continuación cada una de esas transformaciones para cada variable a tratar.

  - Vibración en la dirección X:

$$X = \frac{350}{1024} \cdot (valor - 1024)$$

  - Vibración en la dirección Y: ya se ha comentado anteriormente que los sensores proporcionan datos equivalentes en las direcciones X, Y, con la misma resolución. De ahí que la expresión asociada sea la misma.

$$Y = \frac{350}{1024} \cdot (valor - 1024)$$

- Vibración en la dirección Z:

$$Z = 10 \cdot valor$$

- Presión:

$$P = \frac{500}{255} \cdot valor$$

- Temperatura:

$$T = \frac{80}{255} \cdot valor$$

- Voltaje de la batería:

$$V = \frac{3}{255} \cdot valor$$

- ❖ Una vez transformados, los datos se almacenan en diferentes matrices según el estado, es decir, según sean datos de vibración o datos de presión, temperatura y voltaje de la batería.

- ❖ El último paso es la representación de los datos si el usuario lo ha requerido a través de su interfaz gráfica. Para ello, el programa recorre los listados asociados a cada gráfica. Primero lee el número del sensor y, en caso de coincidir con el que acaba de mandar un dato, se pasa a la lectura de qué variable se está solicitando.

Volviendo al esquema general de la aplicación presentado en la Figura 6.13, puede observarse que hay tres eventos que interrumpen el bucle principal cada cierto tiempo.

El primero de ellos es el incremento de las variables usadas para contabilizar el tiempo. Se incrementan cada 10 milisegundos y cuando alcanzan un cierto valor además se procede al borrado de las gráficas.

Las gráficas de vibración se borran cada segundo, pero sin embargo las de presión, temperatura y voltaje se borran cada cien segundos ya que se reciben muchos menos datos de este tipo. Es por eso que las escalas en el eje horizontal de cada tipo de gráfica son diferentes.

Otro evento que interrumpe al bucle principal es el almacenamiento de los datos y liberación de memoria. Se realiza cada treinta segundos y se trata de pasar toda la información recogida durante ese periodo de tiempo a unos archivos de texto. Por cada sensor se generan dos archivos de texto:

- Uno para guardar toda la información de los datos de vibración. Contiene cuatro columnas que se refieren al instante de tiempo en que se recogió la información y los datos de vibración en cada una de las direcciones del espacio X, Y, Z.

- En el otro se guarda toda la información de los datos de presión, temperatura y voltaje de la batería en el mismo formato que el anterior.

Una vez que los datos son almacenados, se libera memoria mediante el borrado de dicha información. Se reutilizan las matrices en las que se estaban almacenando los datos para continuar con los datos de los siguientes treinta segundos. Haciendo esto, se evita que la gran cantidad de información a almacenar haga a la aplicación trabajar de una manera más lenta.

Por último, el bucle principal de la aplicación puede ser interrumpido y, en este caso abortado, por el evento *Exit*. Al pulsar este botón se almacena en los archivos de texto la información que quede en las matrices y se genera un archivo de texto que recoge información general de lo ocurrido durante la ejecución de la aplicación.

Este archivo general contiene un listado para cada sensor tal y como el que se muestra en la Figura 6.18. Se indican los siguientes parámetros:

- El identificador del sensor.

- El identificador del receptor asociado a dicho sensor, que coincide con el de los mensajes CAN en los que se ha recibido la información.

- Los nombres de los archivos de texto asociados a dicho sensor.

- Recapitulación del número total de mensajes recibidos de cada tipo enviados por dicho sensor.

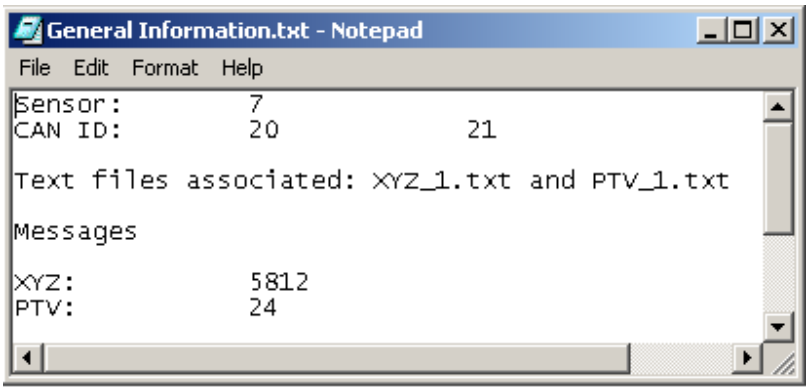


Figura 6.18. Archivo General Generado al Finalizar la Aplicación.

Merece la pena destacar algunos detalles más acerca de la aplicación definitiva. Tal y como se observa en la interfaz gráfica del usuario (Figura 6.14), cada gráfica lleva asociados dos listados. Uno con los identificadores de los sensores con los que se está trabajando y otro con las seis variables medidas por los mismos, es decir, X, Y, Z, P, T, V. En ninguno de ellos se le da la posibilidad al usuario de escribir, solo de elegir de la lista. De esta manera, se evitan posibles confusiones. Cada vez que el usuario cambia alguno de estos parámetros, la gráfica asociada es borrada. Así, se evita que el usuario tienda a comparar datos tomados por distintos sensores o incluso variables diferentes.

A su vez, cada gráfica lleva asociado un cursor deslizante. Se trata de un zoom de manera que cuando el usuario varía su posición, se modifica el tamaño de la escala de la gráfica asociada. Al igual que al cambiar de sensor o de variable, cuando se modifica la posición del cursor deslizante del zoom, la gráfica asociada es borrada. El objetivo vuelve a ser evitar que el usuario pueda comparar datos en distintas escalas, lo cual carecería de sentido. También se varían las etiquetas que indican los valores extremos del eje de ordenadas.

En conclusión, el desarrollo de la aplicación ha resultado tedioso por varios motivos. Entre los cuales destacarían: la falta de información detallada de los datos que los sensores mandan, incertidumbre de las escalas o unidades en las que se recogen los datos, la gran cantidad de datos a almacenar que hacia el proceso más complicado y mucho más lento, problemas inesperados del funcionamiento del *hardware* utilizado, etc. Por todos estos motivos la aplicación ha sido desarrollada buscando flexibilidad y facilidad de adaptación a los posibles cambios venideros.

Capítulo 7

Análisis de los Datos Recogidos

7.1. Introducción

Los sistemas de monitorización de vibración pueden dividirse principalmente en tres partes: adquisición de datos, extracción de características y clasificación. En este trabajo se han intentado abarcar cada una de esas partes. La adquisición de datos ha sido la parte en la que se ha trabajado más profundamente con el desarrollo de la aplicación en *Visual Basic*. En este capítulo se describirán las pruebas realizadas y los resultados obtenidos. Para la obtención de resultados se tratan las otras dos partes: extracción de características y clasificación.

Como se ha explicado en el capítulo anterior, la utilización de la aplicación desarrollada genera unos ficheros de texto en los que se almacena toda la información relevante. Esos ficheros de texto contienen todos los datos recogidos que serán sometidos a análisis posteriormente. Como también se comentó en el capítulo anterior, este análisis posterior se ha realizado utilizando como herramienta el *software Matlab*.

En este capítulo se describirán en detalle los distintos análisis realizados de los datos. De forma cronológica, dichos análisis realizados han sido:

- Un primer programa en *Matlab* que simplemente importa los datos de los archivos de texto creados mediante la ejecución de la aplicación en *Visual Basic* y los representa.

- Análisis de datos previamente recogidos en plataformas de prueba con el objetivo de estudiar y familiarizarse con el tipo de análisis a realizar en los datos recogidos durante este proyecto.

- Recogida de datos reales en plataformas de prueba en laboratorio. Dichas plataformas tienen distintos fallos y con el análisis de los datos se trata de identificarlos y diferenciarlos unos de otros.

  - En una primera etapa se recogieron datos con unos sensores no inalámbricos previamente instalados.

  - Posteriormente, se instalaron los sensores inalámbricos en dichas plataformas y se realizaron los mismos análisis para identificación de fallos. Tratando de obtener la misma información que con los sensores no inalámbricos.

7.2. Representación de los Datos Recogidos

Este primer programa *"Automatic_v2"* es muy simple. Al principio se le pide al usuario el número de sensores con el que se ha trabajado. Con esta información se representarán seis gráficas por cada sensor: vibración en cada dirección del espacio X, Y, Z, presión, temperatura y voltaje de la batería.

A continuación, en las Figuras 7.1-7.6, se presentan como ejemplo dichas gráficas para un solo sensor. Dichos datos fueron recogidos mediante la aplicación en *Visual Basic* en ficheros de texto, agitando el sensor manualmente.

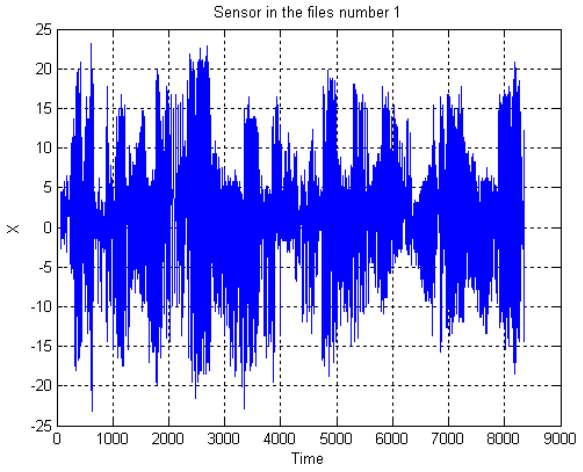


Figura 7.1. Vibración en Dirección X.

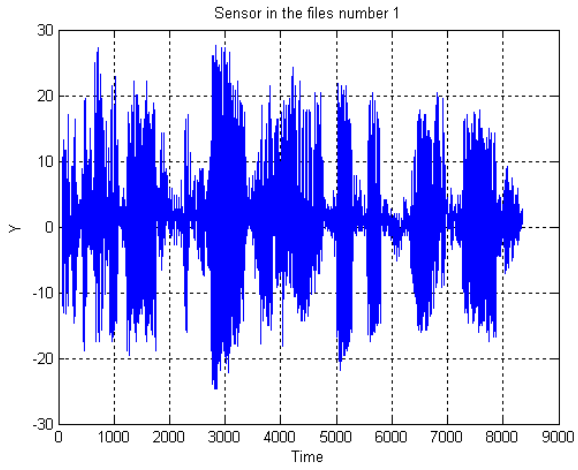


Figura 7.2. Vibración en Dirección Y.

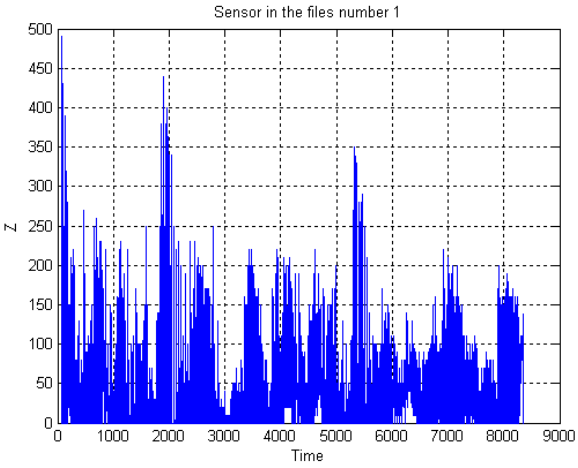


Figura 7.3. Vibración en Dirección Z.

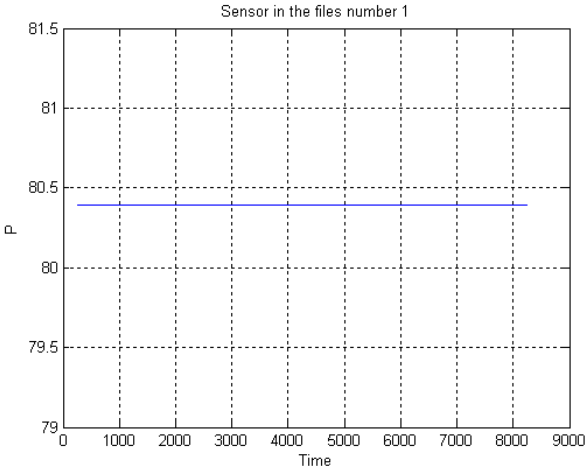


Figura 7.4. Presión.

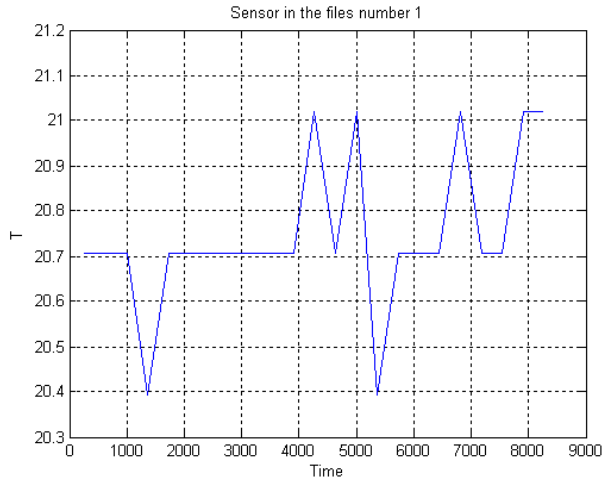


Figura 7.5. Temperatura.

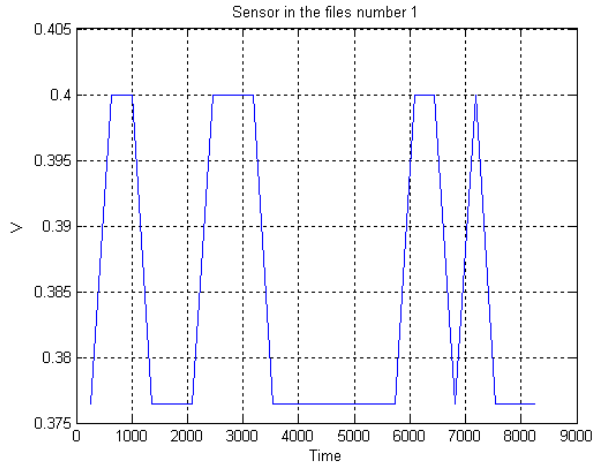


Figura 7.6. Voltaje de la Batería.

7.3. Descripción de las Plataformas Utilizadas

Aprovechando el material disponible en laboratorio, se realizaron distintas pruebas para comprobar el correcto funcionamiento de la aplicación y comenzar a esbozar lo que sería la verdadera aplicación en un futuro, es decir, recogida de datos, análisis de los mismos, detección e identificación de fallos.

Se disponía de seis plataformas de prueba, compuestas por un sistema mecánico rotativo, un acelerómetro de un eje, una tarjeta de adquisición de datos PCI 1711 y un PC. Cada una de ellas con un fallo diferente:

- *Bearing Defects*: fallo en un rodamiento.

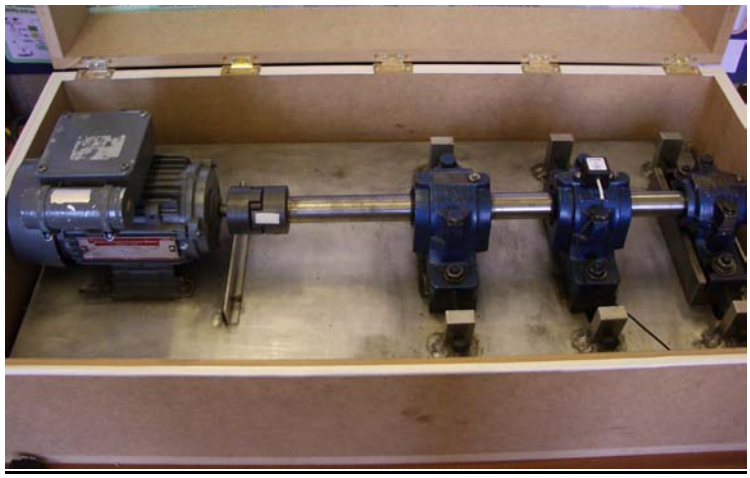


Figura 7.7. Plataforma de Prueba – Fallo Bearing.

- *Gearmesh* o *Gearbox Faults*: engranajes con un diente menos.

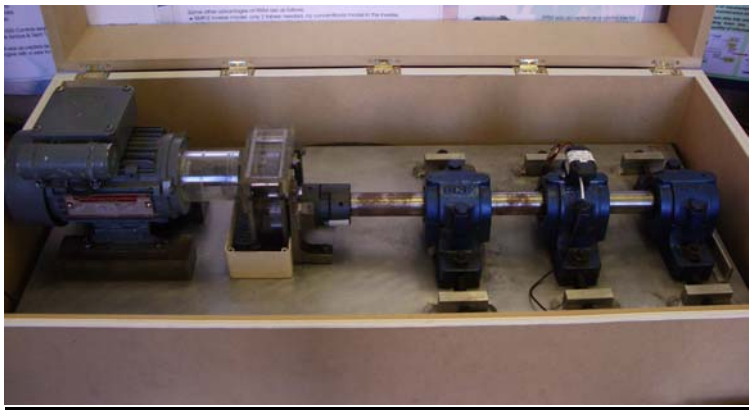


Figura 7.8. Plataforma de Prueba – Fallo Gearmesh.

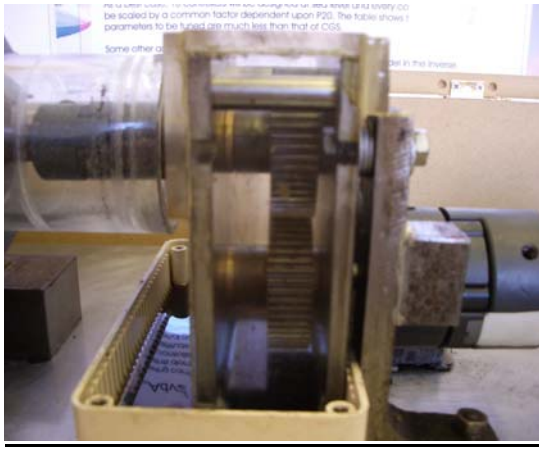


Figura 7.9. Detalle de los Engranajes.

- *Resonance*.

- *Imbalance*: eje desequilibrado.

- *Misalignment*: eje desalineado.

- *Multifault*: se trata de una plataforma que contiene varios de los fallos anteriores. No se ha trabajado en profundidad con ella porque la detección de cada fallo por separado podía resultar más tediosa.

Como se ha comentado, cada una de ellas lleva instalado un acelerómetro. Se trata de un sensor de vibración bastante preciso basado en un material piezoeléctrico integrado con acondicionamiento de señal y regulación electrónica. Sus principales características son que añade poco ruido a la señal y que abarca un amplio ancho de banda, 0.3-10000 Hz. Dicho acelerómetro se muestra en la Figura 7.10.

Figura 7.10. Acelerómetro Crossbow.

Según la posición en la que esta instalado mide la vibración en la dirección perpendicular al eje de rotación, Z, tal y como se muestra en la Figura 7.11.

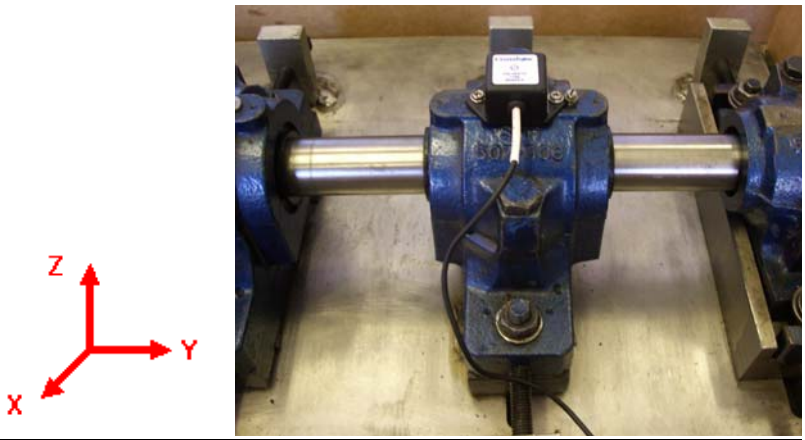


Figura 7.11. Acelerómetro Instalado en las Plataformas de Prueba.

Por tanto, se miden datos de vibración mediante el acelerómetro de un eje y se capturan usando un PC basándose en un sistema de adquisición vía PCI 1711. El sistema de adquisición consiste en el uso de la herramientae *Matlab Simulink* y el *Real-Time Windows Target* (RTWT).

Se usó el modelo en *Simulink* que se muestra en la Figura 7.12 para adquirir, representar y almacenar la señal de vibración durante la ejecución en tiempo real. El modelo en *Simulink "corus.mdl"* está compuesto por 4 bloques:

- Entrada analógica *(Analog Input):* se usa para seleccionar los canales de entradas analógicas y el rango de voltaje.

- *Clock*: es el reloj del sistema y da como salida el tiempo de simulación actual en cada paso de simulación.

- *To Workspace*: es un bloque usado para exportar datos al espacio de trabajo de *Matlab*. Hay dos bloques de este tipo en el modelo usado, uno para exportar los datos de vibración leídos por PCI 1711 y otro para exportar el tiempo del sistema.

- *Scope*: es un bloque para ver los datos representados en tiempo real, en este caso las señales de vibración recogidas.

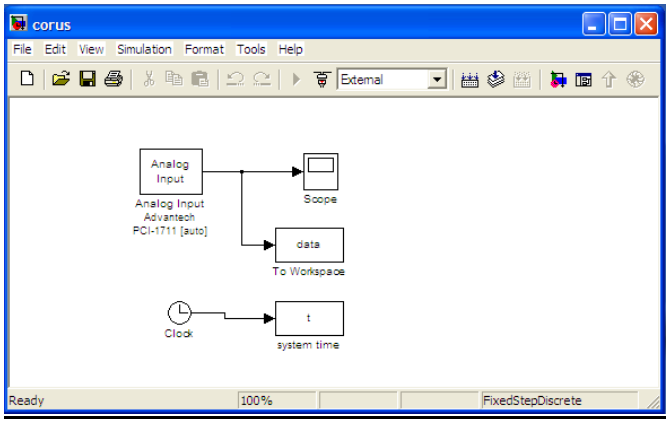


Figura 7.12. Modelo en Simulink.

7.4. Análisis de Datos

7.4.1. Recogidos por Sensores Cableados

Teniendo como objetivo familiarizarse con el tipo de análisis a realizar en los datos, se comenzó analizando datos previamente recogidos. Para ello se

usaron archivos de 50000 datos de cada uno de los fallos individuales previamente mencionados. Cada archivo contiene un vector de medidas de vibración, dichos datos fueron recogidos en cada una de las plataformas con fallos durante 50 segundos a una frecuencia de 1 kHz.

En esta etapa se desarrolló un programa en *Matlab* (*"task2"*) para llevar a cabo el análisis de los vectores de vibración contenidos en los archivos mencionados. Lo primero que hace el programa es cargar los archivos en *Matlab*. Después se lleva a cabo la extracción de características y a partir de ellas la representación para la obtención de resultados.

Para la extracción de características se lleva a cabo un análisis en frecuencia de las señales medidas. Se seleccionaron 4 características basándose en el valor cuadrático medio, media cuadrática o RMS del inglés *Root Mean Square* del cálculo de *PSD (Power Spectral Density).* Así, las características calculadas son:

- *RMS* de la señal.
- *RMS* de las bajas frecuencias, es decir, de 0 a 50 Hz.
- *RMS* de las frecuencias medias, es decir, de 50 a 200 Hz.
- *RMS* de las altas frecuencias, es decir, de 200 a 500 Hz.

Cada vector de vibración de longitud 50000 se divide en 50 vectores de longitud 1000 sin solaparse. Las características se calculan basándose en cada uno de esos bloques que corresponden a 1 segundo de medidas. Se obtendrán por tanto las 4 características de cada bloque.

Se describe a continuación el procesamiento de cada uno de esos bloques de 1000 datos:

- Normalización: cada vector de longitud 1000 se normaliza mediante la sustracción de la media.

$$X\_Normalizado = X - \overline{X} \tag{7.1}$$

  Podemos renombrar a cada vector normalizado $X\_Normalizado$ como $x_i \quad i = 1...50$. A partir de estos vectores normalizados serán calculadas las características.

- Cálculo de *PSD* de cada vector normalizado $x_i \quad i = 1...50$ mediante el método de *Welch*, al vector resultante podemos llamarle $y_i \quad i = 1...50$.

- Primera característica: se trata de la *RMS* de la señal completa.

$$f_1 = \frac{\|y_i\|_2}{\sqrt{n}} \qquad i = 1...50 \tag{7.2}$$

Donde $y_i \quad i = 1...50$ es la *PSD* del vector $x_i \quad i = 1...50$, $\|y_i\|_2$ es la norma 2 de $y_i \quad i = 1...50$ y n es la longitud de cada vector $x_i \quad i = 1...50$, en este caso se sabe que es 1000.

- Segunda característica: se filtra la señal con un filtro paso bajo dejando las frecuencias por debajo de 50 Hz y después se calcula la *PSD*. Se ha usado el filtro *Butterworth*, en particular de orden 11, el cual proporciona una respuesta lo más plana posible tanto en la banda de paso como en la banda parada. Después se halla la *RMS* de las bajas frecuencias mediante la formula 7.2.

- Tercera característica: se filtra la señal con un filtro paso banda, siendo la banda que se deja pasar 50-200 Hz. Se vuelve a usar el filtro *Butterworth*, pero de orden 13 en este caso, y después se calcula la *PSD*. Se halla la *RMS* de las frecuencias medias mediante la fórmula 7.2.

- Cuarta característica: se filtra la señal con un filtro paso alta dejando pasar las frecuencias por encima de 200 Hz y después se calcula la *PSD*. Se vuelve a usar el filtro *Butterworth*, pero de orden 18 en este caso. Después se halla la *RMS* de las altas frecuencias mediante la fórmula 7.2.

Combinando las cuatro características en un vector fila se tiene:

$$F = \begin{bmatrix} f_1 & f_2 & f_3 & f_4 \end{bmatrix}_{1x4} \tag{7.3}$$

Se obtienen así para cada fallo, 50 vectores de características extraídas de los 50 bloques de 1000 datos cada uno. Combinando estos 50 vectores de características en una matriz se obtiene para cada fallo:

$$Fault1 = \begin{bmatrix} F_1 \\ F_2 \\ \vdots \\ F_{50} \end{bmatrix}_{50x4}$$

Y combinando las matrices obtenidas a partir de cada tipo de fallo en una sola matriz:

$$G = \begin{bmatrix} Fault1 \\ Fault2 \\ \vdots \\ Fautl5 \end{bmatrix}_{250x4}$$

Se tienen vectores de características de 4 dimensiones. Para visualizar dichas características, es necesario transformar en datos de 2 dimensiones. Para ello se realiza un análisis de componentes principales.

La Figura 7.13 muestra lo obtenido al representar todos los puntos mediante las 2 componentes principales. Se observa que algunos de los fallos pueden diferenciarse fácilmente, pero otros sin embargo solapan, como pasa con *bearing* e *imbalance*, encontrando ambigüedad en la identificación de dichos fallos.

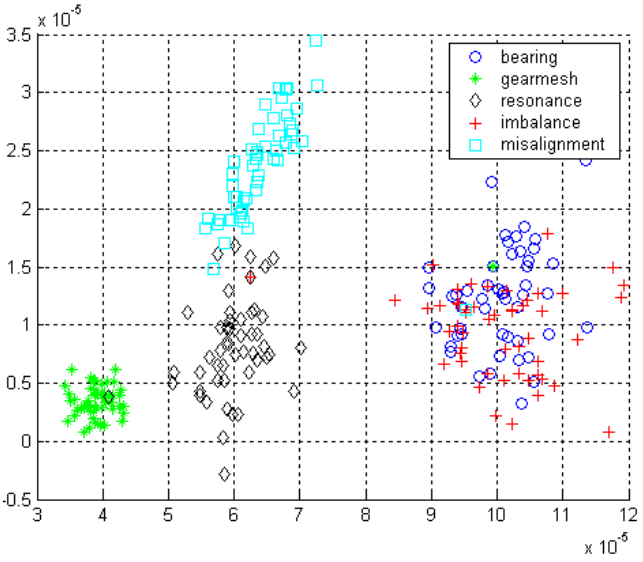


Figura 7.13. Componentes Principales.

Se pensó en usar las componentes no principales o secundarias obteniéndose una representación tal y como se muestra en la Figura 7.14. Parece que, a pesar de lo esperado, los fallos están mejor separados usando estas dos componentes, pero todavía podría haber confusión entre *bearing* e *imbalance*.

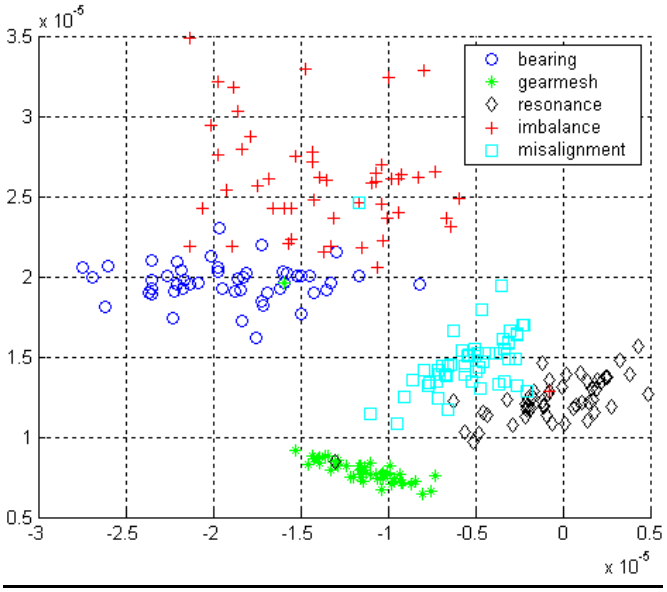


Figura 7.14. Componentes Secundarias.

Intentando perder la ambigüedad se trató una representación en tres dimensiones, usando en este caso las tres componentes principales. El resultado se muestra en la Figura 7.15 observándose que no se consigue la separación total de los fallos.

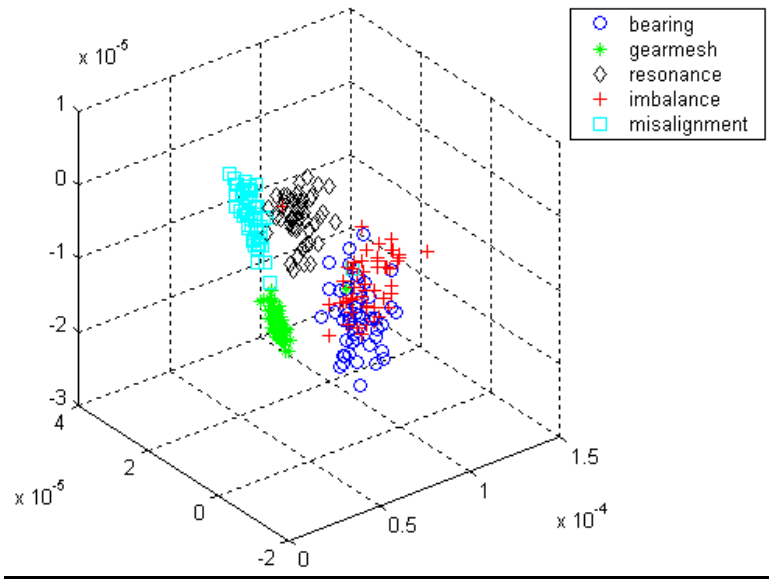


Figura 7.15. Tres Componentes Principales.

Todos estos resultados hacen pensar que la utilización de los sensores inalámbricos, que miden la vibración en las tres direcciones del espacio, podría proporcionar más información y eliminar la ambigüedad obtenida.

7.4.2. Recogidos por Sensores Inalámbricos

Una vez familiarizados con el análisis de datos a realizar, se procedió a la instalación de los sensores inalámbricos en las plataformas de prueba y al análisis de los datos recogidos a través de ellos.

Se trabajó con dos plataformas de prueba, aquellas con fallos *gearmesh*, en la cual se instaló el sensor 7, y *bearing*, en la que se instaló el sensor 8. Con dos plataformas era suficiente para recoger datos y proceder a la identificación de fallos.

Se realizaron varias pruebas para averiguar en qué posición se obtendría la información mas útil a partir de los sensores inalámbricos, teniendo en cuenta que es posible recabar una mayor amplitud de datos por tener medidas en las tres direcciones del espacio. Pero, a su vez, había que tener en cuenta que la resolución en la dirección Z es menor que en las direcciones X e Y.

En la Figura 7.16 se muestra la primera posición que se probó. Se comprobó que se hacía coincidir la dirección de menor resolución, Z, con la dirección en la que el acelerómetro cableado tomaba sus medidas, lo cual no era conveniente.

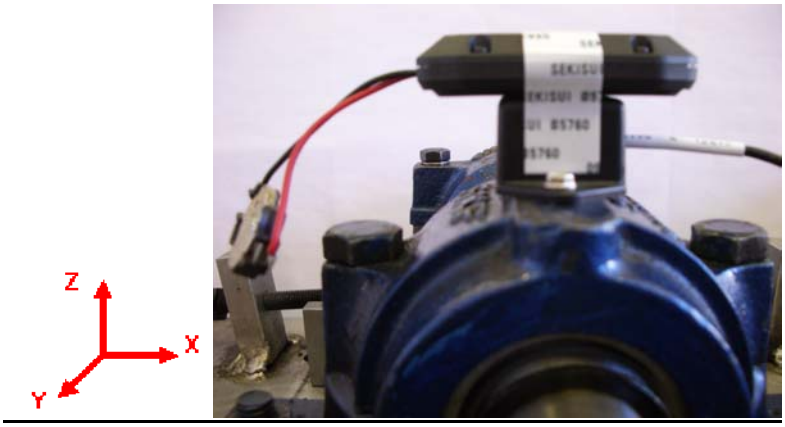


Figura 7.16. Instalación de Sensores Inalámbricos - Primera Posición.

En la Figura 7.17 se presentan tres imágenes. Tratando de aprovechar la resolución en las direcciones X e Y se modificó la posición de los sensores, como se muestra en la imagen de la izquierda. El hecho de que la batería no se encuentre adherida al resto del cuerpo del sensor, añadía un desequilibrio que modificaba la vibración. Por ello, se trató de solucionar intentando centrar el centro de gravedad, como se observa en las otras dos imágenes. Aún así, y debido a los medios de los que se disponía, al separar el sensor tanto del

eje y no poder estabilizarlo por completo, se observaron anomalías en las medidas.

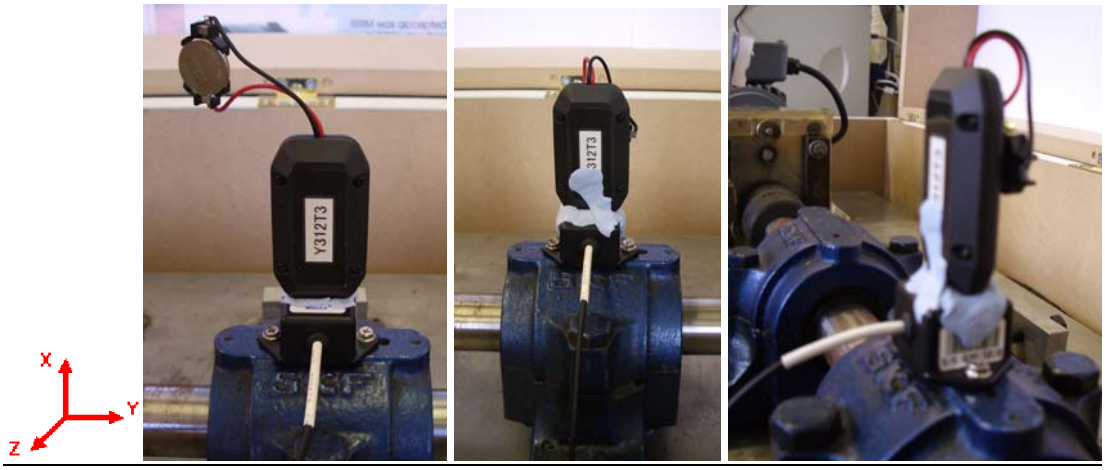


Figura 7.17. Instalación de Sensores Inalámbricos - Segunda Posición.

Es por eso que se recurrió a la posición que se muestra en la Figura 7.18, bastante más estable que la anterior y por tanto más fiable en las medidas. Es en esta posición en la que definitivamente se llevaron a cabo las pruebas.

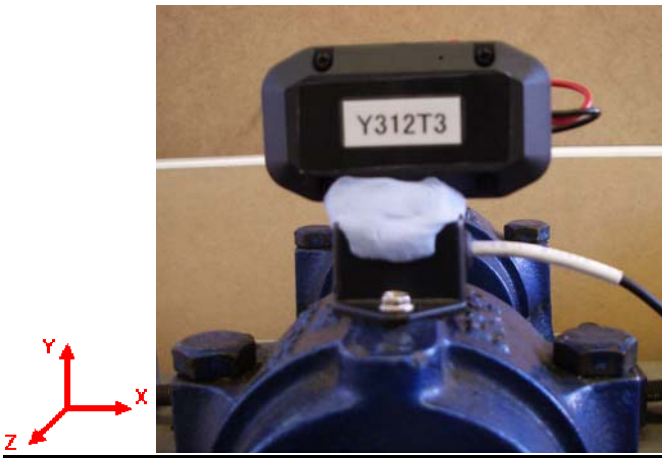


Figura 7.18. Instalación de Sensores Inalámbricos Tercera Posición.

Se programaron en *Matlab* dos programas que son básicamente iguales, *"processing78"* y *"processing87"*. Solo se diferencian en el orden de lectura de los datos, dependiendo de qué sensor fue registrado en primer lugar durante la recogida de datos mediante la aplicación en *Visual Basic*. Es necesario tener en cuenta esto porque la aplicación en *Visual Basic*

almacena los datos en diferentes archivos de texto según el orden en el que registre los sensores durante el proceso de registro.

Explicando brevemente lo que hacen dichos programas, podrían dividirse en los siguientes pasos:

- Cargan los datos recogidos importándolos desde los archivos de texto creados por la aplicación en *Visual Basic* para el almacenamiento de los mismos.

- Comprueban que dichos archivos son lo suficientemente extensos como para poder analizar y representar una cantidad de datos representativa. Se ha considerado que los archivos deben contener al menos 25000 puntos.

- Se procede al análisis de los datos, dividiéndolos en vectores de longitud 5000. Para ello se llama a la función *"calculo_mPSD"* la cual realiza el procesamiento descrito en el punto anterior, es decir, normalización de los datos y cálculo de características. Esta función devuelve la matriz $Fault1$ y además devuelve el resultado del análisis en el dominio de la frecuencia de las señales, el cual se realiza mediante la estimación de la *Power Spectral Density (PSD)* por el método de *Welch* de las medidas contenidas en cada vector de 5000 datos.

  Las gráficas obtenidas mediante la representación de las *PSDs* de cada tipo de fallo se presentan en las Figuras 7.19, 7.20 y 7.21.

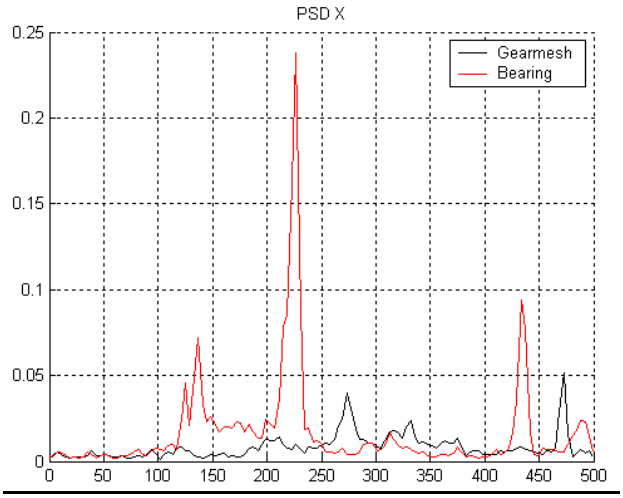


Figura 7.19. Power Spectral Density en Dirección X.

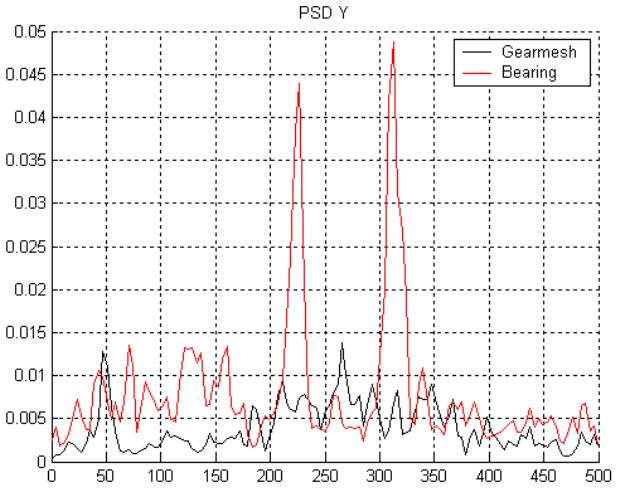


Figura 7.20. Power Spectral Density en Dirección Y.

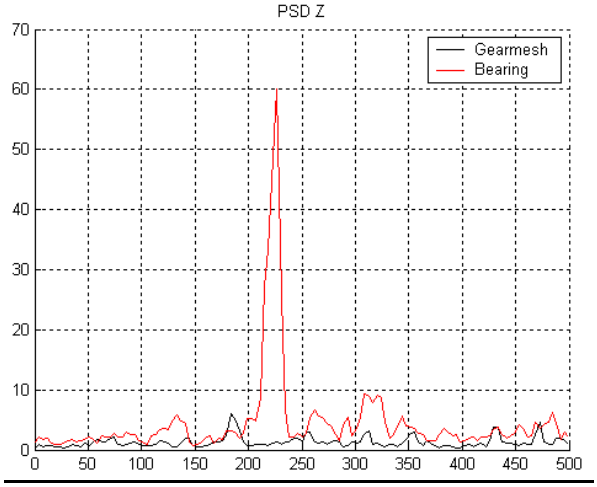


Figura 7.21. Power Spectral Density en Dirección Z.

Se observa que en cada tipo de fallo hay picos característicos diferentes y a distintas frecuencias. Es importante destacar que ha sido observada similitud en la posición y en el tamaño de dichos picos en distintas ejecuciones del programa. Es por tanto obvio que mediante el análisis de dichas representaciones sería posible caracterizar cada tipo de fallo por esos picos y posteriormente pasar a la diferenciación de los fallos.

- Se realiza la transformación en dos dimensiones de los vectores de características y se representan las nubes de puntos obtenidas según las componentes principales.

Los resultados obtenidos se muestran en las gráficas que se presentan a continuación en las Figuras 7.22, 7.23 y 7.24. Cada una de ellas correspondiente a los resultados obtenidos a partir de los datos recogidos en cada dirección del espacio X, Y, Z.

Al igual que las gráficas anteriores de las *PSDs*, se representan en rojo los puntos correspondientes al fallo *Bearing* y en negro los del fallo *Gearmesh*.

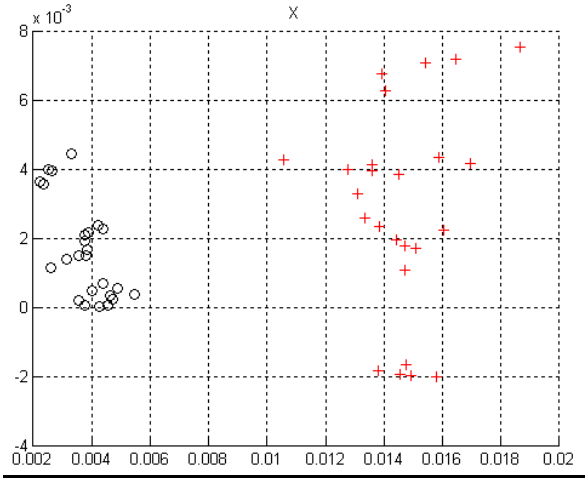


Figura 7.22. Análisis de Componentes Principales en Dirección X.

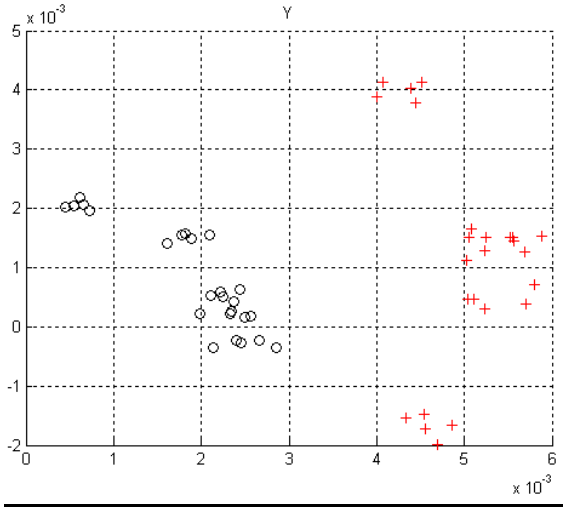


Figura 7.23. Análisis de Componentes Principales en Dirección Y.

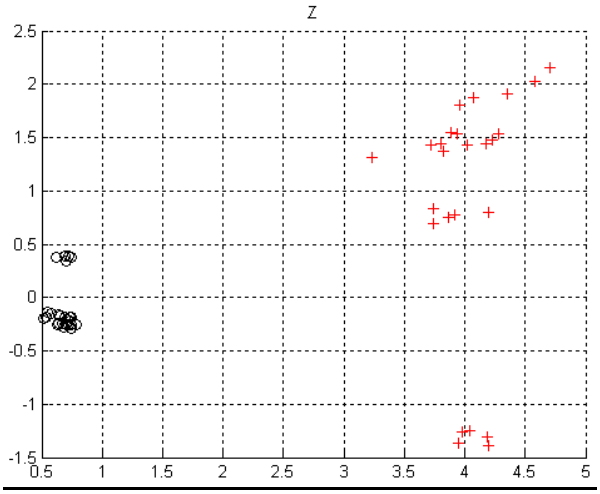


Figura 7.24. Análisis de Componentes Principales en Dirección Z.

Pueden distinguirse de una manera bastante clara dos grupos de puntos en cada gráfica correspondientes a cada tipo de fallo analizado. Así, sería posible implementar un clasificador disponiendo de una cantidad considerable de puntos que sirvieran de entrenamiento y podría llevarse a cabo la detección de fallos de una forma simple y cómoda.

Esta parte no se ha llevado a cabo de una manera explícita en este trabajo, pero es sencillo pensar en la implementación de un clasificador de mínima distancia o del vecino más cercano. Bastaría con tener un grupo significativo de puntos de entrenamiento, caracterizar cada clase de fallo por el punto medio de ese conjunto de puntos de entrenamiento y después en cada medida de vibración analizar los datos y clasificarlos en uno u otro tipo de fallo dependiendo de a qué punto medio se encuentre más cerca el punto obtenido en el análisis.

Matemáticamente, si se caracteriza cada clase por un punto medio con un vector de características tal que:

$$\overline{F_i} = \left[\bar{f}_1 \;\; \bar{f}_2 \;\; \bar{f}_3 \;\; \bar{f}_4\right]_{1x4} \qquad i = 1,2,...,clases$$

Y la muestra en estudio se caracteriza por un vector de características como el indicado en 7.3:

$$F = \left[f_1 \;\; f_2 \;\; f_3 \;\; f_4\right]_{1x4} \qquad (7.3)$$

A partir de estos vectores se hallan las distancias desde el punto en estudio al punto medio de cada clase mediante la siguiente expresión:

$$D_i(F,\overline{F_i}) = \sqrt{\left[\left(f_1-\bar{f}_1\right)^2+\left(f_2-\bar{f}_2\right)^2+\left(f_3-\bar{f}_3\right)^2+\left(f_4-\bar{f}_4\right)^2\right]} \qquad i=1,2,...,clases$$

Se clasificará F como clase i si:

$$D_i < D_j \qquad \forall j \in clases \neq i$$

7.4.3. Comparación

Para llevar a cabo la comparación de los datos recogidos por los sensores inalámbricos con los adquiridos mediante los sensores no inalámbricos se realizaron distintos análisis. Explicando paso a paso, de manera cronológica, se procedió de la siguiente manera:

- Primero se recogieron datos de las dos plataformas de prueba anteriormente utilizadas, *bearing* y *gearmesh*, por medio de los sensores no inalámbricos previamente instalados o sensores *crossbow*. Dichos datos se adquirieron mediante el modelo en *Simulink* presentado anteriormente en este capítulo en la Figura 7.12. Cada vez que se ejecuta el programa se recogen datos durante 5 segundos a una frecuencia de 1 kHz, es decir, se obtienen archivos de 5000 datos.

- Para el fallo *gearmesh*, se compararon dichos datos con los de los vectores de 50000 puntos que habían sido recogidos previamente por esos mismos sensores. Para esta comparación se utilizó el programa en *Matlab "scale"*. La primera diferencia se encontraba en el tamaño de los archivos que se querían comparar, pero esto no suponía algo difícil de subsanar. Sin embargo, en contra de lo esperado, los datos en sí eran muy diferentes como se puede observar en la Figura 7.25.

Se planteó que podía resultar un problema de escalas así que se intentó resolver normalizando los datos. Para la normalización se sustrajo la media a ambos grupos de datos y se halló la *RMS*. Se trató de igualar las escalas multiplicando los datos recogidos por el ratio que se indica a continuación:

$$ratio = \frac{RMS\_\mathrm{Pr}evios}{RMS\_\mathrm{Re}cogidos} \qquad (7.4)$$

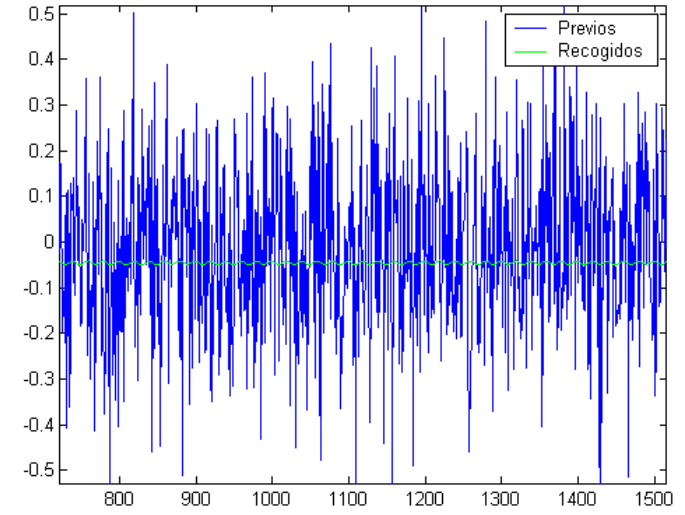


Figura 7.25. Comparación Entre los Datos de los Sensores Cableados o Sensores Crossbow.

El resultado obtenido mejoró el resultado previo sin tratamiento, pero aun así no fue muy exitoso como se muestra en la Figura 7.26.

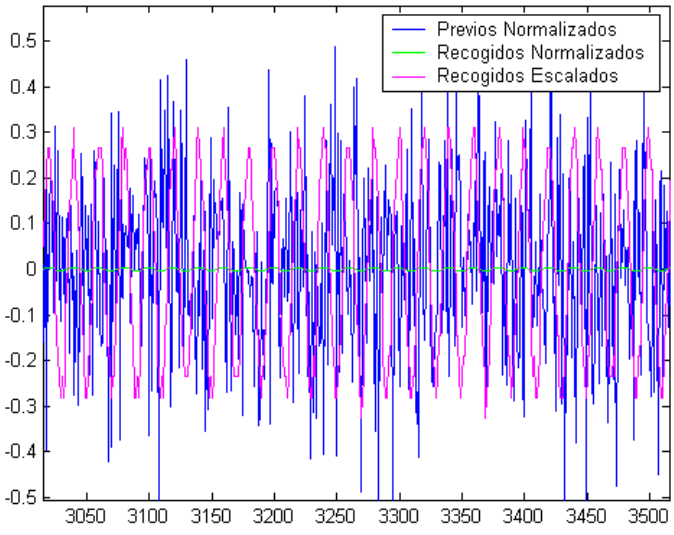


Figura 7.26. Comparación con los Datos Normalizados y Escalados.

- Tratando de solucionar esta discordancia entre los datos recogidos por sensores *crossbow*, se pensó en realizar un tratamiento diferente de los datos. El programa utilizado es *"normalisation"* y básicamente trata de realizar la comparación tras la extracción de características y el análisis de componentes principales.

Se recogieron datos de las dos plataformas de prueba anteriormente utilizadas, *bearing* y *gearmesh*, mediante el modelo en *Simulink* presentado anteriormente en este capítulo en la Figura 7.12.
En un primer paso, se representan las nubes de puntos de los datos que fueron recogidos en un trabajo previo a éste. Después se importan los datos recogidos por los sensores *crossbow* en las plataformas de prueba, se sustrae la media, se calcula la *RMS*, se multiplica por el ratio indicado en 7.4 y se realiza el análisis de componentes principales a los datos tratados.

El resultado obtenido se muestra en la Figura 7.27. Se observa que las nubes de puntos de los datos recogidos están perfectamente separadas en dos clases y que servirían para la identificación de fallos. Sin embargo, dichas nubes de puntos no coinciden con las anteriormente representadas. Al no tener conocimiento exacto de cómo fueron recogidos los datos y de si dichos datos tienen algún tratamiento previo desconocido, pasamos a centrarnos en la comparación con los datos recogidos por los sensores inalámbricos.

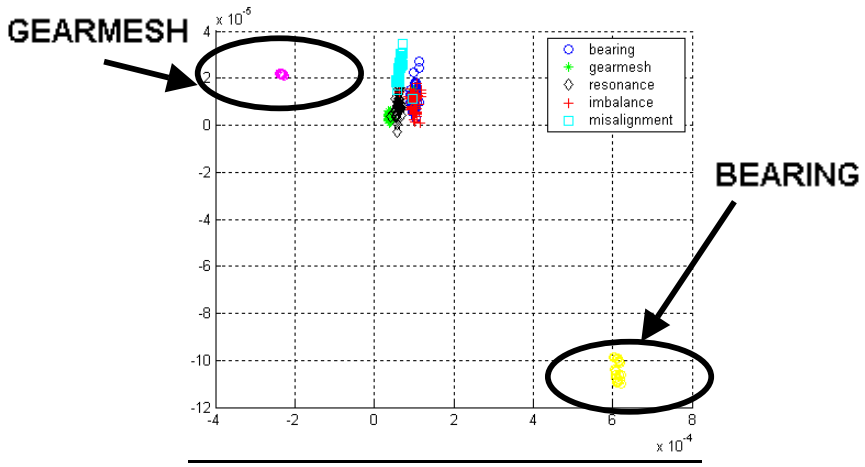


Figura 7.27. Análisis de Componentes Principales. Comparación Datos Previos y Recogidos.

- Se recogieron datos de las dos plataformas de prueba anteriormente utilizadas, *bearing* y *gearmesh*, por medio de ambos tipos de sensores. Los datos de los sensores inalámbricos se recogieron mediante la aplicación en *Visual Basic* desarrollada en este proyecto y los datos de los sensores no inalámbricos se adquirieron mediante el modelo en *Simulink* presentado anteriormente en este capítulo en la Figura 7.12.

En esta etapa se realizó un tipo de procesamiento un poco más profundo a los datos recabados para intentar hacer coincidir los puntos recogidos con los distintos tipos de sensores.

Los archivos de datos recogidos por los sensores *crossbow* son de longitud 5000 y con los sensores inalámbricos es posible recoger tantos datos como se quiera. Es por ello que lo primero que hace el programa *"processing_ojo"* para poder comparar con archivos de misma longitud es mostrarle al usuario la longitud de los archivos de datos recogidos por sensores *wireless* y pedirle desde que valor quiere tomar los 5000 puntos.

Una vez que se cuenta con los archivos de 5000 puntos se pasa a normalizarlos. La normalización llevada a cabo consiste en la sustracción de la media y división por la varianza, es decir:

$$datos\_N = \frac{datos\_R - \bar{x}}{\sigma^2}$$

Siendo $datos\_N$ los datos normalizados, $datos\_R$ los datos recogidos, $\bar{x}$ la media de los datos recogidos y σ^2 la varianza de los mismos. Se realiza esta normalización a todos los datos con los que se va a trabajar.

Una vez hecho esto, se intenta un escalado de los datos. Para ello se hallan los valores máximos y mínimos de los datos recogidos por cada tipo de sensor en cada tipo de fallo y se procede de la siguiente manera:

$$datos\_crossbow = datosC\_N \cdot \frac{datosW\_Nmax - datosW\_Nmin}{datosC\_Nmax - datosC\_Nmin}$$

Obteniendo así los datos *crossbow* a comparar, $datos\_crossbow$, siendo $datosC\_N$ los datos *crossbow* normalizados mediante el procedimiento anteriormente indicado, $datosW\_Nmax$ y $datosW\_Nmin$ los valores máximo y mínimo de los datos recogidos por los sensores *wireless* normalizados y $datosC\_Nmax$ y $datosC\_Nmin$ los de los recogidos por los sensores *crossbow* normalizados.

Con esto, se representan dos gráficas que se muestran a continuación en la Figura 7.28. La primera se refiere a los datos recogidos con cada tipo de sensor en la plataforma de prueba con fallo *bearing* y la segunda en aquella con fallo *gearmesh*.

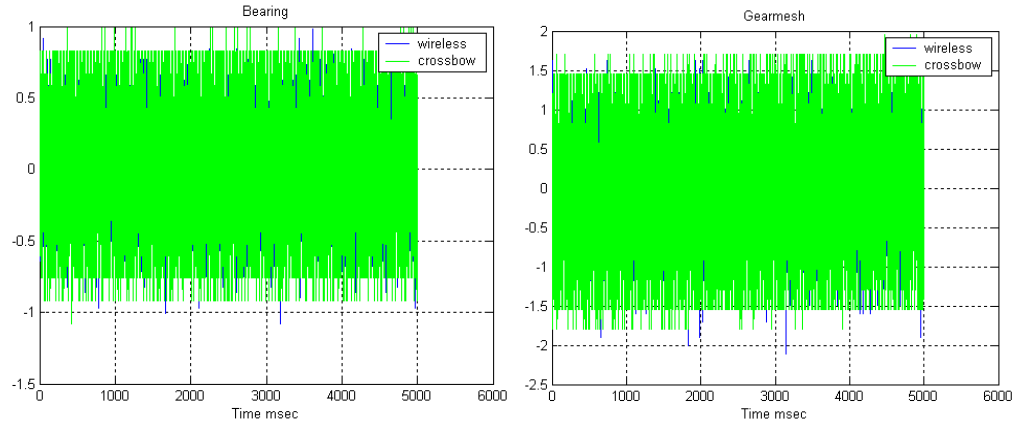


Figura 7.28. Comparación Datos Recogidos por Distintos Sensores. Fallos Bearing y Gearmesh.

El siguiente paso es la llamada a la función ya comentada *"calculo_mPSD"*. Esta función devuelve la matriz $Fault1$ de los datos recogidos por ambos sensores en las plataformas de prueba de cada tipo de fallo y el resultado del análisis en el dominio de la frecuencia de las señales, realizado mediante la estimación por el método de *Welch* de las *Power Spectral Density (PSD)* de las medidas contenidas en cada vector de 5000 datos.

Con estos datos, se realiza el análisis de componentes principales y se obtiene la representación mostrada en la Figura 7.29.

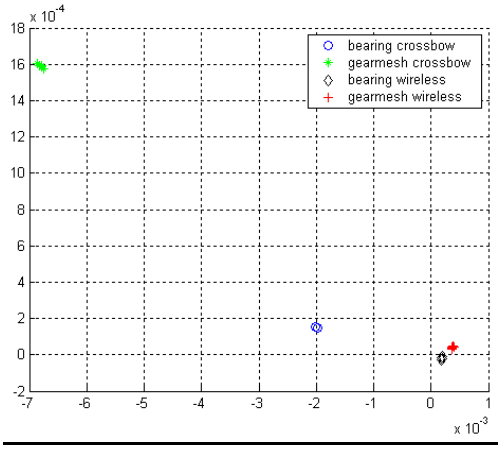


Figura 7.29. Análisis de Componentes Principales. Fallos Bearing y Gearmesh.

Se obtienen cuatro grupos diferentes de nubes de puntos. Concluyendo de nuevo que es posible la diferenciación de cada tipo de fallo con los datos recogidos por ambos sensores, *crossbow* y *wireless*, pero que, sin

embargo, no se obtiene la misma información a partir de los datos recogidos por cada tipo de sensor. El problema es la falta de información del procesamiento realizado en los datos al recogerlos con los sensores *crossbow*.

Se observa la misma discordancia al representar las *PSDs* tal y como se muestra en las Figuras 7.30, 7.31 y 7.32. En la Figura 7.30 se representan todas las *PSDs*, ambos fallos y ambos tipos de sensores. En las siguientes dos gráficas se separa la representación según los fallos para una mayor claridad.

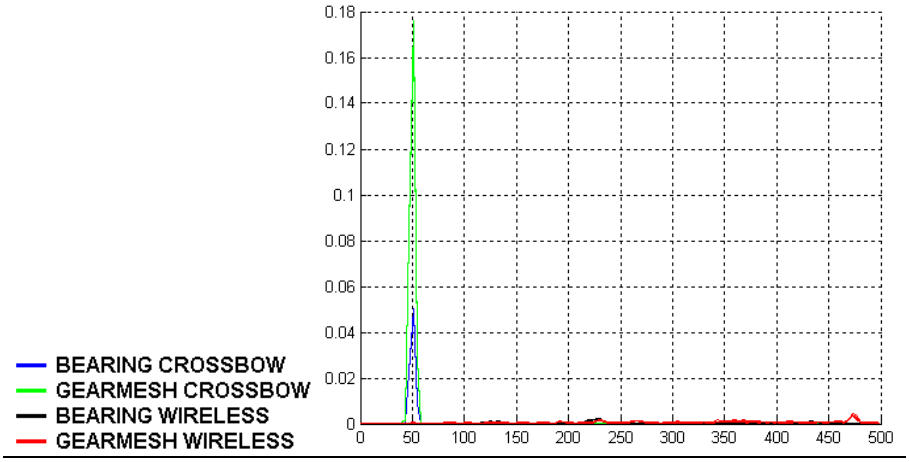


Figura 7.30. PSD - Fallos Bearing y Gearmesh.

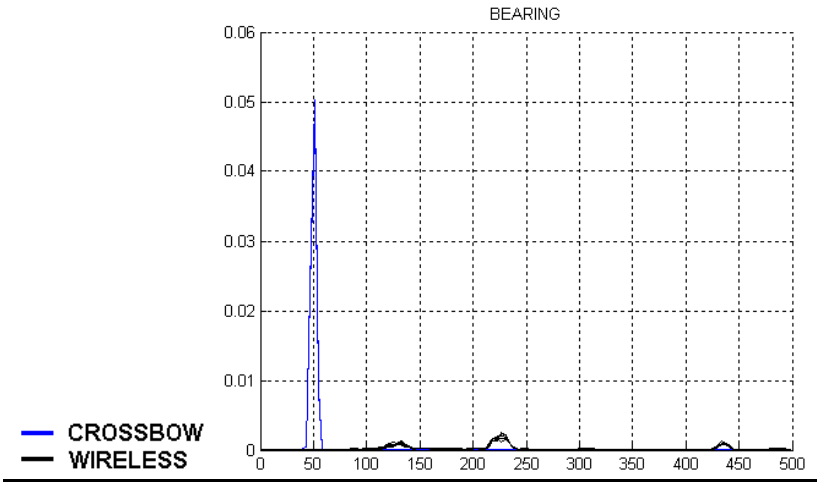


Figura 7.31. PSD - Fallo Bearing.

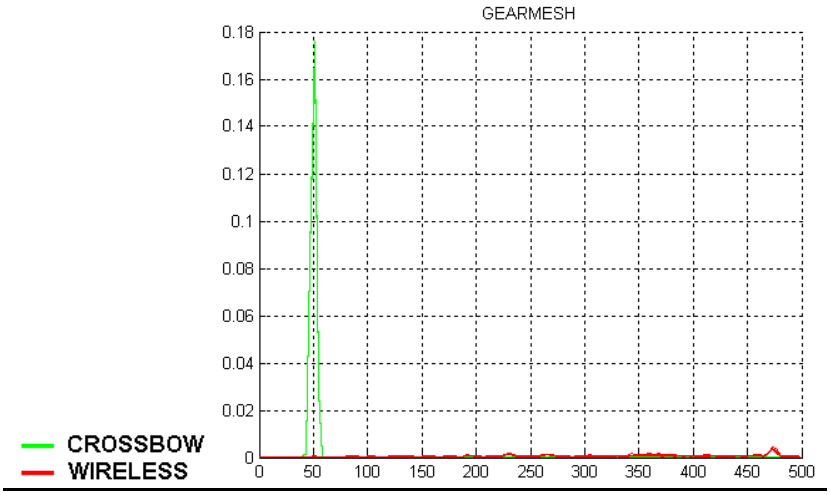


Figura 7.32. PSD - Fallo Gearmesh.

Capítulo 8

Conclusiones y Trabajo Futuro

8.1. Conclusiones

El objetivo del presente proyecto de final de carrera ha sido el desarrollo de una aplicación que permitiera al usuario, mediante una interfaz gráfica, el seguimiento del estado de las turbinas de gas de los aviones. Dicha aplicación informática tenía que ser capaz de recibir datos enviados por un sistema inalámbrico, descodificándolos, transformándolos, representándolos y almacenándolos para posteriores análisis necesarios.

Dicha aplicación se desarrolló con éxito, tratando de hacerla muy flexible y adaptable a posibles cambios venideros. Resulta importante la adaptabilidad porque en el punto de investigación en el que se encontraba el proyecto, había aún mucha información desconocida acerca del sistema inalámbrico. Muchos parámetros fueron deducidos por observación pero sin tener ninguna fuente de confirmación de los mismos.

La aplicación desarrollada se considera muy flexible y es una manera cómoda y sencilla de recoger datos y llevar el seguimiento en tiempo real del estado de las turbinas. Además, las ventajas del uso de un sistema inalámbrico hacen que las aplicaciones futuras en aviación sean casi seguras, pudiendo respaldar esta afirmación debido al interés mostrado en sus visitas al departamento por parte de *Rolls - Royce*. Entre dichas ventajas las más destacables son el ahorro en peso, la facilidad de instalación de los sensores, la rapidez de instalación y reemplazamiento y, por supuesto, el abaratamiento del sistema por ser un sistema desarrollado originalmente para automoción y adaptado a la aviación.

Debido al tiempo del que se disponía se pasó a probar el funcionamiento correcto de la aplicación en plataformas de prueba reales en laboratorio. Resultó una parte muy interesante del proyecto, pudiendo esbozar las

aplicaciones futuras de la aplicación desarrollada. Se recogieron datos mediante la instalación de los sensores en plataformas con distintos fallos procediendo posteriormente al análisis de los datos recogidos.

Los análisis realizados mostraron la posibilidad de detectar fallos y, mediante el procesamiento de los datos, la identificación de los mismos. Esta parte es muy interesante principalmente en los campos que se pretenden abordar en un futuro en el departamento:

- Comprobación del correcto funcionamiento de las turbinas al final del proceso de fabricación. De esta forma, cabría la posibilidad de poder reemplazar ciertas partes de la turbina antes de que un fallo más grave sea alcanzado. Este punto resulta muy interesante para los fabricantes porque haría posible el ahorro de grandes cantidades de dinero.

- Seguimiento de la salud de las turbinas de gas una vez en funcionamiento. Trabajando en el análisis de los datos recogidos se podrían detectar fallos en un temprano estado. De esta forma, se dispondría de más tiempo para reaccionar y más posibilidades de evitar accidentes. Se ha pensado incluso en la posibilidad de predecir fallos antes de que estos ocurran. Esto resultaría posible y fiable siempre y cuando se disponga de multitud de análisis previos.

- Reducción de los costes de mantenimiento mediante la habilidad de detección de fallos cuando estos se están desarrollando y de planificar las fechas de mantenimiento de una manera más eficiente.

8.2. Trabajo Futuro

Es obvio que el trabajo desarrollado, aunque exitoso, es muy amplio y tiene infinidad de aplicaciones tanto en aviación como en muchos otros campos de la industria. Por estos motivos el presente trabajo tiene muchos campos que se podrían modificar para distintas aplicaciones y deja mucho trabajo futuro a desarrollar.

Respecto a en qué se puede seguir trabajando, podríamos dividir el análisis en dos partes: la aplicación en *Visual Basic* desarrollada y las pruebas en laboratorio llevadas a cabo.

Desde el punto de vista de la aplicación en *Visual Basic*, ésta podría modificarse en los siguientes puntos:

- Reintentarse la exportación de los datos a *Excel* de una manera más eficiente. De esta manera, sería posible seguir recogiendo datos a la vez que se analizan los recogidos hasta el momento. Esto es posible en la aplicación desarrollada pero de una manera menos directa. Sería necesario hacer una copia de los archivos de texto mientras estos

siguen siendo modificados por la aplicación. Así podrían analizarse los datos en los archivos de texto copiados.

- Para contabilizar el tiempo en la aplicación se está usando el reloj del sistema. Esto puede resultar un problema cuando el sistema se encuentra muy ocupado ya que la aplicación se haría más lenta. Para solucionar esto, haría falta usar un sistema operativo de tiempo real.

- Debido a esa falta de información que se tiene del sistema inalámbrico sería necesario repasar en un futuro las escalas utilizadas en la transformación de los datos recibidos, ya que las usadas son una estimación deducida de las pruebas.

- Podría añadirse también una manera específica de hacerle saber al usuario cuando alguno de los sensores registrados ha dejado de enviar datos. En la aplicación desarrollada, cuando se dejan de recibir datos de un sensor, el usuario puede darse cuenta siempre y cuando se esté representando algunas de las variables de dicho sensor, ya que dicha gráfica se quedaría en blanco al no tener datos que representar.

Por otro lado, resaltando los puntos débiles de las pruebas en laboratorio desarrolladas podrían comentarse los siguientes puntos:

- Se trabajó con varias plataformas de prueba las cuales tenían algún fallo mecánico. Sería interesante disponer de una plataforma ideal, sin ningún tipo de fallo, con la que poder comparar los datos de vibración recogidos.

- Otro punto a retomar sería la comparación de los datos recogidos por los dos tipos de sensores utilizados, es decir, los sensores inalámbricos y los sensores *crossbow*. Para ello, sería necesario conocer más información de ambos tipos de sensores y ponerse en contacto con la persona que instaló y realizó pruebas con los sensores *crossbow* con anterioridad.

- En caso de disponer de mejores medios para hacerlo, sería deseable buscar una manera más eficiente de adherir los sensores inalámbricos a las plataformas de prueba. De esta manera, se aseguraría la estabilidad de los sensores y, por lo tanto, sería menos probable tener vibraciones parásitas añadidas.

- Otro problema con el que cuenta el *hardware* utilizado es la alimentación por baterías. Desde el punto de vista de las aplicaciones reales que el sistema puede tener, a todos los fabricantes les preocupa el hecho de tener que reemplazar las baterías. Es por eso que actualmente en el departamento se están llevando a cabo diversas investigaciones acerca de recogida de energía, o en inglés *energy harvesting*. Se trata de disponer de unos dispositivos que al vibrar generan energía y cargan la batería, evitando así la necesidad de reemplazarla. Hay mucha variedad de estos dispositivos.

- Para identificar los distintos tipos de fallos de manera automática podría implementarse un clasificador.

REFERENCIAS

- “Protocols and Architectures for Wireless Sensor Networks”, Holger Kart, Andreas Willing

- “Methods of Fault Diagnosis”, S. Leonhardt and M. Ayoubi

- “Considerations for monitoring aeroderivative gas turbines”, Machinery Messages

- “Prognosis of Faults in Gas Turbine Engines”, Tom Brotherton, Intelligent Automation Corporation

- “Recent Advancements in Aircraft Engine Health Management (EHM) Technologies and Recommendations for the Next Step”, Link C. Jaw, Scientific Monitoring, Inc.

- “A Survey of Intelligent Control and Health Managenment Technologies for Aircraft Propulsion Systems”, NASA

- “A Short Survey of Wireless Sensor Networks”, Holger Karl, Andreas Willing

- http://en.wikipedia.org/wiki/Main_Page

- http://www.emb.cl/electroindustria/articulo.mv?xid=302&rank=1

- http://w3.iec.csic.es/URSI/articulos_gandia_2005/articulos/SC4/563.pdf

APÉNDICES

Apéndice 1

APLICACIÓN EN VISUAL BASIC

Form - frmRead

```
'ASSUMPTIONS:

'Each sensor is going to send the information in 2 messages: the first one has
'LEN=7 and the second one has LEN=5. The CAN ID of the messages will be
'consecutive.
'First message:
   'Byte 0: header.
   'Bytes 1-4: sensor ID, although we are only looking at the byte 4.
   'Bytes 5-6: X or P data.
'Second message:
   'Bytes 0-1: Y or T data.
   'Bytes 2-3: Z or V data.
   'Byte 4: status, 0->XYZ or 1->PTV.

'Each receiver is reading the data of each sensor, so it is possible to know
'which sensor is sending data with the CAN ID.

'-----------------------------------------------------------------------------------------

'POSSIBLE MISTAKES:

'The file PCAN_USB.DLL should be in the same directory as the program.

'The program has to be modified at two points with the route where the user
'wants to store the text files. Those two points are into two different Sub
'procedure which are Private Sub cmdExit_Click() and Private Sub
tmrStore_Timer().

'It may be necesssary to check the baud rate in the fuction CAN_Ini in the Sub
'procedure called Private Sub cmdCAN_Init_Click(). It changes depending on
'the sensors.

'-----------------------------------------------------------------------------------------

'DESCRIPTIONS OF THE PUBLIC VARIABLES:

'Variable which stores how many sensors are sending data.
Public n_sensors As Integer

'This variable is used to count how much time has passed since the application
'started, it is increased each 10 msec.
Public tenmsec_aux As Double
```

```
'These variables are also used to count how much time has passed since the
'application started, but broking down the information into small parts.
Public tenmsec As Integer
Public sec As Integer
Public tensec As Integer
Public minute As Integer
Public tenmin As Integer

'These variables are used to relate the time with the beginning of PTV graphs.
Public tensec_aux As Integer
Public flag_aux As Integer

'Auxiliary variable to plot the data: they are necessary because the variables
'which count the time cannot restart when the graphs reach the end, so the time
'variables carry on increasing and these variables restart with the graphs.
Public aux_XYZ As Integer
Public aux_PTV As Integer

'In the program an array of structs is used to store all the information about a
'sensor in each struct. "Beginning" is used to redimension that array at the
'beginning of the program but just once.
Public beginning As Integer

'-------------------------------------------------------------------------------------

Private Sub cboSensor_Change(Index As Integer)
'The user cannot write the number of the sensor which he wants it to be plotted,
'the user has to choose the number in the list.
cboSensor(Index).Text = Format(graph_sensor(Index))
End Sub

'-------------------------------------------------------------------------------------

Private Sub cboSensor_Click(Index As Integer)
'When the user chooses the number of the sensor in the list, the plotted data
'has to change, so in the graph its scale, its labels and its axis have to change
'as well.
graph_sensor(Index) = cboSensor(Index).Text
pctGraph(Index).Cls

'The scale is different depending on the variable which has to be plotted.
Select Case graph_variable(Index)
Case 0
min(Index) = scale_minx
max(Index) = scale_maxx
Case 1
min(Index) = scale_miny
max(Index) = scale_maxy
Case 2
```

```
min(Index) = scale_minz
max(Index) = scale_maxz
Case 3
min(Index) = scale_minp
max(Index) = scale_maxp
Case 4
min(Index) = scale_mint
max(Index) = scale_maxt
Case 5
min(Index) = scale_minv
max(Index) = scale_maxv
End Select
lblMin(Index).Caption = min(Index)
lblMax(Index).Caption = max(Index)
sldZoom(Index).value = 1

'Plot x and y axis
Select Case graph_variable(Index)
Case 0, 1, 2
pctGraph(Index).Scale (-1, max(Index))-(scalet_XYZ, min(Index))
'Vibration graphs x,y,z
pctGraph(Index).Line (-1, 0)-(scalet_XYZ, 0), vbRed        'x axis
pctGraph(Index).Line (0, min(Index))-(0, max(Index)), vbRed 'y axis
pctGraph(Index).PSet (0, 0), vbRed                         'origin
Case 3, 4, 5
pctGraph(Index).Scale (-1, max(Index))-(scalet_PTV, min(Index))
'Pressure, temperature and voltage graphs p,t,v
pctGraph(Index).Line (-1, 0)-(scalet_PTV, 0), vbRed        'x axis
pctGraph(Index).Line (0, min(Index))-(0, max(Index)), vbRed 'y axis
pctGraph(Index).PSet (0, 0), vbRed                         'origin
End Select
End Sub

'-----------------------------------------------------------------------------------

Private Sub cboSensor_KeyPress(Index As Integer, KeyAscii As Integer)
'The user cannot write the number of the sensor which he wants it to be plotted,
'the user has to choose the number in the list.
result = MsgBox("Please select a sensor in the list", 64, "Invalid value")
End Sub

'-----------------------------------------------------------------------------------

Private Sub cboVariable_Change(Index As Integer)
'The user cannot write the name of the variable which he wants it to be plotted,
'the user has to choose the name in the list.
Select Case graph_variable(Index)
Case 0
cboVariable(Index).Text = "X"
Case 1
```

```
cboVariable(Index).Text = "Y"
Case 2
cboVariable(Index).Text = "Z"
Case 3
cboVariable(Index).Text = "P"
Case 4
cboVariable(Index).Text = "T"
Case 5
cboVariable(Index).Text = "V"
End Select
End Sub

'--------------------------------------------------------------------------------

Private Sub cboVariable_Click(Index As Integer)
'When the user chooses the name of the variable in the list, the plotted data has
'to change, so in the graph its scale, its labels and its axis have to change as
'well.
For k = 0 To 5
   If cboVariable(Index).Text = cboVariable(Index).List(k) Then
      graph_variable(Index) = k
      pctGraph(Index).Cls

      'The scale is different depending on the variable which has to be plotted.
      Select Case graph_variable(Index)
      Case 0
      min(Index) = scale_minx
      max(Index) = scale_maxx
      lblvble(Index).Caption = "X"
      Case 1
      min(Index) = scale_miny
      max(Index) = scale_maxy
      lblvble(Index).Caption = "Y"
      Case 2
      min(Index) = scale_minz
      max(Index) = scale_maxz
      lblvble(Index).Caption = "Z"
      Case 3
      min(Index) = scale_minp
      max(Index) = scale_maxp
      lblvble(Index).Caption = "P"
      Case 4
      min(Index) = scale_mint
      max(Index) = scale_maxt
      lblvble(Index).Caption = "T"
      Case 5
      min(Index) = scale_minv
      max(Index) = scale_maxv
      lblvble(Index).Caption = "V"
      End Select
```

```
        lblMin(Index).Caption = min(Index)
        lblMax(Index).Caption = max(Index)
        sldZoom(Index).value = 1

        'Plot x and y axis
        Select Case graph_variable(Index)
        Case 0, 1, 2
        pctGraph(Index).Scale (-1, max(Index))-(scalet_XYZ, min(Index))
        'Vibration graphs x,y,z
        pctGraph(Index).Line (-1, 0)-(scalet_XYZ, 0), vbRed        'x axis
        pctGraph(Index).Line (0, min(Index))-(0, max(Index)), vbRed 'y axis
        pctGraph(Index).PSet (0, 0), vbRed                         'origin
        Case 3, 4, 5
        pctGraph(Index).Scale (-1, max(Index))-(scalet_PTV, min(Index))
        'Pressure, temperature and voltage graphs p,t,v
        pctGraph(Index).Line (-1, 0)-(scalet_PTV, 0), vbRed        'x axis
        pctGraph(Index).Line (0, min(Index))-(0, max(Index)), vbRed 'y axis
        pctGraph(Index).PSet (0, 0), vbRed                         'origin
        End Select
    End If
Next k
End Sub

'--------------------------------------------------------------------------------

Private Sub cboVariable_KeyPress(Index As Integer, KeyAscii As Integer)
'The user cannot write the name of the variable which he wants it to be plotted,
'the user has to choose the name in the list.
result = MsgBox("Please select a variable in the list", 64, "Invalid value")
End Sub

'--------------------------------------------------------------------------------

Private Sub cmdCAN_Close_Click()
'Disconnect from CAN-Hardware.
ret = CAN_Close()
'Display the return code.
lblCAN_Close.Caption = ret
End Sub

'--------------------------------------------------------------------------------

Private Sub cmdCAN_Init_Click()

'Activate de hardware and reserve a send and receive buffer:
'First parameter = register for the baud rate.
'Second parameter = standard or extended message.
'In this case: 500 kBit/s and standard.
ret = CAN_Init(CAN_BAUD_500K, CAN_INIT_TYPE_ST)
'Display the return code.
```

```
lblCAN_Init.Caption = ret

'Plot x and y axis
For Index = 0 To 5
Select Case graph_variable(Index)
Case 0, 1, 2
'Vibration graphs x,y,z
pctGraph(Index).Line (-1, 0)-(scalet_XYZ, 0), vbRed        'x axis
pctGraph(Index).Line (0, min(Index))-(0, max(Index)), vbRed 'y axis
pctGraph(Index).PSet (0, 0), vbRed                         'origin
Case 3, 4, 5
'Pressure, temperature and voltage graphs p,t,v
pctGraph(Index).Line (-1, 0)-(scalet_PTV, 0), vbRed        'x axis
pctGraph(Index).Line (0, min(Index))-(0, max(Index)), vbRed 'y axis
pctGraph(Index).PSet (0, 0), vbRed                         'origin
End Select
Next Index

End Sub

'--------------------------------------------------------------------------------

Private Sub cmdExit_Click()
'Before finishing the program, all the general information is stored in a text file.
'For each sensor: the sensor ID, its associated CAN ID, its associated text files
'and the total number of received messages.
For i = 1 To n_sensors

   Open _
   "C:\Documents and Settings\uos\Desktop\programs\Visual Basic\last
one\General Information.txt" _
   For Append As #i
   Print #i, "Sensor:", sensors_vble(i - 1).iden
   Print #i, "CAN ID:", sensors_vble(i - 1).IDcan, sensors_vble(i - 1).IDcan + 1,
Tab(14)
   Print #i, "Text files associated: XYZ_" & CStr(i) & ".txt and PTV_" & CStr(i) &
_
   ".txt", Tab(14)
   Print #i, "Messages", Tab(14)
   Print #i, "XYZ:", sensors_vble(i - 1).MsgCounter_V
   Print #i, "PTV:", sensors_vble(i - 1).MsgCounter_T, Tab(14)
   Print #i, " ", Tab(14)
   Print #i, " ", Tab(14)
   Close #i

   'If there is some no stored information, it is stored before finishing.
   Open _
   "C:\Documents and Settings\uos\Desktop\programs\Visual Basic\last
one\XYZ_" & CStr(i) & ".txt" _
   For Append As #i
```

```
   If sensors_vble(i - 1).MsgCounter_V_aux <> 0 Then
      For j = 1 To sensors_vble(i - 1).MsgCounter_V_aux - 1
      Print #i, sensors_vble(i - 1).MatrixV(3, j - 1), sensors_vble(i - 1).MatrixV(0,
_
      j - 1), sensors_vble(i - 1).MatrixV(1, j - 1), sensors_vble(i - 1).MatrixV(2, j -
1)
      Next j
      Print #i, sensors_vble(i - 1).MatrixV(3, sensors_vble(i -
1).MsgCounter_V_aux _
      - 1), sensors_vble(i - 1).MatrixV(0, sensors_vble(i - 1).MsgCounter_V_aux
- 1), _
      sensors_vble(i - 1).MatrixV(1, sensors_vble(i - 1).MsgCounter_V_aux - 1),
_
      sensors_vble(i - 1).MatrixV(2, sensors_vble(i - 1).MsgCounter_V_aux - 1)
   End If
   Close #i

   'If there is some no stored information, it is stored before finishing.
   Open _
   "C:\Documents and Settings\uos\Desktop\programs\Visual Basic\last
one\PTV_" & CStr(i) & ".txt" _
   For Append As #i + 1

   If sensors_vble(i - 1).MsgCounter_T_aux <> 0 Then
      For j = 1 To sensors_vble(i - 1).MsgCounter_T_aux - 1
      Print #i + 1, sensors_vble(i - 1).MatrixT(3, j - 1), _
      sensors_vble(i - 1).MatrixT(0, j - 1), sensors_vble(i - 1).MatrixT(1, j - 1), _
      sensors_vble(i - 1).MatrixT(2, j - 1)
      Next j
      Print #i + 1, sensors_vble(i - 1).MatrixT(3, sensors_vble(i -
1).MsgCounter_T_aux _
      - 1), sensors_vble(i - 1).MatrixT(0, sensors_vble(i - 1).MsgCounter_T_aux
- 1), _
      sensors_vble(i - 1).MatrixT(1, sensors_vble(i - 1).MsgCounter_T_aux - 1),
_
      sensors_vble(i - 1).MatrixT(2, sensors_vble(i - 1).MsgCounter_T_aux - 1)
   End If
   Close #i + 1

Next i

'To finish the program.
End
End Sub

'--------------------------------------------------------------------------------

Private Sub Form_Load()
'The application starts asking the number of sensors.
```

```
n_sensors = InputBox("How many sensors?", "Welcome", "1")

'At the beginning the variables for the graph scales, the scales and the labels
'have the default value.
graph_variable(0) = 0
graph_variable(1) = 1
graph_variable(2) = 2
graph_variable(3) = 3
graph_variable(4) = 4
graph_variable(5) = 5

min(0) = scale_minx
lblMin(0).Caption = min(0)
max(0) = scale_maxx
lblMax(0).Caption = max(0)

min(1) = scale_miny
lblMin(1).Caption = min(1)
max(1) = scale_maxy
lblMax(1).Caption = max(1)

min(2) = scale_minz
lblMin(2).Caption = min(2)
max(2) = scale_maxz
lblMax(2).Caption = max(2)

min(3) = scale_minp
lblMin(3).Caption = min(3)
max(3) = scale_maxp
lblMax(3).Caption = max(3)

min(4) = scale_mint
lblMin(4).Caption = min(4)
max(4) = scale_maxt
lblMax(4).Caption = max(4)

min(5) = scale_minv
lblMin(5).Caption = min(5)
max(5) = scale_maxv
lblMax(5).Caption = max(5)

'Graphs scale, first the top left corner and then the bottom right one.
For i = 0 To 2
pctGraph(i).Scale (-1, max(i))-(scalet_XYZ, min(i))
Next i

For i = 3 To 5
pctGraph(i).Scale (-1, max(i))-(scalet_PTV, min(i))
Next i
```

```
End Sub

'-------------------------------------------------------------------------------------

Private Sub sldZoom_Change(Index As Integer)
'The graph scales are changed depending on the zoom. The change is different
'according to the variable.
Select Case graph_variable(Index)
Case 0
min(Index) = scale_minx / sldZoom(Index).value
max(Index) = scale_maxx / sldZoom(Index).value
Case 1
min(Index) = scale_miny / sldZoom(Index).value
max(Index) = scale_maxy / sldZoom(Index).value
Case 2
max(Index) = scale_maxz / sldZoom(Index).value
Case 3
max(Index) = scale_maxp / sldZoom(Index).value
Case 4
max(Index) = scale_maxt / sldZoom(Index).value
Case 5
max(Index) = scale_maxv / sldZoom(Index).value
End Select

'The labels and the scale have to change as well.
lblMin(Index).Caption = min(Index)
lblMax(Index).Caption = max(Index)

pctGraph(Index).Cls

Select Case graph_variable(Index)
Case 0, 1, 2
pctGraph(Index).Scale (-1, max(Index))-(scalet_XYZ, min(Index))
pctGraph(Index).Line (-1, 0)-(scalet_XYZ, 0), vbRed 'x axis
Case 3, 4, 5
pctGraph(Index).Scale (-1, max(Index))-(scalet_PTV, min(Index))
pctGraph(Index).Line (-1, 0)-(scalet_PTV, 0), vbRed 'x axis
End Select

pctGraph(Index).Line (0, min(j))-(0, max(j)), vbRed 'y axis
pctGraph(Index).PSet (0, 0), vbRed                  'origin

End Sub

'-------------------------------------------------------------------------------------

Private Sub tmrTime_Timer()
'This variable is increased each 10 ms and then it is displayed.
tenmsec_aux = tenmsec_aux + 1
txtTime.Text = tenmsec_aux
```

```
'This variable is increased each 10 ms.
tenmsec = tenmsec + 1
'When tenmsec=100, one second has passed.
If tenmsec = 100 Then
   tenmsec = 0
   sec = sec + 1
   'When sec=10, ten seconds have passed.
   If sec = 10 Then
      sec = 0
      tensec = tensec + 1
      'Auxiliary variable to relate the time with the beginning of PTV graphs.
      tensec_aux = tensec_aux + 1
      'When tensec=6, one minute has passed.
      If tensec = 6 Then
         tensec = 0
         minute = minute + 1
         'When min=10, ten minutes have passed.
         If minute = 10 Then
            minute = 0
            tenmin = tenmin + 1
            'When tenmin=6, one hour has passed and the clock begins again.
            If tenmin = 6 Then
               tenmin = 0
            End If
         End If
      End If
      'When tensec_aux=10, it is indicated by flag_aux=1 to delete the
'information plotted in the PTV graphs and rebegin ploting from the origin.
      If tensec_aux = 10 Then
         tensec_aux = 0
         flag_aux = 1
      End If
   End If
End If
'The time variables are displayed.
lblTenmsec.Caption = tenmsec
lblSec.Caption = sec
lblTensec.Caption = tensec
lblMinute.Caption = minute
lblTenmin.Caption = tenmin


'Auxiliary variable to plot the data.
aux_XYZ = aux_XYZ + 1
aux_PTV = aux_PTV + 1
'The graphs have a limit so the information is deleted and ploting rebegins.
If tenmsec = 0 Then

   aux_XYZ = 0
```

```
        For j = 0 To 5
        If graph_variable(j) = 0 Or graph_variable(j) = 1 Or graph_variable(j) = 2
Then
        pctGraph(j).Cls

        pctGraph(j).Line (-1, 0)-(scalet_XYZ, 0), vbRed 'x axis
        pctGraph(j).Line (0, min(j))-(0, max(j)), vbRed 'y axis
        pctGraph(j).PSet (0, 0), vbRed                  'origin
        End If
        Next j

End If

If flag_aux = 1 Then

   aux_PTV = 0

        For j = 0 To 5
        If graph_variable(j) = 3 Or graph_variable(j) = 4 Or graph_variable(j) = 5
Then
        pctGraph(j).Cls

        pctGraph(j).Line (-1, 0)-(scalet_PTV, 0), vbRed 'x axis
        pctGraph(j).Line (0, min(j))-(0, max(j)), vbRed 'y axis
        pctGraph(j).PSet (0, 0), vbRed                  'origin
        End If
        Next j

   flag_aux = 0

End If

End Sub

'--------------------------------------------------------------------------------

Private Sub tmrRead_Timer()

 'Struct to store the received information.
 Dim myMsg As TPCANMsg

 'General counters:
 Static Counter As Double        'It counts received messages.
 Static MsgCounter_V As Double   'It counts received messages of
                                 'vibration(XYZ).
 Static MsgCounter_T As Integer  'It counts received messages of pressure,
                                 'temperature and voltage(PTV).

 'Variables to count the missed messages:
```

```
Static MsgCounter_7 As Integer       'with LEN=7.
Static MsgCounter_5 As Integer       'with LEN=5.
Static MsgCounter_strange As Integer 'with another LEN.

'When the application starts a registering process is necessary:
'This variable indicates if all the sensors have been registered.
Static registered As Integer
'This variable counts how many sensors have been registered.
Static count_registered As Integer
'This variable indicates if the sensor which is treated at the moment has already
'been registered.
Static sensors_registered As Integer

'Array of structs: to store all the information about a sensor in each struct.
If beginning = 0 Then
ReDim Preserve sensors_vble(n_sensors - 1)
beginning = 1
End If

'Get the next message or error out of the receive queue of the device driver.
ret = CAN_Read(myMsg)

'The return value from CAN_Read() is CAN_ERR_OK when the message
'contents a normal message.
While (ret = CAN_ERR_OK)
   'It is known which sensor is sending the data with the CAN ID.
   which_sensor = myMsg.ID
   'Registering process:
   If registered = 0 And myMsg.LEN = 7 Then
   For i = 1 To n_sensors
      If sensors_vble(i - 1).IDcan = which_sensor Then
         sensors_registered = 1 'This sensor has already been registered.
      End If
   Next i
   For i = 1 To n_sensors
      'If the sensor has not been registered yet, an empty place is looked for.
      If sensors_registered = 0 Then
         If sensors_vble(i - 1).firstMsg = 0 Then
            'In the empty place, CAN ID and sensor identifier are stored, and it is
            'indicated that the place is occupied by firstMsg=1.
            sensors_vble(i - 1).IDcan = which_sensor
            sensors_vble(i - 1).iden = myMsg.DATA(4)

            'The number of the sensor is added to the list of the sensors at each
           'graph.
            d = Format(myMsg.DATA(4))
            For m = 0 To 5
            cboSensor(m).AddItem (d), i - 1
            Next m
```

```
            sensors_vble(i - 1).firstMsg = 1

            result = MsgBox("Sensor registered", 64, "Registering Process")
            sensors_registered = 1
            count_registered = count_registered + 1
            'At the beginning all the variables of the first sensor registered are
            'plotted.
            If count_registered = 1 Then
               For k = 0 To 5
               graph_sensor(k) = sensors_vble(i - 1).iden
               cboSensor(k).Text = d
               Next k
            End If

            'The registering process will have been finished when all the sensors
           'will have been registered.
            If count_registered = n_sensors Then
            registered = 1
            result = MsgBox("Registering process finished", 0 + 64, "Registering
Process")
            End If
         End If
      End If
   Next i
   sensors_registered = 0
   End If



   For k = 1 To n_sensors
   'It is known which sensor is sending the data with the CAN ID.
   If which_sensor = sensors_vble(k - 1).IDcan Or which_sensor = _
   sensors_vble(k - 1).IDcan + 1 Then
   'The sensors send a message with LEN=7 and then another one with LEN=5.
   'If the message with LEN=7 is received, it is indicated by flag1=1 and the
   'information is stored.
   If myMsg.LEN = 7 Then
      sensors_vble(k - 1).flag1 = 1
      sensors_vble(k - 1).vector(0) = myMsg.DATA(5)
      sensors_vble(k - 1).vector(1) = myMsg.DATA(6)

      'buffer1 is used to write the message with LEN=7 in a list.
      sensors_vble(k - 1).buffer1 = "ID: " + Format(myMsg.ID, "###000 ") +
"LEN: " _
      + Format(myMsg.LEN, "#0 ")
      For i = 0 To myMsg.LEN - 1
         sensors_vble(k - 1).buffer1 = sensors_vble(k - 1).buffer1 + " " + _
         Format(myMsg.DATA(i), "###000")
      Next i
   Else
```

```
    'The message after a message with LEN=7 should have LEN=5. If it does not
'happen the message with LEN=7 is missed so the missed counter of message
'with LEN=7 is increased and displayed, and it is indicated that there isn't any
'message with LEN=7 stored by flag1=0.
        If myMsg.LEN <> 5 And sensors_vble(k - 1).flag1 = 1 Then
        sensors_vble(k - 1).MsgCounter_7 = sensors_vble(k - 1).MsgCounter_7 +
1
        MsgCounter_7 = MsgCounter_7 + 1
        lblCounterMsg_7.Caption = MsgCounter_7
        sensors_vble(k - 1).flag1 = 0
        End If
    End If

    'If the message LEN is not 7 and neither 5 the strange message counter is
'increased and displayed.
    If myMsg.LEN <> 7 And myMsg.LEN <> 5 Then
    sensors_vble(k - 1).MsgCounter_strange = sensors_vble(k -
1).MsgCounter_strange + 1
    MsgCounter_strange = MsgCounter_strange + 1
    lblCounterMsg_Strange.Caption = MsgCounter_strange
    End If

    'It is checked if the message with LEN=5 is received and if a message with
'LEN=7 has already been received.
    If myMsg.LEN = 5 Then
    If sensors_vble(k - 1).flag1 = 1 Then

    'The last byte of the message with LEN=5 is checked to find out the type of
    'the complete message.
    sensors_vble(k - 1).n = myMsg.DATA(myMsg.LEN - 1)
    sensors_vble(k - 1).n = sensors_vble(k - 1).n And 128
    If sensors_vble(k - 1).n = 128 Then
        sensors_vble(k - 1).n = 1
    End If

    'Counter is increased and displayed.
    sensors_vble(k - 1).Counter = sensors_vble(k - 1).Counter + 1
    Counter = Counter + 1
    lblCounterMsg.Caption = Counter

    'To know if a point or a line has to be plotted, the type of the last message
'has to be known. N_anterior is used with this objective, storing this information.
    'The problem is with the first message, because no previous message exists.
    If sensors_vble(k - 1).Counter = 1 Then
    sensors_vble(k - 1).n_anterior = sensors_vble(k - 1).n
    End If

    'The data is stored.
    sensors_vble(k - 1).vector(2) = myMsg.DATA(0)
    sensors_vble(k - 1).vector(3) = myMsg.DATA(1)
```

```
    sensors_vble(k - 1).vector(4) = myMsg.DATA(2)
    sensors_vble(k - 1).vector(5) = myMsg.DATA(3)


   'buffer2 is used to write the message with LEN=5 in a list.
    sensors_vble(k - 1).buffer2 = "ID: " + Format(myMsg.ID, "###000 ") + "LEN: "
_
    + Format(myMsg.LEN, "#0 ")
    For i = 0 To myMsg.LEN - 1
        sensors_vble(k - 1).buffer2 = sensors_vble(k - 1).buffer2 + " " + _
        Format(myMsg.DATA(i), "###000")
    Next

    'The message type is checked to store and plot the data in a different place.
    'The transformation of the data will be also different.
    'If n=0 it is XYZ data.
    If sensors_vble(k - 1).n = 0 Then
        'XYZ data arrive stored in 2 bytes
        sensors_vble(k - 1).value(0) = sensors_vble(k - 1).vector(0) * 256 + _
        sensors_vble(k - 1).vector(1)
        sensors_vble(k - 1).value(1) = sensors_vble(k - 1).vector(2) * 256 + _
        sensors_vble(k - 1).vector(3)
        sensors_vble(k - 1).value(2) = sensors_vble(k - 1).vector(4) * 256 + _
        sensors_vble(k - 1).vector(5)
        'X data transformation
        sensors_vble(k - 1).value(0) = (350 / 1024) * _
        (sensors_vble(k - 1).value(0) - 1024)
        'Y data transformation
        sensors_vble(k - 1).value(1) = (350 / 1024) * _
        (sensors_vble(k - 1).value(1) - 1024)
        'Z data transformation
        sensors_vble(k - 1).value(2) = 10 * sensors_vble(k - 1).value(2)

        'The received messages XYZ counter is increased and displayed.
        sensors_vble(k - 1).MsgCounter_V = sensors_vble(k - 1).MsgCounter_V +
1
        sensors_vble(k - 1).MsgCounter_V_aux = sensors_vble(k -
1).MsgCounter_V_aux + 1
        MsgCounter_V = MsgCounter_V + 1
        lblCounterMsg_V.Caption = MsgCounter_V

        'The complete information is written on the XYZ list.
        lstMsg_V.AddItem (sensors_vble(k - 1).buffer1)
        lstMsg_V.AddItem (sensors_vble(k - 1).buffer2)

        'The information is stored in the XYZ array, redimensioning it each time.
        'The array has a fixed number of rows: x,y,z,time.
        'Each time a column is added by using the XYZ received messages
        'counter (taking into account that VS begins to count from 0).
        ReDim Preserve sensors_vble(k - 1).MatrixV(3, _
```

```
        sensors_vble(k - 1).MsgCounter_V_aux - 1)
        For i = 0 To 2
        sensors_vble(k - 1).MatrixV(i, sensors_vble(k - 1).MsgCounter_V_aux - 1)
= _
        sensors_vble(k - 1).value(i)
        Next
        sensors_vble(k - 1).MatrixV(3, sensors_vble(k - 1).MsgCounter_V_aux - 1)
= _
        tenmsec_aux

        'To plot the data:
        'Look at each graph to know if the data have to be plotted.
        For h = 1 To 6
        'First look at the sensor identifier.
        If graph_sensor(h - 1) = sensors_vble(k - 1).iden Then
        'Second look at the variable type.
        Select Case graph_variable(h - 1)
        Case 0
        If sensors_vble(k - 1).n_anterior = sensors_vble(k - 1).n Then
           'If the last message was a XYZ message a line is plotted.
           pctGraph(h - 1).Line -(aux_XYZ, sensors_vble(k - 1).value(0))
        Else
           'If the last message was a XYZ message a point is plotted.
           pctGraph(h - 1).PSet (aux_XYZ, sensors_vble(k - 1).value(0))
        End If
        'Plots a vertical line to know the exact received data.
        pctGraph(h - 1).Line (aux_XYZ, 0)-(aux_XYZ, sensors_vble(k -
1).value(0)), vbRed
        Case 1
        If sensors_vble(k - 1).n_anterior = sensors_vble(k - 1).n Then
           'If the last message was a XYZ message a line is plotted.
           pctGraph(h - 1).Line -(aux_XYZ, sensors_vble(k - 1).value(1))
        Else
           'If the last message was a XYZ message a point is plotted.
           pctGraph(h - 1).PSet (aux_XYZ, sensors_vble(k - 1).value(1))
        End If
        'Plots a vertical line to know the exact received data.
        pctGraph(h - 1).Line (aux_XYZ, 0)-(aux_XYZ, sensors_vble(k -
1).value(1)), vbRed
        Case 2
        If sensors_vble(k - 1).n_anterior = sensors_vble(k - 1).n Then
           'If the last message was a XYZ message a line is plotted.
           pctGraph(h - 1).Line -(aux_XYZ, sensors_vble(k - 1).value(2))
        Else
           'If the last message was a XYZ message a point is plotted.
           pctGraph(h - 1).PSet (aux_XYZ, sensors_vble(k - 1).value(2))
        End If
        'Plots a vertical line to know the exact received data.
        pctGraph(h - 1).Line (aux_XYZ, 0)-(aux_XYZ, sensors_vble(k -
1).value(2)), vbRed
```

```
        End Select
        End If
        Next h
    Else
        'If n=1 it is PTV data.
        If sensors_vble(k - 1).n = 1 Then
            'PTV data arrive stored in 1 byte
            sensors_vble(k - 1).value(0) = sensors_vble(k - 1).vector(1)
            sensors_vble(k - 1).value(1) = sensors_vble(k - 1).vector(3)
            sensors_vble(k - 1).value(2) = sensors_vble(k - 1).vector(5)
            'P data transformation
            sensors_vble(k - 1).value(0) = (500 / 255) * sensors_vble(k - 1).value(0)
            'T data transformation
            sensors_vble(k - 1).value(1) = (80 / 255) * sensors_vble(k - 1).value(1)
            'V data transformation
            sensors_vble(k - 1).value(2) = (3 / 255) * sensors_vble(k - 1).value(2)

            'The PTV received messages counter is increased and displayed.
            sensors_vble(k - 1).MsgCounter_T = sensors_vble(k - 1).MsgCounter_T
+ 1
            sensors_vble(k - 1).MsgCounter_T_aux = sensors_vble(k -
1).MsgCounter_T_aux + 1
            MsgCounter_T = MsgCounter_T + 1
            lblCounterMsg_T.Caption = MsgCounter_T

            'The complete information is written on the PTV list.
            lstMsg_T.AddItem (sensors_vble(k - 1).buffer1)
            lstMsg_T.AddItem (sensors_vble(k - 1).buffer2)

            'The information is stored in the PTV array, redimensioning it each time.
            'The array has a fixed number of rows: p,t,v,time.
            'Each time a column is added by using the PTV received messages
            'counter (taking into account that VS begins to count from 0).
            ReDim Preserve sensors_vble(k - 1).MatrixT(3, _
            sensors_vble(k - 1).MsgCounter_T_aux - 1)
            For i = 0 To 2
            sensors_vble(k - 1).MatrixT(i, sensors_vble(k - 1).MsgCounter_T_aux -
1) = _
            sensors_vble(k - 1).value(i)
            Next
            sensors_vble(k - 1).MatrixT(3, sensors_vble(k - 1).MsgCounter_T_aux -
1) = _
            tenmsec_aux

            'To plot the data:
            'Look at each graph to know if the data have to be plotted.
            For h = 1 To 6
            'First look at the sensor identifier.
            If graph_sensor(h - 1) = sensors_vble(k - 1).iden Then
            'Second look at the variable type.
```

```
        Select Case graph_variable(h - 1)
        Case 3
        If sensors_vble(k - 1).n_anterior = sensors_vble(k - 1).n Then
           'If the last message was a PTV message a line is plotted.
           pctGraph(h - 1).Line -(aux_PTV, sensors_vble(k - 1).value(0))
        Else
           'If the last message was a PTV message a point is plotted.
           pctGraph(h - 1).PSet (aux_PTV, sensors_vble(k - 1).value(0))
        End If
        'Plots a vertical line to know the exact received data.
        pctGraph(h - 1).Line (aux_PTV, 0)-(aux_PTV, sensors_vble(k -
1).value(0)), vbRed
        Case 4
        If sensors_vble(k - 1).n_anterior = sensors_vble(k - 1).n Then
           'If the last message was a PTV message a line is plotted.
           pctGraph(h - 1).Line -(aux_PTV, sensors_vble(k - 1).value(1))
        Else
           'If the last message was a PTV message a point is plotted.
           pctGraph(h - 1).PSet (aux_PTV, sensors_vble(k - 1).value(1))
        End If
        'Plots a vertical line to know the exact received data.
        pctGraph(h - 1).Line (aux_PTV, 0)-(aux_PTV, sensors_vble(k -
1).value(1)), vbRed
        Case 5
        If sensors_vble(k - 1).n_anterior = sensors_vble(k - 1).n Then
           'If the last message was a PTV message a line is plotted.
           pctGraph(h - 1).Line -(aux_PTV, sensors_vble(k - 1).value(2))
        Else
           'If the last message was a PTV message a point is plotted.
           pctGraph(h - 1).PSet (aux_PTV, sensors_vble(k - 1).value(2))
        End If
        'Plots a vertical line to know the exact received data.
        pctGraph(h - 1).Line (aux_PTV, 0)-(aux_PTV, sensors_vble(k -
1).value(2)), vbRed
        End Select
        End If
        Next h
     End If
  End If

  'Before finishing, the current message type is stored to use it in the next
'message.
  sensors_vble(k - 1).n_anterior = sensors_vble(k - 1).n
  'In addition, it is indicated that a message with LEN=7 is not stored.
  sensors_vble(k - 1).flag1 = 0

  Else

  'If flag1=0 when the message with LEN=5 is received, it is missed so the
'missed counter of message with LEN=5 is increased and displayed.
```

```
    sensors_vble(k - 1).MsgCounter_5 = sensors_vble(k - 1).MsgCounter_5 + 1
    MsgCounter_5 = MsgCounter_5 + 1
    lblCounterMsg_5.Caption = MsgCounter_5
    End If
    End If

    End If
    Next k

    ret = CAN_Read(myMsg)

  Wend

 'If the program arrives to this line a failure in CAN_Read has been made so the
 'return code is displayed.
 lblCAN_Read.Caption = ret

End Sub

'-------------------------------------------------------------------------------

Private Sub tmrStore_Timer()
'The information is stored each 30 seconds in a text file, afterwards the arrays
'which stored this information can be reused. In this way the memory is saved
'and the speed is increased.
For i = 1 To n_sensors

    Open _
    "C:\Documents and Settings\uos\Desktop\programs\Visual Basic\last
one\XYZ_" & CStr(i) & ".txt" _
    For Append As #i

    For j = 1 To sensors_vble(i - 1).MsgCounter_V_aux
    Print #i, sensors_vble(i - 1).MatrixV(3, j - 1), sensors_vble(i - 1).MatrixV(0, j _
    - 1), sensors_vble(i - 1).MatrixV(1, j - 1), sensors_vble(i - 1).MatrixV(2, j - 1)
    Next j

    sensors_vble(i - 1).MsgCounter_V_aux = 0
    lstMsg_V.Clear

    Close #i

    Open _
    "C:\Documents and Settings\uos\Desktop\programs\Visual Basic\last
one\PTV_" & CStr(i) & ".txt" _
    For Append As #i + 1

    For j = 1 To sensors_vble(i - 1).MsgCounter_T_aux
    Print #i + 1, sensors_vble(i - 1).MatrixT(3, j - 1), sensors_vble(i - 1).MatrixT(0,
j _
```

```
    - 1), sensors_vble(i - 1).MatrixT(1, j - 1), sensors_vble(i - 1).MatrixT(2, j - 1)
    Next j

    sensors_vble(i - 1).MsgCounter_T_aux = 0
    lstMsg_T.Clear

    Close #i + 1

Next i
End Sub
```

Module – Module1

```
Option Explicit

'CAN_Message.
Public Type TPCANMsg
    ID As Long              ' 11/29 Bit-Identifier
    MSGTYPE As Byte         ' Bits from MSGTYPE_*
    LEN As Byte             ' Data length code of the message(from 0 to 8)
    DATA(7) As Byte         ' data bytes 0..7
End Type

'Declarations of the functions.

Public Declare Function CAN_Read Lib "pcan_usb" (ByRef pMsgBuff As
TPCANMsg) As Long

Public Declare Function CAN_Init Lib "pcan_usb" _
       (ByVal wBTR0BTR1 As Integer, ByVal CANMsgType As Integer) As Long

Public Declare Function CAN_Close Lib "pcan_usb" () As Long

'BTR0 BTR1 register.
'Baud rate code = register value BTR0/BTR1.
Public Const CAN_BAUD_1M = &H14         ' 1 MBit / s
Public Const CAN_BAUD_500K = &H1C       ' 500 kBit / s
Public Const CAN_BAUD_250K = &H11C      ' 250 kBit / s
Public Const CAN_BAUD_125K = &H31C      ' 125 kBit / s
Public Const CAN_BAUD_100K = &H432F     ' 100 kBit / s
Public Const CAN_BAUD_50K = &H472F      '  50 kBit / s
Public Const CAN_BAUD_20K = &H532F      '  20 kBit / s
Public Const CAN_BAUD_10K = &H672F      '  10 kBit / s
Public Const CAN_BAUD_5K = &H7F7F       '   5 kBit / s

'Msg Type:
Public Const CAN_INIT_TYPE_EX = &H1     ' Extended Frame.
Public Const CAN_INIT_TYPE_ST = &H0     ' Standard Frame.

'Variables for each sensor.

Public Type SensorVble

    IDcan As Long       'Each sensor sends messages with a different CAN ID.
    iden As Integer     'Each sensor has an associated identifier.

    firstMsg As Integer 'This variable indicates when this sensor sends its first
                        'message. It is used in the registering process.
```

```
    n As Integer            'To know the received message type, (n=0 -> XYZ) or
                            '(n=1 -> PTV).
    n_anterior As Integer   'It stores the last received message type, (n=0 -> XYZ)
                            'or (n=1 -> PTV).

    Counter As Double           'It counts received messages.
    MsgCounter_V As Double      'It counts received messages of vibration(XYZ).
    MsgCounter_T As Integer     'It counts received messages of pressure,
                                'temperature and voltage(PTV).

    'Auxiliary counters to store the information in a text file periodically.
    MsgCounter_V_aux As Integer
    MsgCounter_T_aux As Integer

    buffer1 As Variant    'To display the message with LEN=7.
    buffer2 As Variant    'To display the message with LEN=5.

    'Variables to count the missed messages.
    MsgCounter_7 As Integer         'with LEN=7.
    MsgCounter_5 As Integer         'with LEN=5.
    MsgCounter_strange As Integer 'with another LEN.

    flag1 As Integer 'It indicates if a message with LEN=7 has been received
                     'before receiving a message with LEN=5.

    vector(5) As Double   'It stores the data without knowledge about the
                          'type(XYZ or PTV).
    value(2) As Double   'It stores the data without transformation.
    MatrixV() As Double 'It stores XYZ data.
    MatrixT() As Double 'It stores PTV data.

End Type

'Array of structs:to store all the information about a sensor in each struct.
Public sensors_vble() As SensorVble

'Constants for the graphs scales.
'x axis:
Public Const scalet_XYZ = 100
Public Const scalet_PTV = 10000
'y axis:
'XYZ
Public Const scale_minx = -100
Public Const scale_maxx = 100
Public Const scale_miny = -100
Public Const scale_maxy = 100
Public Const scale_minz = -3
Public Const scale_maxz = 1000
'PTV
Public Const scale_minp = 0
```

```
Public Const scale_maxp = 200
Public Const scale_mint = 0
Public Const scale_maxt = 85
Public Const scale_minv = 0
Public Const scale_maxv = 20

'Variables to change the graph scales depending on the zoom.
Public min(5) As Integer
Public max(5) As Integer

'Variables to store which variable type of which sensor is plotted in each graph.
Public graph_sensor(5) As Integer  'Which sensor is indicated by the sensor
                                   'identifier.
Public graph_variable(5) As Integer 'Which variable is indicated by:
                                    'X=0,  Y=1,  Z=2,  P=3,  T=4,  V=5.
```

Apéndice 2

MATLAB – Automatic_v2

```
%It is a simple program to import the data stored in the text files that
%the application creates and plot them.
clear
clc

nsensors = input('Number of sensors: ');

for i=1:nsensors

fileXYZ=sprintf('XYZ_%d.txt',i);
eval(['load ',fileXYZ]);
tam=size(fileXYZ); tf=tam(2);
fileXYZ=fileXYZ(1:tf-4);
f1=eval(fileXYZ);
tam=size(f1); maxv=tam(2);

for j=1:3
figure
y1=f1(:,j+1);
plot(f1(:,1),y1);grid;
xlabel('Time');
switch j
case 1
ylabel('X');
case 2
ylabel('Y');
case 3
ylabel('Z');
end
text=sprintf('Sensor in the files number %g',i); title(text);
end


filePTV=sprintf('PTV_%d.txt',i);
eval(['load ',filePTV]);
tam=size(filePTV); tf=tam(2);
filePTV=filePTV(1:tf-4);
f1=eval(filePTV);
tam=size(f1); maxv=tam(2);

for j=1:3
figure
y1=f1(:,j+1);
plot(f1(:,1),y1);grid;
xlabel('Time');
```

```
switch j
case 1
ylabel('P');
case 2
ylabel('T');
case 3
ylabel('V');
end
text=sprintf('Sensor in the files number %g',i); title(text);
end

end
```

Apéndice 3

MATLAB – Nubes de Puntos

task2

```
%The program imports the data, normalises them by subtracting the mean of
%the data, carries out the feature extraction(function "calculo" is called)
%and plots the results.

function task2
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\previousData\bearing.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\previousData\gearmesh.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\previousData\resonance.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\previousData\imbalance.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\previousData\misalignment.mat');
Fault1=calculo(bearing);
Fault2=calculo(gearmesh);
Fault3=calculo(resonance);
Fault4=calculo(imbalance);
Fault5=calculo(misalignment);
Fault=zeros(250,4);
Fault(1:50,:)=Fault1;
Fault(51:100,:)=Fault2;
Fault(101:150,:)=Fault3;
Fault(151:200,:)=Fault4;
Fault(201:250,:)=Fault5;

%To calculate the first 2 principal components of Fault and use them in the visualisation
c=corrcoef(Fault); % Calculates a correlation coefficient matrix c of Fault
[v,d]=eig(c); % Find the eigvectors v and the eigenvalues d of Fault

T=[v(:,end)';v(:,end-1)']; % Create the transformation matrix T from the first two principal components
z=T*Fault'; % Create a 2-dimensional feature vector z
figure
grid;hold on;
```

```
plot(z(1,1:50),z(2,1:50),'bo',z(1,50:100),z(2,50:100),'g*',z(1,100:150),z(2,100:15
0),'kd',z(1,150:200),z(2,150:200),'r+',z(1,200:250),z(2,200:250),'cs'); % Scatter
plot of the 2-dimensional features
legend('bearing', 'gearmesh', 'resonance', 'imbalance', 'misalignment')

%If we do the same thing with the other two components
T2=[v(:,2)';v(:,1)']; % Create the transformation matrix T from the first two
principal components
z2=T2*Fault'; % Create a 2-dimensional feature vector z
figure
grid;hold on;
plot(z2(1,1:50),z2(2,1:50),'bo',z2(1,50:100),z2(2,50:100),'g*',z2(1,100:150),z2(2,
100:150),'kd',z2(1,150:200),z2(2,150:200),'r+',z2(1,200:250),z2(2,200:250),'cs');
% Scatter plot of the 2-dimensional features
legend('bearing', 'gearmesh', 'resonance', 'imbalance', 'misalignment')

%Visualisation in 3D
T3=[v(:,4)';v(:,3)';v(:,2)'];
z3=T3*Fault';
figure
plot3(z3(1,1:50),z3(2,1:50),z3(3,1:50),'bo',z3(1,50:100),z3(2,50:100),z3(3,50:10
0),'g*',z3(1,100:150),z3(2,100:150),z3(3,100:150),'kd',z3(1,150:200),z3(2,150:2
00),z3(3,150:200),'r+',z3(1,200:250),z3(2,200:250),z3(3,200:250),'cs'); %
Scatter plot of the 2-dimensional features
hold on; grid;
legend('bearing', 'gearmesh', 'resonance', 'imbalance', 'misalignment')
```

calculo

```
%%Feature extraction for files with 50000 points


function Fault=calculo(Featurefile)
%file could be bearing, gearmesh, imbalance, misalignment or resonance
M=zeros(1000,50);
for i=1:50
    M(:,i)=Featurefile((1+1000*(i-1)):(i*1000),1);
end
%Normalised M, each column of N is xi
N=zeros(1000,50);
for j=1:50
    N(:,j)=M(:,j)-mean(M(:,j));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Feature 1: the Root Mean Square(RMS) of the signal
%PSD of each column of N
y=zeros(129,50);
k=zeros(129,50);
for k=1:50
    [y(:,k),f(:,k)]=pwelch(N(:,k),[],[],[],1000);
end
ONE=zeros(50,1);
for i=1:50
    ONE(i,1)=norm(y(:,i))/sqrt(1000);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Feature 2: RMS of low frequencies 0-50 Hz
%each column of N is filtered
J=zeros(1000,50);
[B,A]=butter(11,0.1);
for j=1:50
    J(:,j)=filter(B,A,N(:,j));
end
%PSD of each column of J
for k=1:50
    [y(:,k),f(:,k)]=pwelch(J(:,k),[],[],[],1000);
end
TWO=zeros(50,1);
for i=1:50
    TWO(i,1)=norm(y(:,i))/sqrt(1000);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Feature 3: RMS of middle frequencies 50-200 Hz
```

```
%each column of N is filtered
J=zeros(1000,50);
[B,A]=butter(13,[0.1 0.4]);
for j=1:50
    J(:,j)=filter(B,A,N(:,j));
end
%PSD of each column of J
for k=1:50
    [y(:,k),f(:,k)]=pwelch(J(:,k),[],[],[],1000);
end
THREE=zeros(50,1);
for i=1:50
    THREE(i,1)=norm(y(:,i))/sqrt(1000);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Feature 4: RMS of high frequencies 200-500 Hz
%each column of N is filtered
J=zeros(1000,50);
[B,A]=butter(18,0.4,'high');
for j=1:50
    J(:,j)=filter(B,A,N(:,j));
end
%PSD of each column of J
for k=1:50
    [y(:,k),f(:,k)]=pwelch(J(:,k),[],[],[],1000);
end
FOUR=zeros(50,1);
for i=1:50
    FOUR(i,1)=norm(y(:,i))/sqrt(1000);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Fault=zeros(50,4);
for m=1:50
    Fault(m,1)=ONE(m,1);
    Fault(m,2)=TWO(m,1);
    Fault(m,3)=THREE(m,1);
    Fault(m,4)=FOUR(m,1);
end
```

Apéndice 4

MATLAB – Procesamiento

processing78

```
function processing78
% Sensor 7: gearmesh files 1
% Sensor 8: bearing files 2

%%%WIRELESS SENSORS
for i=1:2

fileXYZ=sprintf('XYZ_%d.txt',i);
eval(['load ',fileXYZ]);
tam=size(fileXYZ); tf=tam(2);
fileXYZ=fileXYZ(1:tf-4);
f1=eval(fileXYZ);
tam=size(f1); maxv=tam(2);


data_X=f1(:,2);
l=length(data_X);
flag=0;
if l<25000
    nsensors = input('It is necessary a file with 25000 or more points');
    flag=1;
    break
end
if flag==0
dataW_X=zeros(5000,1);
min=1;
max=5000;
if i==1
figure
end
for k=1:5
    dataW_X=data_X(min:max,1);
    %i=1 -> gearmesh
    if i==1
        [Fault_W1_X,f_W1_X,y_W1_X]=calculo_mPSD(dataW_X);

        c_X=corrcoef(Fault_W1_X); % Calculates a correlation coefficient matrix c
of Fault
        [v_X,d_X]=eig(c_X); % Find the eigvectors v and the eigenvalues d of
Fault
```

```
        T_X=[v_X(:,end)';v_X(:,end-1)']; % Create the transformation matrix T from
the first two principal components
        z_X=T_X*Fault_W1_X'; % Create a 2-dimensional feature vector z
        grid;hold on;
        plot(z_X(1,1:5),z_X(2,1:5),'ko'); % Scatter plot of the 2-dimensional
features
    else
        %i=2 -> bearing
        if i==2
            [Fault_W2_X,f_W2_X,y_W2_X]=calculo_mPSD(dataW_X);
            c_X=corrcoef(Fault_W2_X); % Calculates a correlation coefficient matrix
c of Fault
            [v_X,d_X]=eig(c_X); % Find the eigvectors v and the eigenvalues d of
Fault

            T_X=[v_X(:,end)';v_X(:,end-1)']; % Create the transformation matrix T
from the first two principal components
            z_X=T_X*Fault_W2_X'; % Create a 2-dimensional feature vector z
            hold on;
            plot(z_X(1,1:5),z_X(2,1:5),'r+'); % Scatter plot of the 2-dimensional
features
        end
    end
    min=max+1;
    max=max+5000;
end
title('X')
%Legend('Black Gearmesh','Red Bearing')

end
end

for i=1:2

fileXYZ=sprintf('XYZ_%d.txt',i);
eval(['load ',fileXYZ]);
tam=size(fileXYZ); tf=tam(2);
fileXYZ=fileXYZ(1:tf-4);
f1=eval(fileXYZ);
tam=size(f1); maxv=tam(2);

data_Y=f1(:,3);
l=length(data_Y);
flag=0;
if l<25000
    nsensors = input('It is necessary a file with 25000 or more points');
    flag=1;
    break
end
if flag==0
```

```
dataW_Y=zeros(5000,1);
min=1;
max=5000;
if i==1
figure
end
for k=1:5
   dataW_Y=data_Y(min:max,1);
   %i=1 -> gearmesh
   if i==1
      [Fault_W1_Y,f_W1_Y,y_W1_Y]=calculo_mPSD(dataW_Y);

      c_Y=corrcoef(Fault_W1_Y); % Calculates a correlation coefficient matrix c
of Fault
      [v_Y,d_Y]=eig(c_Y); % Find the eigvectors v and the eigenvalues d of
Fault

      T_Y=[v_Y(:,end)';v_Y(:,end-1)']; % Create the transformation matrix T from
the first two principal components
      z_Y=T_Y*Fault_W1_Y'; % Create a 2-dimensional feature vector z
      grid;hold on;
      plot(z_Y(1,1:5),z_Y(2,1:5),'ko'); % Scatter plot of the 2-dimensional
features
   else
      %i=2 -> bearing
      if i==2
         [Fault_W2_Y,f_W2_Y,y_W2_Y]=calculo_mPSD(dataW_Y);
         c_Y=corrcoef(Fault_W2_Y); % Calculates a correlation coefficient matrix
c of Fault
         [v_Y,d_Y]=eig(c_Y); % Find the eigvectors v and the eigenvalues d of
Fault

         T_Y=[v_Y(:,end)';v_Y(:,end-1)']; % Create the transformation matrix T
from the first two principal components
         z_Y=T_Y*Fault_W2_Y'; % Create a 2-dimensional feature vector z
         hold on;
         plot(z_Y(1,1:5),z_Y(2,1:5),'r+'); % Scatter plot of the 2-dimensional
features
      end
   end
   min=max+1;
   max=max+5000;
end
title('Y')
%Legend('Black Gearmesh','Red Bearing')

end
end

for i=1:2
```

```
fileXYZ=sprintf('XYZ_%d.txt',i);
eval(['load ',fileXYZ]);
tam=size(fileXYZ); tf=tam(2);
fileXYZ=fileXYZ(1:tf-4);
f1=eval(fileXYZ);
tam=size(f1); maxv=tam(2);

data_Z=f1(:,4);
l=length(data_Z);
flag=0;
if l<25000
   nsensors = input('It is necessary a file with 25000 or more points');
   flag=1;
   break
end
if flag==0

dataW_Z=zeros(5000,1);

min=1;
max=5000;
if i==1
figure
end
for k=1:5
   dataW_Z=data_Z(min:max,1);
   %i=1 -> gearmesh
   if i==1
      [Fault_W1_Z,f_W1_Z,y_W1_Z]=calculo_mPSD(dataW_Z);

      c_Z=corrcoef(Fault_W1_Z); % Calculates a correlation coefficient matrix c
of Fault
      [v_Z,d_Z]=eig(c_Z); % Find the eigvectors v and the eigenvalues d of Fault

      T_Z=[v_Z(:,end)';v_Z(:,end-1)']; % Create the transformation matrix T from
the first two principal components
      z_Z=T_Z*Fault_W1_Z'; % Create a 2-dimensional feature vector z
      grid;hold on;
      plot(z_Z(1,1:5),z_Z(2,1:5),'ko'); % Scatter plot of the 2-dimensional
features
   else
      %i=2 -> bearing
      if i==2
         [Fault_W2_Z,f_W2_Z,y_W2_Z]=calculo_mPSD(dataW_Z);
         c_Z=corrcoef(Fault_W2_Z); % Calculates a correlation coefficient matrix
c of Fault
         [v_Z,d_Z]=eig(c_Z); % Find the eigvectors v and the eigenvalues d of
Fault
```

```
            T_Z=[v_Z(:,end)';v_Z(:,end-1)']; % Create the transformation matrix T
from the first two principal components
            z_Z=T_Z*Fault_W2_Z'; % Create a 2-dimensional feature vector z
            hold on;
            plot(z_Z(1,1:5),z_Z(2,1:5),'r+'); % Scatter plot of the 2-dimensional
features
        end
    end
    min=max+1;
    max=max+5000;
end
title('Z')
%Legend('Black Gearmesh','Red Bearing')

end
end

if flag==0
figure;
grid;hold on;
plot(f_W1_X(:,3),y_W1_X(:,3),'k');
hold on;
plot(f_W2_X(:,3),y_W2_X(:,3),'r');
title('PSD X')
Legend('Gearmesh','Bearing')

figure;
grid;hold on;
plot(f_W1_Y(:,3),y_W1_Y(:,3),'k');
hold on;
plot(f_W2_Y(:,3),y_W2_Y(:,3),'r');
title('PSD Y')
Legend('Gearmesh','Bearing')

figure;
grid;hold on;
plot(f_W1_Z(:,3),y_W1_Z(:,3),'k');
hold on;
plot(f_W2_Z(:,3),y_W2_Z(:,3),'r');
title('PSD Z')
Legend('Gearmesh','Bearing')
end
```

processing87

```
% Sensor 8: bearing files 1
% Sensor 7: gearmesh files 2

function processing87
%%%WIRELESS SENSORS

for i=1:2

fileXYZ=sprintf('XYZ_%d.txt',i);
eval(['load ',fileXYZ]);
tam=size(fileXYZ); tf=tam(2);
fileXYZ=fileXYZ(1:tf-4);
f1=eval(fileXYZ);
tam=size(f1); maxv=tam(2);

data_X=f1(:,2);
l=length(data_X);
flag=0;
if l<25000
   nsensors = input('It is necessary a file with 25000 or more points');
   flag=1;
   break
end
if flag==0
dataW_X=zeros(5000,1);
min=1;
max=5000;
if i==1
figure
end
for k=1:5
   dataW_X=data_X(min:max,1);
   %i=1 -> bearing
   if i==1
      [Fault_W1_X,f_W1_X,y_W1_X]=calculo_mPSD(dataW_X);

      c_X=corrcoef(Fault_W1_X); % Calculates a correlation coefficient matrix c
of Fault
      [v_X,d_X]=eig(c_X); % Find the eigvectors v and the eigenvalues d of
Fault

      T_X=[v_X(:,end)';v_X(:,end-1)']; % Create the transformation matrix T from
the first two principal components
      z_X=T_X*Fault_W1_X'; % Create a 2-dimensional feature vector z
      grid;hold on;
      plot(z_X(1,1:5),z_X(2,1:5),'ko'); % Scatter plot of the 2-dimensional
features
```

```
    else
        %i=2 -> gearmesh
        if i==2
            [Fault_W2_X,f_W2_X,y_W2_X]=calculo_mPSD(dataW_X);
            c_X=corrcoef(Fault_W2_X); % Calculates a correlation coefficient matrix
c of Fault
            [v_X,d_X]=eig(c_X); % Find the eigvectors v and the eigenvalues d of
Fault

            T_X=[v_X(:,end)';v_X(:,end-1)']; % Create the transformation matrix T
from the first two principal components
            z_X=T_X*Fault_W2_X'; % Create a 2-dimensional feature vector z
            hold on;
            plot(z_X(1,1:5),z_X(2,1:5),'r+'); % Scatter plot of the 2-dimensional
features
        end
    end
    min=max+1;
    max=max+5000;
end
title('X')
Legend('Black Bearing','Red Gearmesh')

end
end

for i=1:2

fileXYZ=sprintf('XYZ_%d.txt',i);
eval(['load ',fileXYZ]);
tam=size(fileXYZ); tf=tam(2);
fileXYZ=fileXYZ(1:tf-4);
f1=eval(fileXYZ);
tam=size(f1); maxv=tam(2);

data_Y=f1(:,3);
l=length(data_Y);
flag=0;
if l<25000
    nsensors = input('It is necessary a file with 25000 or more points');
    flag=1;
    break
end
if flag==0
dataW_Y=zeros(5000,1);
min=1;
max=5000;
if i==1
figure
end
```

```
for k=1:5
   dataW_Y=data_Y(min:max,1);
   %i=1 -> bearing
   if i==1
      [Fault_W1_Y,f_W1_Y,y_W1_Y]=calculo_mPSD(dataW_Y);

      c_Y=corrcoef(Fault_W1_Y); % Calculates a correlation coefficient matrix c
of Fault
      [v_Y,d_Y]=eig(c_Y); % Find the eigvectors v and the eigenvalues d of
Fault

      T_Y=[v_Y(:,end)';v_Y(:,end-1)']; % Create the transformation matrix T from
the first two principal components
      z_Y=T_Y*Fault_W1_Y'; % Create a 2-dimensional feature vector z
      grid;hold on;
      plot(z_Y(1,1:5),z_Y(2,1:5),'ko'); % Scatter plot of the 2-dimensional
features
   else
      %i=2 -> gearmesh
      if i==2
         [Fault_W2_Y,f_W2_Y,y_W2_Y]=calculo_mPSD(dataW_Y);
         c_Y=corrcoef(Fault_W2_Y); % Calculates a correlation coefficient matrix
c of Fault
         [v_Y,d_Y]=eig(c_Y); % Find the eigvectors v and the eigenvalues d of
Fault

         T_Y=[v_Y(:,end)';v_Y(:,end-1)']; % Create the transformation matrix T
from the first two principal components
         z_Y=T_Y*Fault_W2_Y'; % Create a 2-dimensional feature vector z
         hold on;
         plot(z_Y(1,1:5),z_Y(2,1:5),'r+'); % Scatter plot of the 2-dimensional
features
      end
   end
   min=max+1;
   max=max+5000;
end
title('Y')
Legend('Black Bearing','Red Gearmesh')

end
end

for i=1:2

fileXYZ=sprintf('XYZ_%d.txt',i);
eval(['load ',fileXYZ]);
tam=size(fileXYZ); tf=tam(2);
fileXYZ=fileXYZ(1:tf-4);
f1=eval(fileXYZ);
```

```
tam=size(f1); maxv=tam(2);

data_Z=f1(:,4);
l=length(data_Z);
flag=0;
if l<25000
   nsensors = input('It is necessary a file with 25000 or more points');
   flag=1;
   break
end
if flag==0

dataW_Z=zeros(5000,1);

min=1;
max=5000;
if i==1
figure
end
for k=1:5
   dataW_Z=data_Z(min:max,1);
   %i=1 -> bearing
   if i==1
      [Fault_W1_Z,f_W1_Z,y_W1_Z]=calculo_mPSD(dataW_Z);

      c_Z=corrcoef(Fault_W1_Z); % Calculates a correlation coefficient matrix c
of Fault
      [v_Z,d_Z]=eig(c_Z); % Find the eigvectors v and the eigenvalues d of Fault

      T_Z=[v_Z(:,end)';v_Z(:,end-1)']; % Create the transformation matrix T from
the first two principal components
      z_Z=T_Z*Fault_W1_Z'; % Create a 2-dimensional feature vector z
      grid;hold on;
      plot(z_Z(1,1:5),z_Z(2,1:5),'ko'); % Scatter plot of the 2-dimensional
features
   else
      %i=2 -> gearmesh
      if i==2
         [Fault_W2_Z,f_W2_Z,y_W2_Z]=calculo_mPSD(dataW_Z);
         c_Z=corrcoef(Fault_W2_Z); % Calculates a correlation coefficient matrix
c of Fault
         [v_Z,d_Z]=eig(c_Z); % Find the eigvectors v and the eigenvalues d of
Fault

         T_Z=[v_Z(:,end)';v_Z(:,end-1)']; % Create the transformation matrix T
from the first two principal components
         z_Z=T_Z*Fault_W2_Z'; % Create a 2-dimensional feature vector z
         hold on;
         plot(z_Z(1,1:5),z_Z(2,1:5),'r+'); % Scatter plot of the 2-dimensional
features
```

```
        end
    end
    min=max+1;
    max=max+5000;
end
title('Z')
Legend('Black Bearing','Red Gearmesh')

end
end

if flag==0
figure;
grid;hold on;
plot(f_W1_X(:,3),y_W1_X(:,3),'k');
hold on;
plot(f_W2_X(:,3),y_W2_X(:,3),'r');
title('PSD X')
Legend('Bearing','Gearmesh')

figure;
grid;hold on;
plot(f_W1_Y(:,3),y_W1_Y(:,3),'k');
hold on;
plot(f_W2_Y(:,3),y_W2_Y(:,3),'r');
title('PSD Y')
Legend('Bearing','Gearmesh')

figure;
grid;hold on;
plot(f_W1_Z(:,3),y_W1_Z(:,3),'k');
hold on;
plot(f_W2_Z(:,3),y_W2_Z(:,3),'r');
title('PSD Z')
Legend('Bearing','Gearmesh')
end
```

calculo_mPSD

```
%Feature extraction for files with 5000 points

function [Fault,f,y]=calculo_m(Featurefile)
%file could be bearing, gearmesh, imbalance, misalignment or resonance
M=zeros(1000,5);
for i=1:5
    M(:,i)=Featurefile((1+1000*(i-1)):(i*1000),1);
end
%Normalised M, each column of N is xi
N=zeros(1000,5);
for j=1:5
    N(:,j)=M(:,j)-mean(M(:,j));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Feature 1: the Root Mean Square(RMS) of the signal
%PSD of each column of N
y=zeros(129,5);
f=zeros(129,5);
for k=1:5
    [y(:,k),f(:,k)]=pwelch(N(:,k),[],[],[],1000);
end
ONE=zeros(5,1);
for i=1:5
    ONE(i,1)=norm(y(:,i))/sqrt(1000);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Feature 2: RMS of low frequencies 0-50 Hz
%each column of N is filtered
J=zeros(1000,5);
[B,A]=butter(11,0.1);
for j=1:5
    J(:,j)=filter(B,A,N(:,j));
end
%PSD of each column of J
y2=zeros(129,5);
f2=zeros(129,5);
for k=1:5
    [y2(:,k),f2(:,k)]=pwelch(J(:,k),[],[],[],1000);
end
TWO=zeros(5,1);
for i=1:5
    TWO(i,1)=norm(y2(:,i))/sqrt(1000);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Feature 3: RMS of middle frequencies 50-200 Hz
%each column of N is filtered
J=zeros(1000,5);
[B,A]=butter(13,[0.1 0.4]);
for j=1:5
    J(:,j)=filter(B,A,N(:,j));
end
%PSD of each column of J
y3=zeros(129,5);
f3=zeros(129,5);
for k=1:5
    [y3(:,k),f3(:,k)]=pwelch(J(:,k),[],[],[],1000);
end
THREE=zeros(5,1);
for i=1:5
    THREE(i,1)=norm(y3(:,i))/sqrt(1000);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Feature 4: RMS of high frequencies 200-500 Hz
%each column of N is filtered
J=zeros(1000,5);
[B,A]=butter(18,0.4,'high');
for j=1:5
    J(:,j)=filter(B,A,N(:,j));
end
%PSD of each column of J
y4=zeros(129,5);
f4=zeros(129,5);
for k=1:5
    [y4(:,k),f4(:,k)]=pwelch(J(:,k),[],[],[],1000);
end
FOUR=zeros(5,1);
for i=1:5
    FOUR(i,1)=norm(y4(:,i))/sqrt(1000);
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Fault=zeros(5,4);
for m=1:5
    Fault(m,1)=ONE(m,1);
    Fault(m,2)=TWO(m,1);
    Fault(m,3)=THREE(m,1);
    Fault(m,4)=FOUR(m,1);
end
```

Apéndice 5

MATLAB – Comparación

scale

```
%Data collected with crossbow sensors: before this study and during this
%study. When we plot them they are very different. Here we try to get
%similar scales. First the data are normalised by subtracting the mean and
%then we multiply by a ratio.

function scale
%original files
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\gearmesh.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\g1_6.mat');
figure;
plot(gearmesh);
hold on;
plot(g1,'g');
legend('Previous', 'Collected')
%normalisation
m=mean(gearmesh);
gearmesh_N=zeros(50000,1);
for j=1:50000
   gearmesh_N(j)=gearmesh(j)-m;
end
figure;
plot(gearmesh_N);

m=mean(g1);
g1_N=zeros(5000,1);
for j=1:5000
   g1_N(j)=g1(j)-m;
end
hold on;
plot(g1_N,'g');
%root mean square
result=0;
for i=1:50000
   result=result+gearmesh_N(i)^2;
end
rms_gearmesh=(1/50000)*result

result=0;
for i=1:5000
   result=result+g1_N(i)^2;
```

```
end
rms_g1=(1/5000)*result
%ratio
%ratio=sqrt(rms_gearmesh/rms_g1)
ratio=rms_gearmesh/rms_g1

%modified files
new=zeros(50000,1);
for i=1:5000
    new(i)=g1_N(i)*ratio;
end
hold on;
plot(new,'m');
legend('Previous Normalised', 'Collected Normalised', 'Collected Scaled')
```

normalisation

```
%First the same process as in the program "task2", the data from crossbow
%sensors are normalised by subtracting the mean, the RMS is calculates and
%the data are multiplied by a ratio, then principal components analysis is
%done and the results are plotted.

function task2
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\bearing.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\gearmesh.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\resonance.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\imbalance.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\misalignment.mat');
Fault1=calculo(bearing);
Fault2=calculo(gearmesh);
Fault3=calculo(resonance);
Fault4=calculo(imbalance);
Fault5=calculo(misalignment);
Fault=zeros(250,4);
Fault(1:50,:)=Fault1;
Fault(51:100,:)=Fault2;
Fault(101:150,:)=Fault3;
Fault(151:200,:)=Fault4;
Fault(201:250,:)=Fault5;
%To calculate the first 2 principal components of Fault and use them in the visualisation
c=corrcoef(Fault); % Calculates a correlation coefficient matrix c of Fault
[v,d]=eig(c); % Find the eigvectors v and the eigenvalues d of Fault

T=[v(:,end)';v(:,end-1)']; % Create the transformation matrix T from the first two principal components
z=T*Fault'; % Create a 2-dimensional feature vector z
figure
grid;hold on;
plot(z(1,1:50),z(2,1:50),'bo',z(1,50:100),z(2,50:100),'g*',z(1,100:150),z(2,100:150),'kd',z(1,150:200),z(2,150:200),'r+',z(1,200:250),z(2,200:250),'cs'); % Scatter plot of the 2-dimensional features
```

```
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\g1_6.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\g2_6.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\g3_6.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\g4_6.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\g5_6.mat');


m=mean(gearmesh);
gearmesh_N=zeros(50000,1);
for j=1:50000
   gearmesh_N(j)=gearmesh(j)-m;
end

m1=mean(g1);m2=mean(g2);m3=mean(g3);m4=mean(g4);m5=mean(g5);
g1_N=zeros(5000,1);
g2_N=zeros(5000,1);
g3_N=zeros(5000,1);
g4_N=zeros(5000,1);
g5_N=zeros(5000,1);

for j=1:5000
   g1_N(j)=g1(j)-m1;
   g2_N(j)=g2(j)-m2;
   g3_N(j)=g3(j)-m3;
   g4_N(j)=g4(j)-m4;
   g5_N(j)=g5(j)-m5;
end

%root mean square
result=0;
for i=1:50000
   result=result+gearmesh_N(i)^2;
end
rms_gearmesh=(1/50000)*result

result1=0;result2=0;result3=0;result4=0;result5=0;
for i=1:5000
   result1=result1+g1_N(i)^2;
   result2=result2+g2_N(i)^2;
   result3=result3+g3_N(i)^2;
   result4=result4+g4_N(i)^2;
   result5=result5+g5_N(i)^2;
end
rms_g1=(1/5000)*result1
rms_g2=(1/5000)*result2
```

```
rms_g3=(1/5000)*result3
rms_g4=(1/5000)*result4
rms_g5=(1/5000)*result5
%ratio
ratio1=sqrt(rms_gearmesh/rms_g1)
ratio2=sqrt(rms_gearmesh/rms_g2)
ratio3=sqrt(rms_gearmesh/rms_g3)
ratio4=sqrt(rms_gearmesh/rms_g4)
ratio5=sqrt(rms_gearmesh/rms_g5)
%modified files
new1=zeros(50000,1);new2=zeros(50000,1);new3=zeros(50000,1);new4=zeros
(50000,1);new5=zeros(50000,1);
for i=1:5000
    new1(i)=g1_N(i)*ratio1;
    new2(i)=g2_N(i)*ratio2;
    new3(i)=g3_N(i)*ratio3;
    new4(i)=g4_N(i)*ratio4;
    new5(i)=g5_N(i)*ratio5;
end

Fault1=calculo_m(new1);
Fault2=calculo_m(new2);
Fault3=calculo_m(new3);
Fault4=calculo_m(new4);
Fault5=calculo_m(new5);

Fault=zeros(25,4);
Fault(1:5,:)=Fault1;
Fault(6:10,:)=Fault2;
Fault(11:15,:)=Fault3;
Fault(16:20,:)=Fault4;
Fault(21:25,:)=Fault5;

%To calculate the first 2 principal components of Fault and use them in the
visualisation
c=corrcoef(Fault); % Calculates a correlation coefficient matrix c of Fault
[v,d]=eig(c); % Find the eigvectors v and the eigenvalues d of Fault

T=[v(:,end)';v(:,end-1)']; % Create the transformation matrix T from the first two
principal components
z=T*Fault'; % Create a 2-dimensional feature vector z
hold on;
plot(z(1,1:5),z(2,1:5),'mo',z(1,5:10),z(2,5:10),'mo',z(1,10:15),z(2,10:15),'mo',z(1,
15:20),z(2,15:20),'mo',z(1,20:25),z(2,20:25),'mo'); % Scatter plot of the 2-
dimensional features


load('C:\Documents and
Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\y1_6.mat');
```

```
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\y2_6.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\y3_6.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\y4_6.mat');
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow\match\y5_6.mat');

m=mean(bearing);
bearing_N=zeros(50000,1);
for j=1:50000
   bearing_N(j)=bearing(j)-m;
end

m1=mean(y1);m2=mean(y2);m3=mean(y3);m4=mean(y4);m5=mean(y5);
y1_N=zeros(5000,1);
y2_N=zeros(5000,1);
y3_N=zeros(5000,1);
y4_N=zeros(5000,1);
y5_N=zeros(5000,1);

for j=1:5000
   y1_N(j)=y1(j)-m1;
   y2_N(j)=y2(j)-m2;
   y3_N(j)=y3(j)-m3;
   y4_N(j)=y4(j)-m4;
   y5_N(j)=y5(j)-m5;
end

%root mean square
result=0;
for i=1:50000
   result=result+bearing_N(i)^2;
end
rms_bearing=(1/50000)*result

result1=0;result2=0;result3=0;result4=0;result5=0;
for i=1:5000
   result1=result1+y1_N(i)^2;
   result2=result2+y2_N(i)^2;
   result3=result3+y3_N(i)^2;
   result4=result4+y4_N(i)^2;
   result5=result5+y5_N(i)^2;
end
rms_y1=(1/5000)*result1
rms_y2=(1/5000)*result2
rms_y3=(1/5000)*result3
rms_y4=(1/5000)*result4
rms_y5=(1/5000)*result5
```

```
%ratio
ratio1=sqrt(rms_bearing/rms_y1)
ratio2=sqrt(rms_bearing/rms_y2)
ratio3=sqrt(rms_bearing/rms_y3)
ratio4=sqrt(rms_bearing/rms_y4)
ratio5=sqrt(rms_bearing/rms_y5)
%modified files
new1=zeros(50000,1);new2=zeros(50000,1);new3=zeros(50000,1);new4=zeros
(50000,1);new5=zeros(50000,1);
for i=1:5000
    new1(i)=y1_N(i)*ratio1;
    new2(i)=y2_N(i)*ratio2;
    new3(i)=y3_N(i)*ratio3;
    new4(i)=y4_N(i)*ratio4;
    new5(i)=y5_N(i)*ratio5;
end

Fault1=calculo_m(new1);
Fault2=calculo_m(new2);
Fault3=calculo_m(new3);
Fault4=calculo_m(new4);
Fault5=calculo_m(new5);

Fault=zeros(25,4);
Fault(1:5,:)=Fault1;
Fault(6:10,:)=Fault2;
Fault(11:15,:)=Fault3;
Fault(16:20,:)=Fault4;
Fault(21:25,:)=Fault5;

%To calculate the first 2 principal components of Fault and use them in the
visualisation
c=corrcoef(Fault); % Calculates a correlation coefficient matrix c of Fault
[v,d]=eig(c); % Find the eigvectors v and the eigenvalues d of Fault

T=[v(:,end)';v(:,end-1)']; % Create the transformation matrix T from the first two
principal components
z=T*Fault'; % Create a 2-dimensional feature vector z
hold on;
plot(z(1,1:5),z(2,1:5),'yo',z(1,5:10),z(2,5:10),'yo',z(1,10:15),z(2,10:15),'yo',z(1,15
:20),z(2,15:20),'yo',z(1,20:25),z(2,20:25),'yo'); % Scatter plot of the 2-
dimensional features

legend('bearing', 'gearmesh', 'resonance', 'imbalance', 'misalignment')
```

processing_ojo

```
%This program imports data collected with crossbow sensors and with
%wireless sensors and tries to match them.

%IMPORTANT: the data from wireless sensors have to be stored as:
% Sensor 8 - Bearing in Files 1
% Sensor 7 - Gearmesh in Files 2

function processing_ojo

%CROSSBOW
%For both failures: Load the data, normalise them and calculate max and min.

%%%GEARMESH
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow-Wireless\g1_6.mat');

mg1=mean(g1);
vg1=var(g1);
g1_N=zeros(5000,1);
g1_N=g1-mg1;
g1_N=g1_N/vg1;

g1_N_max=max(g1_N)
g1_N_min=min(g1_N)
r1=g1_N_max-g1_N_min


%%%BEARING
load('C:\Documents and Settings\Administrador\Escritorio\programs\Matlab\Crossbow-Wireless\y1_6.mat');
mb1=mean(y1);
vb1=var(y1);
b1_N=zeros(5000,1);
b1_N=y1-mb1;
b1_N=b1_N/vb1;

b1_N_max=max(b1_N)
b1_N_min=min(b1_N)
r2=b1_N_max-b1_N_min

%WIRELESS
%For both failures: Load the data, look at the length of the files and ask
%the user from which value he wants to extract 5000 points to compare,
%normalise the data and calculate max and min
```

```
nsensors = 2;

for i=1:nsensors

fileXYZ=sprintf('XYZ_%d.txt',i);
eval(['load ',fileXYZ]);
tam=size(fileXYZ); tf=tam(2);
fileXYZ=fileXYZ(1:tf-4);
f1=eval(fileXYZ);
tam=size(f1); maxv=tam(2);

%vertical
data=f1(:,2);
length(data)
first=input('First value:');
dataW=zeros(5000,1);
dataW=data(first:(first+5000),1);

mW=mean(dataW);
vW=var(dataW);
dataW=dataW-mW;
dataW=dataW/vW;

dataW_max=max(dataW)
dataW_min=min(dataW)
r=dataW_max-dataW_min

%For both failures: The collected data from crossbow sensors are scaled and
%then the data of each kind of sensor are plotted.

figure
plot(dataW);grid;
xlabel('Time msec');

%i=1 -> bearing
if i==1
   b1_N=b1_N*(r/r2);
   b1_N_max=max(b1_N)
   b1_N_min=min(b1_N)
   hold on;
   plot(b1_N,'g');
   title('Bearing')
   Legend('wireless','crossbow')

   %The function "calculo_mPSD" is called to carry out the feature
   %extraction
   [Fault_b_crossbow,f_b_crossbow,y_b_crossbow]=calculo_mPSD(b1_N);
   [Fault_W1,f_W1,y_W1]=calculo_mPSD(dataW);
else
   %i=2 -> gearmesh
```

```
    if i==2
        g1_N=g1_N*(r/r1);
        g1_N_max=max(g1_N)
        g1_N_min=min(g1_N)
        hold on;
        plot(g1_N,'g')
        title('Gearmesh')
        Legend('wireless','crossbow')

        %The function "calculo_mPSD" is called to carry out the feature
        %extraction
        [Fault_g_crossbow,f_g_crossbow,y_g_crossbow]=calculo_mPSD(g1_N);
        [Fault_W2,f_W2,y_W2]=calculo_mPSD(dataW);
    end
end

end

%PRINCIPAL COMPONENTS ANALYSIS

%CROSSBOW-BEARING
%To calculate the first 2 principal components of Fault and use them in the
visualisation
c=corrcoef(Fault_b_crossbow); % Calculates a correlation coefficient matrix c of
Fault
[v,d]=eig(c); % Find the eigvectors v and the eigenvalues d of Fault

T=[v(:,end)';v(:,end-1)']; % Create the transformation matrix T from the first two
principal components
z=T*Fault_b_crossbow'; % Create a 2-dimensional feature vector z
figure
grid;hold on;
plot(z(1,1:5),z(2,1:5),'bo'); % Scatter plot of the 2-dimensional features

%CROSSBOW-GEARMESH
c=corrcoef(Fault_g_crossbow); % Calculates a correlation coefficient matrix c of
Fault
[v,d]=eig(c); % Find the eigvectors v and the eigenvalues d of Fault

T=[v(:,end)';v(:,end-1)']; % Create the transformation matrix T from the first two
principal components
z=T*Fault_g_crossbow'; % Create a 2-dimensional feature vector z
hold on;
plot(z(1,1:5),z(2,1:5),'g*'); % Scatter plot of the 2-dimensional features

%WIRELESS-BEARING
c=corrcoef(Fault_W1); % Calculates a correlation coefficient matrix c of Fault
[v,d]=eig(c); % Find the eigvectors v and the eigenvalues d of Fault
```

```
T=[v(:,end)';v(:,end-1)']; % Create the transformation matrix T from the first two principal components
z=T*Fault_W1'; % Create a 2-dimensional feature vector z
grid;hold on;
plot(z(1,1:5),z(2,1:5),'kd'); % Scatter plot of the 2-dimensional features

%WIRELESS-GEARMESH
c=corrcoef(Fault_W2); % Calculates a correlation coefficient matrix c of Fault
[v,d]=eig(c); % Find the eigvectors v and the eigenvalues d of Fault

T=[v(:,end)';v(:,end-1)']; % Create the transformation matrix T from the first two principal components
z=T*Fault_W2'; % Create a 2-dimensional feature vector z
grid;hold on;
plot(z(1,1:5),z(2,1:5),'r+'); % Scatter plot of the 2-dimensional features

legend('bearing crossbow','gearmesh crossbow', 'bearing wireless','gearmesh wireless' )

%PSDs
figure;
grid;hold on;
plot(f_b_crossbow,y_b_crossbow,'b');
hold on;
plot(f_g_crossbow,y_g_crossbow,'g');
hold on;
plot(f_W1,y_W1,'k');
hold on;
plot(f_W2,y_W2,'r');

figure;grid;hold on;
plot(f_b_crossbow,y_b_crossbow,'b',f_W1,y_W1,'k');title('BEARING');

figure;grid;hold on;
plot(f_g_crossbow,y_g_crossbow,'g',f_W2,y_W2,'r');title('GEARMESH');
```