REBECA RUBI ARISTA RANGEL

SISTEMA DE ALINEAMIENTO GIROSCÓPICO PARA UNA LÍNEA DE MONTAJE AERONÁUTICA

SEVILLA
2011

UNIVERSIDAD DE SEVILLA

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS

SISTEMA DE ALINEAMIENTO GIROSCÓPICO PARA UNA LÍNEA DE MONTAJE AERONÁUTICA

Proyecto presentado en la Universidad de Sevilla
como uno de los requisitos para la obtención del grado
Ingeniero en Telecomunicación, Especialidad en Automática y Robótica.

POR

REBECA RUBI ARISTA RANGEL

Sevilla, Febrero de 2011.

"Si se puede imaginar... se puede programar"
Arturo M. Lopez

AGRADECIMENTOS

Agradezco a mis padres la gran oportunidad que me dieron de estudiar en el extranjero. A mi padre Sergio por apoyarme y estar conmigo en todas las decisiones de mi vida, y por ser un ejemplo a seguir. A mi madre Rebeca porque siempre de una forma u otra ha estado a mi lado, haciéndome sentir que estoy en casa aún estando al otro lado del mundo.

A mi hermana Karina por ser parte fundamental del 'barco' y por su grandísimo cariño.

Agradezco a mi familia Rangel y Arista por todas sus palabras de aliento, por creer y confiar tanto en mí y ayudarme con todos aquellos buenos deseos, oraciones y demás.

Agradezco a toda mi familia Mejías Naranjo, por el apoyo incondicional que me han dado durante mi estancia completa en Sevilla, y por haberme convencido de estudiar en esta maravillosa ciudad.

A mi novio Alberto, por haberme apoyado en momentos buenos y malos, por su paciencia, sus incansables palabras de aliento y su cariño.

Me gustaría agradecer especialmente a mi tutor Prof. Fabio Gómez-Estern Aguilar por su dedicación, por haber confiado en mí para llevar a cabo este proyecto, por todos sus buenos consejos y por ser además un muy buen amigo. Igualmente al Prof. Francisco Gordillo su ayuda y dedicación. Agradezco a Gilherme su paciencia y todos sus buenos consejos.

A mis niñas y a todos los amigos que hice durante todos estos años en Sevilla, muchísimas gracias por todos los hermosos momentos que me habéis hecho vivir en todos estos años. Los llevo a todos en mi corazón.

Resumen del proyecto presentado en la Universidad de Sevilla como uno de los requisitos necesarios para la obtención del grado de Ingeniero en Telecomunicación, Especialidad en Automática y Robótica.

SISTEMA DE ALINEAMIENTO GIROSCÓPICO PARA UNA LÍNEA DE MONTAJE AERONÁUTICA

Rebeca Rubi Arista Rangel

Febrero/2011

Tutor:	Prof. D. Fabio Gómez-Estern Aguilar, Dr.
Área de Concentración:	Ingeniería de Sistemas y Automática
Palabras-clave:	Sistema de alineamiento, Sistema de medida inercial, Tratamiento de matrices de rotación, Control de trayectoria, Ángulos de Euler
Número de Páginas:	142

Dentro de la colaboración de I+D que mantiene el Grupo de Control no Lineal con las compañías ELIMCO y EADS se ha desarrollado un sistema para alinear en 3 ejes angulares, con precisión inferior a 0.16mrads, una serie de equipos que se instalarán en las aeronaves Airbus A400M en la factoría de de EADS de San Pablo, Sevilla. El sistema realiza un tratamiento matemático automatizado de las matrices de rotación de los cuerpos solidarios a las unidades, y mediante intercambio de datos en red, estima la orientación relativa de ambos cuerpos en términos de ángulos de Euler. Dichos resultados se comparan con una base de datos configurable con información sobre la aeronave, para estimar el error neto de montaje.

Abstract of proyect presented to Universidad de Sevilla as a partial fulfillment of the requirements for the degree of Telecommunications Engineering in Automation and Robotics.

GYROSCOPIC ALIGNMENT SYSTEM FOR AN AERONAUTICAL ASSEMBLY LINE

Rebeca Rubi Arista Rangel

February/2011

Advisor:	Prof. D. Fabio Gómez-Estern Aguilar, Dr.
Area of Concentration:	Automation and Systems Engineering
Keywords:	Alignment System, Inertial measurement system, Rotation matrix treatment, Path control, Euler angles
Number of Pages:	142

Within the R&D cooperation supported by the Non linear Control Group with ELIMCO and EADS companies, a system has been developed to align a series of equipments in 3 angular axes with a precision of less than 0.16mrads, which will be installed in the Airbus A400M aircraft at San Pablo EADS's factory in Seville. The system carries out an automation mathematical treatment of the unity rotation matrixes and using a network data interchange estimates the relative orientation of both unities in terms of Euler angels. To estimate the assembly net error, these results are compared with a configurable data base which contains the aircraft information.

Índice general

Índice de figuras

Índice de tablas

Lista de abreviaturas

IMU	*Inertial Measurement Unit*
CU	*Control Unit*
RMU	*Reference Measurement Unit*
PMU	*Positional Measurement Unit*
GADIRU	*GPS Air Data Inertial Reference Unit*
INS	*Inertial Navigation System*
GPS	*Global Position System*
HUD	*Head-up Display*
EVS	*Enhanced Vision System*
3D	*3 Dimensiones*
NMEA	*National Maritime Electronics Association*
DSP	*Digital Signal Processor*
ZUPT	*Zero Velocity Update*
PPS	*Pulse Per Second*
RISC	*Reduced Instruction Set Computer*
SDRAM	*Synchronous Dynamic Random Access Memory*
GPIO	*General Purpose Input/Output*
TFT	*Thin Film Transistor*
LCD	*Liquid Crystal Display*
USB	*Universal Serial Bus*
CODEC	*Coder/Decoder*
UART	*Universal Asynchronous Receiver-Transmitter*
FIFO	*First In, First Out*
JTAG	*Joint Test Action Group of IEEE*
FLASH	*Memoria no volátil incluso en ausencia de alimentación*
RAM	*Random Access Memory*

Capítulo 1

Introducción

1.1. Objetivo del Proyecto

Revisión, puesta en marcha y pruebas de un sistema de alineamiento de tipo Strapdown para el montaje de precisión de piezas aeronáuticas para el aeronave Airbus A400M. Dentro de la colaboración de I+D que mantiene el Grupo de Control no Lineal con las compañías ELIMCO y EADS se ha desarrollado un novedoso sistema para alinear en 3 ejes angulares con precisión inferior a 0.16mrads una serie de equipos que se instalarán en las aeronaves Airbus A400M en la factoría de de EADS de San Pablo, Sevilla.

En proyectos anteriores se han desarrollado modelos matemáticos para comprobar la validez del sistema de alineamiento (satisfactoriamente) y se ha llegado a implementar una versión preliminar del sistema basada en dos unidades de medida comunicadas por red Ethernet, cada una de ellas con los siguientes elementos:

1. Unidad de medida inercial IXSEA de alta precisión.
2. Placa de control, registro y proceso de señales PC104 ARCOM VIPER, sobre la que corre el sistema operativo Windows CE.
3. Chasis y electrónica de acondicionamiento de señal.

El sistema realiza un tratamiento matemático automatizado de las matrices de rotación de los cuerpos solidarios a las unidades, y mediante intercambio de datos en red, estima a la orientación (actitud) relativa de ambos cuerpos en términos de ángulos de Euler. Dichos resultados se comparan con una base de datos configurable con información sobre la aeronave para estimar el error neto de montaje.

1.2. Alcance del Proyecto

El sistema está actualmente implementado y es preciso una fase de revisión para pasar a la de producción. Para ello es preciso realizar las siguientes tareas:

1. Revisar o realizar una nueva implementación el sistema de conteo de giros realizado por el operario al realizar el montaje para poder limitar su número, y así controlar el error máximo de los algoritmos de integración de trayectorias implementados en la Unidad de Medida Inercial.
2. Realizar modificaciones en el sistema de cálculo del error del montaje de modo que se pueda fácilmente reconfigurar para su uso en nuevos equipos.
3. Asistir a EADS en las pruebas y la configuración en la Línea de Ensamblaje Final del A400M (FAL) si fuera necesario.
4. Depurar y corregir los problemas que pudieran surgir en las pruebas.

1.3. Planificación del Proyecto

En este documento se han estructurado los pasos desarrollados a lo largo del proyecto, que constan de la documentación inicial, el desarrollo de nuevas prestaciones, la depuración del programa y las conclusiones obtenidas a lo largo de la evolución del proyecto.

En el segundo capítulo se describen las características hardware de los dos componentes principales de las unidades de medida, la unidad de medida inercial IMU modelo AIRINS del fabricante iXSea, y la unidad de control CU o microcomputador empotrado VIPER del fabricante Arcom, así como la integración de ambos dentro de las unidades de medida PMU y RMU.

En el tercer capítulo se describen las características software del sistema, se explica el funcionamiento general del programa ejecutado en la unidad de control CU de las unidades de medida, así como el funcionamiento del sistema en el arranque y configuración. A su vez, se describe el modo de comunicación de la unidad CU con la unidad IMU dentro de cada unidad de medida, la comunicación UDP establecida entre ambas unidades de medida y una visión global del papel de ambas unidades dentro del sistema.

En el cuarto capítulo se detalla el procedimiento para obtener la matriz de rotación teórica de los equipos a partir de los parámetros dados por el usuario. Posteriormente se muestra el proceso para el cálculo de la desviación total, y finalmente se comprueba la

validez de este nuevo método de cálculo mediante una comparativa de los resultados con los entregados por el procedimiento anterior.

El quinto capítulo describe las distintas prestaciones que se han implementado en el programa. Éstas son, un contador de vueltas sobre el ángulo heading para monitorizar la correcta ejecución de la trayectoria de un experimento, un filtro de medidas configurable por el usuario, un botón de salida para poder detener la ejecución de un experimento, y un icono de acceso directo al programa en el escritorio para facilitar el lanzamiento del programa.

En el sexto capítulo se describen las pruebas a las que ha sido sometido el sistema para comprobar su correcto funcionamiento y para depurar los posibles fallos encontrados.

El séptimo capítulo esta dedicado a las conclusiones derivadas de los resultados previamente obtenidos, así como las líneas futuras de investigación y posibles mejoras del sistema.

Por último se muestra la bibliografía utilizada a lo largo del desarrollo del proyecto, así como los apéndices, donde se encuentra el manual de usuario desarrollado como herramienta para una fácil implementación del sistema, y el código fuente de las funciones más relevantes del programa “viper.exe” dentro de las clases CDialogMedida, CViperDlg y CCESocket .

Capítulo 2

Estructura Hardware

2.1. Introducción

Como se ha comentado anteriormente, cada unidad de medida consta de una unidad de medida inercial IMU (en inglés *Inertial Maesurement Unity*) comunicada mediante el puerto serie con un computador empotrado. La unidad de medida elegida es el modelo AIRINS de IXSEA, y el computador utilizado es la unidad VIPER de la empresa Arcom. En este capítulo se describen las características principales de estos equipos.

2.2. La IMU de IXSEA

2.2.1. Descripción

La unidad AIRINS es un Sistema de Navegación Inercial (en inglés INS *Inertial Navigation System*) compacto GPS/INS diseñado para cumplir los requerimientos de mapeo aéreo y en aplicaciones de detección remota. Es una solución de IXSEA, para la industria dedicada al estudio aéreo y terrestre, para cubrir la creciente demanda de un posicionamiento de alta precisión, orientación y datos georeferenciados.

La unidad AIRINS muestra su orientación, además de su posición y velocidad, y puede recibir datos por parte de otros sensores permitiendo mejorar su precisión. La información es enviada por seis puertos serie asíncronos. Puede utilizarse también como

un girocompás, y a su vez puede interactuar con un GPS mediante dos conectores series usando el formato NMEA.

La unidad indica su posición con una tasa que puede ser de hasta 100 Hz, generando los siguientes parámetros:

- 3 ángulos de Euler: Heading, Roll y Pitch.
- Tiempo asociado a los ángulos.
- La velocidad Norte y Este.
- La posición 3D.
- El estado de sistema.

En este proyecto solo se usarán los datos de los ángulos de Euler, el tiempo asociado a ellos y el estado del sistema. Una característica muy importante que nos permitirá realizar los experimentos de forma correcta es su capacidad para general un pulso PPS, el cual pondrá a cero el reloj de la unidad de medida.

En el siguiente apartado se describirán las características más importantes de la unidad AIRINS. Para una información más extensa habrá que dirigirse al manual de usuario de la unidad. (IXSEA land & air, 2009)

2.2.2. Características

El corazón de la unidad AIRINS es la IMU alojada en su interior. La IMU se compone de 3 giroscopios de fibra óptica de muy alta gama (0,01 grados/hora) fabricados por IXSEA y por 3 acelerómetros pendulares de alta precisión situados sobre tres ejes ortogonales. Además de la IMU, la unidad AIRINS usa un algoritmo de navegación basado en el Filtro de Kalman, haciendo posible corregir las medidas de la IMU usando sensores externos. Así puede trabajar como una caja negra o incluso ser conectado a otro sistema externo, como por ejemplo un GPS, aumentando en gran medida el rendimiento obtenido con sistemas tradicionales de navegación.

Como podemos ver en la figura 2.1, la unidad se trata de un volumen compacto en forma de cubo de 16 cm de lado y solo 4,5 kg. En el interior de este volumen se encuentran la IMU y un DSP que ejecutará el software de navegación incluido en la unidad. El consumo energético del conjunto está reducido a tan sólo 12 vatios. Como el uso de la

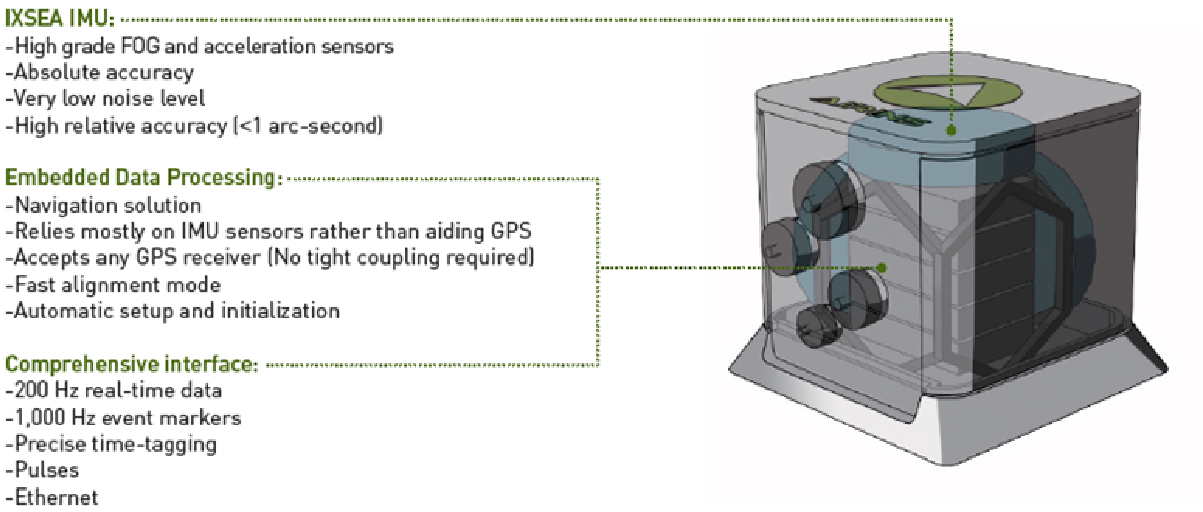

Figura 2.1: Unidad Airins de IXSEA

unidad está orientado para aplicaciones en el aire, el filtro de navegación ha sido modificado para cumplir con las características de este tipo de pruebas, que son:

- Sin límite de velocidad.
- Sin límite de altitud.

También está disponible la posibilidad de situar la unidad en modo ZUPT. Los giroscopios que componen la IMU son de fibra óptica. Hoy en día estos giroscopios de alta precisión son equivalentes o incluso superiores en rendimiento a los mejores giroscopios de láser en anillo.

Una última característica a destacar de la unidad es el modo de alineamiento inicial. La unidad utiliza esta característica para reiniciar las posiciones de los ángulos y eliminar así el posible error de deriva que se pudiera estar cometiendo heredado de medidas anteriores. Durante la fase de alineamiento inicial, la posición es inicializada a la posición más reciente almacenada en la memoria no volátil de la unidad, o bien a la posición introducida manualmente usando un software específico.

Es posible también inicializar la posición a unas coordenadas GPS siempre y cuando se encuentre algún dispositivo GPS conectado a la unidad. La fase de alineamiento inicial es de 5 minutos y durante ese periodo la posición de la unidad no es computada y los ángulos heading, roll y pitch son señalados como "no válidos". El alineamiento puede ser realizado en vuelo, aunque no se recomienda.

El diagrama de bloques del funcionamiento de la unidad se muestra en la figura 2.2.

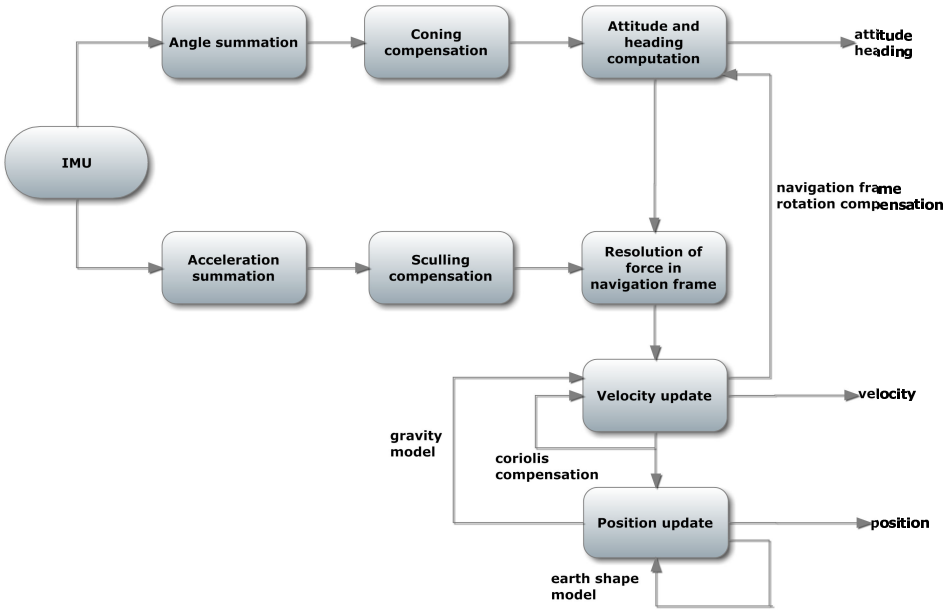

Figura 2.2: Diagrama de bloques de Airins

2.2.3. Acumulación del error

La exactitud de los datos generados por la unidad depende, tanto de la precisión de los acelerómetros y los giroscopios, como de los errores de posición y velocidad iniciales obtenidos después del alineamiento. Debido a que los datos de los giroscopios y los acelerómetros son computados a lo largo del tiempo y, por otra parte, debido a que el cálculo computacional de la velocidad y posición forma un bucle cerrado, todos estos errores se acumulan en el tiempo e influyen unos sobre otros. A esto se le da el nombre de "error de deriva", y refleja una diferencia creciente entre la posición mostrada por la IMU y la posición real en la que se encuentra. La observación del vector gravedad mediante acelerómetros sirve como una observación externa de la vertical en un punto (local). Esto permite corregir la mayoría de los errores de deriva.

2.3. La unidad de control VIPER de ARCOM

2.3.1. Descripción

La unidad de control VIPER es un ordenador empotrado compatible con el estándar PC/104 de muy bajo consumo. Está basado en el procesador Intel® 400MHz PXA255 XScale® RISC. El PXA255 es una implementación del compilador ARM, de la microarquitectura de Intel XScale, junto con una serie de periféricos integrados, incluidos un controlador gráfico de display, un controlador DMA, un controlador de interrupciones, un reloj de tiempo real y múltiples puertos serie. (EUROTECH Imagine, build, succeed, 2010)

La placa base ofrece un amplio rango de características, haciéndola ideal para la labor que requerimos de ella, debido a su bajo consumo, como su gran capacidad de cálculo, además de la inclusión de memoria no volátil, puertos serie y Ethernet. La placa base se muestra en la siguiente figura 2.3.

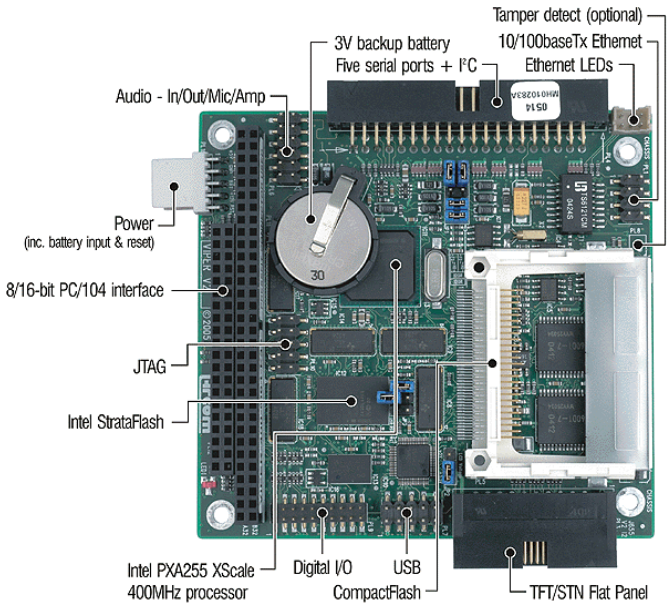

Figura 2.3: Microcomputador VIPER

La unidad de control CU está formada por el propio microcomputador VIPER, la fuente de alimentación, un display táctil que hace la función de interfaz gráfica, y está provista de múltiples puertos USB, puertos serie y un puerto Ethernet. Además contiene el sistema operativo Windows CE 5.0 preinstalado y listo para ser usado. Sobre este sistema operativo, se ejecuta el software diseñado en este Proyecto Final de Carrera llamado “Viper.exe”, el cual se detallará más adelante.

El conjunto tiene el aspecto mostrado en la figura 2.4. Como bien se ha comentado, la unidad de control se encuentra empotrada en el sistema de medida final, por lo que en la figura se muestra un equipo perteneciente al kit de desarrollo proporcionado por el fabricante para la realización de pruebas del software EUROTECH Ltd (2007a). El equipo de medida final se mostrará más adelante.

Figura 2.4: Unidad de control

2.3.2. Características

A continuación se detallan las características principales de la unidad de control. Para consultar con mayor profundidad las especificaciones ver EUROTECH boards & modules (2010) y EUROTECH Ltd (2007b).

Microprocesador PXA255

El microprocesador PXA255 es el cerebro que organiza y da vida a la unidad de control. Se trata de un microprocesador de bajo consumo de tipo RISC. Contiene un DSP que hace las funciones de co-procesador y que potencia las aplicaciones multimedia.

El microcontrolador procesa a una velocidad de 400 MHz gracias al reloj incluido de 3.68645 MHz. Se alimenta de la fuente de alimentación principal de 5V. También posee un controlador de memoria con un bus de memoria a 100 MHz, 32 KB de memoria para datos y 32 KB de memoria para instrucciones.

Además puede gestionar hasta 85 pins de líneas de entrada y salida de propósito general (GPIO), aunque muchas de ellas han sido configuradas para funciones alternativas como la de la tarjeta de sonido y la interfaz CompactFLASH. Por último, destacar que el microprocesador PXS255 es un dispositivo de bajo consumo y no requiere disipador de calor, para temperaturas de hasta 70ºC.

Temporizador

El microprocesador PXA255 contiene un temporizador interno que puede ser usado para proteger al sistema de errores en el software. Los periodos del temporizador pueden ser ajustados, desde un mínimo de 271 ns hasta un máximo de 19 minutos y 25 segundos. Cuando los periodos cumplen, la placa se resetea. En este reseteo, el temporizador es desactivado hasta una nueva activación por software.

Memoria

La CU cuenta con distintos tipos de memoria, cada una con un uso y capacidad distinta. Dispone de una memoria SDRAM de 64MB, óptima para las tareas que realizará la computadora. Esta memoria está montada en la placa mediante soldadura, por lo que no podrá ampliarse su capacidad en un futuro. La memoria puede funcionar a frecuencias que oscilan entre los 50MHz y los 99.5MHz, lo que permite configurar el funcionamiento para conseguir un óptimo balance entre consumo de energía y rendimiento.

Además de la memoria RAM, se dispone de una memora no volátil, imprescindible para almacenar el sistema operativo, el software, y los archivos de configuración, experimentos y de resultados. Ésta tendrá una capacidad de 32 MB, óptima para almacenar lo anteriormente comentado.

Soporte para display LCD

El microprocesador Intel PXA255 integra un controlador de display LCD que permite disponer de una pantalla a color como interfaz gráfica. El display posee una resolución máxima de 600x800 píxeles, suficientes para mostrar por pantalla la interfaz gráfica del software que se ha diseñado. El display montado es táctil, por lo que interactuar con el software resulta intuitivo y sencillo.

Líneas de entrada/salida de propósito general (GPIO)

La unidad CU cuenta con 8 líneas de entrada/salida de propósito general. Mediante éstas, se generará un pulso necesario para sincronizar las dos unidades de medida RMU y PMU. En apartados posteriores, se describirá con mayor detalle la necesidad de este pulso. Para leer de las GPIO, sólo es necesario leer los bytes menos significativos, localizados a partir de una dirección concreta de memoria. Para escribir en estas líneas, se inserta un 1 en los bytes destinados a tal fin. Las líneas de entrada toleran un máximo de 5V de tensión, y las líneas de salida pueden generar 3.3V a 24mA como máximo.

Puertos USB

La unidad de control tiene dos puertos USB 1.1 que soportan velocidades de transferencia de hasta 12 Mbit/s. La existencia de estos puertos es de gran utilidad, ya que nos permiten almacenar rápidamente en la memoria flash nuevas versiones del software diseñado, o almacenar archivos de configuración a partir de una unidad de memoria flash portátil, además de facilitar la recuperación del archivo de resultados.

Conector 10/100BaseTx Ehernet

El controlador Ethernet Lan91C111 de SMSC incluido en la placa provee una interfaz 10/100BaseTX. Ésta interfaz es imprescindible para la comunicación con la unidad de medida, ya que a través de ésta se canalizan las comunicaciones UDP y TCP utilizadas continuamente a lo largo de la ejecución del programa. Como podremos comprobar en capítulos posteriores, la interfaz Ethernet de la unidad control es una característica imprescindible.

Puertos Serie COM

La CU está provista de 5 UARTs de alta velocidad. Cuatro de estos canales (COM1-COM4) pueden ser usados como interfaces estándar RS232, y la restante (COM5) puede ser configurada como RS422 o como RS485. Al igual que el puerto Ethernet, los puertos serie son una característica primordial de esta placa. Serán utilizados en la tarea de recepción continua de datos enviados por parte de la IMU y, a su vez, se utilizarán para enviar tramas de restart y configuración en sentido CU-IMU.

Otras características

Otras prestaciones con las que cuenta la unidad de control son:

- una memoria cache de 32K de datos y 32K de instrucciones
- un controlador de gráficos de pantalla plana TFT de hasta 800x600 de resolución con control de retroiluminación utilizado para mejorar la resolución del display
- un chip CODEC de audio, modelo LM4529 AC´97 de National Semiconductor con amplificador de potencia LM4880, para dar soporte a las características de audio
- un bus de expansión PC/104 que proporciona una mayor escalabilidad
- un puerto JTAG que permite la descarga de datos a la memoria flash

En principio estas funcionalidades no serán utilizadas dentro del desarrollo del programa, pero quedan disponibles para un posible uso en futuras actualizaciones del software.

2.4. Unidades de medida PMU y RMU

Figura 2.5: Unidad de medida

En la figura 2.5 vemos la estructura final de la unidad de medida, compuesta por una unidad de medida inercial IMU y una unidad de control CU. Existen dos unidades de medida, ambas exactamente iguales en cuanto al hardware, y en cuanto a su función, una actúa como sistema de referencia (RMU, del inglés *Reference Measurement Unit*) y otra

como unidad móvil (PMU, del inglés *Position Measurement Unit*), tal y como veremos en el siguiente capítulo 3. Ambas unidades pueden verse en la figura 2.6.

Figura 2.6: Unidades PMU y RMU

Capítulo 3

Estructura Software

3.1. Introducción

En este capítulo se explica el funcionamiento general del programa "viper.exe" ejecutado en la CU de las unidades de medida. Se describirá el funcionamiento del sistema en el arranque y configuración, el modo de comunicación con la IMU, cómo las dos unidades de medida establecen su comunicación UDP y una visión global del papel de ambas unidades dentro del sistema.

El kit de desarrollo de la unidad de control, nos proporciona el software Microsoft eMBedded Visual C++ en su versión 4.00.1610. Sobre éste, se ha realizado un proyecto MFC (Microsoft Foundation Class) usando el lenguaje de programación C++, y posteriormente se han acometido labores de depuración de código.

3.2. Arranque y configuración del sistema

El programa "viper.exe" se encuentra almacenado dentro de la memoria no volatil de ambas unidades de medida, ubicada en la carpeta "FlashDisk". Como se ha mencionado anteriormente, las unidades de medida constan de una CU y una IMU, y aunque ambas unidades de medida sean exactamente iguales en hardware, en funcionamiento se diferencían, una como sistema de referencia y otra como unidad móvil.

Dentro de la CU de cada una de ellas, se ejecuta el sofware diseñado. El programa alojado en ambas es exactamente el mismo, debido a la flexibilidad que esto aporta, y es

el archivo de cofiguración el que define la funcionalidad de cada uno de los sistemas.

Éste archivo, al igual que el programa, se encuentra alojado dentro de la carpeta "Flashdisk" y aporta la información necesaria para definir si se trata de la unidad de medida móvil ó de referencia, así como otros datos para el correcto desarrollo de las funciones a realzar por la unidad. Su utílidad y contenido se detalla con profundidad en el manual de usuario correspondiente al apéndice A.2.2

Al iniciar el software la función *CViperDlg::OnInitDialog()* es la encargada de realizar todas las tareas de arranque e inicialización. Tras la lectura y análisis del fichero de configuración, mostrará la interfaz que corresponda, dependiendo de si estamos en la unidad de medida móvil o en la de referencia.

Además de esto, abre y configura el puerto serie a través del cuál se establece la comunicación entre la IMU y la CU. Para ello utiliza los datos almacenados de la lectura del fichero en la estructura de configuración *datos_sistema*.

El siguiente paso es la creación de un Timer que marcará las lecturas del puerto serie a la frecuencia definida en una constante, llamado *Timer_alm*. Si la creación del Timer no es posible se mostrará un mensaje de error.

El último paso en la inicialización, es la creación y conexión de los sockets encargados de gestionar la futura comunicación UDP entre las dos unidades de medida y la comunicación TCP entre la unidad de medida y el servidor central. Se crean tres sockets, uno para la conexión TCP con el servidor central, y los dos restantes utilizados en la conexión UDP con la otra unidad de medida.

Para la conexión UDP son creados dos sockets, ya que es necesario usar uno para el envío de datagramas y otro distinto para la recepción de los mismos. El sistema mostrará errores si no se consigue crear alguno de los sockets o si no es posible abrir el fichero de configuración o de resultados.

Se adjunta la función *CViperDlg::OnInitDialog()* al completo en el anexo C.1.

3.3. Comunicación con la IMU

3.3.1. Introducción

La comunicación con la IMU se realiza a través del puerto serie de la CU, en el sentido CU-IMU y viceverza. El protocolo que siguen las tramas de datos definido por el fabricante de la IMU es el *Protocolo Qnb Bin*, el cual puede verse con detalle en Ruiz (2008).

Éste protocolo consta básicamente en el envío de tramas en formato binario por parte de la IMU, a una frecuencia constante de 100 Hz. La recepción de tramas por parte de la IMU se realiza en formato cadena de caracteres y se trata solo de peticiones de reinicio de la misma, enviadas por la CU en caso necesario.

Debido a la enorme cantidad de datos recibidos por parte de la IMU es necesario gestionar el puerto serie para que no exista pérdida de datos y manteniendo la mayor robustez posible del sistema. Para ello se implementa una cola circular, con suficiente capacidad como para evitar desbordamientos en condiciones normales.

Para poder gestionar la cola circular en cuanto al almacenamiento y lectura de las tramas, unido a un correcto indexado, se implementa un buffer intermedio entre la fuente de datos y el destino de estos.

El almacenamiento de las tramas provenientes de la IMU en la cola circular y la lectura de éstas, cuando fuera necesario, unido a una actualización correcta de los índices de almacenamiento y recuperación de la cola, es la forma más eficiente de implementar un buffer que haga de almacén intermedio entre la fuente de datos (IMU) y el destino de éstos (función que analiza las tramas).

El envío de tramas presenta muchas menos dificultades, ya que solamente se necesita enviar una trama por el puerto serie hacia la IMU, cuando ésta necesite ser reiniciada. Se encapsulará la trama oportuna en una cadena de caracteres y será enviada carácter a carácter por el puerto serie.

3.3.2. Comunicación en sentido IMU-CU

El protocolo Qnb Bin de la IMU de iXSea impone que se reciban por el puerto serie tramas de 27 bytes, cada una a la frecuencia de 100 Hz, es decir, se recibirán 100 tramas por segundo, lo que hacen un total de 2700 bytes por segundo. Así pues, ésa será la capacidad de lectura que como mínimo deberá tener el programa “viper.exe”.

El flujo constante de tramas que se reciben en la unidad CU se almacenan en primera instancia en una cola circular FIFO que tiene un tamaño de 30000 bytes. Las posiciones de la cola que quedan libres al realizar una extracción de la misma, se reutilizan actualizando adecuadamente los índices de almacenamiento y extracción. La creación de la cola circular y la definición de sus índices de almacenamiento y recuperación puede verse con detalle en Ruiz (2008).

En la recepción de datos por el puerto serie, Windows almacena los datos recibidos en un buffer interno asociado a ese puerto. Como el periodo de recepción es muy bajo puede llegar a desbordarse el buffer interno del puerto serie si no leemos de él lo suficientemente rápido. Es por esto, por lo que una vez creada la cola circular, se debe de crear el Timer que lea periódicamente del buffer interno del puerto serie y almacene en nuestra cola cíclica los datos recibidos.

Hay que tener presente que la frecuencia de almacenamiento no es la misma que la frecuencia a la que llegan las tramas al puerto serie. La frecuencia de almacenamiento se ha definido a 10 Hz, con lo que se trasladarán los datos que haya en el buffer interno del puerto serie hacia la cola circular, 10 veces por segundo.

Se ha elegido ese valor ya que vacía el buffer interno del puerto serie con la suficiente frecuencia, pero sin hacer excesivas transferencias por segundo, con lo que se tendría un innecesario aumento de carga de lectura y escritura.

Una vez que tenemos la cola creada y el Timer corriendo a 10 Hz, cada 100 milisegundos se interrumpirá la ejecución del programa principal para vaciar el contenido del buffer interno de la cola cíclica. De esto se encargará la función *CViperDlg::OnTimer()* cuyo código puede verse en el apéndice C.2.

Cuando el sistema necesite hacer uso de las tramas provenientes de la IMU almacenadas en la cola circular para procesar su información, se extraerán una trama completa de la cola circular, comprobando posteriormente su integridad, todo ello mediante la función *extrae_trama_completa()*. Ésta llama a la función *extrae_trama()* hasta que devuelva un indicador de "trama completa".

La función *extrae_trama()* lee la trama completa almacenada desde la cola cíclica, que es apuntada por el índice de extracción, y además comprueba su integridad haciendo uso del checksum que la acompaña. La función devuelve un número entero indicando el estado de la trama extraída, el cual puede ser:

- Trama correcta: significa que la trama extraída de la cola circular es correcta, es decir, que su checksum coincide con el calculado.

- Trama incorrecta: significa que la trama extraída de la cola circular es errónea, el checksum calculado no coincide con el que acompaña a la trama.

- Trama incompleta: significa que la trama extraída no está completa. Esto puede deberse, a que se haya perdido algún byte en la transmisión o a que la trama esté corrupta. En ese caso, la trama se descarta.

La trama extraída se almacena en la variable global "trama" para que pueda ser usada por el sistema, además antes de finalizar se deben actualizar correctamente los índices de extracción según el número de bytes extraidos de la cola circular, para que en la próxima lectura se eviten pérdidas de tramas correctas.

Cabe destacar, que si se pretende extraer una trama de la cola y ésta se encuentra vacía, la función devolverá "trama incompleta". El código de ambas funciones puede verse dentro del apéndice en C.3.

3.3.3. Comunicación en sentido CU-IMU

La comunicación que parte del microcomputador hacia la unidad de medida inercial, consta del envío de una única trama compuesta por una cadena se caracteres, que provoca el reinicio de la unidad IMU. Cada vez que el sistema necesite reiniciar la IMU se llamará a la función *CViperDlg::resetea_IMU()*.

Dentro de esta función, se escribe sobre una cadena de caracteres de 31 bytes de longitud la trama que el fabricante de la IMU ha facilitado para reiniciar la unidad. Tras esto, solo queda enviar la trama por el puerto serie usando para ello la función *CSerialCom::WriteByte()*. Antes de terminar, se comprueba que el reset ha sido válido mediante la función *CViperDlg::comprueba_reset()*. El código de las funciones puede verse en el apéndice C.4,

3.4. Comunicación UDP entre las 2 unidades de medida

3.4.1. Introducción

Ambas unidades de medida, durante las fases de alineamiento, inicio de trayectoria y fin de trayectoria, generarán un diálogo UDP entre ellas, que será fundamental para la correcta realización de los experimentos y que además, provocará cambios de estado de las dos unidades.

Además de los cambios de estado mencionados, la principal funcionalidad de los mensajes UDP es la de enviar los promedios de los ángulos calculados en la unidad de referencia RMU a la unidad móvil PMU tanto en el momento inicial como final de la trayectoria del experimento.

La unidad móvil será la encargada de comparar las posiciones iniciales y finales de las dos unidades, y con estos datos deberá calcular si el experimento es válido o, si por el contrario el error máximo ha sido superado y por tanto, se ha de repetir la medida.

Para comparar las posiciones, la unidad móvil necesita conocer los ángulos de posición propios y los de la unidad de referencia, es por ello que esta última empaquetará los ángulos de posición adecuados en un mensaje UDP que enviará hacia la unidad móvil para su procesado.

3.4.2. Envío y recepción de mensajes UDP

En el punto inicial de la trayectoria del experimento, la unidad PMU envía a la RMU un paquete UDP ordenándole calcular sus ángulos promedio. Tras un tiempo de espera necesario, la unidad de referencia contestará a la móvil con otro paquete UDP, en el que irán incluidos los ángulos promedios que le fueron pedidos.

El mismo proceso se repite cuando se llega al punto final de la trayectoria. Una vez que la unidad PMU tenga en su poder los ángulos iniciales y finales, estará en disposición de calcular el error existente entre ellos y poder compararlo con el error máximo permitido para el experimento en cuestión.

Antes de enviar los mensajes UDP se escribe la cabecera adecuada y los datos de los ángulos promedios, si fuera necesario, y será entonces cuando se proceda a enviar el mensaje a la otra unidad de medida utilizando la función *CCESocket::Send()*, cuyo código puede verse en D.1. En ella se crea la estructura, se introduce la cabecera y se manda por el socket creado anteriormente para el envío de datagramas.

La recepción de los datagramas se hace de forma asíncrona y automática usando la función *CViperDlg::OnReceive()* a la que se llamará cada vez que se reciba un datagrama por la interfaz Ethernet. El código de esta función puede verse en el apéndice C.5.

Dentro de esta función se leerá el paquete UDP usando la función *CCESocket::Read()* y según sea la cabecera del mismo, se procederá a un cambio de estado del sistema, a la llamada de una función externa o al envío de un nuevo paquete como respuesta al recibido. El código de esta función puede verse en D.2.

Al esperar la recepción de un mensaje UDP, para evitar realizar una espera activa y que el programa quede bloqueado debido a que el sistema no es multitarea, se utiliza la función *procesa_mensajes()* en cada bucle de espera activa, que hace uso de las funciones *PeekMessage()*, *TranslateMessage()* y *DispatchMessage()* del sistema operativo.

Este código puede verse en el apéndice B.16.

La función *PeeKMessage()* busca en la cola de mensajes y si la cola no está vacía devuelve un valor positivo y rellena una estructura del tipo MSG con los datos del mensaje. La función *TranslateMessage()* hace un proceso adicional traduciendo mensajes con teclas virtuales, en mensajes con caracteres.

Por último, la función *DispatchMessage()* se encarga de tomar el mensaje, chequear a qué ventana está destinado y buscar el Window Procedure de dicha ventana. Luego llama a dicho procedimiento enviando como parámetro el handle de la ventana, el mensaje, wparam y lparam.

3.5. Funcionamiento general del sistema

3.5.1. Introducción

En este apartado explicaremos de manera global el funcionamiento del sistema y el papel de cada una de las unidades de medida dentro del proceso de ejecución de los experimentos. La mayor parte del trabajo de cálculo del sistema recae sobre la unidad de medida móvil PMU, siendo las tareas realizadas en la unidad de referencia mucho más ligeras. Para ver con profundidad los pasos a dar para realizar un experimento, dirigirse al apéndice manual de usuario en ejecución paso a paso A.3.

La forma de calcular la desviación total entre las dos unidades de medida, está diseñada de forma que se registra la posición de las dos unidades de medida en un punto inicial, se desplazaba la unidad móvil hasta al punto final y de nuevo se registran las posiciones de las dos unidades. Con estos datos se procede a calcular la matriz de desviación total tal y como se verá en capítulos posteriores.

Las posiciones a medir serán elementos concretos de la aeronave, como por ejemplo el Enhaced Vision System (EVS), el Head-Up Display (HUD) y el Military Radar (MIL). Podrán introducirse tantos elementos a medir en el sistema como se desee. Las posiciones de estos elementos serán medidas respecto a dos posiciones iniciales diferentes denominadas Gadiru 1 y Gadiru 2.

Figura 3.1: Imagen general de los componentes del A400M

3.5.2. Funcionamiento de la unidad de medida móvil PMU

La unidad de medida móvil es la encargada de llevar todo el peso a la hora de realizar el experimento. Se encargará de calcular y almacenar las posiciones iniciales y finales, calcular con ellas el error cometido y comparar el error cometido con el permitido.

Durante el traslado del sistema no se permite superar una velocidad máxima, ni estar durante mucho tiempo a una velocidad alta cercana a la máxima. Además, es necesario contabilizar las vueltas completas que ha dado la unidad de medida en su traslado desde la posición inicial hasta la final, y que el valor de los ángulos pitch y roll de los ejes solidarios de la unidad no excedan unos límites establecidos.

En su viaje de un punto a otro, es posible que el operario haya tenido que rodear obstáculos y podrían haberse realizado una o más vueltas completas sobre el plano horizontal. Para que el operario sea consciente de esta circunstancia, la interfaz del sistema muestra un contador que indica el número de vueltas que ha realizado la unidad de medida desde su punto de partida, ya que al llegar a la posición final y antes de realizar el montaje de la unidad, deberá deshacer las vueltas dadas durante el trayecto. Además, se comprueba que la unidad de medida no viaje a una velocidad cercana a la máxima durante mucho tiempo.

A todas estas tareas habrá que sumarle la continua recepción de tramas en la cola circular y la posterior extracción de éstas tras lo cual se comprueba el checksum de cada una de ellas. Como puede verse la carga computacional que soportará la unidad móvil no es poca, pero el microcomputador proporcionado por Arcom tiene la suficiente capacidad para ejecutar el sistema sin problemas.

3.5.3. Funcionamiento de la unidad de medida de referencia RMU

La unidad de medida de referencia realiza un papel muy sencillo dentro del experimento. El programa "viper.exe" que se ejecuta en la RMU es mucho más simple que la parte ejecutada en la PMU.

La unidad de referencia recibirá paquetes UDP desde la unidad móvil ordenando el reinicio de su IMU, a lo cual responderá con un mensaje UDP de confirmación. Posteriormente la unidad móvil enviará un pulso PPS y acto seguido la orden de calcular los ángulos promedios. Una vez calculados, se devolverán hacia la PMU para que ella los procese.

Este procedimiento se realiza en la posición inicial y final del experimento. Durante el transcurso de la trayectoria de la unidad móvil hasta la posición final, no realiza ningún proceso excepto ir almacenando en la cola cíclica las tramas que recibe de su IMU.

Capítulo 4

Creación de las matrices de rotación de los equipos

4.1. Introducción

Para conseguir una mayor versatilidad del programa se decidió realizar modificaciones en el sistema de cálculo del error del montaje, de modo que fuese fácilmente reconfigurable para su uso en nuevos equipos.

La matriz de rotación teórica de los equipos, esto es, el valor teórico de la rotación del equipo desde el punto de referencia hasta el punto final del montaje del equipo, se encontraba anteriormente dentro del código del programa, siendo tarea solo del programador el acceso a estos datos para realizar cambios o agregar nuevos equipos.

Para solucionar este problema se optó por colocar dentro del archivo de experimentos, el cual es accesible y modificable por el usuario, una parte dedicada a la definición de la matriz de rotación teórica de los equipos. Esto se define mediante una serie de giros alrededor de los ejes 'x', 'y' y 'z', mediante los cuales se obtiene la matriz de rotación teórica.

En este capítulo se detallará el procedimiento para obtener la matriz de rotación teórica de los equipos a partir de los parámetros dados por el usuario, mostrando para ello las partes más relevantes del código, así como las variables y estructuras más importantes. Posteriormente se mostrará el proceso para el cálculo de la desviación total, y finalmente se comprobará la validez de este nuevo método de cálculo mediante una comparativa de los resultados con los entregados por el procedimiento anterior.

4.2. Archivo de experimentos “experimentos.txt”

4.2.1. Introducción

En la unidad de medida móvil PMU se almacena el archivo de experimentos “experimentos.txt” dentro de la memoria no volátil de la unidad de control, ubicada concretamente en el directorio “FlashDisk”. El envío y recepción de este archivo por parte del servidor central se tratará en el apéndice manual de usuario A.3.

4.2.2. Formato del archivo

Se trata de un fichero de texto dividido en 2 partes:

- **Parte de experimentos:** contiene una descripción de cada experimento disponible a realizar, acompañado de unos valores de distintas variables necesarias para seleccionar el equipo de montaje que se está calibrando y las condiciones de trabajo del experimento (velocidades máximas tolerables, tiempo máximo, etc.), que serán monitorizadas en todo momento por el programa. Son las líneas previas a la etiqueta #COMIENZO_EQUIPOS. Para mayor detalle ver el apéndice de manual de usuario A.2.3.

- **Parte de equipos:** contienen la definición de la orientación de cada uno de los equipos de montaje o adaptadores, respecto al equipo GADIRU 1. El sistema ha de conocer esta medida para poder calcular el error de alineamiento. Son las líneas posteriores a la etiqueta #COMIENZO_EQUIPOS.

Cada línea de descripción de equipo consta de un índice para numerar los equipos, el nombre del equipo y una serie de rotaciones en forma de pares de valores numéricos (tantos como se deseen), que definen los giros que llevarían al sistema de medida (RMU) desde la posición GADIRU 1 hasta el punto de anclaje del adaptador, perteneciente al equipo en cuestión.

Dichos giros se pueden descomponer en una serie de pasos, de modo que en cada paso se especifica el ángulo (en grados) que rota la RMU respecto a sus ejes locales. En cada paso, los ejes quedan determinados por el resultado del giro anterior (es decir, siempre hay que referir los giros a ejes solidarios con la RMU). Para mayor detalle de la definición de los giros para la matriz de rotación dentro del archivo de experimentos ver el apéndice manual de usuario A.2.3.

A continuación se muestra un ejemplo para el equipo MIL. La línea del archivo de experimentos sería de la siguiente forma:

```
3;MIL;1;180;2;90;
```

Los giros descritos serían el primero sobre el eje ‘x’ un ángulo de 180 grados, y el segundo sobre el eje ‘y’ un ángulo de 90 grados. Esto se representan en la figura 4.1.

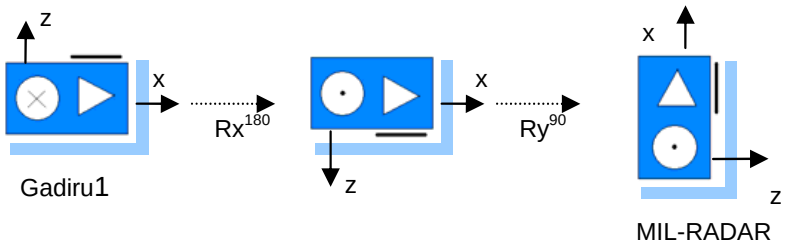

Figura 4.1: Sistema de ejes del movimiento. La banda negra superior representa la pantalla de la RMU, vista lateralmente.

4.2.3. Estructuras formadas en el programa

struct datos_experimento

Contiene la descripción de cada experimento disponible a realizar obtenido de cada línea de descripción de experimento del archivo “experimentos.txt”. Esto se traduce en valores de distintas variables necesarias para seleccionar el equipo de montaje que se está calibrando y las condiciones de trabajo del experimento, que serán monitorizadas en todo momento por el programa.

Para ver la descripción de cada uno de los parámetros de configuración necesarios para cada experimento y el tipo de valores que pueden tener, ver el apéndice manual de usuario A.3.

Campos:

- Índice del experimento
- Nombre del equipo cuyo alineamiento se desea medir
- Posición inicial de la RMU
- Sin uso en esta versión del software
- Tiempo máximo permitido por el experimento
- Velocidad máxima de giro permitido
- Velocidad máxima mantenida de giro

- Máximo número de muestras con velocidad de giro superior a la máxima mantenida
- Sin uso en esta versión del software
- Tamaño del filtro de medidas
- Límite de ángulos pitch y roll soportados

```
struct datos_experimento {
        int indice;
        CString equipo;
        CString gadiru; // GADIRU_L GADIRU_R
        float error_maximo;
        float tiempo_maximo;
        float vel_maxima;
        float vel_alta;
        float num_alto_maximo;
        float error_adaptadores;
        int tamano_filtro_medidas;
        float limite_angulo_vueltas;
};
```

struct datos_equipo

Estructura utilizada para almacenar los datos de los equipos que serán utilizados en los experimentos.

Campos:

- Índice del equipo
- Nombre del equipo
- Rotaciones sobre los ejes (1=X, 2=Y, 3=Z)
- Radianes rotados sobre el eje indicado anteriormente
- Matriz de rotación total del equipo

```
struct datos_equipo {
        int indice;
        CString nombre;
        int eje_rot[MAX_ROT];
        double grad_rot[MAX_ROT];
        double Mat_Rot[3][3];
};
```

4.3. Creación de las matrices de rotación de los equipos

4.3.1. Introducción

Al ejecutar el programa "Viper.exe" en la unidad PMU y pulsar en la ventana del programa el botón "Realización Medida", se abrirá la ventana para la realización del experimento, dando paso a la función *CDialogMedida::OnInitDialog()* B.1. Dentro de esta función se llama a la función *CDialogMedida::lee_experimentos_y_equipos(LPCSTR nombre_fichero)*, la cual se encargará de leer los datos del archivo de experimentos y crear las matrices de rotación.

4.3.2. Matriz de rotación

El código completo de la función *CDialogMedida::lee_experimentos_y_equipos(LPCSTR nombre_fichero)* puede verse en el apéndice B.2. Las operaciones que se realizan dentro de esta función son las siguientes:

- Se lee línea a línea el fichero mediante un while hasta llegar al final del fichero. Dentro del while se compara la línea leída con la cadena separadora "#COMIENZO_EQUIPOS" para determinar si estamos en el proceso de lectura de experimentos o equipos, activando la bandera "segunda_parte" al encontrar esta cadena.

- Si nos encontramos en la primera parte, se llama a la función *CDialogMedida::procesa_experimento(char *linea_leida)*, en la cual se cogerán los datos correspondientes a los experimentos y se colocarán dentro de la estructura datos_experimentos. Al llegar a la segunda parte, se llama a la función *CDialogMedida::procesa_equipo(char *linea_leida)*, en la cual se recogen los datos correspondientes al equipo en cuestión dentro de la estructura datos_equipos.

- Si hemos llegado al final de fichero y no se encuentra activada la bandera "segunda_parte", el programa mostrará un mensaje de error indicando que no se han colocado los datos correspondientes a los equipos dentro del fichero.

El código de ambas funciones, procesa_experimento y procesa_equipo puede verse en el apéndice B.3, B.4.

4.3.3. Proceso de cálculo

Dentro de la función *CDialogMedida::procesa_equipo(char *linea_leida)*, se realiza el cálculo de la matriz de rotación de los equipos. Esta función es llamada desde la función *CDialogMedida::lee_experimentos_y_equipos(LPCSTR nombre_fichero)* tantas veces, como líneas de equipos halla en el fichero, lo cual se contabiliza mediante el contador "contador_equipos".

En primer lugar, pone los valores iniciales de las matrices auxiliares a utilizar a cero. Posteriormente busca la posición del primer punto y coma dentro de la línea que se ha pasado como argumento de la función. Después, copia el contenido de la línea en índice_cadena desde la posición inicial hasta una posición anterior al punto y coma, y convierte esta cadena en un entero (indice_entero_leido). Este primer dato leído es el índice del equipo.

Los datos de los equipos se colocarán dentro de la variable d_equipos, la cual es del tipo struct datos_equipos. Se vuelca el número del equipo hallado anteriormente.

```
d_equipos[contador_equipos].indice=indice_entero_leido
```

Se sitúa el puntero de la línea justo después del punto y coma para poder seguir cargando los datos correspondientes al equipo.

Vuelve a buscar la posición de un nuevo punto y coma dentro de la línea, y copia la línea en la cadena nombre hasta un caracter antes de la posición del punto y coma, obteniendo así el nombre del equipo, el cual es almacenado en la estructura.

```
d_equipos[contador_equipos].nombre=nombre
```

Acto seguido, entra en bucle while para almacenar las duplas de eje y grados de rotación hasta llegar al último punto y coma de la línea. En cada iteración almacena una dupla eje y grados de rotación.

```
d_equipos[contador_equipos].eje_rot[k]=atoi(eje_rot)
d_equipos[contador_equipos].grad_rot[k]=atof(grad_rot)
```

Con cada dupla llama a la función *crea_matriz_rotacion()* para crear la matriz de rotación sobre el eje y con los grados especificados. Dentro de esta función se transforman los grados en radianes y se crea la matriz de la siguiente forma: (Ollero, 2007)

- Si es una rotación de gamma radianes sobre el eje 'x'

$$R(\hat{x},\gamma) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & \sin\gamma \\ 0 & -\sin\gamma & \cos\gamma \end{bmatrix}$$

- Si es una rotación de beta radianes sobre el eje 'y'

$$R(\hat{y},\beta) = \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix}$$

- Si es una rotación de alfa radianes sobre el eje 'z'

$$R(\hat{z},\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Devuelta esta matriz, si se trata de la primera matriz creada, la almacena y vuelve a otra iteración del while para obtener otra dupla eje y grados, en caso de existir más datos en la linea, aumentando el contador "k".

Si por el contrario se trata de la segunda o posteriores iteraciones, se multiplica la matriz recién creada por la matriz almacenada anteriormente, llevando a cabo de esta forma las rotaciones especificadas en la línea de izquierda a derecha para llegar a la matriz de rotación total. La primera rotación a ejecutar es la primera que aparece en la línea.

$$R_{total} = R_n \cdot R_{n-1} \cdots R_2 \cdot R_1$$

Posteriormente, vueleve a otra iteración del bucle while para obtener otra dupla eje y grados en caso de existir más datos, aumentando el contador "k".

Cuando no se encuentre otro punto y coma dentro de la línea (final de la línea) se sale del bucle while, se copia la matriz resultante de todas la iteraciones dentro de la estructura del equipo mediante la función *copiar_matriz(Rot1,d_equipos[contador_equipos].Mat_Rot)*. Se incrementa el contador "contador_equipos" y se vuelve a la función *lee_experimentos_y_equipos* para repetir todo el proceso anterior y almacenar la información de un nuevo equipo en caso de que exista.

El código de las funciones *void copiar_matriz(double origen[3][3], double destino [3][3])* y *void crea_matriz_rotacion(double Rot0[3][3], int eje, double grados)* puede verse en el apéndice B.15.

4.4. Cálculo del error de alineamiento

4.4.1. Introducción

Al inicio de un experimento, conectadas las dos unidades mediante el cable de red y el cable PPS, pulsado el botón "Inicio Medida" de la interfaz de la unidad PMU, se activa la función *CDialogMedida::OnInicioTrayectoria()* en la cual se obtienen los ángulos de Euler de la posición inicial de ambas unidades. El código completo de esta función puede verse en el apéndice B.5.

Al termino de la ejecución del experimento, una vez montada la unidad PMU en su destino y conectada a la unidad RMU mediante el cable de red y el cable PPS, se pulsa el botón "Fin (Conectar PPS!)" el cual activa la función *CDialogMedida::OnFinTrayectoria()*, la cual realiza el proceso de cálculo de la desviación total entre el valor teórico de la posición final de la unidad y el valor real. El código de la función puede verse en el apéndice B.12.

Los ángulos de las posiciones iniciales y finales de ambas unidades, se almacenarán dentro de una estructura del tipo angulos_promedios que está formado por estructuras del tipo angulo.

```
extern struct angulos_promedios iniciales;
extern struct angulos_promedios finales;

struct  angulos_promedios{
        struct angulo orientacion_RMU;
        struct angulo orientacion_PMU;
};

struct  angulo{
        float  heading;
        float  roll;
        float pitch;
        float Q0;
        float Q1;
        float Q2;
        float Q3;
        struct tiempo t;
};
```

4.4.2. Ángulos de Euler de las posiciones iniciales

Al activarse la función *CDialogMedida::OnInicioTrayectoria()*, se vacía el buffer y la cola de las tramas recibidas por parte de la IMU. Posteriormente, se obtiene la última trama completa y se extrae de ésta la hora actual para identificar correctamente el pulso PPS. Recuérdese que en las tramas que envían las IMUs periódicamente está incluido el tiempo en el que se dan los ángulos de posición.

Se envía un pulso PPS durante 2 segundos a la IMU y a la IMU de la unidad de referencia, el cual pondrá a cero el reloj de las dos unidades de medida, quedando sincronizados los dos relojes. Terminado este proceso, se extrae una trama completa, obteniendo la hora de la trama y comparándola con la hora obtenida anteriormente. Si es mayor, se extrae otra trama y se vuelve a comparar hasta encontrar una trama con hora menor. Si es menor, se extrae de la trama los valores correspondientes a los cuaterniones de la posición inicial.

```
iniciales.orientacion_PMU.Q0=viper_dialogo->lee_flotante(&trama[5]);
iniciales.orientacion_PMU.Q1=viper_dialogo->lee_flotante(&trama[9]);
iniciales.orientacion_PMU.Q2=viper_dialogo->lee_flotante(&trama[13]);
iniciales.orientacion_PMU.Q3=viper_dialogo->lee_flotante(&trama[17]);
```

Con estos datos se llama a la función *cuaterniones_a_Euler*, donde se obtienen los ángulos de euler de la posición inicial de la unidad PMU mediante los cuaterniones obtenidos anteriormente.

```
iniciales.orientacion_PMU.heading
iniciales.orientacion_PMU.pitch
iniciales.orientacion_PMU.roll
```

Se guarda también dentro de la estructura la hora obtenida de la trama.

```
iniciales.orientacion_PMU.t=t
```

Posteriormente, ordena a la unidad RMU que calcule su posición inicial y envíe los cuaterniones correspondientes por el puerto UDP. Se vuelcan los ángulos recibidos en la estructura correspondiente a los datos de la RMU.

```
iniciales.orientacion_RMU.Q0=bufferRxUdp.ang_promedio.Q0;
iniciales.orientacion_RMU.Q1=bufferRxUdp.ang_promedio.Q1;
iniciales.orientacion_RMU.Q2=bufferRxUdp.ang_promedio.Q2;
iniciales.orientacion_RMU.Q3=bufferRxUdp.ang_promedio.Q3;
```

Se llama a la función *cuaterniones_a_Euler* para obtener los ángulos de Euler de la posición inicial de la unidad RMU mediante los cuaterniones obtenidos anteriormente.

```
iniciales.orientacion_RMU.heading
iniciales.orientacion_RMU.pitch
iniciales.orientacion_RMU.roll
```

Se almacena también la hora obtenida de la trama proveniente de la conexión UDP entre las unidades.

```
iniciales.orientacion_RMU.t=bufferRxUdp.ang_promedio.t
```

De esta forma contamos con los datos de las posiciones iniciales de ambas unidades, necesarios para el posterior cálculo de la desviación de la posición final.

4.4.3. Ángulos de Euler de las posiciones finales

Al activarse la función *CDialogMedida::OnFinTrayectoria()*, el primer paso que realiza es eliminar el timer "TIMER_CONTROLA_VELOCIDAD", utilizado durante la ejecución del experimento para controlar el proceso del mismo. Esta función se verá con detalle en el siguiente capítulo.

Después de esperar 2 segundos de seguridad, se vacía el buffer y la cola de las tramas recibidas por parte de la IMU de la propia unidad y posteriormente, se extrae una trama completa, obteniendo de esta forma la última trama proveniente de la IMU, de la cual obtenemos la hora actual.

A continuación se envía un pulso PPS a la IMU de la unidad RMU y a la IMU de la propia unidad. Pasado este proceso se entra en un bucle donde se extrae una trama, se obtiene la hora y se compara con la hora obtenida en la trama anterior al pulso PPS. Si es menor, se trata de la primera trama posterior al pulso PPS y sale del bucle, sino vuelve a obtener otra trama y a comparar la hora.

Al salir del bucle obtenemos de la trama los cuaterniones que corresponden a la posición actual de la unidad.

```
finales.orientacion_PMU.Q0=viper_dialogo->lee_flotante(&trama[5]);
finales.orientacion_PMU.Q1=viper_dialogo->lee_flotante(&trama[9]);
finales.orientacion_PMU.Q2=viper_dialogo->lee_flotante(&trama[13]);
finales.orientacion_PMU.Q3=viper_dialogo->lee_flotante(&trama[17]);
```

Con estos datos se llama a la función *cuaterniones_a_Euler*, donde se obtiene los ángulos de Euler de la posición inicial de la unidad PMU mediante los cuaterniones obtenidos anteriormente.

```
finales.orientacion_PMU.heading
finales.orientacion_PMU.pitch
finales.orientacion_PMU.roll
```

Se guarda también dentro de la estructura la hora obtenida de la trama.

```
finales.orientacion_PMU.t=t
```

Posteriormente, ordena a la unidad RMU que calcule su posición final y envíe los cuaterniones correspondientes por el puerto UDP. Se vuelcan los ángulos recibidos en la estructura correspondiente a los datos de la RMU.

```
finales.orientacion_RMU.Q0=bufferRxUdp.ang_promedio.Q0;
finales.orientacion_RMU.Q1=bufferRxUdp.ang_promedio.Q1;
finales.orientacion_RMU.Q2=bufferRxUdp.ang_promedio.Q2;
finales.orientacion_RMU.Q3=bufferRxUdp.ang_promedio.Q3;
```

Se llama a la función *cuaterniones_a_Euler* para obtener los ángulos de Euler de la posición final de la unidad RMU mediante los cuaterniones obtenidos anteriormente.

```
finales.orientacion_RMU.heading
finales.orientacion_RMU.pitch
finales.orientacion_RMU.roll
```

Se almacena también la hora obtenida de la trama proveniente de la conexión UDP entre las unidades.

```
finales.orientacion_RMU.t=bufferRxUdp.ang_promedio.t
```

Obtenidos todos estos datos, se llama a la función *calcular_rotacion_total*, que es donde se realizará el proceso de cálculo del error de desviación total. Espera al término de esta función y almacena los resultados finales en la estructura r_experimento. Se eliminan todos los timers restantes y se vuelca la información de los resultados en el archivo "resultados.csv" alojado en la carpeta /FlashDisk/Startup/.

4.4.4. Proceso de cálculo

En la función *calcular_rotacion_total(angulos_promedios &iniciales, angulos_promedios &finales, double RPY[3])*, se recibe como argumentos las estructuras iniciales y finales correspondientes a ambas unidades y un vector global de orden 3. El código completo de esta función puede verse en el apéndice B.13.

El primer paso que realiza es pasar los ángulos de Euler de las posiciones iniciales y finales de ambas unidades a matrices de rotación, mediante la función *cuaterniones_a_MatrizRotacion*.

```
cuaterniones_a_MatrizRotacion(CRMUi,iniciales.orientacion_RMU);
cuaterniones_a_MatrizRotacion(CPMUi,iniciales.orientacion_PMU);
cuaterniones_a_MatrizRotacion(CRMUf,finales.orientacion_RMU);
cuaterniones_a_MatrizRotacion(CPMUf,finales.orientacion_PMU);
```

Posteriormente llama a la función *calcular_matrices_dependientes_posicion*, cuyo código puede verse en apéndice B.14. Esta función devolverá el valor de la matriz de rotación teórica del equipo respecto a los ejes solidarios del avión y la matriz de rotación teórica respecto a los ejes solidarios del Gadiru_1 ó el Gadiru_2, según sea el caso en el experimento seleccionado.

Debemos recordar que la matriz de rotación teórica introducida mediante el archivo "experimentos.txt" y almacenada en la estructura d_experimentos, es respecto a los ejes solidarios del Gadiru_1.

Veamos el proceso de cálculo de estas dos matrices. En la figura 4.2 se muestra de manera intuitiva el proceso de cálculo de la matriz de rotación teórica del equipo en el punto final del montaje respecto a los ejes solidarios del avión.

El primer paso que se realiza dentro de la función es calcular la matriz de rotación del Gadiru_1 respecto a los ejes solidarios del avión. Esto se realiza mediante la función *crea_matriz_rotacion*, ya que se trata solo de un giro de 180 grados respecto al eje 'x'.

Lo siguiente será buscar dentro de la estructura que contiene los datos de los equipos, la matriz de rotación del equipo sobre el cual estamos realizando el experimento. Esto se hace mediante un bucle while, comparando el nombre del equipo del experimento y el nombre del equipo dentro de la estructura de equipos.

Ahora para poder cambiar el sistema de referencia y obtener la matriz teórica del equipo respecto a los ejes solidarios del avión, multiplicamos la matriz Gadiru_1 respecto al avión por la matriz traspuesta teórica del equipo respecto a Gadiru_1.

$$C_{teorica}^{avion} = C_{G1}^{avion} \cdot C_{teorica}^{G1^T}$$

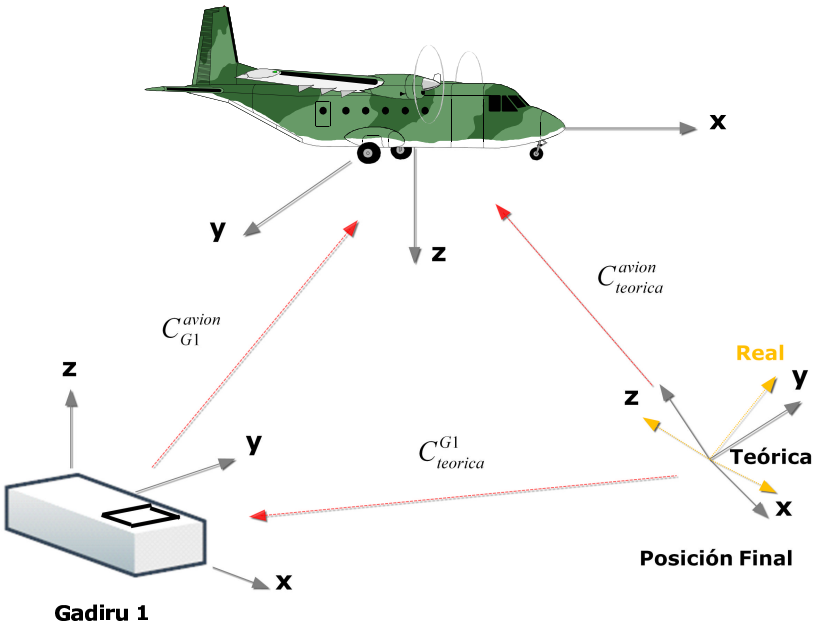

Figura 4.2: Ejes de coordenadas de los equipos y matrices de rotación

El último paso antes de finalizar esta función es mirar si el experimento se realiza respecto al Gadiru_1 ó al Gadiru_2, ya que si es respecto a este último debemos realizar una corrección de un giro de 180 grados sobre el eje 'z' sobre la matriz teórica del equipo para que quede referenciada a los ejes solidarios del Gadiru_2. La función *calcular_matrices_dependientes_posicion* devuelve entonces a la función *calcular_rotacion_total* las matrices CcoordsAvion y CTeorico.

Para encontrar la rotación neta desde la unidad RMU a la unidad PMU, eliminando la desviación en la rotación introducida por la rotación de la tierra, se realiza el siguiente cálculo:

$$C^{RMU}_{PMU} = C^{fin}_{PMU} \cdot C^{inicio^T}_{PMU} C^{inicio}_{RMU} \cdot C^{fin^T}_{RMU}$$

Ahora calculamos la desviación entre la matriz de rotación teórica del equipo y la matriz real:

$$C^{RMU}_{desviacin} = C^{RMU}_{PMU} \cdot C^{T}_{Terico}$$

Multiplicamos ahora la matriz de desviación para que esté referenciada a los ejes solidarios del avión:

$$C^{avion}_{desviacin} = C_{coordsAvion} \cdot C^{RMU}_{desviacin} \cdot C^{T}_{coordsAvion}$$

A partir de esta matriz calculamos los valores de la desviación en ángulos de Euler, mostrando estos valores en miliradianes en un cuadro de texto.

4.5. Comparación con la desviación calculada con el método anterior

Para comprobar que este nuevo proceso de cálculo de las matrices teóricas de los equipos y cálculo de la desviación final no introducía errores significativos debido a la matemática adicional inherente, se realizó una función *diferencia_matrices* para comprobar la diferencia de valores en las matrices respecto al proceso de cálculo que existía anteriormente.

Ejecutando el programa "Viper.exe" mediante la opción de depuración del programa eMBedded Visual C++ e imprimiendo por pantalla los valores de la diferencia entre las matrices calculadas por el nuevo método y el anterior, se obtuvieron los siguientes resultados:

Experimento equipo EVS - Gadiru 1

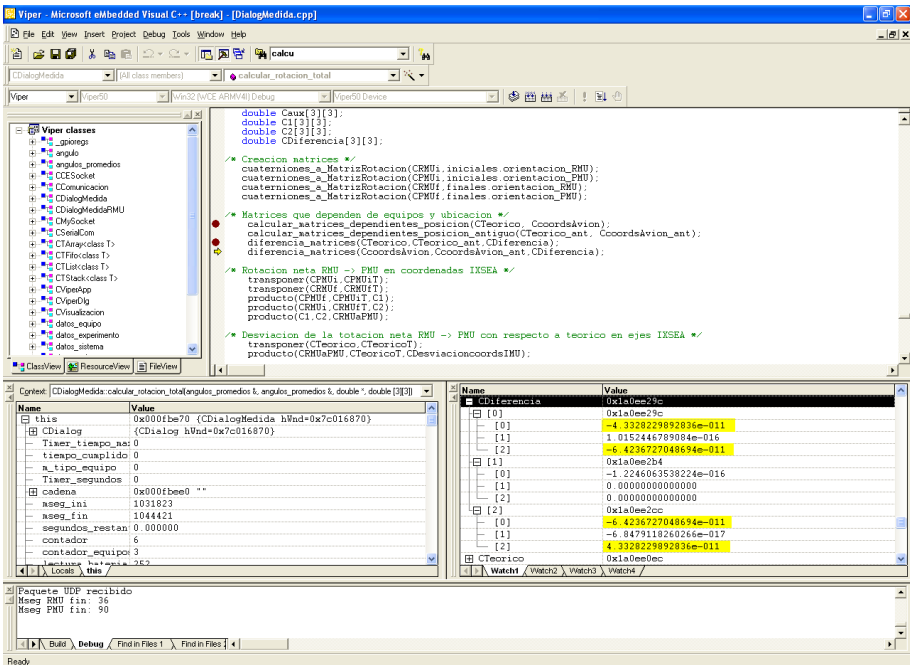

Figura 4.3: Diferencia entre las matrices de rotación teóricas

En el cálculo de la matriz de rotación teórica, que a su vez es el primer paso dentro del cálculo del error, vemos que la diferencia máxima es del orden de exp(-11) 4.3.

Al pasar la matriz de rotación teórica a coordenadas respecto a los ejes solidarios del avión, vemos que la diferencia máxima es del orden de exp(-11) 4.4.

En el cálculo de la desviación entre la matriz teórica y la matriz real, vemos que la diferencia máxima es del orden de exp(-11) 4.5.

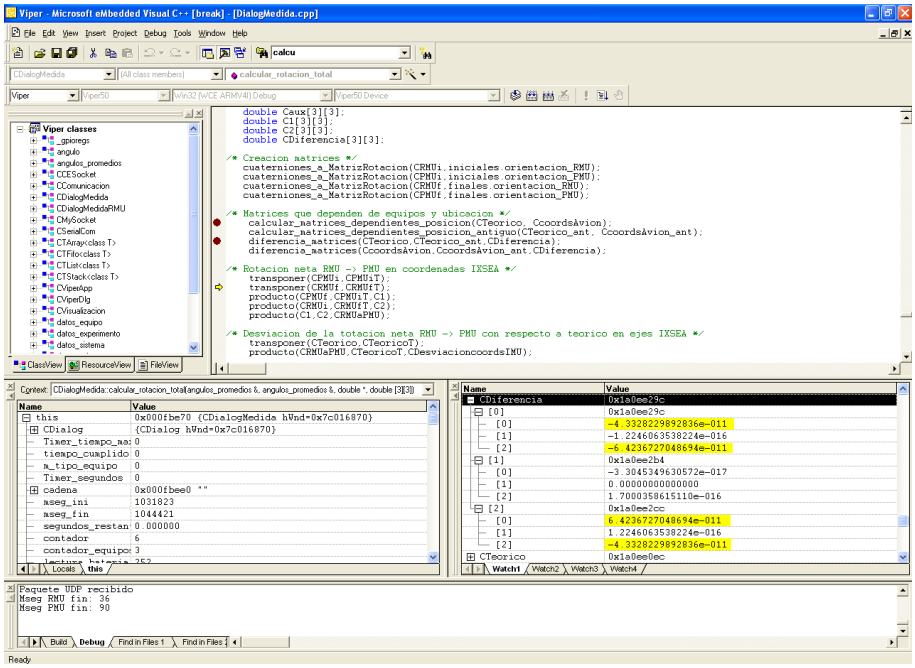

Figura 4.4: Diferencia entre las matrices de rotación teóricas respecto a los ejes solidarios del avión

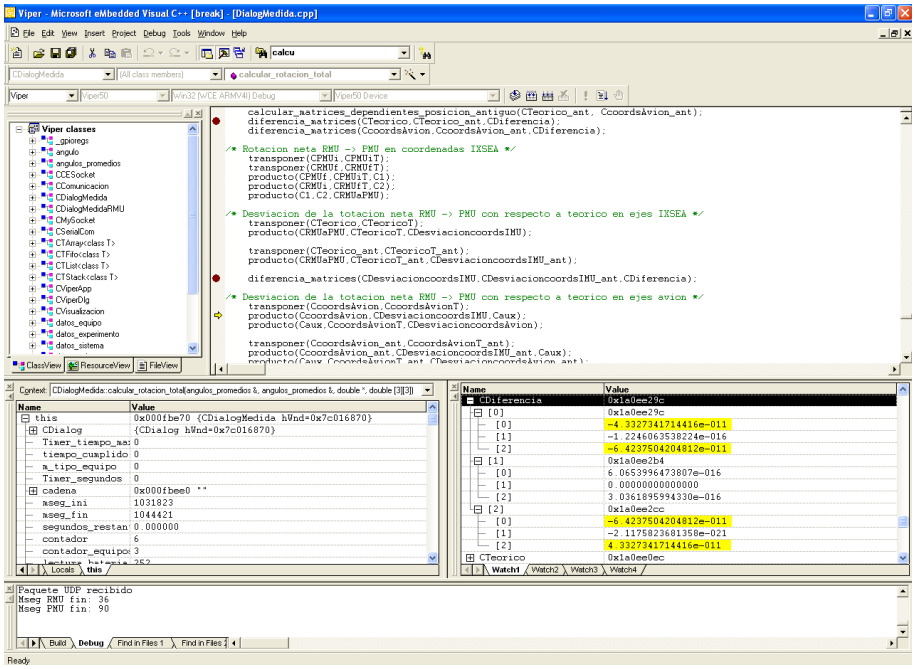

Figura 4.5: Diferencia entre las matrices de desviación respecto a los ejes solidarios de Gadiru_1

Al pasar las matrices de desviación a coordenadas de los ejes solidarios del avión, vemos que la diferencia máxima es del orden de exp(-11) 4.6.

Por tanto, el error máximo introducido en el cálculo de la desviación es del orden de exp(-11), que es mucho menor a la desviación que intentamos medir, la cual debe ser menor a 0.16mrad. Podemos decir entonces, que el error en la medida introducido por las

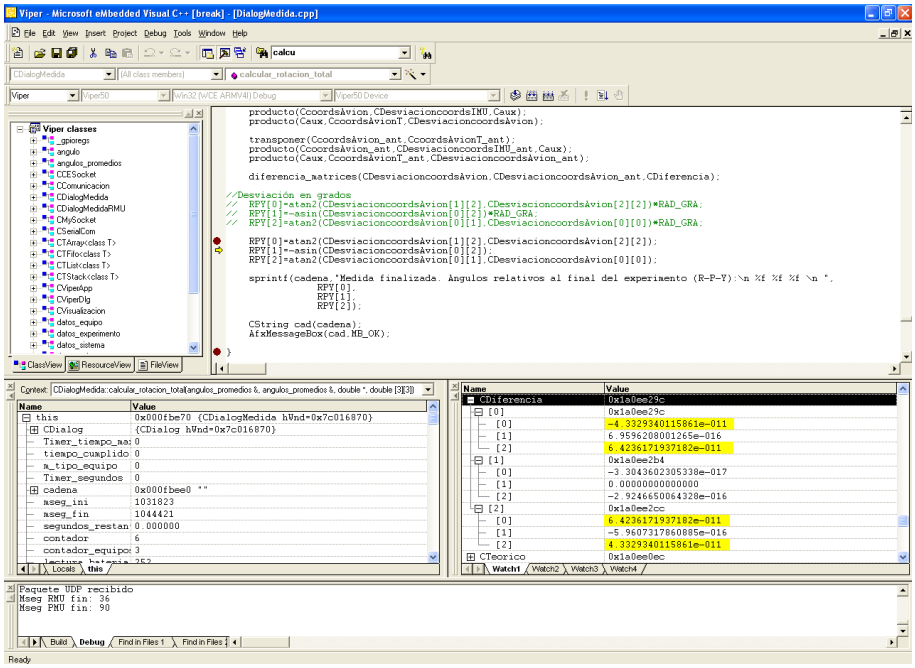

Figura 4.6: Diferencia entre las matrices de desviación respecto a los ejes solidarios del avión

no linealidades de los cálculos del nuevo proceso, no afectan sustancialmente a la medida.

Experimento equipo EVS - Gadiru 2

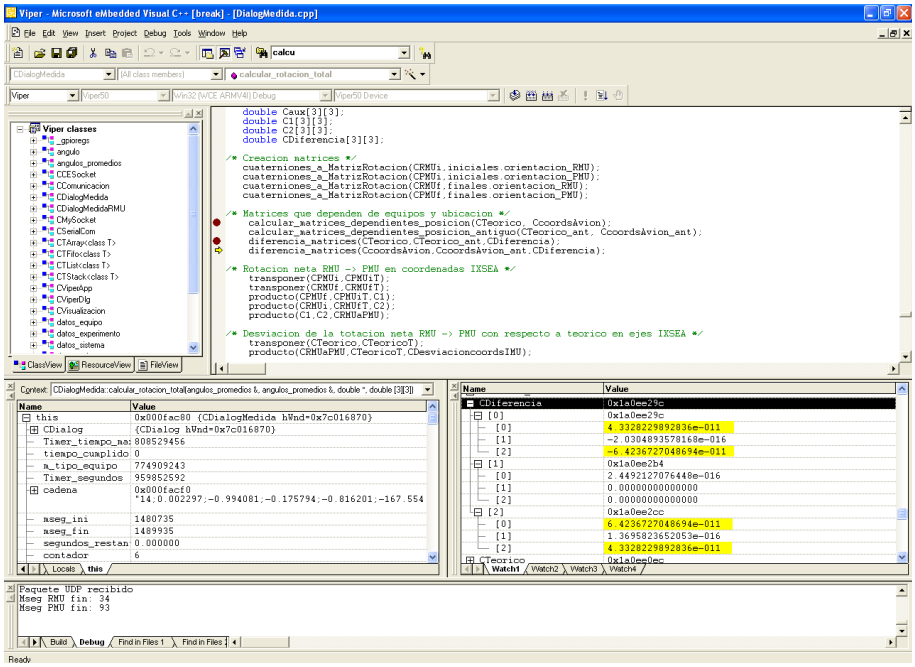

Figura 4.7: Diferencia entre las matrices de rotación teóricas

En la matriz de rotación teórica, tomando en cuenta le rotación extra para referir la

matriz a los ejes solidarios del Gadiru_2, vemos que la diferencia máxima es del orden de exp(-11) 4.7.

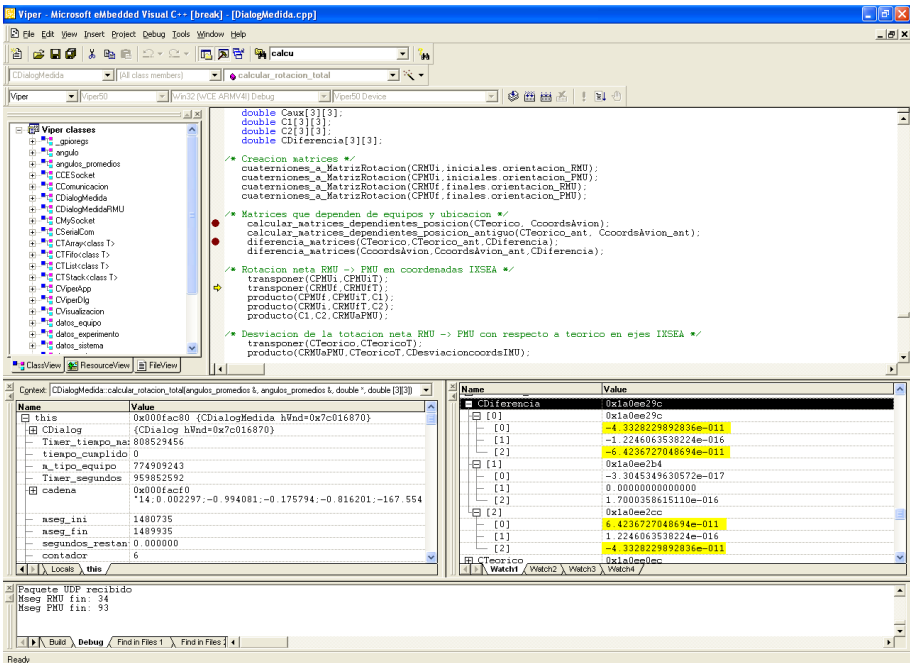

Figura 4.8: Diferencia entre las matrices de rotación teóricas respecto a los ejes solidarios del avión

Al pasar la matriz de rotación teórica a coordenadas respecto a los ejes solidarios del avión vemos que la diferencia máxima es del orden de exp(-11) 4.8.

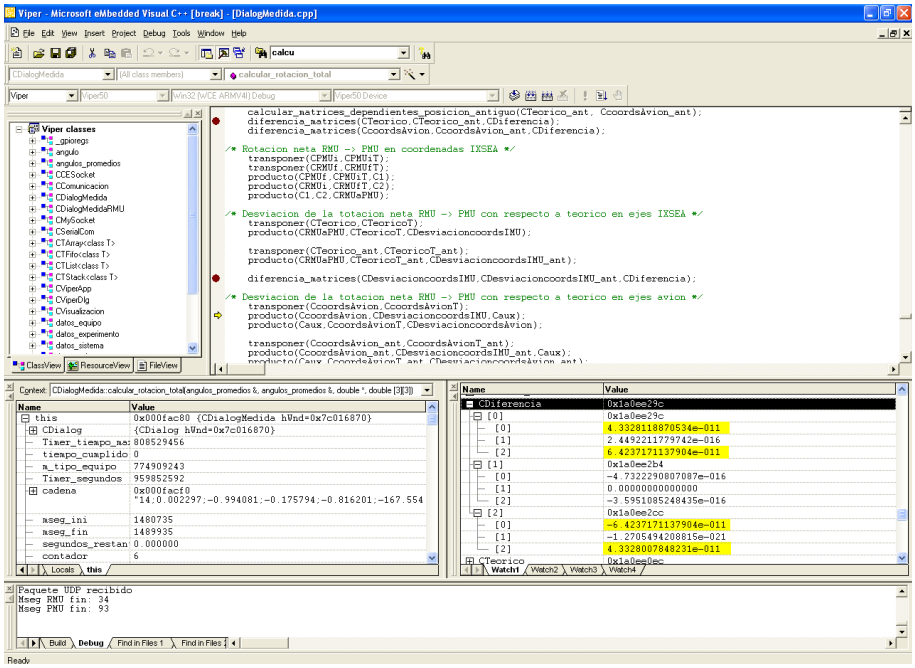

Figura 4.9: Diferencia entre las matrices de desviación respecto a los ejes solidarios de Gadiru_2

En el cálculo de la desviación entre la matriz teórica y la matriz real, vemos que la diferencia máxima es del orden de exp(-11) 4.9.

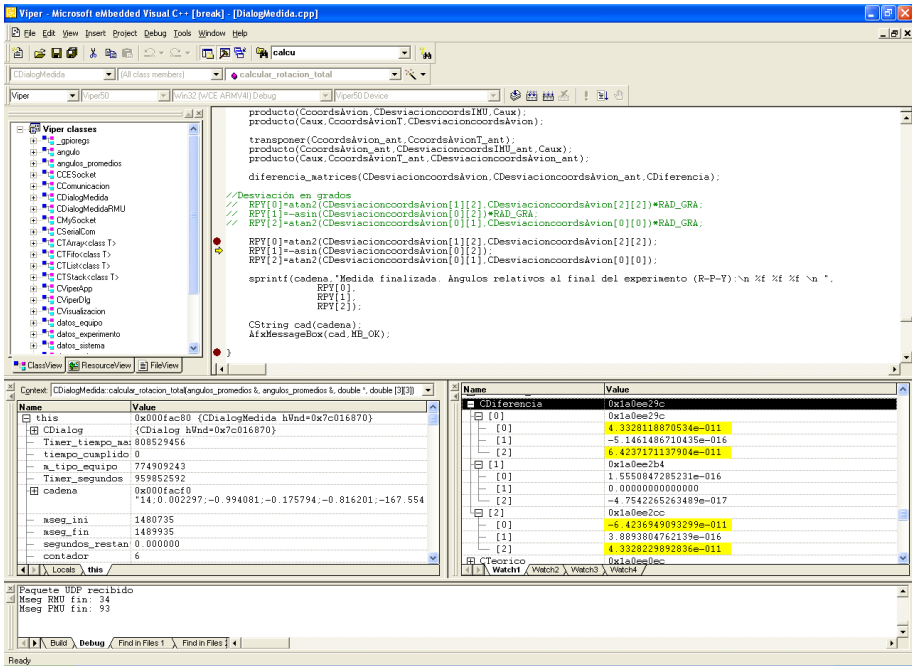

Figura 4.10: Diferencia entre las matrices de desviación respecto a los ejes solidarios del avión

Al pasar las matrices de desviación a coordenadas de los ejes solidarios del avión vemos que la diferencia máxima es del orden de exp(-11) 4.10.

Por tanto el error máximo introducido en el cálculo de la desviación es del orden de exp(-11), que es mucho menor que la desviación que intentamos medir, la cual debe ser menor a 0.16mrad. Podemos decir entonces, que el error en la medida introducido por las no linealidades de los cálculos del nuevo proceso, habiendo en este caso un mayor número de cálculos, no afectan sustancialmente a la medida.

Experimento equipo HUD - Gadiru 1

En la matriz de rotación teórica, que es el primer paso dentro del cálculo del error, vemos que la diferencia máxima es del orden de exp(-16) 4.11. Este es menor que en los experimentos anteriores debido a que se tiene un menor número de rotaciones para formar la matriz de rotación teórica del equipo.

Al pasar la matriz de rotación teórica a coordenadas respecto a los ejes solidarios del avión vemos que la diferencia máxima es del orden de exp(-16) 4.12.

En el cálculo de la desviación entre la matriz teórica y la matriz real, vemos que la diferencia máxima es del orden de exp(-16) 4.13.

Al pasar las matrices de desviación a coordenadas de los ejes solidarios del avión vemos

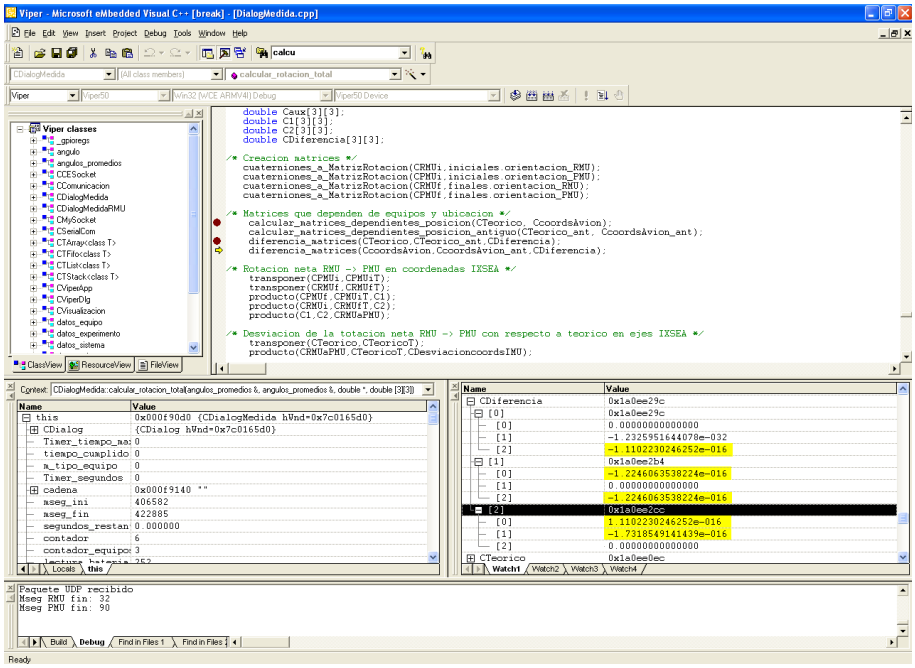

Figura 4.11: Diferencia entre las matrices de rotación teóricas

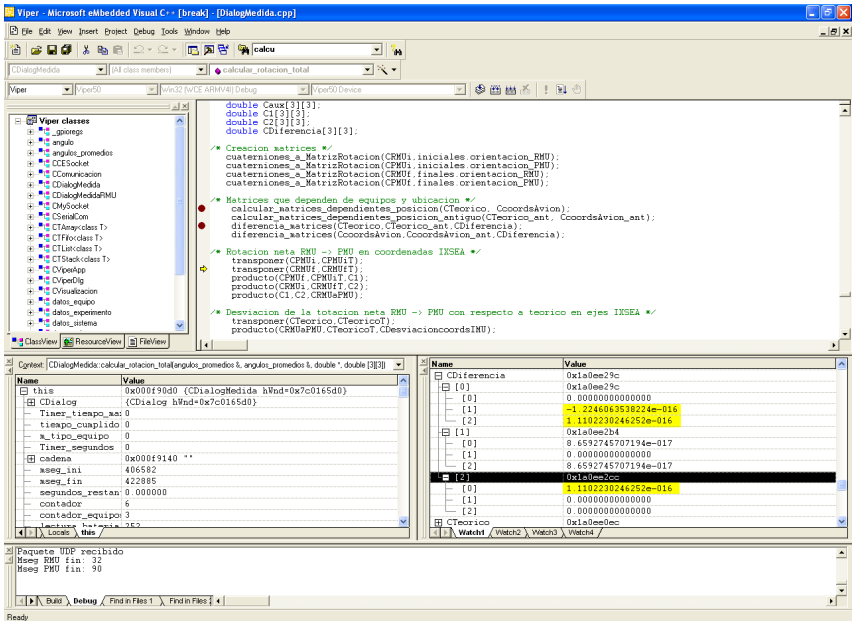

Figura 4.12: Diferencia entre las matrices de rotación teóricas respecto a los ejes solidarios del avión

que la diferencia máxima es del orden de exp(-16) 4.14.

Por tanto el error máximo introducido en el cálculo de la desviación es del orden de exp(-16), que es mucho menor que la desviación que intentamos medir, la cual debe ser menor a 0.16mrad.

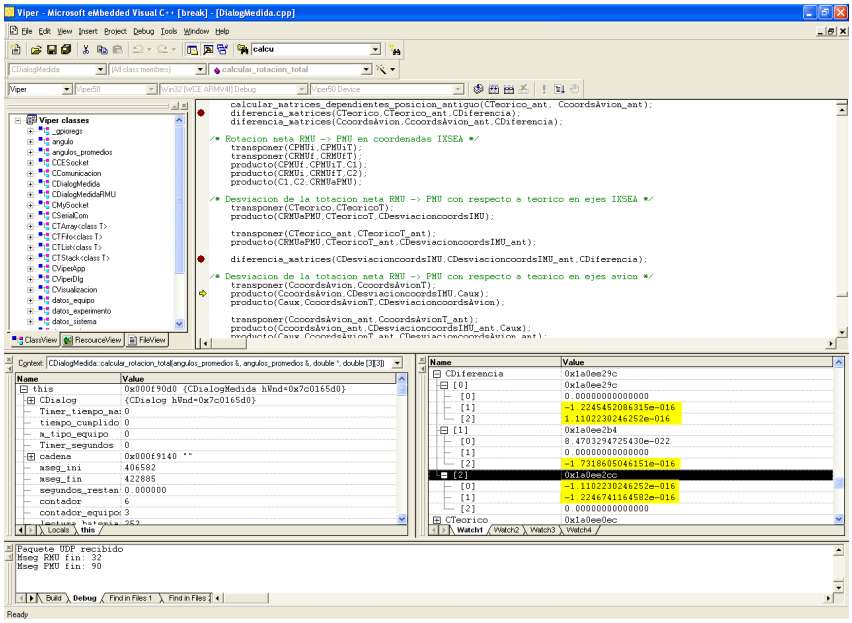

Figura 4.13: Diferencia entre las matrices de desviación respecto a los ejes solidarios de Gadiru_1

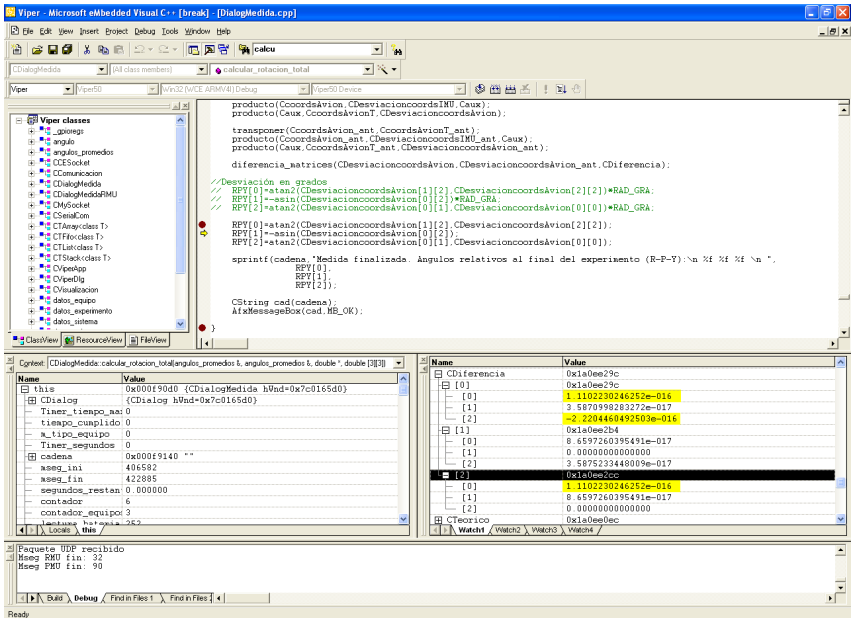

Figura 4.14: Diferencia entre las matrices de desviación respecto a los ejes solidarios del avión

Experimento equipo HUD - Gadiru 2

En la matriz de rotación teórica, tomando en cuenta la rotación extra que supone referenciar el equipo a los ejes solidarios al Gadiru_2, vemos que la diferencia máxima es del orden de exp(-16) 4.15.

Al pasar la matriz de rotación teórica a coordenadas respecto a los ejes solidarios del avión vemos que la diferencia máxima es del orden de exp(-16) 4.16.

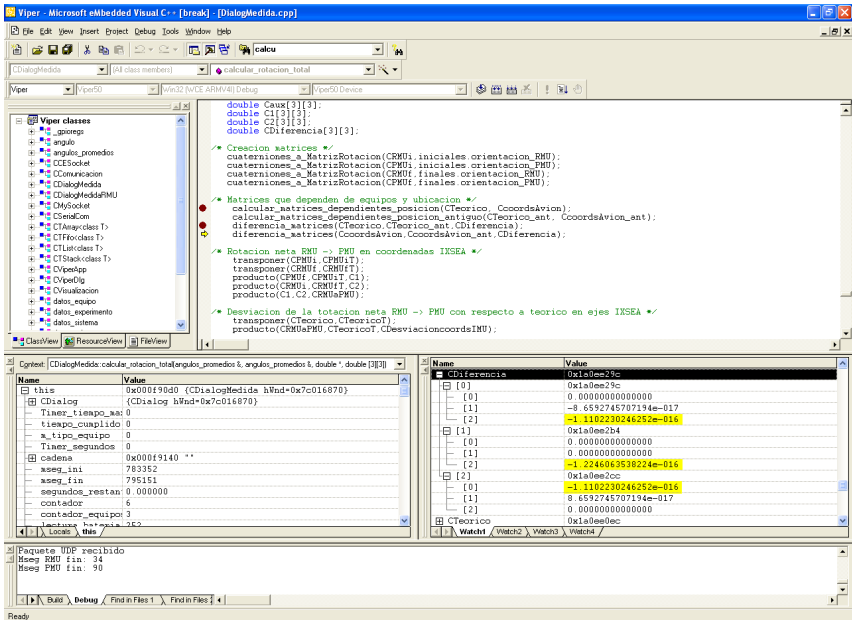

Figura 4.15: Diferencia entre las matrices de rotación teóricas

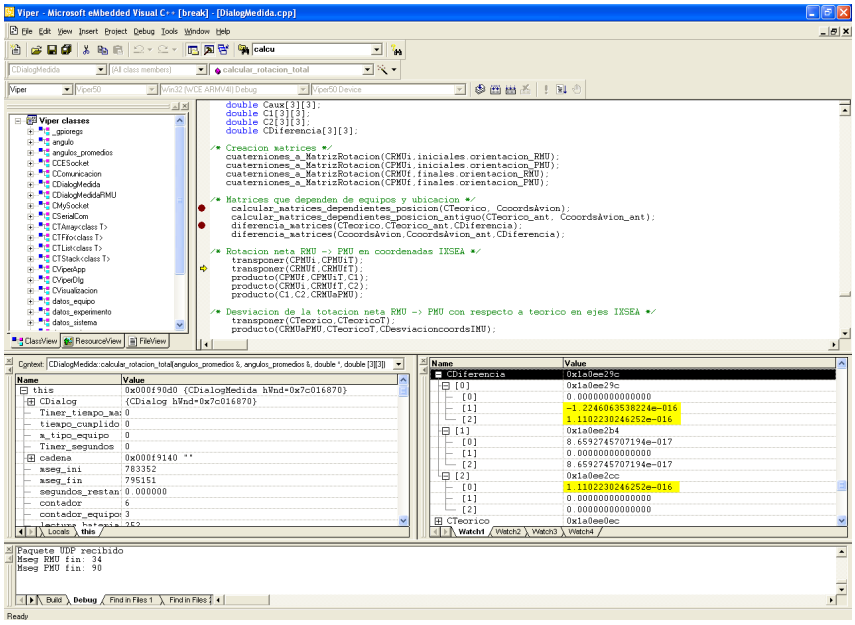

Figura 4.16: Diferencia entre las matrices de rotación teóricas respecto a los ejes solidarios del avión

En el cálculo de la desviación entre la matriz teórica y la matriz real, vemos que la diferencia máxima es del orden de exp(-16) 4.17.

Al pasar las matrices de desviación a coordenadas de los ejes solidarios del avión vemos que la diferencia máxima es del orden de exp(-16) 4.18.

Por tanto el error máximo introducido es del orden de exp(-16) que es mucho menor a la desviación que intentamos medir, la cual debe ser menor a 0.16mrad.

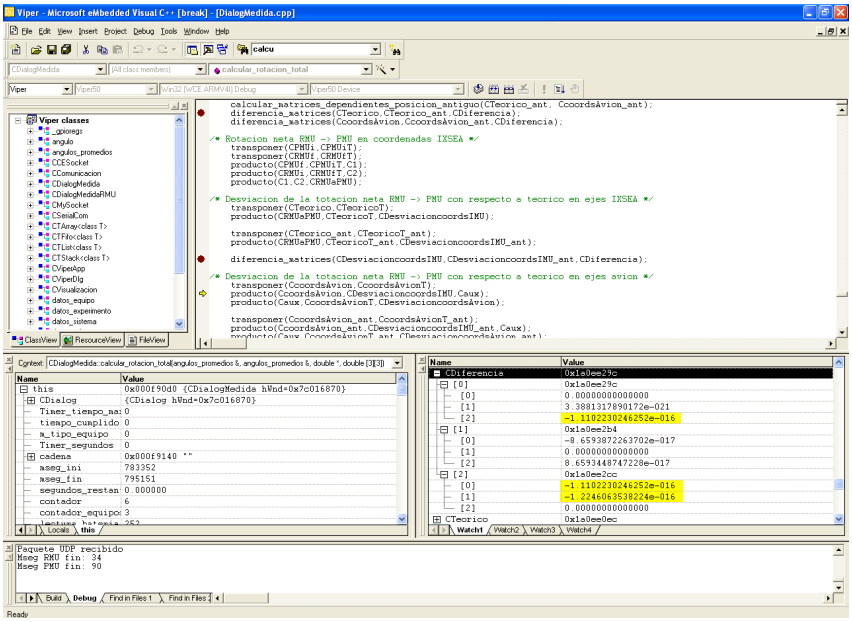

Figura 4.17: Diferencia entre las matrices de desviación respecto a los ejes solidarios de Gadiru_2

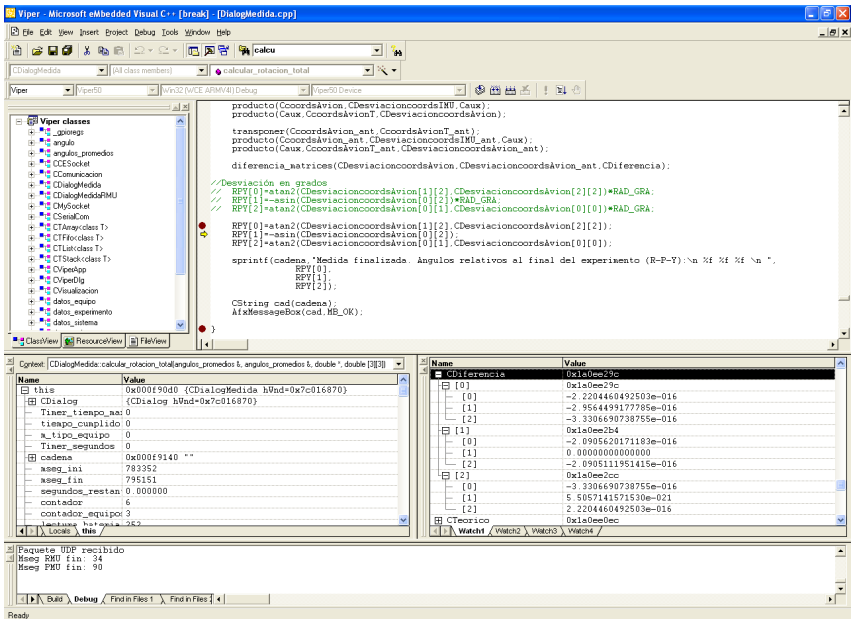

Figura 4.18: Diferencia entre las matrices de desviación respecto a los ejes solidarios del avión

Experimento equipo MIL - Gadiru 1

En la matriz de rotación teórica, que es el primer paso dentro del cálculo del error, vemos que la diferencia máxima es del orden de exp(-16) 4.19.

Al pasar la matriz de rotación teórica a coordenadas respecto a los ejes solidarios del avión vemos que la diferencia máxima es del orden de exp(-16) 4.20.

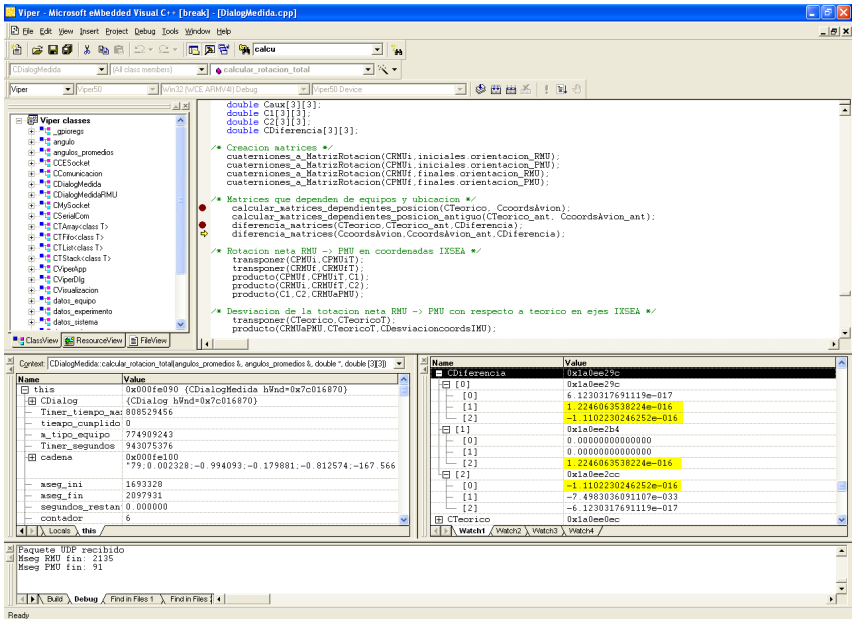

Figura 4.19: Diferencia entre las matrices de rotación teóricas

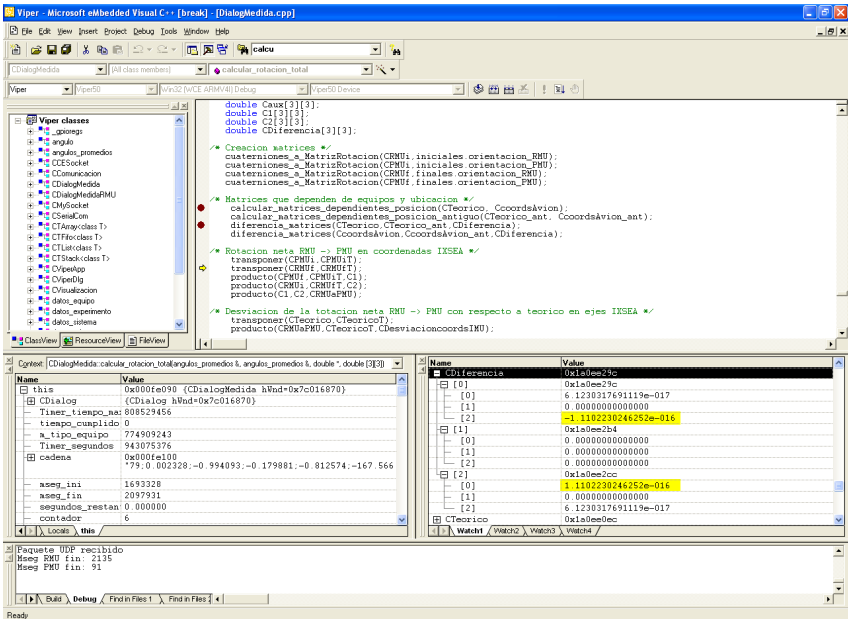

Figura 4.20: Diferencia entre las matrices de rotación teóricas respecto a los ejes solidarios del avión

En el cálculo de la desviación entre la matriz teórica y la matriz real, vemos que la diferencia máxima es del orden de exp(-16) 4.21.

Al pasar las matrices de desviación a coordenadas de los ejes solidarios del avión vemos que la diferencia máxima es del orden de exp(-16) 4.22.

Por tanto el error máximo introducido en el cálculo de la desviación es del orden de exp(-16), que es mucho menor que la desviación que intentamos medir, la cual debe ser menor a 0.16mrad.

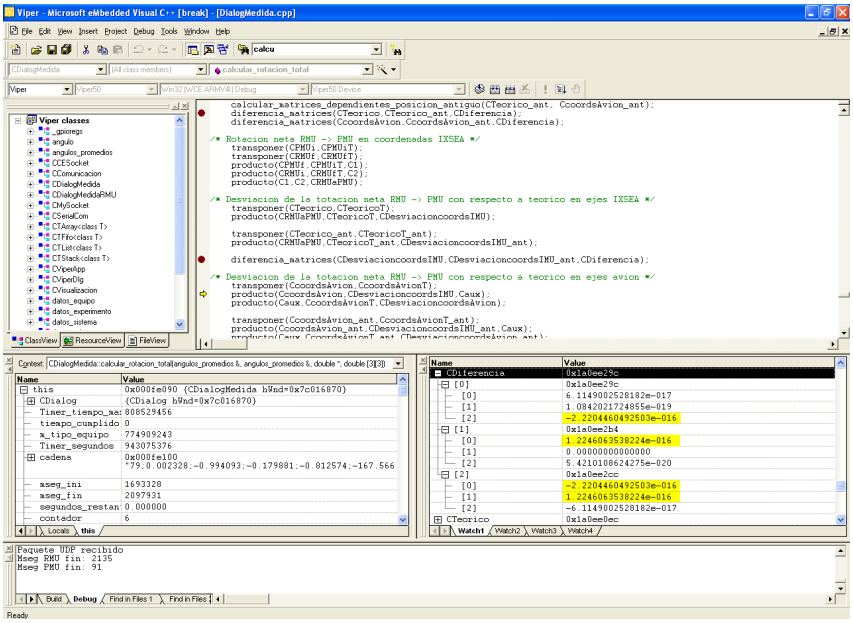

Figura 4.21: Diferencia entre las matrices de desviación respecto a los ejes solidarios de Gadiru_1

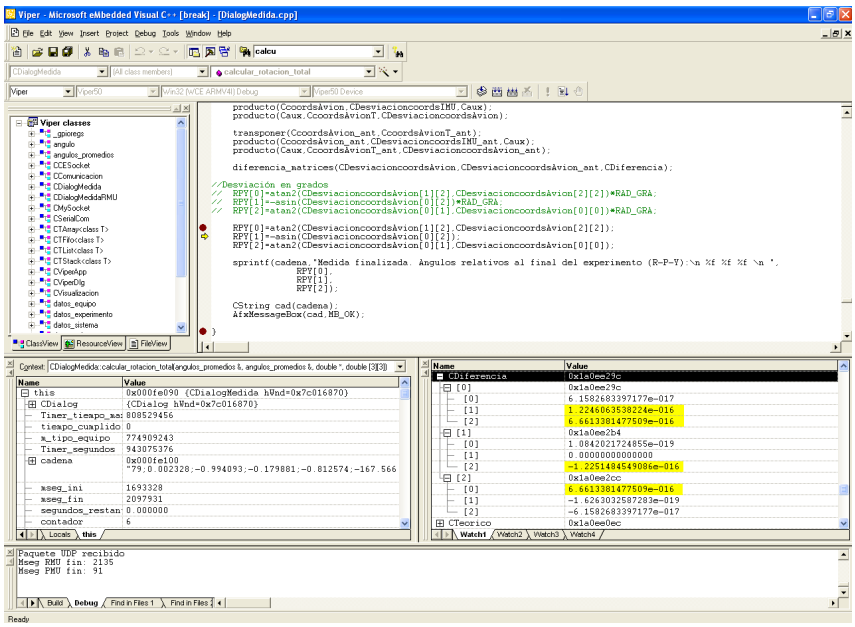

Figura 4.22: Diferencia entre las matrices de desviación respecto a los ejes solidarios del avión

Experimento equipo MIL - Gadiru 2

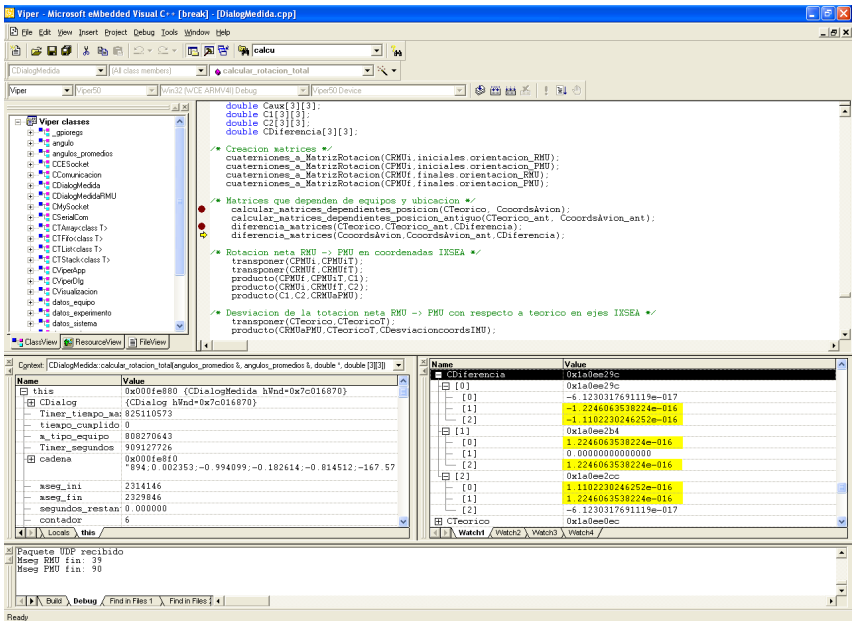

Figura 4.23: Diferencia entre las matrices de rotación teóricas

En la matriz de rotación teórica, que es el primer paso dentro del cálculo del error, vemos que la diferencia máxima es del orden de exp(-16) 4.23.

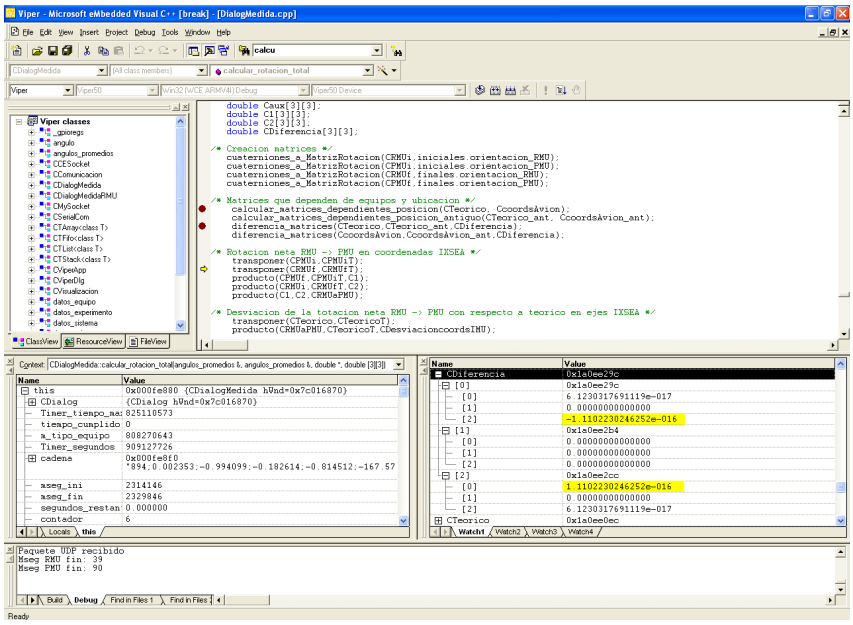

Figura 4.24: Diferencia entre las matrices de rotación teóricas respecto a los ejes solidarios del avión

Al pasar la matriz de rotación teórica a coordenadas respecto a los ejes solidarios del avión vemos que la diferencia máxima es del orden de exp(-16) 4.24.

En el cálculo de la desviación entre la matriz teórica y la matriz real, vemos que la diferencia máxima es del orden de exp(-16) 4.25.

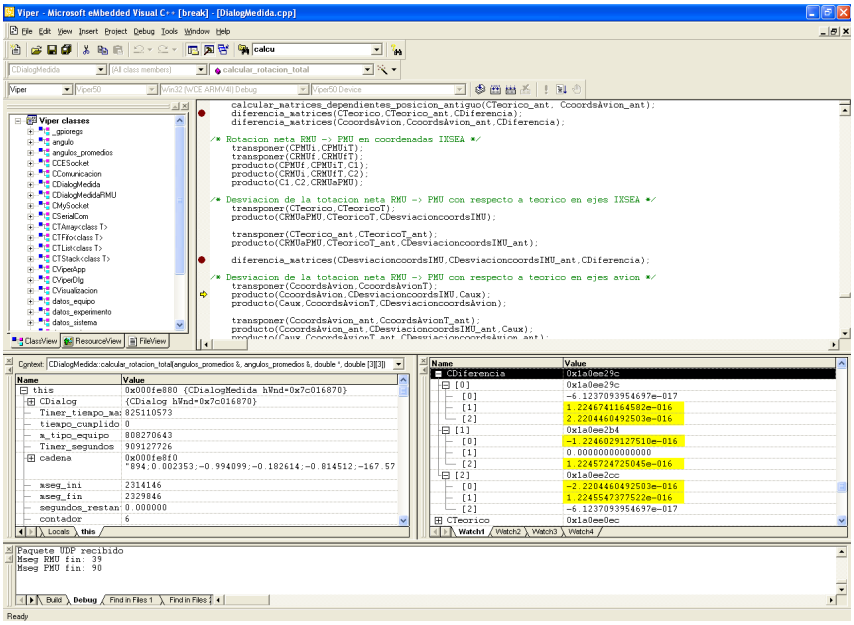

Figura 4.25: Diferencia entre las matrices de desviación respecto a los ejes solidarios de Gadiru_2

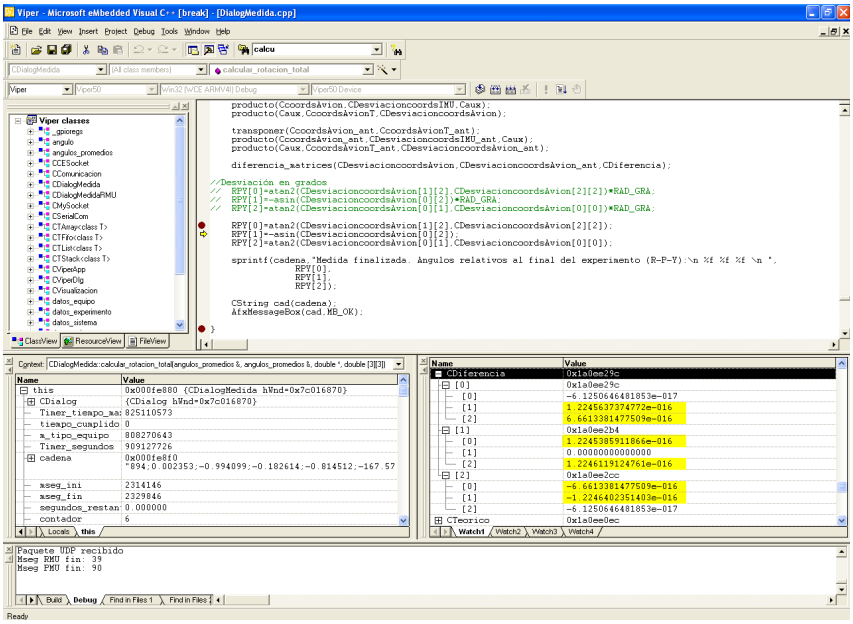

Figura 4.26: Diferencia entre las matrices de desviación respecto a los ejes solidarios del avión

Al pasar las matrices de desviación a coordenadas de los ejes solidarios del avión vemos que la diferencia máxima es del orden de exp(-16) 4.26. El error máximo introducido en el cálculo de la desviación es del orden de exp(-16), que es mucho menor a 0.16mrad.

Por tanto, cuanto mayor es el número de rotaciones necesarias para crear la matriz de rotación teórica del equipo, mayor es el error introducido por las no linealidades de los cálculos, como en el caso del equipo EVS. Sin embargo, es muy pequeño y no afecta a la medida de la desviación.

Capítulo 5

Implementación de nuevas funcionalidades en el programa

5.1. Introducción

En este capítulo se describen las distintas prestaciones que se han implementado en el programa. Primero, se ha introducido un contador de vueltas sobre el ángulo heading para monitorizar la correcta ejecución de la trayectoria de la unidad de medida móvil en un experimento.

Posteriormente, se detallan las correcciones realizadas en el cálculo de la velocidad instantánea respecto a la versión del código anterior. Acto seguido, se describe la colocación de un filtro de medidas configurable por el usuario, para tomar un numero de muestras deseado y calcular su valor promedio, para computar los valores iniciales y finales de las unidades de medida.

A su vez, se detalla la implantación un botón de salida para poder detener la ejecución de un experimento en cualquier momento y volver a la ventana principal del programa. Por último, se describe la introducción de un icono de acceso directo al programa en el escritorio de las unidades CU, para facilitar el lanzamiento del programa.

5.2. Desarrollo de un contador de vueltas del ángulo heading

5.2.1. Introducción

Durante la ejecución de un experimento, al trasladar la unidad de medida PMU desde el punto de inicio al punto final, es posible que el operario haya tenido que rodear obstáculos y podrían haberse realizado una o más vueltas completas sobre el plano horizontal.

Por especificación del fabricante de la unidad IMU, la unidad de medida deberá tener un número de vueltas sobre su propio eje igual a cero antes de realizar el montaje final, donde los ángulos pitch y roll se verán incrementados de un valor anterior aproximado de cero.

Por tanto, es necesario contabilizar las vueltas completas que ha dado la unidad de medida en su traslado desde la posición inicial hasta la final, y monitorizar que el valor de los ángulos pitch y roll de los ejes solidarios de la unidad no excedan unos límites establecidos.

Para que el operario sea consciente de esta circunstancia, la interfaz del sistema muestra un contador que indica el número de vueltas que ha realizado la unidad de medida desde su punto de partida, ya que al llegar a la posición final y antes de realizar el montaje de la unidad, deberá deshacer las vueltas dadas durante el trayecto. A su vez, se indica mediante un cuadro de texto de información el estado actual de la trayectoria del sistema.

5.2.2. Ángulo heading y corrección de salto

El ángulo de heading devuelto por la IMU está dentro del rango [-π,π] y es de la forma de la figura 5.1.

Al calcular el incremento del heading, hacemos la resta entre el ángulo actual y el anterior. Si al calcular el incremento se da un cambio de signo entre el ángulo actual y el anterior, el incremento calculado será erróneo. Por ejemplo:

- En sentido horario:

$$h_0 = 140°, h_1 = -160°$$
$$\Delta h = h_1 - h_0 = -300°$$

  el incremento correcto es de $\Delta h = 60°$

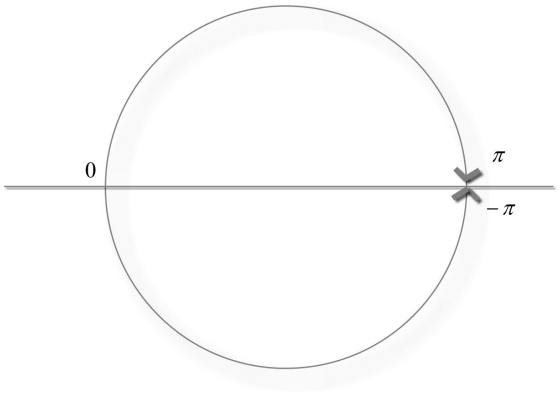

Figura 5.1: Rango del ángulo de heading

- En sentido antihorario:
$$h_0 = -155°, h_1 = 170°$$
$$\Delta h = h_1 - h_0 = -325°$$
el incremento correcto es de $\Delta h = -35°$

Lo que debemos hacer es corregir el desfase que existe en el rango, sumando 2π al ángulo negativo y posteriormente realizar la resta:

- En sentido horario:

$$h_0 = 140°, h_1 = -160° \rightarrow h_1' = 360° - 160° = 200°$$
$$\Delta h = h_1 - h_0 = 60°$$

- En sentido antihorario:

$$h_0 = -155°, h_1 = 170° \rightarrow h_0' = 360° - 155° = 205°$$
$$\Delta h = h_1 - h_0 = -35°$$

Ahora bien, para darnos cuenta de que existe un problema al calcular el incremento debido al cambio de signo, y realizar el correspondiente ajuste, comprobamos que el incremento, perteneciente al rango [-2π,2π], no exceda en valor absoluto un cierto porcentaje de 2π. Este porcentaje tendrá un compromiso con el salto real más grande que podamos computar.

Veamos el razonamiento mediante un ejemplo mostrado en la figura 5.2:

- Salto permitido $<\pi$:
Tomamos por ejemplo un salto máximo permitido de 150°. Realizamos un salto en sentido antihorario (color amarillo):

$$h_0 = 70°, h_1 = -90° \rightarrow \Delta h = h_1 - h_0 = -160°$$

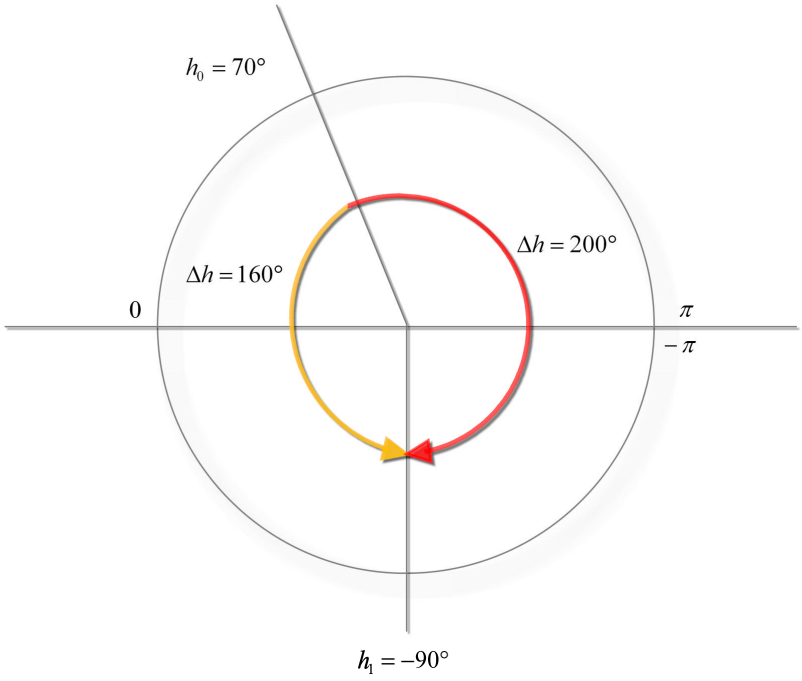

Figura 5.2: Posibilidad de giro horario o antihorario

incremento correcto pero cumple la restricción y se realiza la compensación:

$$h_1' = 360° - 90° = 270°$$
$$\Delta h = h_1 - h_0 = 200°$$

- Salto permitido $>\pi$:
  Tomamos por ejemplo un salto máximo permitido de 190°. Realizamos un salto en sentido horario (color rojo):

$$h_0 = 70°, h_1 = -90° \rightarrow \Delta h = h_1 - h_0 = -160°$$

  incremento erróneo y no se compensa porque no se cumple la restricción.

Por tanto, el valor óptimo para realizar este ajuste es de π, ya que tomando un ángulo menor que este, dejaría de ser distinguible si se trata de una vuelta en sentido horario o antihotario, y escogiendo un ángulo mayor, existirían demasiados errores que no se tomarían en cuenta

Debido a que el rango del ángulo de heading es de [-π,π], si el incremento del ángulo devuelve en valor absoluto un valor mayor que π, sabemos que alguno de los dos ángulos debe ser de signo contrario al otro, por lo cual debemos realizar una corrección del desfase al ángulo negativo.

A efectos prácticos, durante la realización de la trayectoria, esto se traducirá en que entre 2 tramas consecutivas sobre las cuales calculemos el incremento, no debe haber una

diferencia mayor a π o no se compensará el error.

Debido a que la frecuencia de almacenamiento de tramas procedentes de la IMU se realiza con una frecuencia de 10 Hz (0.1 seg), tenemos un factor limitante para calcular el incremento entre 2 tramas seguidas, ya que éste es el tiempo mínimo que existe para obtener una trama y procesarla.

Esto se traduce en que, tomando la frecuencia de cálculo del incremento igual a la frecuencia de almacenamiento, la diferencia entre 2 tramas consecutivas es, para las primeras 2 tramas de 0.2 segundos, y en las posteriores de 0.1 segundos, ya que compara la trama actual con la anterior ya computada. Durante este tiempo entre trama y trama, no puede existir un incremento del ángulo de heading mayor que π, o éste será computado de manera errónea.

Cabe destacar que las vueltas realizadas en sentido horario se muestran con un valor positivo, mientras que las vueltas en sentido antihorario se muestran con un valor negativo.

5.2.3. Diagrama de estados

El diagrama de estados implementado se muestra en la figura 5.3. Como podemos ver, se trata de 4 estados dependientes del número de vueltas sobre el eje de la unidad, y del valor de los ángulos pitch y roll.

El primer estado corresponde al estado "VUELTAS_OK_ANGULO_OK", en el cual permanecemos mientras el valor absoluto del número entero de vueltas sea igual a cero, y el valor de los ángulos pitch y roll sea menor al valor especificado en el experimento y que se encuentra almacenado en la variable d_experimento.limite_angulo_vueltas. Se muestra por pantalla en el cuadro de información el mensaje "Trayectoria correcta...".

Si el valor absoluto del contador de vueltas se vuelve mayor o igual que uno, se pasa al segundo estado correspondiente a "VUELTAS_KO_ANGULO_OK" y se muestra el mensaje "Nº vueltas no nulo: deshacer vueltas.". Si el valor absoluto del contador de vueltas vuelve a cero, se vuelve al estado uno correspondiente a "VUELTAS_OK_ANGULO_OK", y si por el contrario, el contador entero de vueltas permanece distinto de cero y el límite permitido de los ángulos pitch y roll se excede, pasamos al tercer estado.

En el tercer estado, correspondiente a "VUELTAS_KO_ANGULO_KO", se activa la función *CDialogMedida::cancela_trayectoria()* la cual realiza los pasos necesarios para cancelar el experimento actual, ya que se ha realizado la trayectoria de manera incorrecta y deberá realizarse de nuevo desde el principio. A su vez, se muestra un cuadro de diálogo con el mensaje "Exceso de vueltas. Reinicie.".

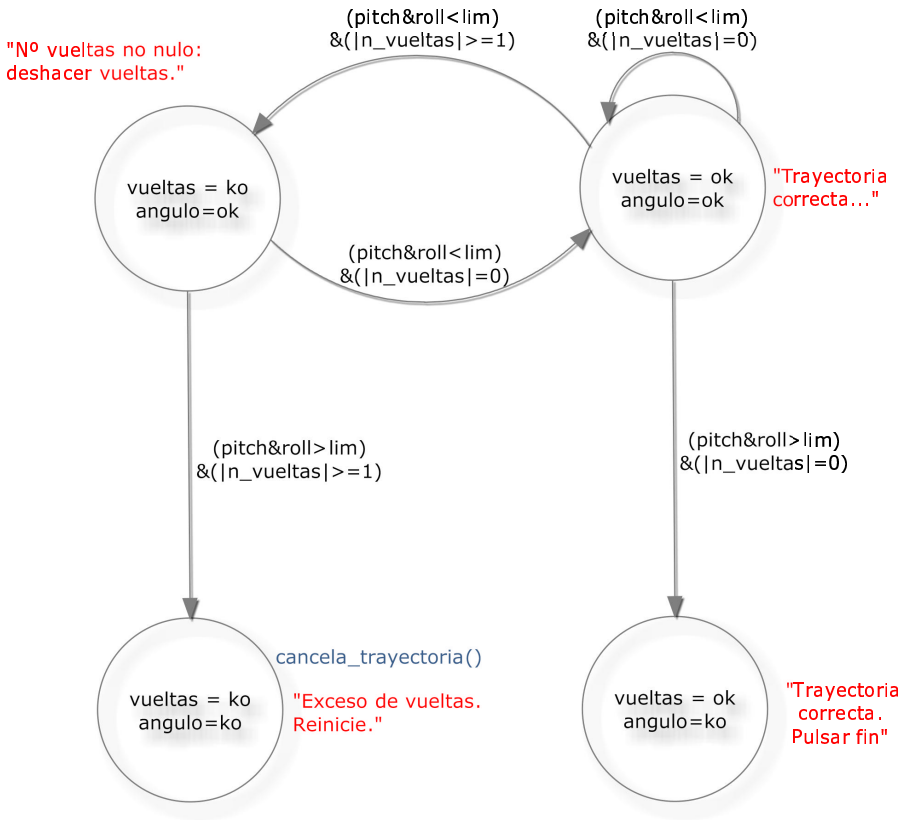

Figura 5.3: Diagrama de estados del contador de vueltas

Si en el primer estado correspondiente a "VUELTAS_OK_ANGULO_OK", esto es, cuando el valor absoluto del número entero de vueltas es cero y los ángulos pitch y roll dentro del límite, se excede el valor límite de los ángulos, se pasa al cuarto estado que corresponde con "VUELTAS_OK_ANGULO_KO". Éste se toma como la posición final del experimento y se muestra el mensaje "Trayectoria correcta. Pulsar fin".

Veamos ahora la implementación del diagrama de estados del contador de vueltas dentro del programa.

5.2.4. Implementación del contador

Al iniciar un experimento y activarse la función *CDialogMedida::OnInicioTrayectoria()*, después de realizarse el procedimiento para obtener la posición inicial de las unidades visto en el capítulo anterior, se pone el estado del contador de vueltas a estado_vueltas=VUELTAS_OK_ANGULO_OK y se crea un timer llamado "TIMER_CONTROLA_VELOCIDAD", para llamar periódicamente a la función *CDialogMedida::controla_velocidad()* con una frecuencia 10 Hz, esto es cada 0.1 segundos.

```
result=SetTimer(TIMER_CONTROLA_VELOCIDAD,(1/FRECUENCIA_CONTROLA)*1000,0);
```

Al iniciarse la trayectoria de la unidad PMU, cada vez que se active el timer se llama a la funcion *CDialogMedida::OnTimer()*, en la cual se identifica el evento con el timer "TIMER_CONTROLA_VELOCIDAD" y llama a la función *CDialogMedida::controla _velocidad()*. Esté código puede verse en el apéndice B.8.

Dentro de la función *CDialogMedida::controla _velocidad()*, cuyo código completo puede verse en el apéndice B.9, se calcula la velocidad instantánea a partir de 2 tramas consecutivas, acto seguido se estima el incremento angular acumulado en el último periodo de muestreo actualizando el estado del sistema, y finalmente se comprueban los límites de velocidad.

El código referente al cálculo de la velocidad instantánea y las correcciones realizadas en éste respecto a la versión anterior, se detallan en el siguiente apartado 5.3. En adelante, nos centraremos únicamente en el procedimiento del contador de vueltas.

Como primer paso, se extrae una trama completa del buffer de datos, obteniendo la hora y los ángulos de Euler actuales, los cuales se almacenan en la variable mseg_k1 y en el vector angulo_k1. Se comprueba si no existe una medida anterior, y si es el caso se pone a cero el incremento angular acumulado, se actualiza el valor de los ángulos anteriores y se sale de la función.

```
// Solo se ejecuta 1 vez para tener un valor antiguo de angulo_k0
if(!existe_medida_anterior)
{
        incremento_angular_acumulado=0;
        existe_medida_anterior=true;
        angulo_k0=angulo_k1; // Actualizo la medida anterior
        mseg_k0=mseg_k1;
        t_k0=t_k1;
        return;
}
```

Si se trata de una medida posterior, se calcula el incremento de los ángulos de Euler llamando a la función *CDialogMedida::calcula_incremento_modulo()* cuyo código completo puede verse en B.10.

```
        incremento_h=calcula_incremento_modulo(angulo_k1.heading,angulo_k0.heading,-M_PI,M_PI);
        incremento_r=calcula_incremento_modulo(angulo_k1.roll,angulo_k0.roll,-M_PI,M_PI);
        incremento_p=calcula_incremento_modulo(angulo_k1.pitch,angulo_k0.pitch,-M_PI/2.0,M_PI/2.0);
```

Dentro de la función *calcula_incremento_modulo()* calculamos el incremento angular tomando en cuenta la corrección necesaria expuesta en secciones anteriores. Entregamos como argumento para cada ángulo el valor del ángulo actual angulo_k1, el valor del ángulo anterior angulo_k0 y el valor mínimo y máximo de dicho ángulo.

```
        salto=maximo-minimo;
        // Calculamos el incremento angular realizando un ajuste si el modulo del mismo
        nos hace pensar que se ha producido desborde
        if(abs(theta1-theta0)>(salto*0.5))
        {
                if(theta1>theta0)
                        {theta0+=salto;}
                else theta1+=salto;
        }
        return (theta1-theta0);
```

Un vez obtenido el valor del incremento para cada ángulo, se actualiza la medida anterior haciendo angulo_k0=angulo_k1 y mseg_k0=mseg_k1. De este modo se obtienen los ángulos centrados en el ángulo inicial y no sometidos a la operación modulo. Posteriormente se calcula el incremento angular acumulado del ángulo heading y se obtiene el valor del contador de vueltas para imprimirlo por pantalla.

```
incremento_angular_acumulado+=incremento_h;

if(incremento_angular_acumulado>0)
{
        contador_vueltas=floor(10*(incremento_angular_acumulado)/(2*M_PI))/10;
}
else contador_vueltas=ceil(10*(incremento_angular_acumulado)/(2*M_PI))/10;

sprintf(cadena,"%.1f",contador_vueltas);
m_vueltas=cadena;
```

El contador se mostrará como un número con una cifra decimal, para que de esta forma el operador, antes de realizar el montaje final, coloque el contador lo mas cercano posible al valor óptimo de 0.0, ya que una medida de 0.9 sin tomar en cuenta ninguna décima mostraría un número de vueltas completas igual a cero, sin ser esta la posición optima a la que debería colocarse el equipo para el montaje final.

El procedimiento para obtener una cifra decimal es el siguiente. Si el valor del incremento es positivo, esto es, se trata de vueltas dadas en sentido horario, se multiplica el valor del incremento por diez y se divide entre 2π, después mediante la función *floor()* se obtiene el valor entero por debajo y se divide entre 10.

Si se trata de un valor negativo del incremento, esto es, se trata de vueltas dadas en sentido antihorario, se multiplica el valor del incremento por diez y se divide entre 2π, después mediante la función *ceil()* se obtiene el valor entero por arriba y se divide entre 10.

Posteriormente, se comprueba el estado de sistema. Si el valor absoluto de los ángulos pitch y roll está dentro del límite y el valor absoluto del contador de vueltas es menor a

1.0, significa que estamos en el estado uno del diagrama mostrado en el apartado anterior. Esto corresponde a estado_vueltas=VUELTAS_OK_ANGULO_OK.

Si proviene del estado dos del diagrama donde el número de vueltas es mayor igual a 1.0, ésto es el estado_vueltas=VUELTAS_KO_ANGULO_OK, y se ha cumplido que el número de vueltas es menor que 1.0, se actualiza el estado a estado_vueltas=VUELTAS_OK_ANGULO_OK y se muestra el mensaje "Trayectoria correcta...".

```
if((abs(angulo_k1.pitch)<(d_experimento.limite_angulo_vueltas))
        &&(abs(angulo_k1.roll)<(d_experimento.limite_angulo_vueltas))&& abs(contador_vueltas)<1.0)
{
        if(estado_vueltas==VUELTAS_KO_ANGULO_OK)
        {
                m_info="Trayectoria correcta...";
                UpdateData(FALSE);
                estado_vueltas=VUELTAS_OK_ANGULO_OK;
        }
}
```

Si el valor absoluto de los ángulos pitch y roll está dentro del límite y el valor absoluto del contador de vueltas es mayor o igual a 1.0, significa que estamos en el estado dos del diagrama mostrado en el apartado anterior. Esto corresponde a estado_vueltas=VUELTAS_KO_ANGULO_OK.

Si proviene del estado uno del diagrama, donde el número de vueltas es menor a 1.0, ésto es el estado_vueltas=VUELTAS_OK_ANGULO_OK, y se ha cumplido que el número de vueltas es mayor o igual a 1.0, se actualiza el estado a estado_vueltas=VUELTAS_KO_ANGULO_OK y se muestra el mensaje "Nº vueltas no nulo: deshacer vueltas.".

```
if((abs(angulo_k1.pitch)<(d_experimento.limite_angulo_vueltas))
        &&(abs(angulo_k1.roll)<(d_experimento.limite_angulo_vueltas))&&abs(contador_vueltas)>=1.0)
{
        if(estado_vueltas==VUELTAS_OK_ANGULO_OK)
        {
                m_info="Nº vueltas no nulo: deshacer vueltas.";
                UpdateData(FALSE);
                estado_vueltas=VUELTAS_KO_ANGULO_OK;
        }
}
```

Si se excede el valor absoluto de alguno de los ángulos pitch y roll mientras el valor absoluto del contador de vueltas es menor a 1.0, significa que hemos llegado al punto final de la trayectoria y se esta realizando el montaje final. Esto corre-

sponde al estado cuatro del diagrama mostrado en el apartado anterior, donde estado_vueltas=VUELTAS_OK_ANGULO_KO.

Si proviene del estado uno del diagrama, donde el valor absoluto de los ángulos pitch y roll está dentro del límite, ésto es el estado_vueltas=VUELTAS_OK_ANGULO_OK, y se ha excedido el valor absoluto de alguno de los ángulos pitch y roll, se actualiza el estado a estado_vueltas=VUELTAS_OK_ANGULO_KO y se muestra el mensaje "Trayectoria correcta. Pulsar fin".

```
if(((abs(angulo_k1.pitch)>(d_experimento.limite_angulo_vueltas))
        ||(abs(angulo_k1.roll)>(d_experimento.limite_angulo_vueltas)))&&abs(contador_vueltas)<1.0)
{
        if(estado_vueltas==VUELTAS_OK_ANGULO_OK)
        {
                m_info="Trayectoria correcta. Pulsar fin";
                UpdateData(FALSE);
                estado_vueltas=VUELTAS_OK_ANGULO_KO;
        }
}
```

Si se excede el valor absoluto de alguno de los ángulos pitch y roll mientras el valor absoluto del contador de vueltas es mayor o igual a 1.0, significa que se ha producido un error en la realización de la trayectoria, y corresponde al estado tres del diagrama mostrado en el apartado anterior donde estado_vueltas=VUELTAS_KO_ANGULO_KO.

Si proviene del estado dos del diagrama donde el valor absoluto de los ángulos pitch y roll está dentro del límite y el contador de vueltas es mayor o igual a 1.0, ésto es el estado_vueltas=VUELTAS_KO_ANGULO_OK, y se ha excedido el valor absoluto de alguno de los ángulos pitch y roll, se actualiza el estado a estado_vueltas=VUELTAS_KO_ANGULO_KO.

Se activa la función *CDialogMedida::cancela_trayectoria()*, la cual realiza el procedimiento para terminar la ejecución del experimento, pasando al estado de reposo, eliminando el timer "TIMER_CONTROLA_VELOCIDAD" que llama a la función *controla_velocidad()* y volviendo a la ventana principal del programa. Éste código puede verse en el apéndice B.11. A su vez, se abre un cuadro de diálogo para mostrar el mensaje "Exceso de vueltas. Reinicie.".

```
if(((abs(angulo_k1.pitch)>(d_experimento.limite_angulo_vueltas))
        ||(abs(angulo_k1.roll)>(d_experimento.limite_angulo_vueltas)))&&abs(contador_vueltas)>=1.0)
{
        if(estado_vueltas==VUELTAS_KO_ANGULO_OK)
        {
        cancela_trayectoria();
        if(msgboxactivo!=true)
                {
```

```
                msgboxactivo=true;
                AfxMessageBox(_T("Exceso de vueltas. Reinicie."));
                msgboxactivo=false;
                }
        return;
        }
}
```

5.3. Corrección del cálculo de la velocidad instantánea

Durante la revisión del código existente, se atajó un problema relacionado con el control de velocidad, ya que existía una singularidad al pasar el sistema por 90^o. Las lecturas de heading y roll experimentaban saltos de $\pm 90^o$ y $\pm 180^o$, debido a la indefinición de los mismos cuando el ángulo pitch alcanzaba los 90^o.

Este problema se resolvió calculando la matriz de rotación C_k de la PMU en cada instante de muestro, obteniendo el incremento angular mediante la operación $\Delta C = C_k C_{k-1}^T$, y el posterior cálculo de los ángulos de Euler de ΔC. Con los incrementos angulares exactos en cada periodo de muestreo divididos por el incremento de tiempo se obtiene la velocidad instantánea sin riesgo de singularidad.

La razón de este fallo se debe a que inicialmente no se previeron pasos por 90^o (la propia IMU hubo de ser reprogramada para enviar la información en cuaterniones, a fin de evitar la mencionada singularidad). Después de llevar a cabo repetidas pruebas y de cientos de horas de funcionamiento, se consideró que la funcionalidad es la adecuada y la robustez es satisfactoria.

5.4. Introducción del filtro de medidas

5.4.1. Introducción

Por petición del cliente se introdujo en el programa la posibilidad de que el cálculo de las posiciones iniciales y finales de las unidades de medida se realizara tomando en cuenta no solo una trama proveniente de de la IMU, sino tomando el valor promedio de una serie de tramas consecutivas.

Sin embargo, hay que tomar en cuenta que la rotación de la tierra podría introducir un

error significativo en el cálculo del promedio. La tierra realiza aproximadamente un giro de 360º cada 24 horas, transformándolo a segundos se trata de 86400 segundos. Calculando la velocidad de rotación vemos que:

$$v = \frac{360^\circ}{86400seg} = 4{,}16 \cdot 10^{-3} \circ /seg \cdot \frac{\pi}{180^\circ} = 7{,}27 \cdot 10^{-5} rad/seg = 0{,}0727 mrad/seg$$

Sabemos que se recibe una muestra por parte de la IMU cada 0.1 segundos, por tanto cada 10 muestras que tome el filtro se traducirá en una espera de 1 segundo que introducirá un error de 0.0727 miliradianes. Recordemos que el orden del valor de la desviación entre el valor teórico y real de la posición que intentamos medir debe ser máximo de 0.16 miliradianes, por lo que el error introducido por el filtro es muy grande comparado con este.

Si por ejemplo, tomamos un filtro de 50 muestras tardaría 5 segundos por lo que el error introducido por la rotación de la tierra sería de 0.36 miliradianes, el cual es mucho mas grande que 0.16 miliradianes.

Teniendo en cuenta este problema, se ha introducido el filtro en el programa por petición expresa del cliente, siendo el número de muestras a tomar configurable por el cliente introduciéndolo como uno de los datos de cada experimento en las líneas de experimentos dentro del fichero "experimentos.txt".

Si se desea que no se implemente el promedio de las muestras, se da un valor de 1 al número de muestras a tomar. Para ver con mas detalle como introducir el valor del filtro ver el apéndice de manual de usuario A.2.3.

5.4.2. Desarrollo del filtro

Para calcular los ángulos de Euler de la posición inicial de las unidades, al pulsar el botón "Inicio Medida" de la interfaz de la unidad PMU y activarse la función *CDialogMedida::OnInicioTrayectoria()* se realizan los mismos pasos para calcular la posición inicial vistos en el capítulo 4.4.2 pero añadiendo un filtro de medidas, tal y como veremos a continuación.

Después de enviar el pulso PPS hacia ambas unidades IMU y coger la primera trama posterior a este pulso, ponemos a cero los valores de las variables que alojarán los cuaterniones para calcular la media como suma parcial. Mediante un bucle for, obtenemos los cuaterniones de la trama actual y obtenemos otra trama para que, al volver a otra iteración del bucle for, podamos sumar los cuaterniones de esta última trama al valor almacenado anteriormente para cada uno de ellos.

```
iniciales.orientacion_PMU.Q0=0;
```

```
iniciales.orientacion_PMU.Q1=0;
iniciales.orientacion_PMU.Q2=0;
iniciales.orientacion_PMU.Q3=0;

for (i=0; i< d_experimento.tamano_filtro_medidas; i++)
{
        iniciales.orientacion_PMU.Q0+=viper_dialogo->lee_flotante(&trama[5]);
        iniciales.orientacion_PMU.Q1+=viper_dialogo->lee_flotante(&trama[9]);
        iniciales.orientacion_PMU.Q2+=viper_dialogo->lee_flotante(&trama[13]);
        iniciales.orientacion_PMU.Q3+=viper_dialogo->lee_flotante(&trama[17]);
        viper_dialogo->extrae_trama_completa();
}

// Dividimos para obtener el promedio
iniciales.orientacion_PMU.Q0/=d_experimento.tamano_filtro_medidas;
iniciales.orientacion_PMU.Q1/=d_experimento.tamano_filtro_medidas;
iniciales.orientacion_PMU.Q2/=d_experimento.tamano_filtro_medidas;
iniciales.orientacion_PMU.Q3/=d_experimento.tamano_filtro_medidas;
cuaterniones_a_Euler(&iniciales.orientacion_PMU.heading,&iniciales.orientacion_PMU.pitch,
        &iniciales.orientacion_PMU.roll,iniciales.orientacion_PMU.Q0,iniciales.orientacion_PMU.Q1,
        iniciales.orientacion_PMU.Q2,iniciales.orientacion_PMU.Q3);
```

Existirán tantas iteraciones en el bucle como se indique en el campo d_experimento.tamano_filtro_medidas, el cual, es el tamaño del filtro de medidas para ese experimento en concreto obtenido a partir del fichero de experimentos. Después de obtener la suma de tantas muestras como se halla indicado en el tamaño del filtro, se divide este valor entre el tamaño del filtro para obtener de esta forma el promedio de las medidas.

Posteriormente, se llama a la función *cuaterniones_a_Euler* con los cuaterniones obtenidos como suma parcial para transformarlos en ángulos de Euler, obteniendo así los ángulos de la posición inicial de la unidad PMU, tal y como se había visto en el capítulo anterior.

Para obtener ahora los ángulos de Euler de la posición inicial de la unidad RMU, si se ha logrado enviar el pulso PPS, se ordena a la RMU mediante un mensaje UDP por el puerto serie con la cabecera CALCULA_PROMEDIO_INICIAL, que calcule el promedio de los cuaterniones de su posición inicial y los devuelva a través del puerto serie.

La unidad PMU se queda en espera, activando la función *procesa_mensajes()* que procesa los mensajes recibidos por el puerto serie, hasta que reciba los cuaterniones de la posición inicial de la RMU.

En la unidad RMU al recibirse el mensaje UDP con la orden CALCULA_PROMEDIO_INICIAL, se activa la función *CViperDlg::OnReceive()*, cuyo código puede verse en el apéndice C.5. Dentro de esta función se identifica el evento relacionado al mensaje CALCULA_PROMEDIO_INICIAL, se cambia el estado del sistema y se llama a la función *calcula_promedio*, incluyendo como argumento el tamaño del filtro de medidas.

```
case CALCULA_PROMEDIO_INICIAL:
        estado_sistema=CALCULA_PROMEDIO_INICIAL;
        pDialogMedidaRMU->calcula_promedio(bufferRxUdp.parametro,false);
        break;
```

Dentro de la función *CDialogMedidaRMU::calcula_promedio()*, cuyo código completo puede verse en el apéndice B.7, se coge la primera trama posterior al pulso PPS, y se ponen a cero los valores de las variables que almacenarán la media como suma parcial de los cuaterniones, los cuales pertenecen al buffer de transmisión, para poder enviarlos posteriormente hacia la PMU con la cabecera PROMEDIO_RMU.

```
bufferTxUdp.cabecera=PROMEDIO_RMU;
bufferTxUdp.ang_promedio.t=obtener_hora(&trama[21]);
bufferTxUdp.ang_promedio.Q0=0;
bufferTxUdp.ang_promedio.Q1=0;
bufferTxUdp.ang_promedio.Q2=0;
bufferTxUdp.ang_promedio.Q3=0;

for (i=0; i< tamano_filtro_medidas; i++)
{
// Almacenamos los datos de la primera trama
        bufferTxUdp.ang_promedio.Q0+=viper_dialogo->lee_flotante(&trama[5]);
        bufferTxUdp.ang_promedio.Q1+=viper_dialogo->lee_flotante(&trama[9]);
        bufferTxUdp.ang_promedio.Q2+=viper_dialogo->lee_flotante(&trama[13]);
        bufferTxUdp.ang_promedio.Q3+=viper_dialogo->lee_flotante(&trama[17]);
        viper_dialogo->extrae_trama_completa();
}

// Dividimos para obtener el promedio
bufferTxUdp.ang_promedio.Q0/=tamano_filtro_medidas;
bufferTxUdp.ang_promedio.Q1/=tamano_filtro_medidas;
bufferTxUdp.ang_promedio.Q2/=tamano_filtro_medidas;
bufferTxUdp.ang_promedio.Q3/=tamano_filtro_medidas;
cuaterniones_a_Euler(&heading,&pitch,&roll, bufferTxUdp.ang_promedio.Q0,
        bufferTxUdp.ang_promedio.Q1, bufferTxUdp.ang_promedio.Q2,bufferTxUdp.ang_promedio.Q3);

m_socket_envia_datagramas->Send((char *)(&bufferTxUdp),sizeof(bufferTxUdp));
TRACE(_T("Angulos enviados RPY: %.3f %.3f %.3f\n"),heading*RAD_GRA,pitch*RAD_GRA,roll*RAD_GRA);
```

Después, mediante un bucle for, obtenemos los cuaterniones de la trama actual y extraemos otra trama para que al volver a otra iteración del bucle for, sumemos los cuaterniones de esta última trama al valor almacenado anteriormente.

Existirán tantas iteraciones en el bucle como se indique en la variable tamano_filtro_medidas pasada como argumento de la función. Después de obtener la suma de tantas muestras como se halla indicado en el tamaño del filtro, se divide este valor entre el tamaño del filtro para obtener de esta forma el promedio de las medidas.

Posteriormente, se llama a la función *cuaterniones_a_Euler* con los cuaterniones obtenidos como suma parcial para transformarlos en ángulos de Euler, obteniendo así los ángulos de la posición inicial de la unidad RMU. Por último, se envía un datagrama con los valores promedio de los cuaterniones hacia la unidad PMU y se muestra por pantalla un mensaje con los ángulos de Euler enviados.

Una vez recibidos los cuaterniones en la unidad PMU correspondientes a la posición de la RMU, se vuelcan en la estructura iniciales.orientacion_RMU, se llama a la función *cuaterniones_a_Euler* para obtener los ángulos de Euler a partir de estos, y de esta forma obtener todos los valores iniciales necesarios de ambas unidades dentro de la estructura correspondiente, para utilizarlos posteriormente en el cálculo de la desviación total.

```
iniciales.orientacion_RMU.Q0=bufferRxUdp.ang_promedio.Q0;
iniciales.orientacion_RMU.Q1=bufferRxUdp.ang_promedio.Q1;
iniciales.orientacion_RMU.Q2=bufferRxUdp.ang_promedio.Q2;
iniciales.orientacion_RMU.Q3=bufferRxUdp.ang_promedio.Q3;
cuaterniones_a_Euler(&iniciales.orientacion_RMU.heading,&iniciales.orientacion_RMU.pitch,
        &iniciales.orientacion_RMU.roll,iniciales.orientacion_RMU.Q0,iniciales.orientacion_RMU.Q1,
        iniciales.orientacion_RMU.Q2,iniciales.orientacion_RMU.Q3);
iniciales.orientacion_RMU.t=bufferRxUdp.ang_promedio.t;
```

Al termino de la ejecución del experimento, pulsado el botón “Fin (Conectar PPS!)” en la ventana de la unidad PMU y activada la función *CDialogMedida::OnFinTrayectoria()*, se calculan los ángulos de Euler de las posiciones finales tal como se vé en el capítulo 4.4.3, pero agregando el filtro de medidas como veremos a continuación.

Una vez extraída la primera trama posterior al pulso PPS, igual que para el cálculo de la posición inicial, se ponen a cero los valores de las variables que almacenarán los cuaterniones para calcular la media como suma parcial.

Mediante un bulcel for, tomamos la suma de tantas muestras como se indique en el campo d_experimento.tamano_filtro_medidas, y dividimos este valor entre el tamaño del filtro, obteniendo así el promedio de las muestras, y con estos valores se llama a la función *cuaterniones_a_Euler* para obtener los ángulos de Euler de la posición final de la unidad PMU.

```
finales.orientacion_PMU.Q0=0;
finales.orientacion_PMU.Q1=0;
finales.orientacion_PMU.Q2=0;
finales.orientacion_PMU.Q3=0;

for (i=0; i< d_experimento.tamano_filtro_medidas; i++)
{
// Almacenamos los datos de la primera trama
finales.orientacion_PMU.Q0+=viper_dialogo->lee_flotante(&trama[5]);
finales.orientacion_PMU.Q1+=viper_dialogo->lee_flotante(&trama[9]);
```

```
finales.orientacion_PMU.Q2+=viper_dialogo->lee_flotante(&trama[13]);
finales.orientacion_PMU.Q3+=viper_dialogo->lee_flotante(&trama[17]);
viper_dialogo->extrae_trama_completa();
}

// Dividimos para obtener el promedio
finales.orientacion_PMU.Q0/=d_experimento.tamano_filtro_medidas;
finales.orientacion_PMU.Q1/=d_experimento.tamano_filtro_medidas;
finales.orientacion_PMU.Q2/=d_experimento.tamano_filtro_medidas;
finales.orientacion_PMU.Q3/=d_experimento.tamano_filtro_medidas;

cuaterniones_a_Euler(&finales.orientacion_PMU.heading,&finales.orientacion_PMU.pitch,
        &finales.orientacion_PMU.roll,finales.orientacion_PMU.Q0,finales.orientacion_PMU.Q1,
        finales.orientacion_PMU.Q2,finales.orientacion_PMU.Q3);
finales.orientacion_PMU.t=t;
```

Para obtener ahora los ángulos de Euler de la posición final de la unidad RMU, se realiza un procedimiento similar al caso del cálculo de la posición inicial. Si se ha logrado enviar el pulso PPS, se ordena a la RMU mediante un mensaje UDP por el puerto serie con la cabecera CALCULA_PROMEDIO_FINAL, que calcule el promedio de los cuaterniones de su posición final y los devuelva a través del puerto serie.

La unidad PMU se queda en espera, activando la función *procesa_mensajes()*, que procesa los mensajes recibidos por el puerto serie, hasta que reciba los cuaterniones de la posición final de la RMU.

En la unidad RMU, al recibirse el mensaje UDP con la orden CALCULA_PROMEDIO_FINAL, se activa la función *CViperDlg::OnReceive()*, donde se identifica el evento relacionado al mensaje CALCULA_PROMEDIO_FINAL, se cambia el estado del sistema y se llama a la función *calcula_promedio*, incluyendo como argumento el tamaño del filtro de medidas.

```
case CALCULA_PROMEDIO_FINAL:
        estado_sistema=CALCULA_PROMEDIO_FINAL;
        pDialogMedidaRMU->calcula_promedio(bufferRxUdp.parametro,true);
        break;
```

Dentro de la función *CDialogMedidaRMU::calcula_promedio()*, se realiza exactamente el mismo procedimiento que para en el caso del cálculo de la posición inicial, almacenando la media como suma parcial de los cuaterniones en el buffer de transmisión para poder enviarlos posteriormente hacia la PMU con la cabecera PROMEDIO_RMU, mostrando por pantalla un mensaje con los ángulos de Euler enviados.

Una vez recibidos los cuaterniones en la unidad PMU correspondientes a la posición de la RMU, se vuelcan en la estructura finales.orientacion_RMU, se llama a la función *cuaterniones_a_Euler* para obtener los ángulos de Euler a partir de estos.

```
finales.orientacion_RMU.Q0=bufferRxUdp.ang_promedio.Q0;
finales.orientacion_RMU.Q1=bufferRxUdp.ang_promedio.Q1;
finales.orientacion_RMU.Q2=bufferRxUdp.ang_promedio.Q2;
finales.orientacion_RMU.Q3=bufferRxUdp.ang_promedio.Q3;
cuaterniones_a_Euler(&finales.orientacion_RMU.heading,&finales.orientacion_RMU.pitch,
        &finales.orientacion_RMU.roll,finales.orientacion_RMU.Q0,finales.orientacion_RMU.Q1,
        finales.orientacion_RMU.Q2,finales.orientacion_RMU.Q3);
finales.orientacion_RMU.t=bufferRxUdp.ang_promedio.t;
```

Una vez realizado todo el procedimiento anterior, contamos con los valores promedio de un determinado número de muestras para la posición inicial y final de ambas unidades de medida, y podemos calcular la desviación total entre la posición teórica y real de los equipos con el procedimiento visto en el capítulo anterior 4.4.4.

5.5. Salida del programa en ejecución

Para poder abandonar la ejecución del experimento en cualquier momento, se ha creado un botón de salida dentro de la ventana de realización de medida de ambas unidades, el cual permite una salida del proceso de medida en ejecución de manera ordenada.

Al presionar el botón "salir" se llama a la función *CDialogMedida::OnSalir()*, la cual lanza un cuadro de diálogo donde se pregunta si se desea abandonar la ejecución, al pulsar "sí" se cierra el puerto serie para no recibir más datos provenientes de la IMU y se devuelve al menú principal del programa.

```
void CDialogMedida::OnSalir()
{
        int result;
        result=AfxMessageBox(_T(''¿Desea abandonar?''),MB_YESNO);

        if(result==IDYES)
        {
                SerialPort.ClosePort();
                exit(0);
        }
}
```

5.6. Creación de un link del programa en el escritorio

El programa se encuentra almacenado dentro de la carpeta "c:\FlashDisk\Startup", ya que es ahí donde se encuentra la memoria no volátil de la CU. Muchas veces acceder hasta ésta para ejecutar el programa, se vuelve una tardea tediosa, por ello se ha optado por crear un link del programa en el escritorio.

Como sabemos, el escritorio no forma parte de la memoria no volátil de la CU por lo que cada vez que se encienda la unidad deberá crearse el link. El procedimiento por tanto será crear un link del programa en la carpeta no volátil "c:\FlashDisk\Startup" y al encender la unidad, copiar el link de la carpeta al escritorio.

Al encender la unidad se ejecuta automáticamente el programa "Viper.exe", mostrando la ventana principal del programa, la cual llama a la función *CViperDlg::OnInitDialog()* cuyo código completo puede verse en el apéndice C.1. Dentro de esta función, se ejecuta la función *CopyFile()*, la cual copia el link del programa de la carpeta FlashDisk al escritorio.

```
CopyFile( \_T(‘‘/FlashDisk/Viper.lnk’’),\_T(‘‘/Windows/Desktop/Viper.lnk’’),FALSE);
```

Este procedimiento se realiza en ambas unidades de medida, PMU y RMU.

Capítulo 6

Depuración del sistema

6.1. Introducción

En éste capítulo se detallan las pruebas realizadas sobre el programa para comprobar su correcta ejecución y los posibles fallos presentados. A su vez, se muestran los experimentos llevados a cabo para comprobar la repetitividad de las medidas y la repercusión del filtro de medidas en el error de desviación calculado.

6.2. Depuración del programa

Los experimentos llevados a cabo en las instalaciones de EADS en el aeropuerto de San Pablo, sobre la estructura del avión para probar la robustez del programa, fueron realizados en su gran mayoría por parte de las personas de las empresas ELIMCO y EADS. Posteriormente, se nos informaban los resultados obtenidos y los cambios y chequeos que había que realizar en el programa.

Sin embargo, durante la programación, cuando las unidades de medida se encontraban en el laboratorio de la escuela, con las estructuras disponibles en el área de trabajo, sin anclajes especiales, ni bases con angulatura definida, se realizaron las pruebas pertinentes para monitorizar la correcta ejecución del programa, y tanto del contador de vueltas, como los estados de la trayectoria y los posibles fallos que presentara la ejecución.

Debido a que existía un plazo muy corto de entrega del sistema por parte de ELIMCO a la empresa EADS, no querían que las unidades saliesen de sus instalaciones para llevarlas al laboratorio de la escuela, por lo que el proceso de pruebas no fue una tarea fácil y en la última fase hubo que trasladarse a las instalaciones de ELIMCO para trabajar en la depuración del programa.

6.3. Pruebas de repetitividad y error

6.3.1. Introducción

Para comprobar la ejecución del programa y la repetitividad de las medidas entregadas, se realizaron una serie de pruebas los días 10 y 11 de Diciembre del 2009 en las instalaciones del laboratorio de la escuela, en las cuales se introdujo un equipo ficticio dentro del archivo "experimentos.txt", el cual no tiene matriz de rotación respecto al Gadiru 1. La línea de definición de este equipo se muestra a continuación:

```
#COMIENZO_EQUIPOS
1;PRUEBA;
```

Utilizamos un equipo sin rotación debido a que, para comprobar la repetitividad sin el uso de bases y anclajes para montar los equipos en distintas posiciones, si teníamos que realizar algún movimiento en las unidades no podíamos asegurar que las posiciones de los equipos en 2 experimentos distintos fuesen exactamente las mismas. Por la misma razón los experimentos se ejecutaron sin mover de posición las unidades colocándolas sobre una plataforma en paralelo tal como se muestra en la figura 6.1.

Figura 6.1: Montaje de las unidades para la ejecución de pruebas

6.3.2. Comprobación de repetitividad de las medidas

Se realizaron cuatro experimentos sobre el equipo ficticio comentado anteriormente, tomando solo una muestra para el filtro de medidas y con una duración de 10 minutos

cada experimento. La línea correspondiente a este experimento dentro del archivo "experimentos.txt" es la siguiente:

```
1;PRUEBA;GADIRU_L;10.11;900;30.33;40.44;5;60.66;1;
```

Los resultados de los experimentos se muestran en la tabla 6.3.2, donde se dan los valores de la desviación de los ángulos de Euler entre la posición teórica del equipo y la posición real, y la diferencia respecto a los valores obtenidos en el experimento anterior.

	Heading	Pitch	Roll	Diferencia con experimento anterior		
Exp. 1	-0,038465	0,0914	0,056517			
Exp. 2	0,01779	0,062498	0,109914	0,056255	0,028902	0,053397
Exp. 3	0,028017	-0,046727	0,095146	0,010227	0,109225	0,014768
Exp. 4	-0,00091	-0,160803	0,097807	0,028927	0,114076	0,002661

Tabla 6.1: Experimentos con equipo sin rotación y sin filtro de medidas

Como podemos ver, la desviación existente entre la posición teórica y real de los equipos, es la posible desviación existente entre la posición real de las unidades en el montaje del laboratorio, y la diferencia en todos los casos es mucho menor a 0.16 miliradianes. En cuanto a la repetitividad de las medidas, vemos que la diferencia de los ángulos de un experimento a otro es, en el peor de los casos de 0.11 miliradianes. Éste valor de repetitividad se toma como válido en cuanto a software, debido a que las condiciones del experimento no son óptimas, en cuanto a una medida tan precisa.

6.3.3. Comprobación de repetitividad de las medidas utilizando un filtro de muestras

Se realizaron cinco experimentos sobre el equipo ficticio sin rotación, tomando 50 muestras para el filtro de medidas y con una duración de 10 minutos cada experimento. La línea correspondiente a este experimento dentro del archivo "experimentos.txt" es la siguiente:

```
1;PRUEBA;GADIRU_L;10.11;900;30.33;40.44;5;60.66;50;
```

Los resultados se muestran en la tabla 6.3.3, donde se dan los valores de la desviación de los ángulos de Euler entre la posición teórica del equipo y la posición real, y la diferencia respecto a los valores obtenidos en el experimento anterior. Los valores de la diferencia entre experimentos que exceden el valor máximo permitido en un experimento de 0.16 miliradianes se muestran de color rojo.

Como podemos ver, la desviación existente entre la posición teórica y real de los equipos es más grande que en el caso anterior y la mayoría excede el valor permitido de

	Heading	Pitch	Roll	Diferencia con experimento anterior		
Exp. 1	-0,653926	-0,545662	-0,312054			
Exp. 2	-1,481602	-1,591278	-0,764649	0,827676	1,045616	0,452595
Exp. 3	-1,217045	-1,516902	-0,554472	0,264557	0,074376	0,210177
Exp. 4	-0,233833	-0,418771	0,010755	0,983212	1,098131	0,565227
Exp. 5	1,127883	1,161671	0,409771	1,361716	1,580442	0,399016

Tabla 6.2: Experimentos con equipo sin rotación y con filtro de 50 muestras

0.16 miliradianes. Ésto se debe a que, como explicamos en el capítulo 5.4.1, un filtro de 50 muestras tardaría 5 segundos en procesar las tramas, por lo que el error introducido por la rotación de la tierra sería de 0.36 miliradianes, el cual es mucho mas grande que 0.16 miliradianes.

En cuanto a la repetitividad de las medidas, vemos que la diferencia de los ángulos de un experimento a otro en casi todos los casos excede el valor permitido de error de 0.16 miliradianes. Por tanto, como habíamos comentado anteriormente, en el filtro de medidas no es aconsejable el uso de un número mayor a 10 muestras, ya que el error introducido solo por éste filtro, es mayor que el permitido en el experimento completo.

Capítulo 7

Conclusión y líneas futuras

Este proyecto tiene especial interés, puesto que se trata de un sistema actualmente implementado en la industria aeronáutica, para el cual fue preciso una fase previa de revisión antes de su empleo en la producción en serie. Conforme a ello, se necesitó el desarrollo de nuevos elementos en el programa para asegurar su flexibilidad, robustez y una correcta ejecución del mismo en todo momento.

En primer lugar, se modificó el sistema de cálculo del error de montaje, de modo que pudiese ser fácilmente reconfigurado por el usuario para su uso en nuevos equipos, aportando de esta forma una gran flexibilidad al sistema. Posteriormente, se implementó un conteo de giros realizado por el operario durante la trayectoria para limitar su número al realizar el montaje, y así controlar el error máximo de los algoritmos de integración de trayectorias implementados en la Unidad de Medida Inercial.

Durante la realización de algunas pruebas sobre la estructura del avión dentro de las instalaciones de EADS, en la Línea de Ensamblaje Final del A400M (FAL), tuve la oportunidad de asistir y ver la aplicación real del sistema, junto con la implementación del programa en cada experimento, hecho que aclaró la visión global que poseía sobre el propósito del proyecto, facilitándome la posterior tarea de la depuración del programa.

Dicho proceso de depuración, se realizó en su mayor parte en las instalaciones del laboratorio de la escuela, donde a partir de los datos proporcionados por Elimco y EADS, concernientes al funcionamiento del sistema dentro de las numerosas pruebas que realizaron en la estructura del avión, se llevaron a cabo las correspondientes correcciones y chequeos de los problemas que pudieran surgir en las pruebas.

Debido a la naturaleza empresarial del proyecto, existían plazos estrictos de entrega que hicieron que el trabajo fuese a contrareloj. Además durante la fase final de pruebas, hubo que trasladarse a las instalaciones de la empresa Elimco y trabajar en conjunto con ellos, para asegurar el correcto funcionamiento del programa y resolver los problemas surgidos durante este proceso.

La realización del proyecto me ha dado la posibilidad de ampliar y afianzar mis conocimientos en el campo de la programación orientada a objetos, y en concretos obre programación en C++ de sistemas embebidos. Además, los problemas surgidos me han dado la oportunidad de potenciar mis destreza en el tema, ya que las soluciones debían darse en un tiempo limitado debido a los mencionados pazos de entrega existentes.

El sistema entregado, muestra un funcionamiento robusto, en el que se han conseguido implementar todas las necesidades que el cliente espera de él, por lo que se ha alcanzado un resultado final satisfactorio. Sin embargo, en futuras versiones podrían añadirse mejoras que han quedado fuera de la versión final por falta de tiempo.

Bibliografía

C. S. Cazorla. Sistema de navegación inercial empotrado para el montaje de componentes aeronáuticos. Technical report, Escuela Técnica Superior de Ingenieros. PFC.

H. M. Deitel. *Cómo programar en C++*. Pearson Educacion, Mexico, 6a ed. edition, 2009.

EUROTECH boards & modules. Reference Guide. Technical report, EUROTECH, 2010. URL http://www.eurotech.com.

EUROTECH Imagine, build, succeed. PXA255 PC/104 Single Board Computer VIPER. Technical Report p: 888.941.2224, EUROTECH, 2010. URL http://www.eurotech-inc.com/single-board-computer-pxa255-pc104-viper.asp.

EUROTECH Ltd. Windows CE 5.0 Development Kit for VIPER Single Board Computer. Technical Report FM 12961, EUROTECH, 2007a. URL http://www.eurotech-ltd.co.uk.

EUROTECH Ltd. VIPER. Technical report, EUROTECH, 2007b. URL http://www.eurotech-ltd.co.uk.

I. Horton. *Beginning Visual C++ 6*. Wrox, cop., Birmingham, 2003.

IXSEA land & air. AIRINS Georeferencing and Orientation System. Technical Report 2009-01-BR-AIR, IXSEA, 2009. URL http://www.ixsea.com.

A. Ollero. *Robótica, Manipuladores y robots móviles*. marcombo, 2007. ISBN 84-267-1313-0.

E. J. Ruiz. Sistema de alineamiento giroscópico de tipo Strapdown. Technical report, Escuela Técnica Superior de Ingenieros, Septiembre 2008. PFC.

Apéndice A

Manual de usuario

A.1. Puesta a punto

A.1.1. Introducción

En este capítulo se describe el conexionado a utilizar a nivel de aplicación, así como la configuración del mismo. A su vez, se describe el software que debe instalarse en el ordenador utilizado como servidor central y en las unidades de control CU alojadas en las unidades de medida. Se describen los pasos para establecer una primera comunicación entre el servidor central y las unidades de medida, y una vez puesto en marcha el sistema de comunicación, cómo establecer futuras conexiones.

A.1.2. Conexionado para aplicación

En la figura A.1 se muestra el tipo de conexionado que debe realizarse para la transferencia y recepción de archivos entre las unidades de medida y el ordenador utilizado como servidor.

Como podemos ver, la conexión se realiza utilizando cables RJ45 y un switch ethernet, que facilitará la gestion y el tráfico de datos. La configuración TCP/IP del sistema se realiza asignando una dirección IP fija a cada uno de los componentes junto con una máscara, asegurándonos de que todas las direcciones se encuentren dentro de la misma red haciendo la comprobación IP_PMU & Mask = IP_RMU & Mask = IP_PC & Mask.

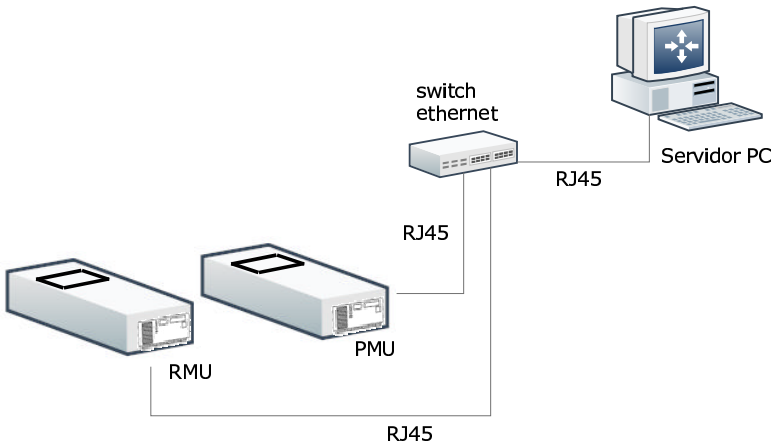

Figura A.1: Conexionado de aplicación entre las unidades de medida y el servidor.

A manera de ejemplo, utilizaremos la dirección IP 192.168.1.0 con una máscara de 24 bits (255.255.255.0), con lo que tendremos un total de 254 direcciones IP asignables dentro de la misma red. Podremos asignar cualquier dirección dentro de este rango a la unidad PMU, RMU y al servidor central. Tomaremos por ejemplo las siguientes:

- IP_PC = 192.168.1.10
- IP_PMU = 192.168.1.20
- IP_RMU = 192.168.1.30

Multipicando mediante AND lógico todas las direcciones por la máscara utilizada, podemos comprobar que todas se encuentran dentro de la misma red. Las direcciones se asignan a cada unidad de medida a través del panel de control de Windows C.E en conexiones de red, utilizando la pantalla táctil. Igualmente se asigna la dirección correspondiente al servidor central a través del panel de control. Éstas direcciones serán las que se utilicen durante todo el proceso de intercambio de datos, así como en los ficheros de configuración de las unidades de medida y dentro del programa “ServidorPC.exe”.

A.1.3. Conexionado para desarrollador

En este apartado se detalla la arquitectura hardware, el conexionado y software necesario para depurar le código fuente de la aplicación. Para una operación normal o un cambio en la configuración, estos pasos no son necesarios.

Software necesario

A continuación se detalla el software que debe instalarse para establecer una conexión del servidor central con las unidades de medida a través del programa ActiveSync, así como el sotware necesario para cargar una nueva versión o realizar una ejecución de depuración del programa "viper.exe".

Dentro del servidor central Sobre Windows XP Service Pack 2 o superior:

- **Paso 1: Embedded Visual C++ 4.0 de Microsoft Windows.** Deberá instalarse el software Microsof Windows Embedded Visual C++ 4.0 ya sea utilizando el cd-rom o a través del download center de Microsoft (www.microsoft.com/downloads).
- **Paso 2: Service Pack 4 para Embedded Visual C++ 4.0.** Si el software instalado en el paso anterior no contiene el Service Pack 4, deberá instalarse por separado, de nuevo desde el cd-rom o a través del download center de Microsoft (www.microsoft.com/downloads).
- **Paso 3: Viper SDK 5.0 de ARCOM.** Se trata de una librería SDK proporcionada por ARCOM, necesaria para trabajar con la unidad de control Viper.
- **Paso 4: ActiveSync Version 3.8.** Debe instalarse la versión 3.8 de ActiveSync. Si existe una versión superior a ésta instalada en el ordenador, se deberá realizar el procedimiento detallado en la sección A.1.3.

Dentro de la unidad de control Viper En Windows CE:

- **Paso 1: descargar el ejecutable repllog.** Puede descargarse desde la página web *http://ftp.eurotech-ltd.co.uk/mblackwell/repllog.zip*
- **Paso 2: copiar el ejecutable dentro de la carpeta \Flashdisk\Arcom bajo el nombre "repllog2.exe".** De esta forma el ejecutable se almacena dentro de la memoria no volátil de la unidad de control.

Puesta a punto de la conexión por ActiveSync

ActiveSync es una utilidad de Microsoft para acceder y sincronizar dispositivos móviles basados en Windows CE desde un PC con Windows XP o superior. En nuestra arquitectura, ActiveSync tiene los siguientes usos:

- Permite la transferencia de ficheros entre el PC y las unidades de medida.

- Permite la carga de programas en las unidades de medida desde Microsoft Embedded Studio.
- Permite la depuración cruzada (ejecución paso a paso de instrucciones desde Microsoft Embedded Studio)

Aunque es más sencillo emplear ActiveSync mediante un cable serie, la velocidad de transferencia puede ser demasiado lenta para la depuración y transferencia de ficheros. Por ello, configuraremos la aplicación para su funcionamiento mediante Ethernet. Desgraciadamente, esta funcionalidad se ha abandonado en las versiones recientes de ActiveSync y es preciso por tanto retroceder a la versión 3.8 para el funcionamiento por red. En cualquier caso, para la operación normal de las unidades de medida no es preciso ActiveSync.

I. ***Instalación del software ActiveSync versión 3.8***

- **Paso 1: desinstalar la versión más reciente de ActiveSync**
- **Paso 2: instalar la versión 3.8 de ActiveSync.**
- **Paso 3: abrir el archivo "wcesetup.log" que se encuentra en la carpeta C:\archivos de programa\activesync\.**
  Este archivo muestra en detalle de los archivos que han sido copiados y aquellos que no lo han sido durante la instalación del programa. Dentro de éste, hay que buscar los siguientes avisos:
  WARNING: not copied (old ver = x.x.x.xxxx, source ver = x.x.x.xxxx)
  Esto mostrará los archivos que no han sido remplazados durante la instalación del programa.
- **Paso 4: borrar cada uno de los archivos enlistados y reinstalar ActiveSync versión 3.8.**
  De esta manera se copiarán los archivos inexistentes.

II. ***Establecimiento de la primera conexión a través de ActiveSync***

Para establecer la conexión entre el servidor central y cada una de las unidades de medida, se deberán seguir los pasos que se detallan a continuación. Este proceso se realiza sólo en la primera conexión y debe realizarse para cada una de las unidades de medida de forma independiente.

- Paso 1: se conecta la unidad de medida (PMU o RMU) con el servidor central mediante un cable RS-232, como se muestra en la figura A.2.
- Paso 2: lanzar ActiveSync 3.8 desde el servidor central. La ventana principal de ActiveSync 3.8 se muestra en la figura A.3
- Paso 3: activar la opción de permitir conexiones Ethernet en Archivo -> Configuración de conexión como puede verse en la figura A.4
- Paso 4: hacer click en Archivo -> Conectar
  Aparece el mensaje CONECTANDO en la ventana principal de ActiveSync
- Paso 5: en la unidad CU pulsar start-> run-> repllog2/remote. Esto enviará una respuesta al ordenador estableciendo la conexión.

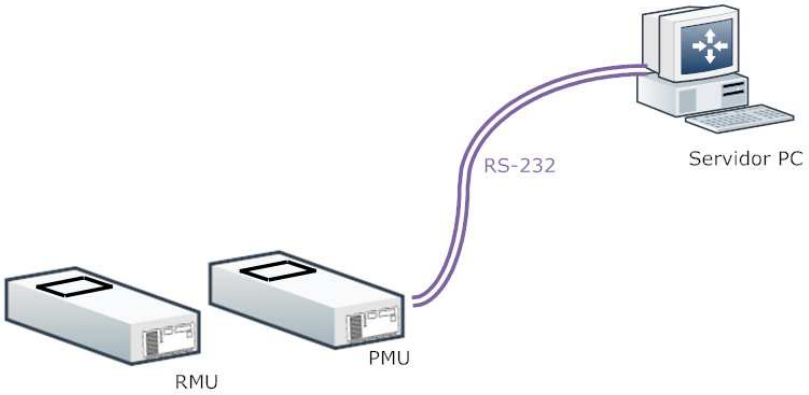

Figura A.2: Conexionado entre las unidades de medida y el servidor.

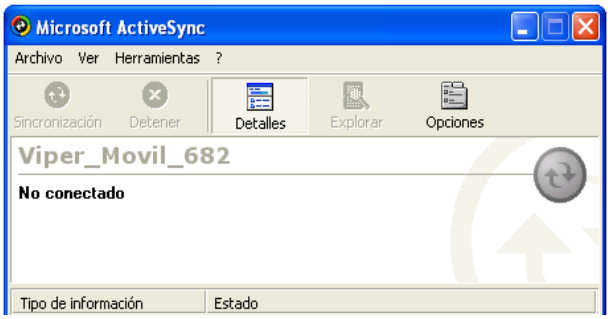

Figura A.3: Ventana principal de ActiveSync 3.8

- Paso 6: en la unidad CU pulsar \Flashdisk\Setup\savereg.exe. Esto guarda la configuración de la comunicación.

- Paso 7: Desconectar el cable RS-232

Una vez realizado este proceso para las dos unidades de medida, las posteriores conexiones se pueden realizar con tan solo lanzar el ActiveSync en el servidor central y pulsando en la unidad CU start-> run-> repllog2/remote, estando los sistemas conectados directamente a través de un cable RJ45 o a través de un switch como se muestra en la figura A.1. Cabe destacar, que el servidor central solo puede sincronizarse con una unidad de medida a la vez.

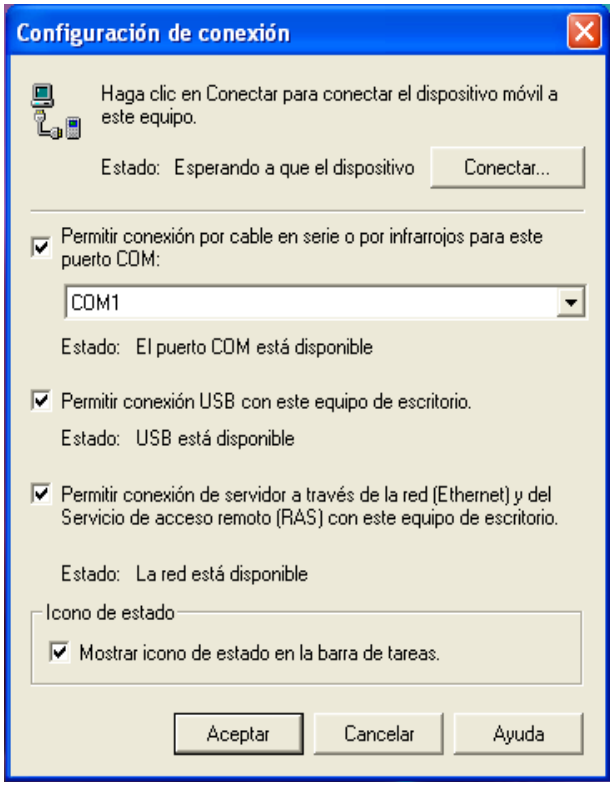

Figura A.4: Activación del funcionamiento mediante Ethernet de ActiveSync

A.2. Ficheros del sistema

A.2.1. Introducción

En este capítulo se describen los diferentes archivos utilizados por el sistema diseñado. Existen tres archivos distintos llamados de configuración, de experimentos y de resultados. En los siguientes apartados se describe la utilidad y diseño de cada uno de ellos.

Las dos diferentes unidades de medida utilizan distintos archivos. La unidad de medida móvil PMU utiliza, tanto el archivo de configuración como el de experimentos, y genera el archivo de resultados; mientras que la unidad de medida de referencia RMU sólo utiliza el archivo de configuración, ya que como veremos mas adelante ésta no realiza ningún tipo de cálculo, limitándose a enviar su información a la PMU.

Los distintos archivos son almacenados dentro de la memoria no volátil de la CU en cada una de las unidades de medida, ubicada concretamente en el directorio “FlashDisk”. El envío y recepción de estos archivos por parte del servidor central se tratará en el capitulo A.3.

A.2.2. Archivo de configuración “conf.txt”

Se trata de un fichero de texto que contiene los valores de configuración necesarios para la comunicación de ambas unidades de medida entre sí y el servidor central. Los archivos de configuración deberán ser enviados a las unidades de medida desde el servidor central antes de comenzar los experimentos. El proceso de envío del fichero por parte del operario se detalla en el capítulo siguiente A.3.5. Para este envío se utiliza una conexión TCP.

Este archivo se almacena con el nombre “conf.tx”, debido a que el sistema está programado para buscarlo con este nombre, pero existe flexibilidad ya que se puede cambiar fácilmente el código del programa. La estructura del archivo es simple, en cada línea se coloca el nombre de la variable seguido de su valor, separado por el signo igual (=). El orden de las variables dentro del archivo no es estricto, el sistema está programado de tal forma que sea capaz de leer todas las variables incluidas, independientemente del orden que éstas ocupen dentro del archivo.

La descripción de cada una de las variable contenidas en este fichero se muestran en la siguiente tabla A.1, junto con los posibles valores que pueden tener.

Variable	Descripción	Valor permitido
ip_local	Indica la dirección IP local de la unidad CU. Aunque para el correcto funcionamiento del sistema, la CU no necesita conocer su dirección local, se ha optado por incluirla para evitar confusiones.	Una dirección IP válida
ip_viper_remota	Indica la dirección IP de la otra unidad CU. Se necesita esta dirección para realizar la comunicación UDP necesaria en la sincronización de las unidades de medida.	Una dirección IP válida
ip_pc	Indica la dirección IP del servidor central. Las unidades de medida necesitan conocer esta dirección para establecer la comunicación TCP con el servidor central.	Una dirección IP válida
puerto_udp	Indica el puerto UDP usado en la comunicación UDP entre las unidades de medida.	Un puerto UDP válido.
puerto_serie	Indica el puerto COM usado por la CU para comunicarse con la unidad IMU.	com1:,com2:,com3: ó com4:
viper_referencia	Indica si la unidad de medida es la unidad móvil o la de referencia. Si la variable vale 1, la unidad será la de referencia (RMU). Si vale 0, la unidad asumirá que es la movil(PMU).	0 ó 1

Tabla A.1: Variables del archivo de configuración.

Un ejemplo del archivo de configuración se muestra en la figura A.5.

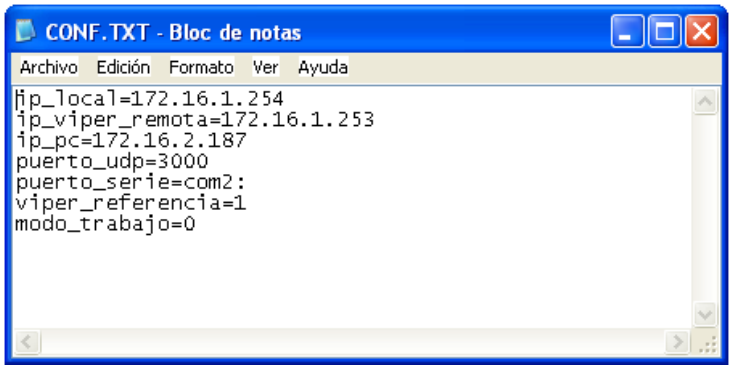

Figura A.5: Ejemplo de archivo de configuración

A.2.3. Archivo de experimentos "experimentos.txt"

Se trata de un fichero de texto instalado en la PMU dividido en 2 partes:

- **Parte de experimentos:** contiene una descripción de cada experimento disponible a realizar, acompañado de unos valores de distintas variables, necesarias para seleccionar el equipo de montaje que se está calibrando y las condiciones de trabajo del experimento (velocidades máximas tolerables, tiempo máximo, etc.), que serán monitorizadas en todo momento por el programa. Son las líneas previas a la etiqueta #COMIENZO_EQUIPOS.
- **Parte de equipos:** contienen la definición de la orientación de cada uno de los equipos de montaje o adaptadores, respecto al equipo GADIRU 1. El sistema ha de conocer esta medida para poder calcular el error de alineamiento. Son las líneas posteriores a la etiqueta #COMIENZO_EQUIPOS.

Este archivo se utiliza sólo en la unidad de medida móvil PMU. Se almacena en la memoria no volátil con el nombre "experimentos.txt", debido a que el sistema está programado para buscarlo como tal. Un ejemplo de dicho fichero se muestra en la figura A.6.

Como ocurría con el archivo de configuración, el sistema necesita conocer la descripción de los experimentos y los equipos, junto con los valores de las variables asociados a ellos para llevar a cabo todo el proceso de medida, por lo que el archivo de experimentos deberá ser enviado a la unidad de medida móvil desde el servidor central antes de comenzar los experimentos, utilizando para este envío una conexión TCP (ver A.3.5).

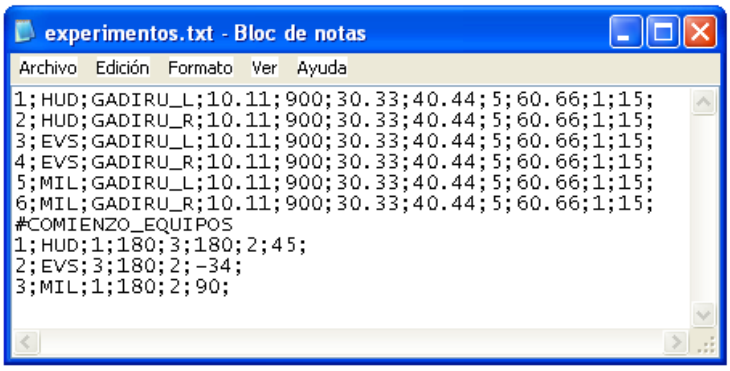

Figura A.6: Ejemplo de archivo de experimentos

Líneas de definición de experimento

En las líneas iniciales (anteriores a #COMIENZO_EQUIPOS) se muestran las opciones de cada uno de los experimentos (operaciones de montaje), acompañado de unos valores de distintas variables, necesarios para comprobar que el experimento se ejecuta bajo las condiciones adecuadas.

La estructura del archivo es sencilla, por cada línea del archivo se tienen los valores de los distintos campos que describen cada uno de los experimentos, separados por punto y coma (;). A diferencia de lo que pasaba con el archivo de configuración, en el archivo de experimentos los campos incluidos en cada línea tienen una posición fija, y no pueden ser intercambiados entre ellos. Se debe respetar el orden de los campos para que el sistema los interprete correctamente y además, no puede omitirse ninguno de los campos. Deben de estar separados obligatoriamente por punto y coma (;) e igualmente cada línea debe estar terminada por un punto y coma (;).

A continuación, en la tabla A.2, se describen cada uno de los parámetros de configuración necesarios para cada experimento, la posición en la que aparecen y el tipo de valores que pueden tener.

Posición	Parámetro	Descripción	Tipo
1	Índice	Índice del experimento, para localizarlo en el listado que se muestra en el inicio de la aplicación Windows CE	Un número entero positivo.
2	Tipo Equipo (HUD, EVS...)	El nombre del equipo cuyo alineamiento se desea medir. Debe coincidir con alguno de los disponibles en las líneas posteriores a la etiqueta #COMIENZO_EQUIPOS	Alfanumérico
Continuación en la siguiente página			

Tabla A.2: Variables de las líneas de definición de experimentos.

Tabla A.2 – continuación de la página anterior

Posición	Parámetro	Descripción	Tipo
3	Posición inicial de la RMU	GADIRU_L: izquierda del avión, también conocido como GADIRU 1. GADIRU_R: derecha del avión (rotado 180º grados en torno a Z respecto a GADIRU_L)	Alfanumérico
4	No usado	Este campo no tiene uso en esta versión del software, pero debe contener un valor 0 por compatibilidad.	Un cero.
5	Tiempo máximo	Tiempo máximo que debe emplear el operario en el montaje (segundos). Habitualmente son 15 minutos (900s)	Un número positivo.
6	Velocidad máxima	Velocidad máxima de giro que se permitirá al operario en las maniobras de montaje (radianes/segundo)	Un número positivo.
7	Velocidad alta	Velocidad máxima mantenida de giro (límite en la integral de velocidad) que se permitirá al operario en las maniobras de montaje (radianes/segundo). El tiempo máximo durante el cual se considera aceptable esta velocidad viene dado por el parámetro num_alto_maximo descrito a continuación.	Un número positivo.
8	num_alto _maximo	Máximo número de muestras (a 100ms por muestra) en las que se toleran velocidades de giro superiores al parámetro Velocidad alta.	Un número entero positivo.
9	No usado	Este campo no tiene uso en esta versión del software, pero debe contener un valor 0 por compatibilidad.	Un cero.
10	Tamaño filtro medidas	Si es mayor que uno, indica que se deben realizar los cálculos a partir de una media de las medidas proporcionadas por la IMU. Su valor indica el número de muestras consecutivas que intervendrán en el cálculo de la media. Sirve para suavizar la medida.	Un número entero positivo.
11	Límite ángulo vueltas	Indica el máximo valor posible en grados de los ángulos pitch y roll que soporta la unidad de medida PMU durante la trayectoria de ejecución del experimento. Se utiliza como umbral de ejecución correcta dentro del contador de vueltas.	Un número real.

Líneas de definición de equipo

Las líneas posteriores a la etiqueta #COMIENZO_EQUIPOS, sirven para definir la orientación de cada uno de los equipos de montaje o adaptadores, respecto al equipo GADIRU 1. El sistema ha de conocer esta medida para poder calcular el error de alineamiento.

A continuación, se describen cada uno de los parámetros que aparecen para cada equipo definido, la posición en la que aparecen y el tipo de valores que pueden tener. Al igual que en el caso anterior, todos los parámetros pertenecientes a una línea son separados por punto y coma(;), y cada línea debe estar terminada por un punto y coma (;).

Posición	Parámetro	Descripción	Tipo
1	Índice	Índice del equipo, se emplea por legibilidad.	Un número entero positivo.
2	Tipo Equipo (HUD, EVS...)	El nombre del equipo cuyo alineamiento se desea medir. Debe coincidir con el segundo parámetro del experimento seleccionado.	Alfanumérico
3	Eje	Eje sobre el cual se realiza la rotación.	1=X, 2=Y, 3=Z
4	Ángulo (grad)	Ángulo de giro en torno al eje especificado en el parámetro Eje.	Un número real.
Resto	Pares sucesivos eje-ángulo, del mismo modo que se muestran en los puntos 3 y 4.		

Tabla A.3: Variables de las líneas de definición de equipos

Posterior al índice y al nombre del equipo, aparecen una serie de rotaciones en forma de pares de valores numéricos (tantos como se deseen), que definen los giros que llevarían al sistema de medida (RMU), desde la posición GADIRU 1 hasta el punto de anclaje del adaptador perteneciente al equipo en cuestión.

Dichos giros se pueden descomponer en una serie de pasos, de modo que en cada paso se especifica el ángulo (en grados) que rota la RMU respecto a sus ejes locales. En cada paso, los ejes quedan determinados por el resultado del giro anterior (es decir, siempre hay que referir los giros a ejes solidarios con la RMU).

El sentido positivo de los giros viene dado por la regla del sacacorchos, y el sistema de ejes inicial de la RMU viene representado en la figura A.7.

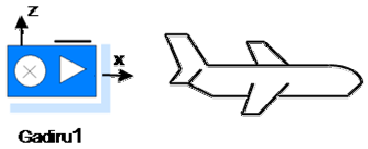

Figura A.7: Sistema de ejes al inicio del movimiento. La banda negra superior representa la pantalla de la RMU, vista lateralmente. El eje Y apunta hacia el interior del papel.

A.2.4. Archivo de resultados "resultados.csv"

Este archivo contiene los resultados arrojados después del procesado de los datos acumulados durante un experimento, y es creado únicamente en la unidad móvil PMU una vez finalizado el experimento. Este archivo se crea con el nombre "resultados.csv", en caso de no existir anteriormente, o es actualizado en caso contrario. Se trata de un fichero que contiene una línea de información por cada experimento realizado. Esta información consiste en varios campos separados por punto y coma (;), mostrando un aspecto como el de la figura A.8.

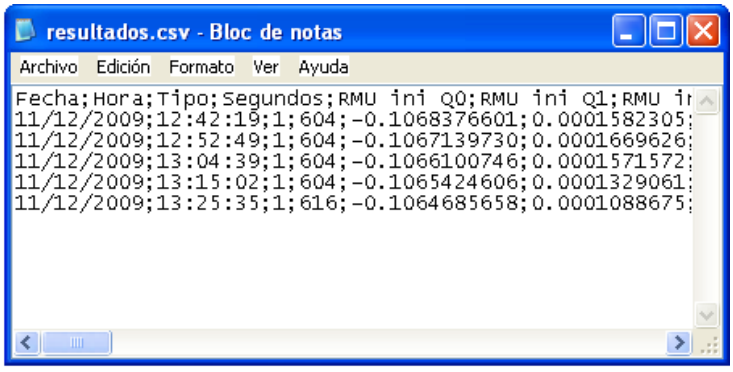

Figura A.8: Ejemplo de archivo resultados.cvs

Se ha optado por la separación de los datos de resultados mediante punto y coma para facilitar así la importación de los datos a Microsoft Excel. Es por ello por lo que el archivo de resultados posee la extensión csv (del inglés comma-separated values). El archivo importado muestra la forma de la figura A.9.

Este archivo, podrá ser descargado al servidor central cuando sea necesario. Para ello, se hace uso de la conexión TCP, que es posible establecer entre la unidad de medida y el propio servidor central (ver A.3.8). A continuación, se describen cada uno de los campos de resultados incluidos en cada línea, o lo que es lo mismo, los datos obtenidos por cada experimento realizado:

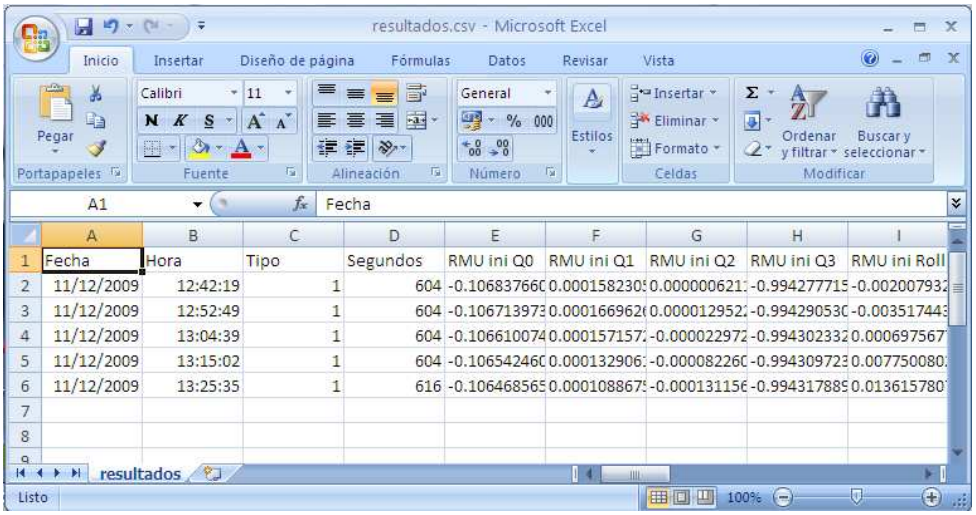

Figura A.9: Ejemplo de archivo resultados.cvs en Excel

Posición	Parámetro	Descripción	Tipo
1	Fecha	Indica la fecha en la que se realizó el experimento en cuestión.	Fecha válida en formato DD/MM/AAAA
2	Hora	Indica la hora exacta a la que se grabaron los resultados del experimento en cuestión.	Hora válida en formato HH:MM:SS
3	Índice	Indica el tipo de experimento, identificado por un índice numérico. Coincide con el índice del archivo de experimentos.	Un número entero positivo.
4	Segundos	Indica el tiempo en segundos, empleado para la realización del experimento en cuestión.	Un número entero positivo.
5, 6, 7 y 8	Posición inicial RMU (cuat)	Indica los 4 valores correspondientes a los cuaterniones de la posición inicial de la unidad RMU.	Un número real.
9, 10 y 11	Posición inicial RMU (deg)	Indica los valores de los ángulos Roll, Pitch y Yaw correspondientes a la posición inicial de la unidad RMU.	Un número real.
12, 13, 14 y 15	Posición inicial PMU (cuat)	Indica los 4 valores correspondientes a los cuaterniones de la posición inicial de la unidad PMU.	Un número real.
16, 17 y 18	Posición inicial PMU (deg)	Indica los valores de los ángulos Roll, Pitch y Yaw correspondientes a la posición inicial de la unidad PMU.	Un número real.
19, 20, 21 y 22	Posición final RMU (cuat)	Indica los 4 valores correspondientes a los cuaterniones de la posición final de la unidad RMU.	Un número real.

Continuación en la siguiente página

Tabla A.4: Variables del archivo de resultados.

Tabla A.4 – continuación de la página anterior

Posición	Parámetro	Descripción	Tipo
23, 24 y 25	Posición final RMU (deg)	Indica los valores de los ángulos Roll, Pitch y Yaw correspondientes a la posición final de la unidad RMU.	Un número real.
26, 27, 28 y 29	Posición final PMU (cuat)	Indica los 4 valores correspondientes a los cuaterniones de la posición final de la unidad PMU.	Un número real.
30, 31 y 32	Posición final PMU (deg)	Indica los valores de los ángulos Roll, Pitch y Yaw correspondientes a la posición final de la unidad PMU.	Un número real.
33, 34 y 35	Error Total (mrad)	Indica el error total calculado tras procesar y comparar las posiciones inicial y final de las dos unidades de medida RMU y PMU. El error es calculado para cada uno de los ángulos Roll, Pitch y Yaw, y está expresado en miliradianes.	Un número real.
36	Tamaño de Filtro	Indica el tamaño del filtro de medidas utilizado.	Un número entero positivo.

A.3. Ejecución paso a paso

A.3.1. Introducción

En este capítulo se detalla la ejecución del sistema paso a paso, siguiendo el orden de prueba de una ejecución real. De esta forma, puede verse de manera mas gráfica el funcionamiento del sistema y cada uno de los pasos a seguir al realizar un experimento. Se tratará un caso general, si bien es cierto que existen algunas variables a la hora de realizar un experimento que pueden modificar la secuencia o algunos de los pasos que veremos a continuación.

La ejecución paso a paso se divide en los siguientes tramos:

- Arranque de las unidades de medida (CU + IMU)
- Arranque del servidor central
- Intercambio de ficheros entre unidades de medida y servidor central
- Elección de experimento y alineamiento de las unidades de medida

- Realización del experimento y medidas
- Envío del fichero de resultados desde la unidad móvil PMU hacia el servidor central

A.3.2. Arranque de las unidades de medida

El primer paso a dar para realizar un experimento, será obligatoriamente el encendido de la unidad de medida. Como se conoce, la unidad de medida contiene la unidad IMU y el ordenador empotrado, por lo que se necesita arrancar los dos componentes. Durante el arranque de la IMU, ésta pasa a modo de alineamiento inicial, paso necesario para eliminar posibles errores de deriva. El arranque de la CU es similar al de cualquier ordenador personal: será necesario esperar a que se cargue el sistema operativo instalado, sobre el que correrá la aplicación diseñada. En nuestro proyecto el sistema operativo elegido es Microsoft Windows CE 5.0.

Una vez que el sistema operativo ha terminado su inicio, a través de la pantalla táctil de la que dispone la unidad CU y que hará de interfaz de usuario, se ejecutará la aplicación diseñada en el presente proyecto. Ésta se encontrará en forma de un archivo ejecutable llamado "Viper.exe" dentro de la carpeta "c:\FlashDisk\Startup", de esta manera se ejecuta cada vez que se inicia la unidad CU, aunque también, estará disponible desde el escritorio un acceso directo al ejecutable. La interfaz gráfica, que mostrará el sistema una vez ejecutado, es la que aparece en la figura A.10.

Figura A.10: Interfaz principal de la aplicación "Viper.exe"

Es importante señalar, que al ejecutar la aplicación "Viper.exe", ésta intentará leer la configuración a adoptar desde el archivo "conf.txt" que debe encontrarse previamente alojado en la ruta "c:\FlashDisk\Startup". Este procedimiento lo realizan tanto la unidad de medida móvil como la de referencia. El título de la ventana que aparece

al arrancar el sistema, dependerá del archivo de configuración que la CU contenga y mostrará "UNIDAD DE REFERENCIA" o "UNIDAD MOVIL" según sea el caso. Si la CU no tuviese este archivo alojado en su memoria, se encontrara corrupto, o fuera imposible su lectura, el programa mostrará un mensaje de error avisando de tal circunstancia.

Es lógico, que la primera vez que se arranque la unidad de medida, no se disponga del archivo de configuración en su interior. El paso del archivo de configuración y del de experimentos desde el servidor central hacia las unidades de medida será el tercer paso a dar en el ejemplo que se describe, y se verá más adelante. Una vez que la unidad de medida posea el archivo de configuración, no será necesario enviárselo de nuevo a no ser que se quiera cambiar su configuración de funcionamiento.

El cuadro de diálogo mostrado por el software es simple y muestra 4 opciones. Si se pulsa sobre el primer botón "COMUNICACIÓN PC", se mostrará una nueva ventana que nos permitirá comunicar la unidad de medida con el servidor central. El proceso de comunicación se detalla más adelante en el apartado A.3.5. Si, en cambio, se pulsa sobre el segundo botón "REALIZACIÓN MEDIDA", un nuevo cuadro de diálogo pasará a primer plano y haciendo uso de él, se podrán realizar los experimentos y medidas oportunas. Se tratará sobre esto en los apartados A.3.6 y A.3.7. Al pulsar sobre el tercer botón "VISUALIZACION TRAMAS", se muestra la posición actual de la unidad de medida, es decir, los ángulos heading, pitch y roll (expresados en radianes), junto con los cuaterniones correspondientes devueltos por la IMU en ese instante. Esta visualización se explica con detalle en la sección A.3.3

Por último, si se pulsa sobre el botón "SALIR", el programa termina y se cierra la ventana del mismo. Cabe destacar que en la parte inferior del cuadro de diálogo, se muestra en todo momento el nivel de batería actual del sistema.

A.3.3. Visualización de la posición actual de la unidad de medida

Al pulsar sobre el botón "VISUALIZACION TRAMAS", se abre una ventana que muestra la posición actual de la unidad de medida, es decir, los ángulos heading, pitch y roll (expresados en radianes), junto con los cuaterniones correspondientes devueltos por la IMU en una trama, que indica la posición del sistema. Esta ventana puede verse en la figura A.11.

Al pulsar sobre el botón "Actualizar", se toma la última trama enviada por la IMU informando de la posición del sistema, y se muestra por pantalla. Si seleccionamos la casilla "Reseteando" y pulsamos sobre el botón "Reset IMU", provocará que se reinicie la IMU incluida en la unidad móvil. Este proceso puede durar alrededor de cinco minutos.

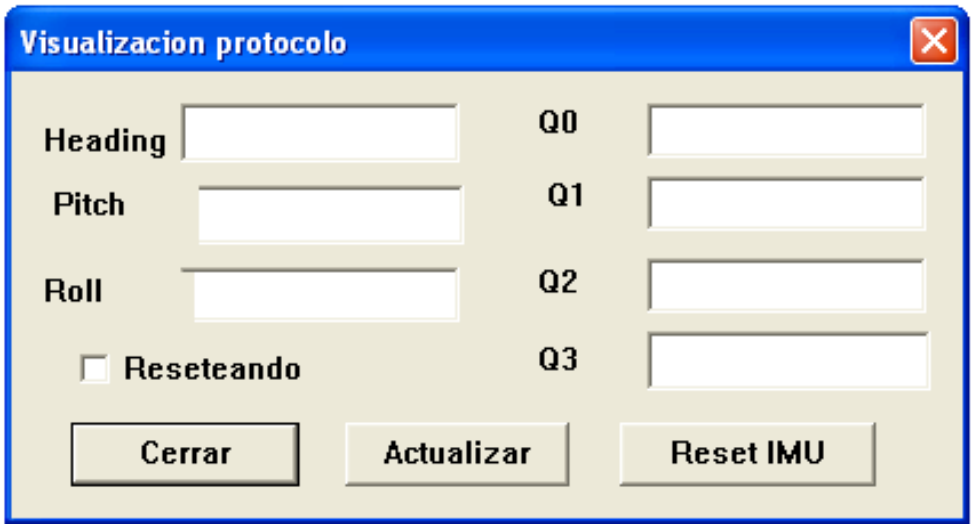

Figura A.11: Ventana de visualización de tramas devueltas por la IMU

Si al pulsar el botón anterior la casilla se encontraba desmarcada, el sistema no realiza el proceso de reinicio. Por último, al pulsar sobre el botón "Cerrar", se cerrará la ventana actual, volviendo a la ventana principal del programa "Viper.exe".

A.3.4. Arranque del servidor central

El servidor central del sistema será un PC, sobre el que se ejecutará un programa específicamente diseñado para la realización de las tareas necesarias. Es importante dejar claro que el programa que hace las funciones de servidor central, cuyo archivo ejecutable está nombrado como "ServidorPC.exe", es diferente e independiente al programa ejecutado en las unidades CU ("Viper.exe").

Una vez aclarado lo anterior, el segundo paso a dar tras la puesta en marcha de las unidades de medida, sería el arranque del PC que albergue el servidor central y la ejecución del mismo. La interfaz gráfica del servidor es lo más simple posible, para facilitar así la labor del operario que haga uso de ella. Básicamente, está dividida en dos zonas: una para comunicarse con la unidad de medida móvil (PMU), y otra para comunicarse con la unidad de medida de referencia (RMU).

Como puede apreciarse en la figura A.12, existe la posibilidad de enviar archivos de configuración y de experimentos hacia la unidad móvil. Además, se pueden descargar los resultados de los experimentos desde la misma unidad. Sin embargo, como se desprende del funcionamiento del sistema en su totalidad, hacia la unidad de referencia sólo podremos enviar archivos de configuración, ya que no necesita los archivos de experimentos y no genera ningún archivo de resultados.

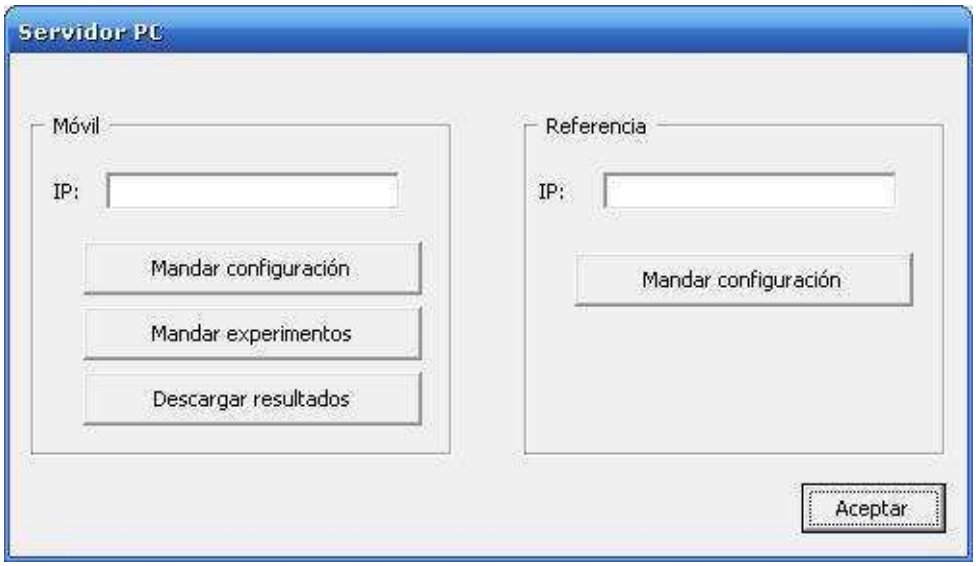

Figura A.12: Interfaz del servidor

A.3.5. Intercambio de ficheros entre unidades de medida y servidor central

Tras el arranque de las unidades de medida y del servidor central, el siguiente paso a dar es la transmisión de los archivos de configuración a ambas unidades de medida y del archivo de experimentos, a la unidad de medida móvil. Como paso previo, es necesario realizar la conexión entre el servidor central y las unidades de medida, a través de un cable de red y sus tarjetas de red como hemos visto en el apartado A.1. Recordemos que la transmisión de los archivos se hará sobre TCP/IP.

Tras la unión mediante cables, en la ventana principal del programa ejecutado en la unidad de medida, debemos pulsar sobre el primer botón "COMUNICACIÓN PC", tras lo cual se mostrará una nueva ventana que posee dos botones, que dan acceso a la recepción del archivo de configuración y de experimentos como se ve en la figura A.13.

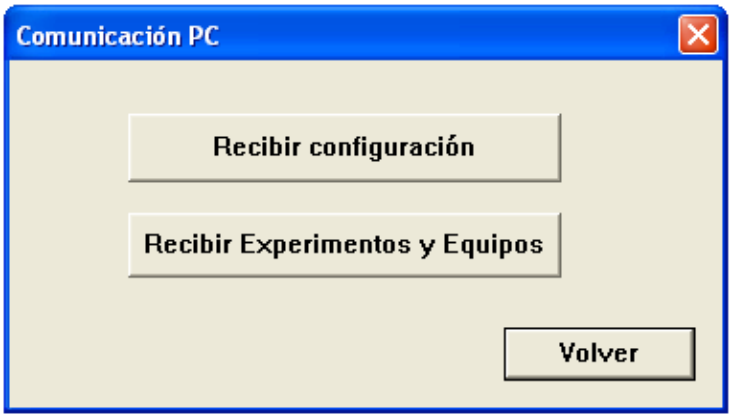

Figura A.13: Interfaz de comunicación con PC de la PMU

Cabe destacar que el botón “Recibir Experimentos y Equipos” del cuadro de diálogo, estará deshabilitado si nos encontramos en la unidad de medida de referencia, ya que ésta unidad no necesita ese archivo para realizar su tarea. Como puede apreciarse, la opción de enviar los resultados al servidor central no aparece en ninguna de las dos unidades de medida. Esto se debe, a que los resultados pueden ser enviados en cualquier momento de la ejecución, no es necesario acceder a este cuadro de diálogo para enviar los resultados. Más adelante se explicará el paso correspondiente al envío de resultados al servidor central.

El proceso para enviar los ficheros desde el servidor central a las unidades de medida, se debe realizar siguiendo los pasos que se muestras a continuación:

I. En el cuadro de diálogo del servidor central, se introduce la dirección IP de la unidad CU a la que se quiere transmitir el archivo en cuestión.

II. El siguiente paso a dar, es pulsar sobre el botón “Mandar configuración” o “Mandar experimentos”, según sea el archivo A.12.

III. Tras esto, aparecerá un cuadro de diálogo mediante el cual se puede seleccionar el archivo a mandar, moviéndonos para ellos por el árbol de ficheros A.14.

Figura A.14: Cuadro de diálogo para seleccionar archivo a mandar

IV. Una vez seleccionado el archivo, pulsamos sobre el botón “Abrir” del cuadro de diálogo.

V. Tras los pasos anteriores, la interfaz mostrará el mensaje “ENVIANDO CONFIGURACIÓN A LA UNIDAD VIPER”.

VI. Es ahora y sólo ahora, cuando se debe pulsar en la ventana de la CU sobre el botón “Recibir configuración” o “Recibir Experimentos y Equipos”, en función del archivo a recibir por parte de la unidad de medida A.13.

VII. Una vez enviado el fichero, se mostrará en la pantalla del servidor central un mensaje comunicando si la operación se ha realizado correctamente o, si por el contrario, ha habido algún tipo de error.

Hay que tener en cuenta que el envío correcto de los archivos por parte del servidor central, está garantizado solo si se siguen correctamente los pasos descritos.

Es muy importante recalcar, que una vez recibidos los archivos de configuración y el archivo de experimentos por parte de las unidades de medida, éstas deben ser reiniciadas para que los cambios sean efectivos. Esto se le indica al operario mediante un mensaje emergente A.15.

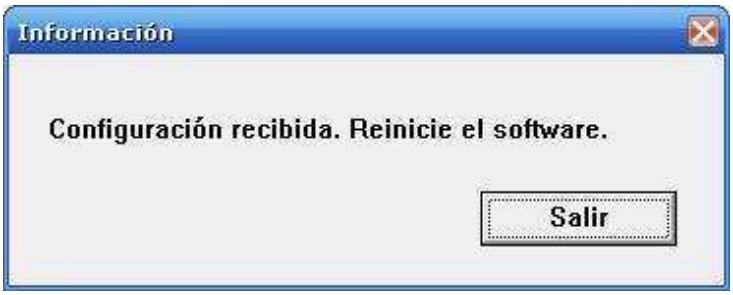

Figura A.15: Mensaje de confirmación mostrado

A.3.6. Elección de experimento y alineamiento de las unidades de medida

Una vez completado el paso anterior, y después de reiniciar las dos unidades de medida para que se hagan efectivas las configuraciones asignadas a cada una de ellas, se puede continuar con el siguiente paso. De nuevo nos encontramos en la ventana principal del programa "Viper.exe" mostrada en la figura A.10. Como ahora el objetivo es realizar el experimento, se debe pulsar el botón "REALIZACIÓN MEDIDA". Esto generará una nueva ventana, que será distinta para cada una de las unidades de medida.

Como se puede observar, el cuadro de diálogo de la unidad de referencia RMU A.16 es mucho más simple que el de la móvil PMU A.17. Para continuar la ejecución del experimento, no se necesita interactuar más con la interfaz de la unidad de referencia, el operario sólo deberá manejar la interfaz de la unidad móvil.

El operario deberá elegir el experimento a realizar de entre los disponibles en la lista desplegable, que se encuentra en la parte superior de la ventana. Evidentemente, esta lista de experimentos se corresponde con los experimentos incluidos en el archivo de experimentos, anteriormente importado a la unidad de medida desde el servidor central.

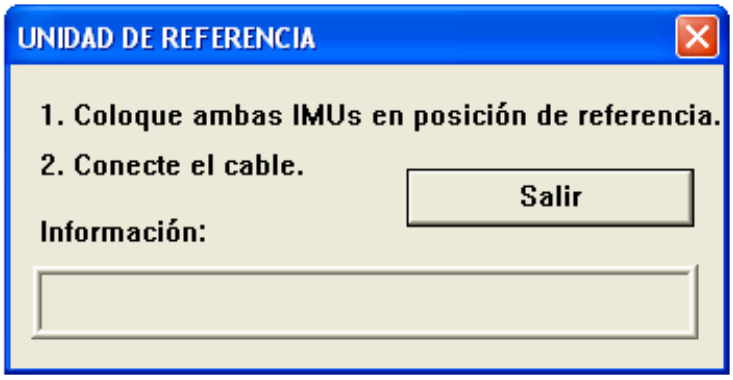

Figura A.16: Interfaz mostradas por la unidad RMU

Figura A.17: Interfaz mostradas por la unidad PMU

Una vez elegido el experimento, y con las dos unidades de medida en la posición de referencia, se procederá a conectarlas mediante un cable de red, a través de sus tarjetas de red como se ha expuesto en el capítulo A.1. Se recuerda que la comunicación entre ambas unidades de medida se realiza mediante UDP, y para ello, se usará el cable de red que podrá conectarse directamente de una unidad a la otra, o a través de un switch como se muestra en la figura A.1.

Estando las dos unidades conectadas mediante el cable y situadas en la posición de referencia, se procederá a la fase de alineamiento. Si se trata de una primera alineación o se desea que las IMUs se reinicien se marcará la casilla “Reset IMUs”, lo cual aparecerá por defecto. Si por el contrario se han reiniciado ya las IMUs con anterioridad y no queremos repetir este proceso, habrá que desmarcar esta casilla. A continuación, se pulsará sobre el botón “IMUs preparadas”.

Si la casilla está marcada, al pulsar el botón provocará que se reinicien y alineen

ambas IMU, la incluida en la unidad móvil y la incluida en la unidad de referencia. Este proceso puede durar alrededor de cinco minutos y se irá informando de su progreso a través del cuadro de texto situado en la parte inferior de la ventana. Si al pulsar el botón anterior la casilla se encontraba desmarcada, el sistema no realiza el proceso de reinicio, terminando la fase de alineamiento y dando paso a la siguiente tarea.

En todo momento el cuadro de información irá mostrando mensajes de progreso, así como los posibles errores que se pudieran producir. Esto será muy útil a la hora de poder identificar rápidamente el motivo de los errores, así como el siguiente paso a dar por el operario. En la unidad de medida de referencia también se incluye el cuadro de texto de información, mostrando los mensajes relativos a su funcionamiento.

La fase de alineamiento terminará cuando en el cuadro de información se muestre el mensaje "Pulse -Inicio Trayectoria-" en la unidad de medida móvil. El cable que une los dos puertos Ethernet debe permanecer conectado, ya que en el siguiente paso será necesaria una comunicación UDP entre ambas unidades de medida.

A.3.7. Realización del experimento y medidas

Una vez elegido el experimento a realizar, antes de desconectar el cable de red que une las unidades de medida, el operario deberá conectar ambas unidades mediante el cable PPS y pulsar el botón "Inicio Medida" en la interfaz de la unidad de medida móvil. Esto provocará la puesta a cero de los tiempos de las dos unidades de medida y el cálculo de las posiciones iniciales de ambas.

Después de una serie de operaciones realizadas por la unidad de medida PMU, en el cuadro de texto de información se mostrará un mensaje que da luz verde al operario a realizar la trayectoria. Se deberá entonces desconectar el cable de red y el cable PPS que une a las dos unidades y se podrá desanclar la unidad de medida móvil para realizar la trayectoria desde el punto de origen hasta el de destino.

Para realizar la trayectoria se dispone de un tiempo máximo que se indica en el cuadro de texto "Tiempo (seg)" incluido para tal fin. A su vez, se indicará en todo momento el nivel restante de batería a través del cuadro de diálogo "Batería". La trayectoria desde la posición inicial a la posición final deberá realizarse moviendo la unidad PMU con sumo cuidado, con movimientos lentos para no sobrepasar el umbral de velocidad soportado por la unidad de medida y prestando especial atención al número de vueltas que se da sobre su propio eje.

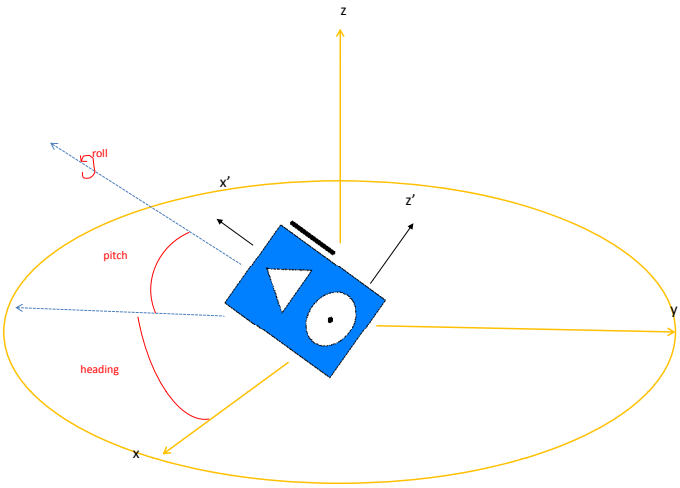

Figura A.18: Sistema de referencia de la unidad PMU

En la figura A.18 se muestra el sistema de referencia de la unidad de medida. Durante la trayectoria del experimento si se realiza un giro completo en torno al propio eje de la unidad (ángulo de heading) se avisará a través del cuadro de diálogo "Vueltas" con un número positivo si se trata de una vuelta en sentido horario y negativo si es antihorario. En el cuadro de diálogo de información se indicará al operador que deberá deshacer estas vueltas antes del montaje de la unidad en la posición final. A su vez, los ángulos pitch y roll no deben exceder el umbral especificado en el fichero de experimentos A.2.

Si el número de vueltas es distinto de cero y se exceden los ángulos de pitch y roll, el experimento no será válido y se mostrará un mensaje de error en el cuadro de diálogo pidiendo al operador que reinicie el experimento. Si por el contrario se exceden los ángulos de pitch y roll y el contador de vueltas es cero, se tomará como el final de la trayectoria y el experimento será válido.

Cuando el operario llegue al punto de destino con la unidad móvil, deberá anclarla en su posición para que quede fija. Tras esto y antes de pulsar el botón "Fin (conectar PPS!)" se deberán conectar de nuevo las dos unidades de medida mediante el cable de red, ya que se establecerá una nueva comunicación UDP entre ambas unidades, y el cable PPS. Sólo cuando los cables estén conectados se podrá pulsar sobre el botón "Fin (conectar PPS!)". Esta pulsación provocará que las unidades de medida calculen su posición final.

Será la unidad de medida móvil PMU la encargada de calcular el resultado final del experimento y de almacenarlo en el archivo correspondiente. Una vez terminado el experimento, en la interfaz de la unidad móvil se mostrará un mensaje informando del éxito o fracaso de la medida, según sea el caso, junto con el resultado del cálculo del error de desviación total entre los ángulos teóricos y prácticos de heading, pitch y roll expresados en miliradianes. Será entonces cuando la fase de medida haya concluido y se podrá desconectar el cable de red y el cable PPS.

Al término de un experimento, la unidad móvil creará si no existiera, o actualizará el fichero de resultados alojado en su memoria no volátil. Por cada experimento realizado, ya sea correcta o incorrectamente, se añadirá una línea al fichero de resultados con los datos oportunos que fueron comentados en el capítulo A.3.8. Este archivo contendrá por tanto tantas líneas como experimentos se hayan realizado, y podrá enviarse al servidor central utilizando el programa alojado en el PC que hace de servidor central. Este proceso se detalla en el siguiente apartado.

A.3.8. Descarga del fichero de resultados en el servidor central

Tras la realización de los experimentos necesarios, el último paso a dar será el de enviar el fichero de resultados desde la unidad móvil PMU, que es donde se encuentra almacenado, hacia el servidor central. Este paso no necesita ninguna acción sobre el programa "Viper.exe" de la unidad móvil y se puede realizar estando en cualquier ventana del programa.

Tras haber conectado de nuevo la unidad móvil con el servidor central mediante el cable de red A.1, el operario deberá pulsar sobre el botón "Descargar resultados" del cuadro de diálogo del programa "servidorPC.exe". Esto provocará que emerja un nuevo diálogo en el que se podrá elegir la ruta y nombre con el que se desea guardar el archivo en el disco duro del servidor central. Tras elegir la ruta, nombre deseado y pulsar sobre el botón "Aceptar" se realizará la transmisión del archivo.

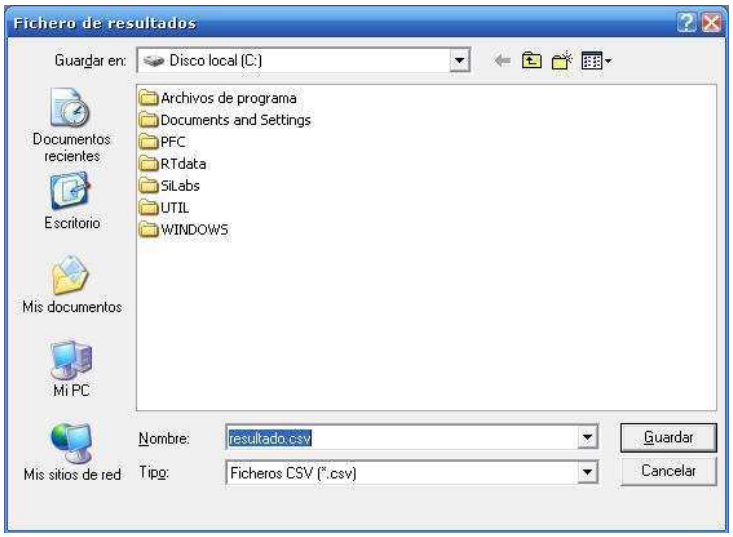

Figura A.19: Cuadro de diálogo para guardar el archivo resultado.csv

Una vez finalizado el envío del archivo, el sistema nos informará de si todo ha ido correctamente o si por el contrario se ha producido algún error.

Apéndice B

CDialogMedida

B.1. CDialogMedida::OnInitDialog()

```
BOOL CDialogMedida::OnInitDialog()
{
        int result;
        CDialog::OnInitDialog();

        // Punteros a los botones del cuadro de dialogo
        CButton* p_imu_alineada = (CButton *)GetDlgItem(IDC_IMU_ALINEADA);
        CButton* p_inicio = (CButton *)GetDlgItem(IDC_INICIO_TRAYECTORIA);
        CButton* p_fin = (CButton *)GetDlgItem(IDC_FIN_TRAYECTORIA);
        // En principio los botones de trayectoria estarán deshabilitados

        p_fin->EnableWindow(FALSE);
        p_inicio->EnableWindow(FALSE);

        CWnd::SetWindowText(_T("UNIDAD MOVIL"));
        // En principio estamos en reposo
        estado_sistema=REPOSO;
        estado_sistema_ventana_medida=TRUE;

        try{
        // Rellenamos la lista de experimentos a partir del fichero
        if (!lee_experimentos_y_equipos("/FlashDisk/Startup/experimentos.txt"))
        {
                // Si tenemos problemas a leer del fichero lo comunicamos y deshabilitamos botones
                p_imu_alineada->EnableWindow(FALSE);
                m_info="ERROR al leer fichero de experimentos";
```

```
                UpdateData(false);
                exit(0);
        }
        }
        catch(...)
        {
                        AfxMessageBox(_T("Verifique la sintaxis de los ficheros TXT"));
                        exit(0);
        }

        m_combo_experimento.SetCurSel(0);
        d_experimento=d_experimentos[0];
        lectura_bateria=GPIORead();
        m_bateria=calcular_nivel_bateria(lectura_bateria);

        m_Reset=TRUE;
        UpdateData(false);
        medida_en_marcha=false;

        // Lanzamos el temporizador de 1 segundo, para tener actualizaciones de bateria
        result=SetTimer(TIMER_1SEG_PMU,1000,0);

        return TRUE;  // return TRUE unless you set the focus to a control
                      // EXCEPTION: OCX Property Pages should return FALSE
}
```

B.2. CDialogMedida::lee_experimentos_y_equipos()

```
bool CDialogMedida::lee_experimentos_y_equipos(LPCSTR nombre_fichero)
{
        UpdateData(TRUE);
        bool fSuccess = FALSE;
        CString cadena;
        int newline='\n';
        // Buffer donde almacenamos cada línea leída
        char linea_leida[CARACTERES_LINEA];
        // Descriptor de fichero
        FILE *fichero;
        fichero = fopen(nombre_fichero,"rb");
        contador=0; // Reset de la lista indexada de experimentos
        contador_equipos=0; // Reset de la lista indexada de experimentos
        bool segunda_parte=false;
        char cadena_separadora[20]="#COMIENZO_EQUIPOS";
        int contador_iteraciones=0;
```

```
cadena_separadora[17]=13;
cadena_separadora[18]=10;
cadena_separadora[19]='\0';


// Reseteamos la lista del combobox
m_combo_experimento.ResetContent();
// Comprobamos que el fichero se ha podido abrir
if (fichero != NULL)
{
        // Recorremos el fichero entero hasta llegar al final, leyendo linea por linea
        while (!feof(fichero))
        {
                if((contador_iteraciones++)>=MAX_ITERACIONES)
                {
                        AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
                        exit(0);
                }
                // Leemos solo una línea
                fgets(linea_leida,CARACTERES_LINEA,fichero);
                if(feof(fichero)) break;
                if( strstr(linea_leida,"#FIN") ) break;

                if( !strcmp(cadena_separadora,linea_leida) )
                {
                        segunda_parte=true;
                }
                else
                {                       // Creamos la estrucutra que contiene los datos del experimento
                        if(!segunda_parte)
                        {
                        procesa_experimento(linea_leida);
                        // Lo metemos en un CString
                        cadena=linea_leida;
                        // Una vez que tenemos la línea leida añadimos la entrada al combobox
                        m_combo_experimento.AddString(cadena);
                        }
                        else procesa_equipo(linea_leida);
                        // Creamos la estrucutra que contiene los datos de los equipos
                }
        }
        fSuccess = TRUE;
}
else
{
        AfxMessageBox(_T("Fallo con el fichero"));
        fSuccess = FALSE;
}
// Cerramos el fichero
```

```
        fclose(fichero);

        if(!segunda_parte)
        {
                AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
                exit(0);
        }
        UpdateData(FALSE);
        return fSuccess;
}
```

B.3. CDialogMedida::procesa_experimento()

```
void CDialogMedida::procesa_experimento(char *linea_leida)
{
        int punto_coma=';';
        char *posicion;
        char indice_cadena[10];
        int indice_entero_leido;
        char error_maximo[10];
        char tiempo_maximo[10];
        char vel_maxima[10];
        char vel_alta[10];
        char num_alto_maximo[10];
        char error_adaptadores[10];
        char tipo_equipo[10];
        char equipo[20];
        char gadiru[20];
        char tamano_filtro_medidas[10];
        char limite_angulo_vueltas[10];

        // Busco donde esta el primer punto y coma
        posicion=strchr(linea_leida,punto_coma);
        if(posicion==NULL)
        {
                AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
                exit(0);
        }
        strncpy(indice_cadena,linea_leida,posicion-linea_leida);
        indice_cadena[posicion-linea_leida]='\0';
        indice_entero_leido=atoi(indice_cadena);
        linea_leida+=posicion-linea_leida+1;        // Situo el puntero de la linea justo despues del punto y coma

// Cargamos los datos del experimento en la estructura d_experimento
        d_experimentos[contador].indice=indice_entero_leido;
```

```
posicion=strchr(linea_leida,punto_coma);
if(posicion==NULL)
{
        AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
        exit(0);
}
strncpy(equipo,linea_leida,posicion-linea_leida);
equipo[posicion-linea_leida]='\0';
linea_leida+=posicion-linea_leida+1;
d_experimentos[contador].equipo=equipo;

d_experimentos[contador].indice=indice_entero_leido;
posicion=strchr(linea_leida,punto_coma);
if(posicion==NULL)
{
        AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
        exit(0);
}
strncpy(gadiru,linea_leida,posicion-linea_leida);
gadiru[posicion-linea_leida]='\0';
linea_leida+=posicion-linea_leida+1;
d_experimentos[contador].gadiru=gadiru;

posicion=strchr(linea_leida,punto_coma);
if(posicion==NULL)
{
        AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
        exit(0);
}
strncpy(error_maximo,linea_leida,posicion-linea_leida);
error_maximo[posicion-linea_leida]='\0';
linea_leida+=posicion-linea_leida+1;
d_experimentos[contador].error_maximo=atof(error_maximo);

posicion=strchr(linea_leida,punto_coma);
if(posicion==NULL)
{
        AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
        exit(0);
}
strncpy(tiempo_maximo,linea_leida,posicion-linea_leida);
tiempo_maximo[posicion-linea_leida]='\0';
linea_leida+=posicion-linea_leida+1;
d_experimentos[contador].tiempo_maximo=atof(tiempo_maximo);

posicion=strchr(linea_leida,punto_coma);
if(posicion==NULL)
{
        AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
```

```
        exit(0);
}
strncpy(vel_maxima,linea_leida,posicion-linea_leida);
vel_maxima[posicion-linea_leida]='\0';
linea_leida+=posicion-linea_leida+1;
d_experimentos[contador].vel_maxima=atof(vel_maxima);

posicion=strchr(linea_leida,punto_coma);
if(posicion==NULL)
{
        AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
        exit(0);
}
strncpy(vel_alta,linea_leida,posicion-linea_leida);
vel_alta[posicion-linea_leida]='\0';
linea_leida+=posicion-linea_leida+1;
d_experimentos[contador].vel_alta=atof(vel_alta);

posicion=strchr(linea_leida,punto_coma);
if(posicion==NULL)
{
        AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
        exit(0);
}
strncpy(num_alto_maximo,linea_leida,posicion-linea_leida);
num_alto_maximo[posicion-linea_leida]='\0';
linea_leida+=posicion-linea_leida+1;
d_experimentos[contador].num_alto_maximo=atof(num_alto_maximo);

posicion=strchr(linea_leida,punto_coma);
if(posicion==NULL)
{
        AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
        exit(0);
}
strncpy(error_adaptadores,linea_leida,posicion-linea_leida);
error_adaptadores[posicion-linea_leida]='\0';
linea_leida+=posicion-linea_leida+1;
d_experimentos[contador].error_adaptadores=atof(error_adaptadores);

posicion=strchr(linea_leida,punto_coma);
if(posicion==NULL)
{
        AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
        exit(0);
}
strncpy(tamano_filtro_medidas,linea_leida,posicion-linea_leida);
tamano_filtro_medidas[posicion-linea_leida]='\0';
linea_leida+=posicion-linea_leida+1;
```

```
        d_experimentos[contador].tamano_filtro_medidas=atoi(tamano_filtro_medidas);

        posicion=strchr(linea_leida,punto_coma);
        if(posicion==NULL)
        {
                AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
                exit(0);
        }
        strncpy(limite_angulo_vueltas,linea_leida,posicion-linea_leida);
        tamano_filtro_medidas[posicion-linea_leida]='\0';
        linea_leida+=posicion-linea_leida+1;
        d_experimentos[contador].limite_angulo_vueltas=atof(limite_angulo_vueltas)/180*M_PI;

        contador++;

}
```

B.4. CDialogMedida::procesa_equipo()

```
void CDialogMedida::procesa_equipo(char *linea_leida)
{
        int punto_coma=';';
        int newline='\n';
        char *posicion;
        int i=0;
        int j=0;
        int k=0;
        char indice_cadena[10];
        int indice_entero_leido;
        char nombre[100];
        char eje_rot[10];
        char grad_rot[20];
        double Rot0[3][3];
        double Rot1[3][3];
        double Aux[3][3];
        double CTeorico_ant[3][3];

        //        Puesta a cero de las matrices
                for (i=0;i<3;i++)
                        for (j=0;j<3;j++)
                                {
                                Rot0[i][j]= 0.0;
                                 Rot1[i][j]= 0.0;
                                Aux[i][j]= 0.0;
                        }
```

```
        // Busco donde esta el primer punto y coma
        posicion=strchr(linea_leida,punto_coma);
        if(posicion==NULL)
        {
                AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
                exit(0);
        }
        strncpy(indice_cadena,linea_leida,posicion-linea_leida);
        indice_cadena[posicion-linea_leida]='\0';
        indice_entero_leido=atoi(indice_cadena);
        // Situo el puntero de la linea justo despues del punto y coma
        linea_leida+=posicion-linea_leida+1;

// Cargamos los datos del equipo en la estructura d_equipos
        d_equipos[contador_equipos].indice=indice_entero_leido;
        posicion=strchr(linea_leida,punto_coma);
        if(posicion==NULL)
        {
                AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
                exit(0);
        }
        strncpy(nombre,linea_leida,posicion-linea_leida);
        nombre[posicion-linea_leida]='\0';
        linea_leida+=posicion-linea_leida+1;
        d_equipos[contador_equipos].nombre=nombre;

// Cargamos todas la rotaciones necesarias respecto al Gadiru_L
                        while(strchr(linea_leida,punto_coma))
                        {
                                posicion=strchr(linea_leida,punto_coma);
                                strncpy(eje_rot,linea_leida,posicion-linea_leida);
                                eje_rot[posicion-linea_leida]='\0';
                                linea_leida+=posicion-linea_leida+1;
                                d_equipos[contador_equipos].eje_rot[k]=atoi(eje_rot);

                                posicion=strchr(linea_leida,punto_coma);
                                if(posicion==NULL)
                                {
                                        AfxMessageBox(_T("Verifique la sintaxis del fichero \n experimentos.txt"));
                                        exit(0);
                                }
                                strncpy(grad_rot,linea_leida,posicion-linea_leida);
                                grad_rot[posicion-linea_leida]='\0';
                                linea_leida+=posicion-linea_leida+1;
                                d_equipos[contador_equipos].grad_rot[k]=atof(grad_rot);
                                crea_matriz_rotacion(Rot0,d_equipos[contador_equipos].eje_rot[k],
                                        d_equipos[contador_equipos].grad_rot[k]);
```

```
                                k++;
                                if(k>1)
                                {
                                producto(Rot0,Rot1,Aux);
                                copiar_matriz(Aux,Rot1);
                                }
                                else copiar_matriz(Rot0,Rot1);
                        }
                        //Almacenamos la matriz de rotación teórica del equipo respecto al Gadiru_L
                        copiar_matriz(Rot1,d_equipos[contador_equipos].Mat_Rot);
                        contador_equipos++;
}
```

B.5. CDialogMedida::OnInicioTrayectoria()

```
void CDialogMedida::OnInicioTrayectoria()
{
        int mseg=0;
        int mseg_rmu=0;
        struct tiempo t;
        int i;
        int result;
        int mseg_espera;
        tiempo_cumplido=0;

        // TODO: Add your control notification handler code here
        CButton* p_imu = (CButton *)GetDlgItem(IDC_IMU_ALINEADA);
        CButton* p_inicio = (CButton *)GetDlgItem(IDC_INICIO_TRAYECTORIA);
        CButton* p_fin = (CButton *)GetDlgItem(IDC_FIN_TRAYECTORIA);
        CButton* p_reset = (CButton *)GetDlgItem(IDC_IMU_RESET);

        p_inicio->EnableWindow(FALSE);
        p_fin->EnableWindow(FALSE);
        p_imu->EnableWindow(FALSE);
        p_reset->EnableWindow(FALSE);
        UpdateData(TRUE);

        // Creamos el contador que controla el tiempo maximo del experimento
        m_info="Espere...calculando posición inicial";
        UpdateData(false);

        // Operaciones Pre-PPS
        vaciar_buffer_y_cola();
        // Extraemos la hora actual para identificar correctamente el pulso PPS ya que su hora sera anterior
        viper_dialogo->extrae_trama_completa();
```

```
        tiempo_antes_de_pps=obtener_hora(&trama[21]);
        msec_antes_de_pps=tiempo_antes_de_pps.mseg+tiempo_antes_de_pps.seg*1000
                +tiempo_antes_de_pps.min*60*1000+tiempo_antes_de_pps.hora*60*60*1000;

        // Mandar PPS con 2 segundos de duración
        if (!mandarPPS(TIEMPO_PPS))
        {
                m_info="ERROR creando pulso PPS";
                UpdateData(false);
                // Cambio el estado del sistema
                estado_sistema=REPOSO;
                p_imu->EnableWindow(TRUE);
                return;
        }

        mseg_ini=GetTickCount();
        // coger primera trama posterior al PPS
        do{
                        viper_dialogo->extrae_trama_completa();
                        t=obtener_hora(&trama[21]);
                        // transformo la hora en milisegundos
                        mseg=t.mseg+t.seg*1000+t.min*60*1000+t.hora*60*60*1000;
                        mseg_espera=GetTickCount()-mseg_ini;
        } while ((mseg>=msec_antes_de_pps)&&(mseg_espera<10000));

        if(mseg_espera>=10000)
        {
                                AfxMessageBox(_T("Fallo PPS en IMU local"));
                                exit(0);
        }
// Repetir la operacion extrae_trama_completa(); N-1 veces

// Ponemos a 0 los valores angulares para calcular la media como suma parcial
        iniciales.orientacion_PMU.Q0=0;
        iniciales.orientacion_PMU.Q1=0;
        iniciales.orientacion_PMU.Q2=0;
        iniciales.orientacion_PMU.Q3=0;

        for (i=0; i< d_experimento.tamano_filtro_medidas; i++)
        {
        // Almacenamos los datos de la primera trama
                iniciales.orientacion_PMU.Q0+=viper_dialogo->lee_flotante(&trama[5]);
                iniciales.orientacion_PMU.Q1+=viper_dialogo->lee_flotante(&trama[9]);
                iniciales.orientacion_PMU.Q2+=viper_dialogo->lee_flotante(&trama[13]);
                iniciales.orientacion_PMU.Q3+=viper_dialogo->lee_flotante(&trama[17]);
                viper_dialogo->extrae_trama_completa();
        }

        // Dividimos para obtener el promedio
```

```
iniciales.orientacion_PMU.Q0/=d_experimento.tamano_filtro_medidas;
iniciales.orientacion_PMU.Q1/=d_experimento.tamano_filtro_medidas;
iniciales.orientacion_PMU.Q2/=d_experimento.tamano_filtro_medidas;
iniciales.orientacion_PMU.Q3/=d_experimento.tamano_filtro_medidas;
cuaterniones_a_Euler(&iniciales.orientacion_PMU.heading,&iniciales.orientacion_PMU.pitch,
        &iniciales.orientacion_PMU.roll,iniciales.orientacion_PMU.Q0,iniciales.orientacion_PMU.Q1,
        iniciales.orientacion_PMU.Q2,iniciales.orientacion_PMU.Q3);
iniciales.orientacion_PMU.t=t;

// Volcamos la medida actual en el dato miembro de la clase.
medida_anterior=iniciales.orientacion_PMU;
incremento_angular_acumulado=0; // para el conteo de las vueltas

// Si se consigue mandar el PPS, ordeno a RMU que calcule el promedio y lo devuelva
bufferTxUdp.cabecera = CALCULA_PROMEDIO_INICIAL;
bufferTxUdp.parametro= d_experimento.tamano_filtro_medidas;
m_socket_envia_datagramas->Send((char *)(&bufferTxUdp),sizeof(bufferTxUdp));

// Espero hasta que la RMU mande a la PMU o los datos o un error de PPS en la RMU
while (estado_sistema!=PROMEDIO_RMU_RECIBIDO && estado_sistema!=ERROR_PPS_RMU)
{
        procesa_mensajes();
}

//  Volcamos los angulos recibidos en la estructura inicial
mseg_rmu=bufferRxUdp.ang_promedio.t.mseg+bufferRxUdp.ang_promedio.t.seg*1000
        +bufferRxUdp.ang_promedio.t.min*60*1000+bufferRxUdp.ang_promedio.t.hora*60*60*1000;

TRACE(_T("Mseg RMU inicio: %d\n"),mseg_rmu);
TRACE(_T("Mseg pMU inicio: %d\n"),mseg);

// Tambien almaceno los datos de inicio de la RMU
iniciales.orientacion_RMU.Q0=bufferRxUdp.ang_promedio.Q0;
iniciales.orientacion_RMU.Q1=bufferRxUdp.ang_promedio.Q1;
iniciales.orientacion_RMU.Q2=bufferRxUdp.ang_promedio.Q2;
iniciales.orientacion_RMU.Q3=bufferRxUdp.ang_promedio.Q3;
cuaterniones_a_Euler(&iniciales.orientacion_RMU.heading,&iniciales.orientacion_RMU.pitch,
        &iniciales.orientacion_RMU.roll,iniciales.orientacion_RMU.Q0,iniciales.orientacion_RMU.Q1,
        iniciales.orientacion_RMU.Q2,iniciales.orientacion_RMU.Q3);
iniciales.orientacion_RMU.t=bufferRxUdp.ang_promedio.t;

TRACE(_T("Angulos iniciales calculados\n"));

// vacio completamente el buffer de entrada.
vaciar_buffer_y_cola();

// Una vez que tenemos los angulos promedios iniciales de RMU y PMU se puede comenzar medida
p_fin->EnableWindow(TRUE);
m_info="Ya puede liberar la PMU y empezar el montaje";
```

```
        // Cambio el estado del sistema
        estado_sistema=TRAYECTORIA;
        medida_en_marcha=true;
        estado_vueltas=VUELTAS_OK_ANGULO_OK;

        // Creación del timer para llamar a la funcción controla_velocidad
        result=SetTimer(TIMER_CONTROLA_VELOCIDAD,(1/FRECUENCIA_CONTROLA)*1000,0);

        UpdateData(FALSE);
        TRACE(_T("Se devuelve el control a Windows\n"));
}
```

B.6. CDialogMedida::mandarPPS()

```
bool CDialogMedida::mandarPPS (int milisegundos)
{
                int lineas=0xff;
                GPIOWrite(lineas);              // Pongo lineas a 1
                Sleep(milisegundos);        // Durante los segundos que necesite
                lineas=lineas^0xff;
                GPIOWrite(lineas);              // Pongo lineas a 0
                return TRUE;
}
```

B.7. CDialogMedidaRMU::calcula_promedio()

```
void CDialogMedidaRMU::calcula_promedio(int tamano_filtro_medidas, bool fin)
{
        int mseg;
        float heading=0;
        float roll=0;
        float pitch=0;
        float aux_heading=0;
        float aux_roll=0;
        float aux_pitch=0;
        int result;
        struct tiempo t;
        int i;
        static int vez=1;
        int mseg_ini,mseg_ellapsed;
```

```
mseg_ini=GetTickCount();

// coger primera trama posterior al PPS
do{
        viper_dialogo->extrae_trama_completa();
        t=obtener_hora(&trama[21]);
        mseg=t.mseg+t.seg*1000+t.min*60*1000+t.hora*60*60*1000;        // transformo la hora en milisegundos
        mseg_ellapsed=GetTickCount();
} while ((mseg>=msec_antes_de_pps)&&((mseg_ellapsed-mseg_ini)<10000));

if((mseg_ellapsed-mseg_ini)>=10000)
{
AfxMessageBox(_T("PPS no detectado"));
estado_sistema_ventana_medida=FALSE;
EndDialog(0);
return;
}

// Ponemos a 0 los valores angulares para calcular la media como suma parcial
bufferTxUdp.cabecera=PROMEDIO_RMU;
bufferTxUdp.ang_promedio.t=obtener_hora(&trama[21]);
bufferTxUdp.ang_promedio.Q0=0;
bufferTxUdp.ang_promedio.Q1=0;
bufferTxUdp.ang_promedio.Q2=0;
bufferTxUdp.ang_promedio.Q3=0;

for (i=0; i< tamano_filtro_medidas; i++)
{
// Almacenamos los datos de la primera trama
        bufferTxUdp.ang_promedio.Q0+=viper_dialogo->lee_flotante(&trama[5]);
        bufferTxUdp.ang_promedio.Q1+=viper_dialogo->lee_flotante(&trama[9]);
        bufferTxUdp.ang_promedio.Q2+=viper_dialogo->lee_flotante(&trama[13]);
        bufferTxUdp.ang_promedio.Q3+=viper_dialogo->lee_flotante(&trama[17]);
        viper_dialogo->extrae_trama_completa();
}

// Dividimos para obtener el promedio
bufferTxUdp.ang_promedio.Q0/=tamano_filtro_medidas;
bufferTxUdp.ang_promedio.Q1/=tamano_filtro_medidas;
bufferTxUdp.ang_promedio.Q2/=tamano_filtro_medidas;
bufferTxUdp.ang_promedio.Q3/=tamano_filtro_medidas;
cuaterniones_a_Euler(&heading,&pitch,&roll, bufferTxUdp.ang_promedio.Q0,
        bufferTxUdp.ang_promedio.Q1, bufferTxUdp.ang_promedio.Q2,bufferTxUdp.ang_promedio.Q3);

m_socket_envia_datagramas->Send((char *)(&bufferTxUdp),sizeof(bufferTxUdp));
// Cambio el estado del sistema
Sleep(2000);
m_info="Posición enviada";
UpdateData(false);
```

```
        // Operaciones Pre-PPS
        vaciar_buffer_y_cola();
// Extraemos la hora actual para identificar correctamente el pulso PPS ya que su hora sera anterior
        viper_dialogo->extrae_trama_completa();
        tiempo_antes_de_pps=obtener_hora(&trama[21]);
  msec_antes_de_pps=tiempo_antes_de_pps.mseg+tiempo_antes_de_pps.seg*1000+
    tiempo_antes_de_pps.min*60*1000+tiempo_antes_de_pps.hora*60*60*1000;

        TRACE(_T("Angulos enviados RPY: %.3f %.3f %.3f\n"),heading*RAD_GRA,pitch*RAD_GRA,roll*RAD_GRA);
        estado_sistema=REPOSO;

        if(fin)
        {
                estado_sistema_ventana_medida=FALSE;
                EndDialog(0);
        }
}
```

B.8. CDialogMedida::OnTimer()

```
void CDialogMedida::OnTimer(UINT nIDEvent)
{
        // TODO: Add your message handler code here and/or call default
        static int cuenta_segundos_bateria=0;

        switch (nIDEvent)
        {
        // Si se cumple el Timer de 5 minutos que espera el restart de la IMU:
        case TIMER_5MIN_PMU:
                estado_sistema=TIEMPO_CUMPLIDO;
                break;

        // Si se cumple el Timer de 1 minuto que controla el calculo del promedio inicial de la PMU
        case TIMER_1MIN_PMU:
                estado_sistema=TIEMPO_CUMPLIDO;
                break;

        // Actualizamos el tiempo restante del experimento cada 1 segundo. Fabio: comprobar si llega aqui
        case TIMER_1SEG_PMU:
        // Lectura de la batería cada 10 segundos. Independiente del modo
                if((cuenta_segundos_bateria++)>=10)
                {
                        lectura_bateria=GPIORead();
                        m_bateria=calcular_nivel_bateria(lectura_bateria);
```

```
                        UpdateData(false);
                        cuenta_segundos_bateria=0;
                }
                if(!medida_en_marcha) return;
                if (tiempo_cumplido) return; // Mensaje timer no necesario.
                // Gestión del tiempo
                mseg_fin=GetTickCount();
                m_tiempo_restante=d_experimento.tiempo_maximo-((mseg_fin-mseg_ini)/1000.0);

                if ((m_tiempo_restante<=0)&&(!tiempo_cumplido))
                {
                        tiempo_cumplido=1;
                        m_tiempo_restante=0;
                        AfxMessageBox(_T("Tiempo superado. Debe reiniciar la operación"));
                        estado_sistema=REPOSO;
                        m_info="Tiempo máximo superado";
                        CButton *p_inicio = (CButton *)GetDlgItem(IDC_INICIO_TRAYECTORIA);
                        CButton *p_fin = (CButton *)GetDlgItem(IDC_FIN_TRAYECTORIA);
                        CButton* p_imu = (CButton *)GetDlgItem(IDC_IMU_ALINEADA);
                        p_inicio->EnableWindow(FALSE); // Fabio Prueba
                        p_fin->EnableWindow(FALSE);
                        p_imu->EnableWindow(TRUE);
                        // vaciar la cola de mensajes
                }
                        UpdateData(false);
                break;

        // Controla Velocidad
        case TIMER_CONTROLA_VELOCIDAD:
                controla_velocidad();
                break;

        // Si se agota el tiempo máximo para el experimento
        case TIMER_T_MAX_EXP:
                break;
        }

        CDialog::OnTimer(nIDEvent);
}
```

B.9. CDialogMedida::controla_velocidad()

```
void CDialogMedida::controla_velocidad()
{
        static struct tiempo t_k0;
```

```
        static int mseg_k0;
        static struct tiempo t_k1;
        static int mseg_k1;
        double vel_h;
        double vel_r;
        double vel_p;
        uint a;
        uint mask=0x30;
        static int num_alto=0;
        int caso;
        static struct angulo angulo_k0;
        static struct angulo angulo_k1;
        double incremento_h;
        double incremento_r;
        double incremento_p;
        static BOOL existe_medida_anterior=false;
        static int seg\_espera\_sms=0;
        bool msgboxactivo=false;
        char cadena[100];

        // Pasos:
        // 1. Calcular la velocidad instantánea a partir de 2 tramas consecutivas
        // 2. Estimar el incremento angular acumulado en el último periodo de muestreo empleando medida_anterior y angulo_k0,
        //                y actualiza el estado del sistema.
        // 3. Compruebo los límites de velocidad

// PASO 1.
        // Capturamos la muestra actual
        viper_dialogo->extrae_trama_completa();
        vaciar_buffer_y_cola();
        t_k1=obtener_hora(&trama[21]);
        obtener_angulos_euler(angulo_k1);

        // transformo la hora en milisegundos
        mseg_k1=t_k1.mseg+t_k1.seg*1000+t_k1.min*60*1000+t_k1.hora*60*60*1000;

        // Solo se ejecuta 1 vez para tener un valor antiguo de angulo_k0
        if(!existe_medida_anterior)
        {
                incremento_angular_acumulado=0;
                existe_medida_anterior=true;
                angulo_k0=angulo_k1; // Actualizo la medida anterior
                mseg_k0=mseg_k1;
                t_k0=t_k1;
                return;
        }
        // Calculo la velocidad instantanea considerando las operaciones modulo
        incremento_h=calcula_incremento_modulo(angulo_k1.heading,angulo_k0.heading,-M_PI,M_PI);
        incremento_r=calcula_incremento_modulo(angulo_k1.roll,angulo_k0.roll,-M_PI,M_PI);
```

```
        incremento_p=calcula_incremento_modulo(angulo_k1.pitch,angulo_k0.pitch,-M_PI/2.0,M_PI/2.0);

        vel_h=incremento_h/(mseg_k1-mseg_k0);
        vel_r=incremento_r/(mseg_k1-mseg_k0);
        vel_p=incremento_p/(mseg_k1-mseg_k0);
        // Actualizo la medida anterior
        angulo_k0=angulo_k1;
        mseg_k0=mseg_k1;

        // De este modo se obtiene un heading centrado en el angulo inicial y no sometido a la operación modulo

// PASO 2. Calculo el giro en heading acumulado
        incremento_angular_acumulado+=incremento_h;

        if(incremento_angular_acumulado>0)
        {
                contador_vueltas=floor(10*(incremento_angular_acumulado)/(2*M_PI))/10;
        }
        else contador_vueltas=ceil(10*(incremento_angular_acumulado)/(2*M_PI))/10;

        sprintf(cadena,"%.1f",contador_vueltas);
        m_vueltas=cadena;

        // Revisa que el estado de la trayectoria es correcto y envía un mensaje
        if((abs(angulo_k1.pitch)<(d_experimento.limite_angulo_vueltas))
                &&(abs(angulo_k1.roll)<(d_experimento.limite_angulo_vueltas))&& abs(contador_vueltas)<1.0)
        {
                if(estado_vueltas==VUELTAS_KO_ANGULO_OK)
                {
                        m_info="Trayectoria correcta...";
                        UpdateData(FALSE);
                        estado_vueltas=VUELTAS_OK_ANGULO_OK;
                }
        }

        if((abs(angulo_k1.pitch)<(d_experimento.limite_angulo_vueltas))
                &&(abs(angulo_k1.roll)<(d_experimento.limite_angulo_vueltas))&&abs(contador_vueltas)>=1.0)
        {
                if(estado_vueltas==VUELTAS_OK_ANGULO_OK)
                {
                        m_info="Nº vueltas no nulo: deshacer vueltas.";
                        UpdateData(FALSE);
                        estado_vueltas=VUELTAS_KO_ANGULO_OK;
                }
        }

        if(((abs(angulo_k1.pitch)>(d_experimento.limite_angulo_vueltas))
                ||(abs(angulo_k1.roll)>(d_experimento.limite_angulo_vueltas)))&&abs(contador_vueltas)<1.0)
        {
```

```
                if(estado_vueltas==VUELTAS_OK_ANGULO_OK)
                {
                        m_info="Trayectoria correcta. Pulsar fin";
                        UpdateData(FALSE);
                        estado_vueltas=VUELTAS_OK_ANGULO_KO;
                }
        }

        if(((abs(angulo_k1.pitch)>(d_experimento.limite_angulo_vueltas))
                ||(abs(angulo_k1.roll)>(d_experimento.limite_angulo_vueltas)))&&abs(contador_vueltas)>=1.0)
        {
                if(estado_vueltas==VUELTAS_KO_ANGULO_OK)
                {
                cancela_trayectoria();
                if(msgboxactivo!=true)
                        {
                        msgboxactivo=true;
                        AfxMessageBox(_T("Exceso de vueltas. Reinicie."));
                        msgboxactivo=false;
                        }
                return;
                }
        }

// PASO 3. Compruebo límites de velocidad
        // Si supero la velocidad máxima instantanea
        if (vel_h>d_experimento.vel_maxima || vel_r>d_experimento.vel_maxima
                ||vel_p>d_experimento.vel_maxima)
        {
                cancela_trayectoria();
                if(msgboxactivo!=true)
                {
                        msgboxactivo=true;
                        AfxMessageBox(_T("Velocidad máxima instantanea superada"));
                        msgboxactivo=false;
                }
                return;
        }

        // Si supero la velocidad alta lo contabilizo
        if (vel_h>d_experimento.vel_alta || vel_r>d_experimento.vel_alta
                ||vel_p>d_experimento.vel_alta)
        {
                num_alto++;
        }

        // Si supero la velocidad máxima acumulada
        if (num_alto > d_experimento.num_alto_maximo)
        {
```

```
                cancela_trayectoria();
                if(msgboxactivo!=true)
                {
                        msgboxactivo=true;
                        AfxMessageBox(_T("Velocidad máxima acumulada superada"));
                        msgboxactivo=false;
                }
                return;
        }

        // Si se detectan error en el STATUS
        a=*((uint*)(&trama[1]));
        if((a&mask) )  // mask=0x30;
        {
                cancela_trayectoria();
                if(msgboxactivo!=true)
                {
                        msgboxactivo=true;
                        AfxMessageBox(_T("Error en el byte de STATUS"));
                        msgboxactivo=false;
                }
                return;
        }
}
```

B.10. CDialogMedida::calcula_incremento_modulo()

```
double CDialogMedida::calcula_incremento_modulo(double theta1,double theta0,double minimo,double maximo)
{
        double salto;
        salto=maximo-minimo;
        // Calculamos el incremento angular realizando un ajuste si el modulo del mismo
        nos hace pensar que se ha producido desborde
        if(abs(theta1-theta0)>(salto*0.5))
        {
                if(theta1>theta0)
                        {theta0+=salto;}
                else theta1+=salto;
        }
        return (theta1-theta0);
}
```

B.11. CDialogMedida::cancela_trayectoria()

```
void CDialogMedida::cancela_trayectoria()
{
        // Punteros a los botones del cuadro de dialogo
        CButton* p_imu_alineada = (CButton *)GetDlgItem(IDC_IMU_ALINEADA);
        CButton* p_inicio = (CButton *)GetDlgItem(IDC_INICIO_TRAYECTORIA);
        CButton* p_fin = (CButton *)GetDlgItem(IDC_FIN_TRAYECTORIA);
        CButton* p_reset = (CButton *)GetDlgItem(IDC_IMU_RESET);

        p_fin->EnableWindow(FALSE);
        p_inicio->EnableWindow(FALSE);
        p_imu_alineada->EnableWindow(TRUE);
        p_reset->EnableWindow(TRUE);

        estado_sistema=REPOSO;
        medida_en_marcha=false;

        // Estado controles
        lectura_bateria=GPIORead();
        m_bateria=calcular_nivel_bateria(lectura_bateria);
        m_Reset=TRUE;
        UpdateData(false);

        KillTimer(TIMER_CONTROLA_VELOCIDAD);
        return;
}
```

B.12. CDialogMedida::OnFinTrayectoria()

```
void CDialogMedida::OnFinTrayectoria()
{
        int mseg=0;
        int mseg_rmu=0;
        int i;
        float suma_heading=0;
        float suma_roll=0;
        float suma_pitch=0;
        float aux_heading=0;
        float aux_roll=0;
        float aux_pitch=0;
        double error_total=0;
        struct tiempo t;
```

```
	// Variable de estado
	medida_en_marcha=false;

	KillTimer(TIMER_CONTROLA_VELOCIDAD);

	// Espera de 2 segundos
	m_info="Esperando 2 segundos de seguridad...";
	UpdateData(false);
	Sleep(2000);

	CButton *p_inicio = (CButton *)GetDlgItem(IDC_INICIO_TRAYECTORIA);
	CButton *p_fin = (CButton *)GetDlgItem(IDC_FIN_TRAYECTORIA);
	CButton *p_imu = (CButton *)GetDlgItem(IDC_IMU_ALINEADA);
	p_inicio->EnableWindow(FALSE);
	p_fin->EnableWindow(FALSE);
	p_imu->EnableWindow(FALSE);

	m_info="Espere...calculando posición final";
	UpdateData(false);

	estado_sistema=CALCULANDO_POSICION_FINAL;

	// Operaciones Pre-PPS
	vaciar_buffer_y_cola();
	// Extraemos la hora actual para identificar correctamente el pulso PPS ya que su hora sera anterior
	viper_dialogo->extrae_trama_completa();
	tiempo_antes_de_pps=obtener_hora(&trama[21]);
msec_antes_de_pps=tiempo_antes_de_pps.mseg+tiempo_antes_de_pps.seg*1000
 +tiempo_antes_de_pps.min*60*1000+tiempo_antes_de_pps.hora*60*60*1000;

	// Enviamos PPS a las 2 IMUs
	if (!mandarPPS(TIEMPO_PPS))
	{
		m_info="Error creando pulso PPS";
		UpdateData(false);
		estado_sistema=REPOSO;
		return;
	}
	mseg_fin=GetTickCount();

	// coger primera trama posterior al PPS
	do{
	viper_dialogo->extrae_trama_completa();
	t=obtener_hora(&trama[21]);
	// transformo la hora en milisegundos
	mseg=t.mseg+t.seg*1000+t.min*60*1000+t.hora*60*60*1000;
	} while (mseg>=msec_antes_de_pps);

	// Ponemos a 0 los valores angulares para calcular la media como suma parcial
```

```
	finales.orientacion_PMU.Q0=0;
	finales.orientacion_PMU.Q1=0;
	finales.orientacion_PMU.Q2=0;
	finales.orientacion_PMU.Q3=0;

	for (i=0; i< d_experimento.tamano_filtro_medidas; i++)
	{
	// Almacenamos los datos de la primera trama
	finales.orientacion_PMU.Q0+=viper_dialogo->lee_flotante(&trama[5]);
	finales.orientacion_PMU.Q1+=viper_dialogo->lee_flotante(&trama[9]);
	finales.orientacion_PMU.Q2+=viper_dialogo->lee_flotante(&trama[13]);
	finales.orientacion_PMU.Q3+=viper_dialogo->lee_flotante(&trama[17]);
	viper_dialogo->extrae_trama_completa();
	}

	// Dividimos para obtener el promedio
	finales.orientacion_PMU.Q0/=d_experimento.tamano_filtro_medidas;
	finales.orientacion_PMU.Q1/=d_experimento.tamano_filtro_medidas;
	finales.orientacion_PMU.Q2/=d_experimento.tamano_filtro_medidas;
	finales.orientacion_PMU.Q3/=d_experimento.tamano_filtro_medidas;

	cuaterniones_a_Euler(&finales.orientacion_PMU.heading,&finales.orientacion_PMU.pitch,
		&finales.orientacion_PMU.roll,finales.orientacion_PMU.Q0,finales.orientacion_PMU.Q1,
		finales.orientacion_PMU.Q2,finales.orientacion_PMU.Q3);
	finales.orientacion_PMU.t=t;

	// Si se consigue mandar el PPS, ordeno a RMU que calcule el promedio y lo devulva
	bufferTxUdp.cabecera = CALCULA_PROMEDIO_FINAL;
	bufferTxUdp.parametro= d_experimento.tamano_filtro_medidas;
	m_socket_envia_datagramas->Send((char *)(&bufferTxUdp),sizeof(bufferTxUdp));

// Espero hasta que la RMU mande a la PMU o los datos o un error de PPS en la RMU
	while (estado_sistema!=PROMEDIO_RMU_RECIBIDO && estado_sistema!=ERROR_PPS_RMU)
	{
		procesa_mensajes();
	}
	mseg_rmu=bufferRxUdp.ang_promedio.t.mseg+bufferRxUdp.ang_promedio.t.seg*1000
		+bufferRxUdp.ang_promedio.t.min*60*1000+bufferRxUdp.ang_promedio.t.hora*60*60*1000;
	TRACE(_T("Mseg RMU fin: %d\n"),mseg_rmu);
	TRACE(_T("Mseg PMU fin: %d\n"),mseg);

	// Tambien almaceno dos datos finales de la RMU
	finales.orientacion_RMU.Q0=bufferRxUdp.ang_promedio.Q0;
	finales.orientacion_RMU.Q1=bufferRxUdp.ang_promedio.Q1;
	finales.orientacion_RMU.Q2=bufferRxUdp.ang_promedio.Q2;
	finales.orientacion_RMU.Q3=bufferRxUdp.ang_promedio.Q3;
	cuaterniones_a_Euler(&finales.orientacion_RMU.heading,&finales.orientacion_RMU.pitch,
		&finales.orientacion_RMU.roll,finales.orientacion_RMU.Q0,finales.orientacion_RMU.Q1,
		finales.orientacion_RMU.Q2,finales.orientacion_RMU.Q3);
```

```
        finales.orientacion_RMU.t=bufferRxUdp.ang_promedio.t;

        // Se calcula el error total
        calcular_rotacion_total(iniciales, finales,  RPY_final);
        r_experimento.cod_error=0;

        // Se almacenan los resultados en fichero de log.
        r_experimento.hora=CTime::GetCurrentTime();
        r_experimento.indice=m_combo_experimento.GetCurSel() + 1; // Ya que el primer elemento del combo es 0
        r_experimento.error_total=error_total;

        KillTimer(TIMER_T_MAX_EXP);
        KillTimer(TIMER_1SEG_PMU);
        KillTimer(TIMER_1MIN_PMU);

        if (!escribe_resultados("/FlashDisk/Startup/resultados.csv", r_experimento))
                AfxMessageBox(_T("ERROR al almacenar los resultados"));
        EndDialog(0);
}
```

B.13. CDialogMedida::calcular_rotacion_total()

```
void CDialogMedida::calcular_rotacion_total(angulos_promedios &iniciales, angulos_promedios &finales, double RPY[3])
{
        double CRMUi[3][3];
        double CPMUi[3][3];
        double CRMUf[3][3];
        double CPMUf[3][3];
        double CRMUfT[3][3];
        double CPMUiT[3][3];
        double CTeorico[3][3];
        double CRMUaPMU[3][3];
        double CDesviacioncoordsIMU[3][3];
        double CDesviacioncoordsAvion[3][3];
        double CcoordsAvion[3][3];
        double CTeoricoT[3][3];
        double CcoordsAvionT[3][3];
        double Caux[3][3];
        double C1[3][3];
        double C2[3][3];

        /* Creacion matrices */
        cuaterniones_a_MatrizRotacion(CRMUi,iniciales.orientacion_RMU);
        cuaterniones_a_MatrizRotacion(CPMUi,iniciales.orientacion_PMU);
        cuaterniones_a_MatrizRotacion(CRMUf,finales.orientacion_RMU);
```

```
        cuaterniones_a_MatrizRotacion(CPMUf,finales.orientacion_PMU);

        /* Matrices que dependen de equipos y ubicacion */
        calcular_matrices_dependientes_posicion(CTeorico, CcoordsAvion);

        /* Rotacion neta RMU -> PMU en coordenadas IXSEA */
        transponer(CPMUi,CPMUiT);
        transponer(CRMUf,CRMUfT);
        producto(CPMUf,CPMUiT,C1);
        producto(CRMUi,CRMUfT,C2);
        producto(C1,C2,CRMUaPMU);

        /* Desviacion de la rotacion neta RMU -> PMU con respecto a teorico en ejes IXSEA */
        transponer(CTeorico,CTeoricoT);
        producto(CRMUaPMU,CTeoricoT,CDesviacioncoordsIMU);

        /* Desviacion de la totacion neta RMU -> PMU con respecto a teorico en ejes avion */
        transponer(CcoordsAvion,CcoordsAvionT);
        producto(CcoordsAvion,CDesviacioncoordsIMU,Caux);
        producto(Caux,CcoordsAvionT,CDesviacioncoordsAvion);

        RPY[0]=atan2(CDesviacioncoordsAvion[1][2],CDesviacioncoordsAvion[2][2]);
        RPY[1]=-asin(CDesviacioncoordsAvion[0][2]);
        RPY[2]=atan2(CDesviacioncoordsAvion[0][1],CDesviacioncoordsAvion[0][0]);

        sprintf(cadena,"Medida finalizada. Desviaciones finales (R-P-Y) en mrads:\n %f %f %f \n ",

        CString cad(cadena);
        AfxMessageBox(cad,MB_OK);
}
```

B.14. CDialogMedida::calcular_matrices_dependientes_posicion

```
void CDialogMedida::calcular_matrices_dependientes_posicion(double CTeorico[3][3], double CcoordsAvion[3][3])
{
        int i,j;
        int contador_busqueda=0;
        double CAvion_GL[3][3];
        double CTeoricoT[3][3];
        double Rot_GR[3][3];
        double Aux[3][3];

        //        Puesta a cero de la matrices
        for (i=0;i<3;i++)
                for (j=0;j<3;j++)
```

```
                {
                        CTeorico[i][j]= 0.0;
                        CTeoricoT[i][j]= 0.0;
                        CcoordsAvion[i][j]=0.0;
                        CAvion_GL[i][j]=0.0;
                        Rot_GR[i][j]=0.0;
                        Aux[i][j]=0.0;
                }

        // Coordenadas de Gaduru_L respecto al avion
        // esta matriz es una rotación sobre el eje X de 180 grados
        crea_matriz_rotacion(CAvion_GL,1,180);

        //Buscamos la matriz teórica del equipo y la volcamos en CTeorico
        while(d_equipos[contador_busqueda].nombre!=d_experimento.equipo) contador_busqueda++;

        copiar_matriz(d_equipos[contador_busqueda].Mat_Rot, CTeorico);
        //Creamos la matriz de rotación del equipo respecto al avion
        transponer(CTeorico,CTeoricoT);
        producto(CAvion_GL,CTeoricoT,CcoordsAvion);

   /* GADIRU_L */
        if(d_experimento.gadiru=="GADIRU_L")
                        GADIRU_L=TRUE;
        else GADIRU_L=FALSE;

  if(GADIRU_L==FALSE) /* GADIRU_R */
   {
                // Si se trata del Gadiru_R realizamos una rotación
                // en la matriz teórica del equipo de 180 grados sobre el eje z
                crea_matriz_rotacion(Rot_GR,3,180);
                producto(CTeorico,Rot_GR,Aux);
                copiar_matriz(Aux, CTeorico);
   }
}
```

B.15. Funciones de cálculo matemático

```
void copiar_matriz(double origen[3][3], double destino [3][3])
{
        int i,j;

        for (i=0;i<3;i++)
                for (j=0;j<3;j++)
                {
```

```
                destino[i][j]= origen[i][j];
                }
}

void crea_matriz_rotacion(double Rot0[3][3], int eje, double grados)
{
        int i,j;
        double radianes=grados/180.0*M_PI;
        //        Puesta a cero de las matrices
        for (i=0;i<3;i++)
                for (j=0;j<3;j++)
                        {
                        Rot0[i][j]= 0.0;
                }

        // Creamos las matrices de rotacion dependiendo del eje de rotación
        switch(eje)
        {
        case 1:
                //Rotación del eje X
                Rot0[0][0] = 1;
                Rot0[1][1] = cos(radianes);
                Rot0[2][2] = cos(radianes);
                Rot0[1][2] = sin(radianes);
                Rot0[2][1] = -sin(radianes);
                break;

        case 2:
                //Rotación del eje Y
                Rot0[0][0] = cos(radianes);
                Rot0[0][2] = -sin(radianes);
                Rot0[1][1] = 1;
                Rot0[2][0] = sin(radianes);
                Rot0[2][2] = cos(radianes);
                break;

        case 3:
                //Rotación del eje Z
                Rot0[0][0] = cos(radianes);
                Rot0[0][1] = sin(radianes);
                Rot0[1][0] = -sin(radianes);
                Rot0[1][1] = cos(radianes);
                Rot0[2][2] = 1;
                break;
        }
}
```

B.16. Funciones de procesado de mensajes

```
void procesa_mensajes(void)
{
        MSG message;

        while(PeekMessage(&message,NULL,0,0,PM_REMOVE))
        {
        TranslateMessage(&message);
        DispatchMessage(&message);
        }
}

WINUSERAPI
BOOL
WINAPI
PeekMessageW(
    PMSG pMsg,
    HWND hWnd ,
    UINT wMsgFilterMin,
    UINT wMsgFilterMax,
    UINT wRemoveMsg);
WINUSERAPI
BOOL
WINAPI
PeekMessageA(
    PMSG pMsg,
    HWND hWnd ,
    UINT wMsgFilterMin,
    UINT wMsgFilterMax,
    UINT wRemoveMsg);
#ifdef UNICODE
#define PeekMessage  PeekMessageW
#else
#define PeekMessage  PeekMessageA
#endif // !UNICODE
/* PeekMessage() Options */
#define PM_NOREMOVE         0x0000
#define PM_REMOVE           0x0001
#define PM_NOYIELD          0x0002

BOOL
WINAPI
TranslateMessage(
    CONST MSG   *pMsg
    );

WINUSERAPI
```

```
LONG
WINAPI
DispatchMessageA(
    CONST MSG *lpMsg);
WINUSERAPI
LONG
WINAPI
DispatchMessageW(
    CONST MSG *lpMsg);
#ifdef UNICODE
#define DispatchMessage  DispatchMessageW
#else
#define DispatchMessage  DispatchMessageA
#endif // !UNICODE
```

Apéndice C

CViperDlg

C.1. CViperDlg::OnInitDialog()

```
BOOL CViperDlg::OnInitDialog()
{
        CString puerto;
        CDialog::OnInitDialog();
        DWORD lectura_bateria;

        SetIcon(m_hIcon, TRUE);                         // Set big icon
        SetIcon(m_hIcon, FALSE);                // Set small icon
        CButton* p_medida = (CButton *)GetDlgItem(IDC_MEDIDA);
        CenterWindow(GetDesktopWindow());        // center to the hpc screen

// Creación del puntero global del ViperDlg
        viper_dialogo=this;

// Creación de un icono en el escritorio
        CreateProcess(_T("/FlashDisk/StartUp/link.bat"),NULL,NULL,NULL,FALSE,0,NULL,NULL,NULL,
                &ProcInfo);
        CopyFile( _T("/FlashDisk/Viper.lnk"),_T("/Windows/Desktop/Viper.lnk"),FALSE);

// Leemos la configuración desde un fichero de texto
        if ( lee_fichero("/FlashDisk/Startup/conf.txt",&configuracion_sistema) )
        {
                // Colocamos en el título de la ventana si somos Referencia o Móvil
                if (configuracion_sistema.viper_referencia)
                        CWnd::SetWindowText(_T("UNIDAD DE REFERENCIA"));
                else
```

```
        CWnd::SetWindowText(_T("UNIDAD MOVIL"));

switch(configuracion_sistema.port_serie)
{
case 1:
        puerto = "COM1:";
        break;
case 2:
        puerto = "COM2:";
        break;
case 3:
        puerto = "COM3:";
        break;
case 4:
        puerto = "COM4:";
        break;
case 5:
        puerto = "COM5:";
        break;
default: break;
}

TRACE(_T("Abriendo puerto:"));
// Configuración del puerto serie
if(!SerialPort.OpenPort(puerto)) PostQuitMessage(0);
if(!SerialPortReset.OpenPort("COM4:")) PostQuitMessage(0);
TRACE(_T("Puertos abiertos: %d\n"),configuracion_sistema.port_serie);

SerialPort.ConfigurePort(CBR_115200, 8, false, NOPARITY , ONESTOPBIT);
SerialPort.SetCommunicationTimeouts(MAXDWORD,0,0,0,0);
SerialPortReset.ConfigurePort(CBR_57600, 8, false, ODDPARITY , TWOSTOPBITS);

// Se crea el timer de lectura del puerto serie a la frecuencia definida
Timer_alm = SetTimer(TIMER_ALM, (1/FRECUENCIA_ALM)*1000, 0);
if (!Timer_alm)
{
        AfxMessageBox(_T("ERROR: Timer_alm"));
        PostQuitMessage(0);
}

// Creación de la clase CMySocket y conexión con el PC
// Dos sockets son para la conexión UDP y el otro para la TCP
m_socket_envia_datagramas = new CMySocket(this);
m_socket_recibe_datagramas = new CMySocket(this);
m_socket_comunicacion = new CMySocket(this);

// Llamadas a Create
m_socket_comunicacion->Create(SOCK_STREAM);
if (!m_socket_comunicacion->Accept(PUERTO_TCP_RESULTADOS))
```

```
        {
                AfxMessageBox(_T("ERROR: Accept TCP"));
        }

        // Ponemos socket asíncrono a la escucha
        if (!m_socket_recibe_datagramas->Create(SOCK_DGRAM) == TRUE)
        AfxMessageBox(_T("ERROR: Create UDP"));
        else if (!m_socket_recibe_datagramas->Accept(3000))
        AfxMessageBox(_T("ERROR: Accept UDP"));

        // Comprobamos que podemos conectarnos al equipo
        TRACE(_T("Sockets Creados\n"));
        if (m_socket_envia_datagramas->Create(SOCK_DGRAM) == TRUE)
        {
                // Realizamos la conexión. Esta conexión es virtual ya que estamos usando UDP y este protocolo
                // es connectionless. Realizamos esto para guardar los datos del ordenador remoto.
                if (!m_socket_envia_datagramas->Connect(configuracion_sistema.ip_viper_remota,
                configuracion_sistema.port_udp) )
                AfxMessageBox(_T("ERROR: Connect UDP"));
                else estado_sistema = REPOSO;
        }
        else
                AfxMessageBox(_T("ERROR: Create UDP"));
}
else
{
// Si no podemos leer el fichero de configuracion lo notificamos
        p_medida->EnableWindow(FALSE);
        AfxMessageBox(_T("ERROR al leer el fichero de configuración"));
}

if ( fopen("/FlashDisk/StartUp/experimentos.txt","rb") == NULL )
{
        // Si no podemos abrir el fichero de experimentos lo notificamos
        p_medida->EnableWindow(FALSE);
        AfxMessageBox(_T("ERROR al leer el fichero de experimentos"));
}

TRACE(_T("Abriendo líneas GPIO"));
if (GPIOInit())
{
        TRACE(_T("Líneas GPIO abiertas"));
        GPIOWrite(0x00); //Ponemos a 0 todas las líneas GPIO para evitar estados inestables
}
else TRACE(_T("Error apertura Líneas GPIO\n"));

// Actualización lectura batería
lectura_bateria=GPIORead();
m_bateria_general=((lectura_bateria&0x01)<<1)+((lectura_bateria&0x02)>>1);
```

```
        UpdateData(false);
        return TRUE;  // return TRUE  unless you set the focus to a control
}
```

C.2. CViperDlg::OnTimer()

```
void CViperDlg::OnTimer(UINT nIDEvent)
{
        BYTE bufferByte;
        static int cuenta_refrescos_visuales=0;
        static int tope_diez_segundos=10*FRECUENCIA_ALM;

        if (flag_desborde) return;

        switch(nIDEvent)
        {
        int lectura_bateria;
        // Si se cumple el timer extraigo del puerto serie todos los datos que haya y los almaceno en la cola_circular
        case TIMER_ALM:
        // Extraigo del puerto serie todos los datos que haya y los almaceno en la cola_circular
                        while ( SerialPort.ReadByte(bufferByte))
                        {
                                alm_cola_c(bufferByte);
                        }
                        if(cuenta_refrescos_visuales++>=FRECUENCIA_ALM) //1 SEGUNDO
                        {
                        // Actualización lectura batería
                        lectura_bateria=GPIORead();
                        m_bateria_general=((lectura_bateria&0x01)<<1)+((lectura_bateria&0x02)>>1);
                        UpdateData(false);
                        cuenta_refrescos_visuales=0;
                        }
                break;
        }
        CDialog::OnTimer(nIDEvent);
}
```

C.3. CViperDlg::extrae_trama_completa()

```
int CViperDlg::extrae_trama_completa(void)
{
        BYTE bufferByte;
```

```
        // Espera a que se reciba una trama completa
        while ( extrae_trama()!=TRAMA_CORRECTA )
        {
                SerialPort.ReadByte(bufferByte);
                alm_cola_c(bufferByte);
        }
        return 1;
}

int CViperDlg::extrae_trama()
{
        BYTE bufferByte=0;
        unsigned short crc=0;
        unsigned short crc_recibido=0;
        unsigned short crc_byte1=0;
        unsigned short crc_byte2=0;
        int i;
        int ind_sig_dolar=0;
        int restantes=0;

        restantes=(ind_alm-ind_rec);

        if(restantes==0)
        {
                return TRAMA_INCOMPLETA;
        }

        if (restantes < 0)
                restantes+=TAM_COLA_C;

        if (restantes >= LONG_TRAMA)
        {
                // Busco la cabecera de trama. Paro cuando la encuentre o cuando se vacie la cola_circular
                while ((bufferByte != 0x24) && (ind_rec!=ind_alm))
                {
                        bufferByte=rec_cola_c();
                }
                // Si despues de encontrar la cabecera compruebo que hay menos de 22 bytes en la
                                cola_circular, estamos en el caso de TRAMA_INCOMPLETA.

                restantes=(ind_alm-ind_rec);
                if (restantes < 0)
                        restantes+=TAM_COLA_C;

                if(restantes<LONG_TRAMA)
                {
                        if (ind_rec == 0)
                                ind_rec=TAM_COLA_C-1;
                        else
```

```
                        ind_rec--;
                return TRAMA_INCOMPLETA;
        }

        // En caso contrario extraigo una trama completa
        ind_sig_dolar=ind_rec;
        trama[0]=bufferByte;
        for (i=1; i<LONG_TRAMA; i++)
        {
                trama[i]=rec_cola_c();
        }
        // Compruebo el crc recibido en la trama con el calculado
        crc=blkcrc (&trama[1], LONG_TRAMA-3);
        crc_byte1=(unsigned short)(*(&trama[LONG_TRAMA-2]));
        crc_byte2=(unsigned short)(*(&trama[LONG_TRAMA-1]));
        crc_recibido=((crc_byte1 << 8) | crc_byte2);
        if (crc==crc_recibido)
        {
                return TRAMA_CORRECTA;
        }
        else
        {
                ind_rec=ind_sig_dolar;
                return TRAMA_ERRONEA;
        }
    }
    else
        return TRAMA_INCOMPLETA;
}
```

C.4. CViperDlg::resetea_IMU()

```
int CViperDlg::resetea_IMU()
{
    char restart[40];
    int contador;

    sprintf (restart, "$PIXSE,CONFIG,WAKEUP*40\r\n");

    for(contador=0;contador<20;contador++)
    {
        SerialPortReset.WriteByte(restart);
        Sleep(200);
        procesa_mensajes();
    }
```

```
        Sleep(500);
        sprintf (restart, "$PIXSE,CONFIG,RESET_*57\r\n");

        // 100 intentos de reset (20 segundos)
        for(contador=0;contador<100;contador++)
        {
                SerialPortReset.WriteByte(restart);
                Sleep(200);
                procesa_mensajes();
                // chequeamos el reset válido
                if(comprueba_reset())
                {
                        TRACE(_T("Reset IMU enviado\n"));
                        Sleep(500);
                        vaciar_buffer_y_cola();
                        return 1;
                }
        }
        TRACE(_T("Reset IMU fallido\n"));
        return 0;
}

BOOL CSerialCom::WriteByte(char *compactada)
{
        BOOL fRes = TRUE;
        int i=0;
        int num_enviados=0;

        num_enviados=strlen(compactada);

        while(i<num_enviados)
        {
                // Aunque WriteFile devuelva un valor no significa que haya terminado
                // de enviar datos por el puerto serie. Luego se comprueba mediante WaitForSingleObject
                if (!WriteFile(hComm, &compactada[i], 1, &iBytesWritten, NULL))
                {
                        // HA HABIDO UN ERROR. ACTIVAMOS LA ALARMA
                        fRes = FALSE;
                }
                else
                        // La operación se completó inmediatamente
                        fRes = TRUE;
                i++;
        }
        return fRes;
}

bool CViperDlg::comprueba_reset()
{
```

```
                uint a,mask;
                vaciar_buffer_y_cola();
                extrae_trama_completa();
                a=*((uint*)(&trama[1]));
                mask=0x30;
                if((a&mask)!=0)        return true;
                        else return false;
}
```

C.5. CViperDlg::OnReceive()

```
LRESULT CViperDlg::OnReceive(WPARAM wParam, LPARAM lParam)
{
        int len;
        TRACE(_T("Paquete UDP recibido\n"));

        if(!estado_sistema_ventana_medida)
        {
                // Si no está abierta la ventana de medida, vaciamos el buffer y rechazamos el datagrama.
                len=m_socket_recibe_datagramas->GetDataSize();
                m_socket_recibe_datagramas->Read((char *)(&bufferRxUdp), len);
                return 0;
        }

        len=m_socket_recibe_datagramas->GetDataSize();// Comprobamos que hay datos en el buffer de entrada

        if(len > 0)
        {
                m_socket_recibe_datagramas->Read((char *)(&bufferRxUdp), len);

                switch(bufferRxUdp.cabecera) // Analizamos la cabecera del paquete que nos ha llegado
                {
                // La cabecera nos marca el siguiente paso a dar el tipo de mensaje UDP recibido
                case ORDEN_RESTART:
                        if(!(pDialogMedidaRMU->IsWindowVisible())) return 0;
                        // Pongo el estado de la RMU en Reseteando y reseteo la IMU
                        estado_sistema=RESETEANDO;
                        if(bufferRxUdp.parametro) // Si se ha solicitado el reset
                        {
                                if (!resetea_IMU())
                                {
                                        bufferTxUdp.cabecera = ERROR_RESET_RMU;
                                        m_socket_envia_datagramas->Send((char *)(&bufferTxUdp),sizeof(bufferTxUdp));
                                        return 0;
                                }
```

```
                        }
                        // Envio a la PMU el ACK
                        bufferTxUdp.cabecera = ACK_ORDEN_RESTART;
                        m_socket_envia_datagramas->Send((char *)(&bufferTxUdp),sizeof(bufferTxUdp));
                        // Compruebo que al IMU se ha reseteado
                        pDialogMedidaRMU->reseteando(bufferRxUdp.parametro);
                        estado_sistema=RESETEADO;
                        break;
                case ACK_ORDEN_RESTART:
                        // Pongo el estado de la PMU en Recibido_ack_orden_restart
                        estado_sistema=RECIBIDO_ACK_ORDEN_RESTART;
                        break;
                case ERROR_RESET_RMU:
                        // Pongo el estado de la PMU en Recibido_ack_orden_restart
                        estado_sistema=RECIBIDO_ERROR_RESET_RMU;
                        break;
                case FIN_RESTART:
                        // Pongo el estado de la PMU en RMU_Reseteda y se lo comunico a la RMU
                        estado_sistema_remoto=RMU_RESETEADA;
                        bufferTxUdp.cabecera = ACK_FIN_RESTART;
                        m_socket_envia_datagramas->Send((char *)(&bufferTxUdp),sizeof(bufferTxUdp));
                        break;
                case ACK_FIN_RESTART:
                        break;
                case ERROR_PPS:
                        // Pongo el estado de la PMU en Error_pps_rmu
                        estado_sistema=ERROR_PPS_RMU;
                        break;
                case CALCULA_PROMEDIO_INICIAL:
                        // Estado de la RMU = Calcula_Promedio y calculo el promedio de los N angulos siguientes a T0
                        estado_sistema=CALCULA_PROMEDIO_INICIAL;
                        pDialogMedidaRMU->calcula_promedio(bufferRxUdp.parametro,false);
                        break;
                case CALCULA_PROMEDIO_FINAL:
                        // Estado de la RMU = Calcula_Promedio y calculo el promedio de los N angulos siguientes a T0
                        estado_sistema=CALCULA_PROMEDIO_FINAL;
                        pDialogMedidaRMU->calcula_promedio(bufferRxUdp.parametro,true);
                        break;
                case PROMEDIO_RMU:
                        // Pongo el estado de la PMU en promedio_rmu_recibido
                        estado_sistema=PROMEDIO_RMU_RECIBIDO;
                        break;
                default:
                        break;
                }
        }
        return 0;
}
```

Apéndice D

CCESocket

D.1. CCESocket::Send()

```
int CCESocket::Send(const char* buf, int len)
{
        int dataPtr = 0;
        int sentBytes = 0;

        if(!m_bWSAStarted)
                return SOCKET_ERROR;

        if(m_socketState < CONNECTED)//No podemos enviar datos si el socket no esta conectado
        {
                m_errorCode = WSAENOTCONN;
                return SOCKET_ERROR;
        }

        if(m_socketState == ACCEPTING && m_socketType == SOCK_STREAM)
        {
                m_errorCode = WSAENOTCONN;
                return SOCKET_ERROR;
        }

        if(!buf || len <= 0)//Comprueba que existe un buffer válido
        {
                m_errorCode = WSAEFAULT;
                return SOCKET_ERROR;
        }
```

```
	switch(m_socketType)
	{
	case SOCK_STREAM:
		while(len > 0)
		{
			sentBytes = send(s, &buf[dataPtr], len, 0);
			if(sentBytes == SOCKET_ERROR)
			{
				m_errorCode = WSAGetLastError();
				return SOCKET_ERROR;
			}
			dataPtr += sentBytes;
			len -= sentBytes;
		}
		break;
	case SOCK_DGRAM:
		if(m_udpReadyToSend)
		{
			while(len > 0)
			{
				if(m_socketState == ACCEPTING)
					sentBytes = sendto(s, &buf[dataPtr], len, 0, (SOCKADDR*) &m_localAddress,
					 sizeof(m_localAddress));
				else
					sentBytes = sendto(s, &buf[dataPtr], len, 0, (SOCKADDR*) &m_remoteAddress,
					 sizeof(m_remoteAddress));
				if(sentBytes == SOCKET_ERROR)
				{
					m_errorCode = WSAGetLastError();
					return SOCKET_ERROR;
				}
				dataPtr += sentBytes;
				len -= sentBytes;
			}

			if(m_readThreadState == CLOSED)
			{
				m_readThreadState = RUNNING;
				m_readThread = CreateThread(NULL, 0, StartThread, this, 0, NULL );
			}
		}
	}
	return dataPtr;
}
```

D.2. CCESocket::Read()

```
int CCESocket::Read(char* buf, int len)
{
        int readBytes;
        DataPacket *data;

        if(!buf || len <= 0)
                return 0;

        readBytes = 0;

        while (m_ReadBuffer.GetNumEntries() > 0 && len > 0)
        {
                data = m_ReadBuffer.GetHead();        //copia la cabecera y la elimina del buffer
                if(len >= data->len)
                {
                        memcpy(&buf[readBytes], data->pos, data->len);
                        len -= data->len;
                        readBytes += data->len;
                        delete[] data->buf;

                        #ifdef _WCE_SECTION
                                EnterCriticalSection(&CCESocket::m_readLock);
                        #else
                                CSingleLock csl(&CCESocket::m_readLock);
                                csl.Lock();
                        #endif

                        delete m_ReadBuffer.RemoveHead();

                        #ifdef _WCE_SECTION
                                LeaveCriticalSection(&CCESocket::m_readLock);
                        #else
                                csl.Unlock();
                        #endif
                }
                else
                {
                        memcpy(&buf[readBytes], data->pos, len);
                        data->len -= len;
                        data->pos += len;
                        readBytes += len;
                        break;
                }
        }
        m_availableData -= readBytes;
```

```
        return readBytes;
}
```